UNIVERSITÀ
DEGLI STUDI
DI PADOVA

DIPARTIMENTO
DI INGEGNERIA
DELL'INFORMAZIONE

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

CORSO DI LAUREA IN INGEGNERIA INFORMATICA

"Virtual private mobile network mediante Wi-Fi"

Relatore: Prof. Stefano Tomasin

Laureando: Simone Roznowicz

ANNO ACCADEMICO 2020 - 2021

Data di laurea: 20 Settembre 2021

Contenuti

Capitolo 1

Introduzione

L’obiettivo della trattazione è quello di analizzare una possibile implementazione di una Virtual Private Mobile Network (VPMN) e, in seguito, di fornirne una versione semplificata. In essa, prima viene creata la connessione peer-to-peer tramite WiFi Direct; poi, viene instaurata una comunicazione semplificata fra tre dispositivi, utilizzando quello centrale per l’inoltro del messaggio.

Lo scopo originario era di sviluppare una vera rete Mesh in cui ogni nodo permetteva un’estensione della connettività inoltrando messaggi ad altri nodi. La fase d'inoltro, in un tale contesto deve essere gestita, idealmente, in modo autonomo dall’applicazione Android mostrando solo all’utente finale l’avvenuta ricezione del messaggio. Gli utenti dei dispositivi intermedi, invece, devono essere all’oscuro del servizio d'inoltro che stanno eseguendo in modo da preservare la riservatezza tra i due utenti in comunicazione.

Con l’avvento della quinta (5G) e ancor più della sesta generazione di reti cellulari (6G) viene sensibilmente migliorata la precisione con cui un utente viene localizzato [1] e viene posta maggiore enfasi sulle questioni legate alla privacy, in quanto sono numerosi modi illeciti applicati da malintenzionati per carpire informazioni [2]. Nella letteratura è già stata affrontata la problematica legata all’utilizzo sconveniente della posizione dell’utente: “*L’informazione della posizione deve essere protetta da un uso improprio della stessa a opera di operatori di rete mobile ma anche di malintenzionati che origliano e che possono usare ricevitori passivi o false stazioni base [3]*” [4].

Inoltre, fornitori di servizi over-the-top (OTT) possono talvolta ricorrere alla posizione dell’utente per scopi commerciali.

Per preservare la privacy dell’utente e renderlo non localizzabile, nella trattazione viene proposta

una soluzione specifica che faccia uso di una VPMN (Virtual Private Mobile Network) la quale è in grado di nascondere la posizione del singolo dispositivo rendendola indistinguibile da quelle di tutti gli altri dispositivi all'interno della VPMN. In essa solo un dispositivo è connesso alla rete mobile; gli altri invece possono avere accesso a Internet richiedendo informazioni, direttamente o indirettamente, a quell'unico dispositivo. In tal modo, l'unica locazione effettivamente rilevabile è quella dell'unico dispositivo che accede alla rete cellulare. Invece, gli altri dispositivi, direttamente o indirettamente connessi ad esso, a seconda dell'estensione della rete Mesh, estendono l'imprecisione nella loro localizzazione potenzialmente all'infinito.

Per realizzare una VPMN viene proposta una soluzione basata sull'utilizzo dello standard di comunicazione WiFi Direct o Wi-Fi P2P (Wi-Fi Peer-to-Peer) che permette a due o più dispositivi di scambiare informazioni se distanti al più 200 m, connettendosi direttamente l'un l'altro senza usare la rete domestica tradizionale, la rete di un ufficio o un hotspot.

Il vantaggio di un approccio basato su WiFi Direct è la creazione di una rete più estesa rispetto all'uso di Bluetooth in cui il singolo messaggio viene direttamente scambiato fra due dispositivi, senza l'intermediario di un access point. Ogni dispositivo all'interno della rete, comportandosi al contempo da client e da server, ha la capacità di ricevere e di inviare messaggi oppure anche di inoltrarli per conto di altri.

Nella trattazione si è proceduto ad analizzare le seguenti sezioni:

- **SCAMBIO DI INFORMAZIONI FRA DISPOSITIVI IN UNA RETE MESH**
    - ♦ **Inoltro del messaggio in una rete Mesh**
    - ♦ **Scelta identificativo**
- **PROGETTI PRESENTI ONLINE E SCELTA PROGETTO DI PARTENZA**
    - ♦ **Rete Mesh e obiettivi del progetto**
    - ♦ **Idea del progetto**
    - ♦ **Tabella riassuntiva progetti online**
    - ♦ **Descrizione progetto scelto**
- **ELABORAZIONE DEL PROGETTO**
    - ♦ **Creazione connessione P2P**
    - ♦ **Tipologia di messaggio**
    - ♦ **Creazione connessione a tre dispositivi**
    - ♦ **Test, risultati e considerazioni**

Scambio di informazioni fra dispositivi in una rete Mesh: Vengono discusse alcune scelte opportune per un'applicazione commerciale. Infatti, essendo ogni dispositivo all'interno di una rete Mesh, viene analizzata la questione di un efficiente inoltro dei messaggi, sia nella scelta dell'algoritmo per identificare il destinatario, sia nella scelta dell'identificativo di ogni dispositivo nella rete.

Progetti presenti online e scelta progetto di partenza: viene discusso l'intento originale e l'obiettivo finale da raggiungere nel progetto. Dopo aver mostrato possibili progetti Android che implementano reti Mesh, disponibili online, viene presentato e descritto il progetto di base scelto. Vengono quindi spiegate le più importanti parti di codice in associazione ai passi eseguiti dall'utente per inviare un messaggio.

Elaborazione del progetto:

Il progetto si compone in due fasi: la prima consiste nel realizzare una connessione fra due dispositivi in modo che possano entrambi scambiarsi informazioni reciprocamente; la seconda, invece, permette di estendere la tradizionale connessione peer-to-peer rendendo possibile un inoltro del messaggio. Infine, viene presentata la modalità con cui sono stati effettuati test e vengono effettuate delle considerazioni sui limiti dell'applicazione realizzata.

Capitolo 2

Scambio di informazioni fra dispositivi in una rete Mesh

2.1 Inoltro del messaggio in una rete Mesh

In una rete Mesh con diversi nodi, può diventare fondamentale determinare un percorso per permettere di raggiungere la destinazione avendo informazioni incomplete sulla struttura generale del grafo.
Un problema fondamentale è, infatti, che il grafo viene costruito dinamicamente: non si conosce la topologia a priori. Il grafo iniziale è costituito dal nodo che invia il messaggio e da quelli ad esso contigui.
Inoltre non è possibile salvare informazioni dinamicamente a livello centralizzato: poiché il codice viene eseguito in modo distribuito, non si può tenere traccia, ad esempio, di tutti i dispositivi che vengono visitati.
A seguito di tali considerazioni sono stati valutati i seguenti algoritmi per l'inoltro del messaggio: l'algoritmo Breadth First Search (BFS), l'algoritmo di Dijkstra e l'algoritmo Depth First Search (DFS).
Un possibile approccio è applicare l'algoritmo BFS che presenta una complessità temporale O(N+E), con N numero di nodi distinti ed E numero di rami presenti nel grafo (rappresentati dalla somma delle connessioni per ogni dispositivo).
Similmente un'altra possibile alternativa consiste nell'utilizzare l'algoritmo di Dijkstra: esso permette di valutare, a differenza di BFS, la distanza fra i diversi nodi del grafo, nonostante il diverso peso dei rami possa assumere poco significato, dato che la distanza massima fra due

dispositivi rimane inferiore a 200 m per limite di WiFi-Direct.

Queste due alternative risulterebbero tuttavia irrealizzabili perché entrambi gli algoritmi si fondano sulla necessità di aggiornare globalmente delle informazioni riguardanti la topologia del grafo (una coda dei dispositivi da visitare al prossimo passo), cosa che si è detta non realizzabile. L'algoritmo DFS, con complessità temporale ancora O(N+E), permette di esaminare il grafo procedendo in profondità. Al contrario dei due algoritmi precedenti non necessita di alcuna informazione globale. Le informazioni possono quindi essere salvate localmente a livello di ogni nodo e pertanto DFS risulta realizzabile. Con tale algoritmo si procede a ispezionare separatamente ogni possibile ramo dell'albero (Figura 2).

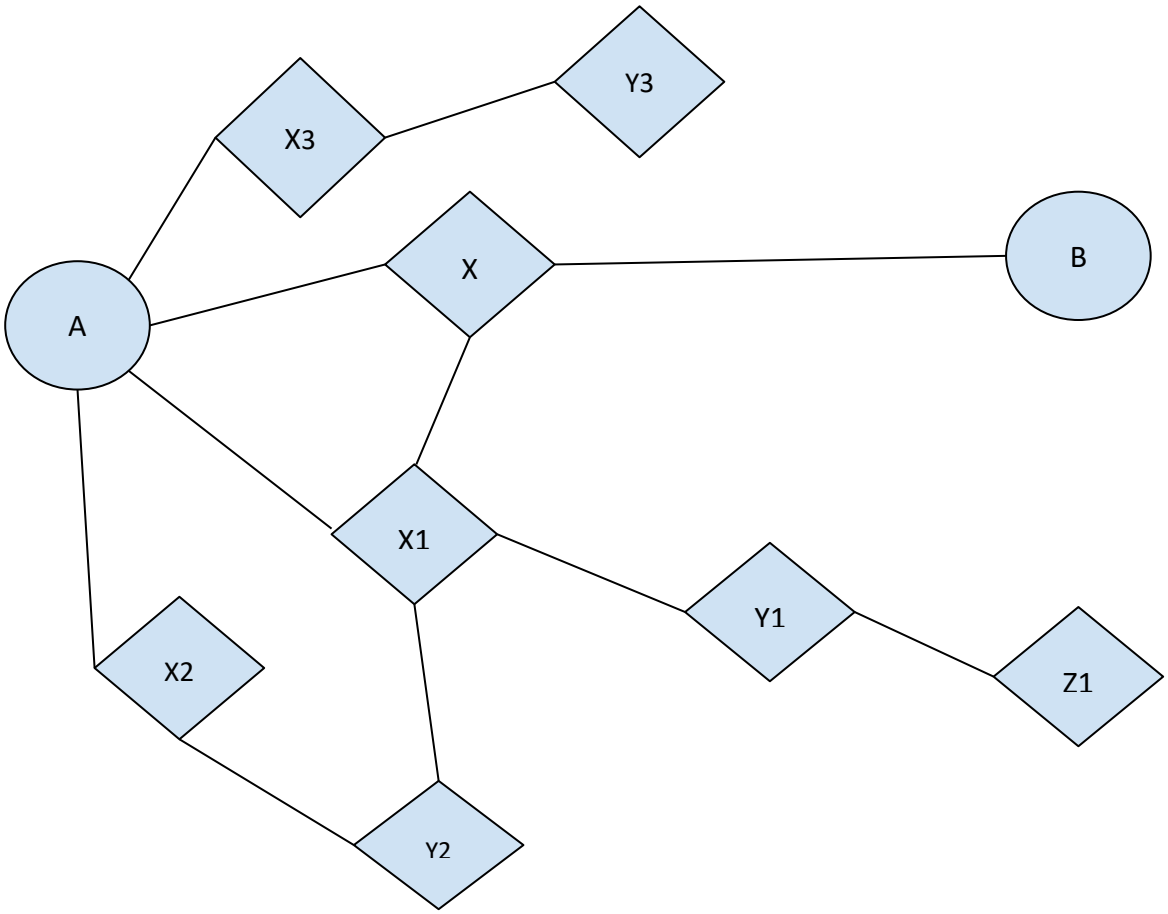


Figura 1: mostra un possibile grafo di una semplice rete Mesh in cui ogni dispositivo è rappresentato da un nodo.

2.2 Scelta identificativo

La scelta dell'identificativo è necessaria per permettere ai dispositivi di interagire fra loro senza equivoci. Nell'API Android viene sconsigliato o impedito di utilizzare identificativi hardware immodificabili (per esempio l'indirizzo MAC oppure il numero di serie).
L'API di Android, invece, suggerisce di utilizzare apposite funzioni e classi che permettano di gestire gli identificativi di ogni utente, all'interno di database.
Anche se non implementata, una soluzione alternativa sarebbe permettere una registrazione dell'utente con indirizzo e-mail (o numero di telefono) e password. Così facendo, ogni dispositivo può essere identificato univocamente. Nel progetto si è assunto, per i test, che il nome dei dispositivi fosse unico e non vi fosse quindi possibilità di equivoco.

Capitolo 3

Progetti presenti online e scelta progetto di partenza

3.1 Rete Mesh e obiettivi del progetto

Originariamente, l'intento principale del progetto era quello di riprodurre, all'interno di un'applicazione Android, una rete Mesh nella quale un solo dispositivo aveva accesso alla rete mobile. In un contesto di applicazione di messaggistica, ciò implicava la possibilità di inoltrare un messaggio un numero indefinito di volte. Tuttavia, la libreria di Android non permette l'immediata creazione di una rete Mesh e, al contrario, specifica chiaramente che Android non supporta ad *hoc mode*. Vengono invece forniti gli strumenti per realizzare una connessione P2P fra due dispositivi oppure per creare un gruppo (si veda la Figura 2), ovvero un insieme di client connessi ad un unico host. L'obiettivo del progetto, pertanto, non è stato la creazione di una vera rete Mesh, bensì di una rete semplificata in cui un'informazione può essere inoltrata una volta.

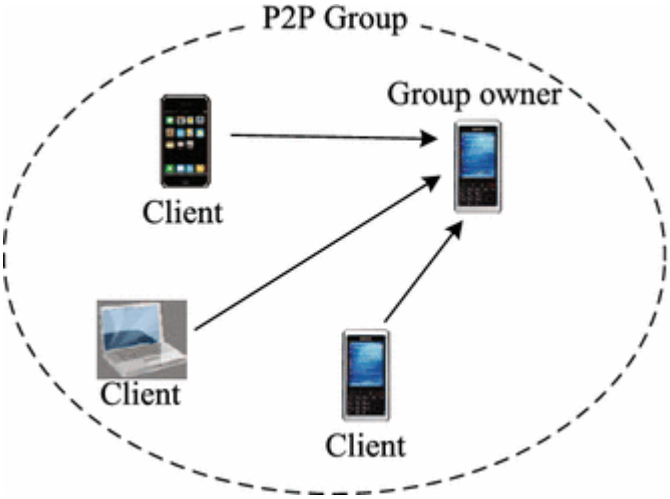


Figura 2: mostra il concetto di P2P group, per cui uno o più dispositivi, detti client, entrano in connessione con un unico dispositivo host, ovvero il group owner.

3.2 Idea del progetto

Nella realizzazione del progetto Android, si è posto l'obiettivo di permettere almeno un inoltro del messaggio. Come si può vedere in Figura 3, dati tre dispositivi, A, X e B, in cui l'applicazione è in esecuzione, l'obiettivo è mandare un messaggio da A a B nonostante i due dispositivi non riescano a rilevarsi a vicenda perché troppo distanti. Il dispositivo X, che si suppone essere in grado di raggiungere sia A che B, funge da intermediario permettendo l'inoltro del messaggio.
In tal modo viene di fatto estesa la connettività WiFi Direct (che in via teorica può raggiungere al più 200 m). In realtà, a causa di condizioni esterne od ostacoli fisici, il raggio di estensione di tale standard è spesso molto minore.
In Figura 4 si può vedere una possibile rete realizzabile con il progetto implementato.

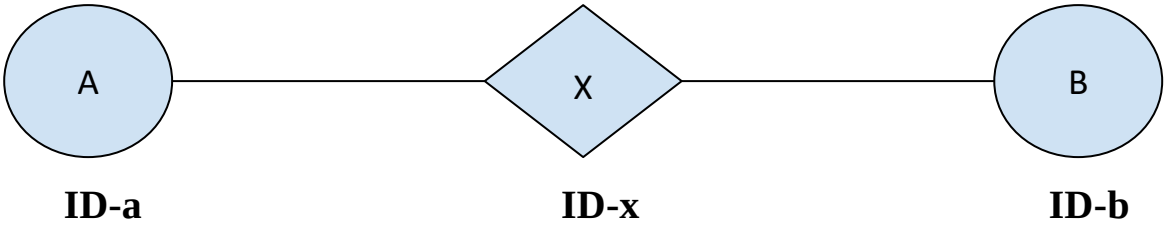


Figura 3: mostra tre possibili dispositivi in cui: A e B non possono rilevarsi a vicenda dato che troppo distanti. X invece funge da mediatore fra i due estendendo la connettività di WiFI Direct.

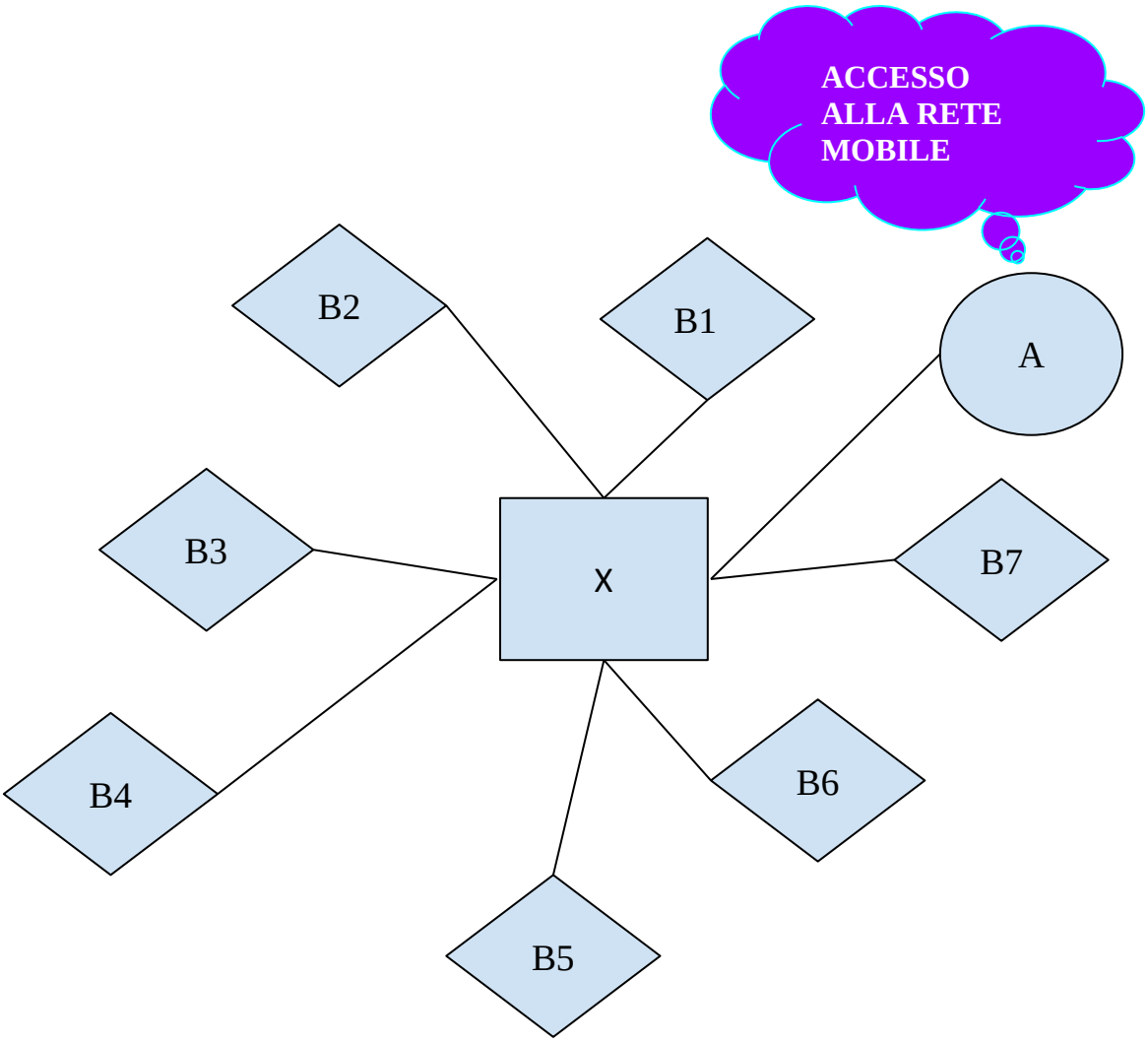


Figura 4: mostra una possibile configurazione realistica con l'applicazione realizzata. Ogni dispositivo Bn (con n intero variabile con valore massimo non definito) può entrare in contatto con il dispositivo A, direttamente connesso alla rete mobile, tramite il dispositivo mediatore X.

3.3 Tabella riassuntiva progetti online

La seguente tabella riassume i principali progetti che implementano reti mesh in applicazioni Android (facendo uso di Wi-Fi o Wi-Fi Direct, senza l'ausilio di altri dispositivi che non siano telefoni cellulari, tablet), visibili online.

Nome	Link	Attività	Disponibilità sorgente
RightMesh	https://rightmesh.io/ https://github.com/RightMesh https://appdevelopermagazine.com/mobile-mesh-networking-apps-via-new-sdk-from-rightmesh/	azienda privata: diversi progetti che fanno uso di reti Mesh	Viene messa a disposizione di utenti autorizzati una libreria apposita per gestire reti Mesh
MeshNetworking_Android-Wi-Fi-Direct	https://github.com/mmoyaferrer/MeshNetworking_Android-Wi-Fi-Direct	MANET: progetto di tesi	gratuito e disponibile con specifica licenza
echo	https://github.com/Breakend/echo	progetto personale, applicazione di messaggistica	gratuito e disponibile
Open Garden	https://opengarden.com/	azienda privata (app presente nel play store)	non disponibile
Briar	https://code.briarproject.org/briar/briar	open source (app presente nel play store) Applicazione di messaggistica	gratuito e disponibile
Automatic Android-based Wireless Mesh Networks	http://www.informatica.si/ojs-2.4.3/index.php/informatica/article/viewFile/713/583	progetto spiegato nell'articolo pubblicato	non disponibile
The Serval Project	https://github.com/servalproject/batphone http://www.servalproject.org/	differenti progetti che implementano reti mesh. Serval Mesh (o Batphone) permette di condividere chiamate vocali phone-to-phone, SMS e file attraverso Wi-Fi	gratuito e disponibile
Underdark android	https://github.com/udark/underdark-android	libreria di connessione peer-to-peer per Android, con	gratuito e disponibile

		supporto Wi-Fi e Bluetooth	

Per la realizzazione della rete mesh esemplificativa, si è cercato un progetto Android open source che implementasse una comunicazione peer-to-peer tra due dispositivi, permettendo la ricezione e l'invio dei messaggi da e verso entrambi i lati in questione. Tuttavia, dopo aver testato numerosi progetti open source parzialmente o completamente non funzionanti, si è optato per il progetto che garantiva maggiore stabilità, anche se incompleto. (Di seguito è riportato il link al progetto GitHub: https://github.com/akshay-ap/WifiP2P).
Tale progetto permette infatti una comunicazione P2P solo parziale: solo un dispositivo, ovvero il group owner che corrisponde all'host, può ricevere i messaggi; analogamente l'altro dispositivo, il client, può inviare messaggi ma non riceverne.
L'invio del messaggio si basa essenzialmente sulla creazione di un socket di comunicazione fra client e host. Tale processo, che avviene nella classe client, è reso possibile utilizzando la seguente funzione di libreria che restituisce l'indirizzo IP del group owner (generalmente il group owner di ha indirizzo IP 192.168.49.1).
Essendo il server in ascolto, non appena sopraggiunge una richiesta di connessione dal client, essa viene accettata dal server.

3.4 Descrizione progetto scelto

Il progetto di partenza, sopra menzionato, si articola nei seguenti passaggi:

- *La ricerca di dispositivi nelle vicinanze:* la funzione discoverPeers(), se giunge a buon fine, permette di iniziare a ricercare dispositivi adiacenti. In caso di insuccesso, invece, viene segnalata l'impossibilità di iniziare la ricerca tramite la comparsa di una notifica. I motivi del fallimento possono essere l'incapacità del dispositivo di supportare lo standard WiFi Direct (P2P UNSUPPORTED), un errore interno (ERROR) oppure il sistema occupato (BUSY).

- *La selezione di un dispositivo dall'apposita lista:* nel momento in cui vengono rilevati dei dispositivi, viene mostrato il loro nome in una lista interattiva con l'utente. Fra questi è possibile selezionare il dispositivo con cui avrà poi inizio la connessione.

- *La connessione al dispositivo scelto:* tramite il pulsante “connect” vengono connessi il dispositivo su cui viene eseguito il codice e il dispositivo selezionato fra la lista dei disponibili.

- *La configurazione:* tramite il pulsante “configure” vengono richieste informazioni riguardanti la connessione instaurata e viene stabilito il dispositivo group owner, quindi l’host, e di conseguenza l’altro che sarà client.

- *L’avvio di un nuovo oggetto ServerSocketThread:* premendo sul pulsante “server start”, l’host rimane in ascolto in background, accettando socket creati dal client che hanno come parametro host l’indirizzo IP del server stesso.

- *L’avvio di un nuovo oggetto ClientSocket:* in background, nel momento in cui viene premuto il pulsante “send string” viene creato un nuovo socket che viene connesso al server tramite la funzione connect:
  nomeSocket.connect((new InetSocketAddress(GrupOwnerIP, port)), timeout);
  La stringa *GroupOwnerIP*, che restituisce l’indirizzo IP del group owner in una rete di telecomunicazione, viene ottenuta precedentemente in fase di configurazione. Pertanto, dato che l’indirizzo IP del group owner è sempre disponibile, la connessione e l’invio di messaggi dal client al server risulta semplice e immediata.
  A livello utente, premendo il pulsante “send string” sul dispositivo client, il messaggio scritto nell’apposita casella di testo sottostante, viene mandato al server che, ricevuto il messaggio, provvede a proiettarlo nello spazio dedicato alla ricezione dei messaggi.

Capitolo 4

Elaborazione del progetto

4.1 Creazione connessione P2P

Il progetto originario, appena descritto, è stato significativamente modificato in modo da implementare le seguenti specifiche:

- Creare una connessione P2P completa, ossia che permetta di scambiare messaggi da client a server e viceversa.
- Creare una rete mesh semplificata in cui il group owner può provvedere a inoltrare messaggi ricevuti da un client a un altro. Quindi possono esserci più client, che non riescono a rilevarsi a vicenda ma che sono connessi a un unico server.

Precedentemente erano presenti solo le classi ServerSocketThread e ClientSocket che avevano il compito di permettere la ricezione dei messaggi e il loro invio, rispettivamente. Ora sono state introdotte le classi ServerSocketThreadSender e ClientSocketListener. La prima scrive sul socket creato dal client e invia il messaggio all'altra estremità del socket, come in Figura 5.

```
1.  public void serverSendMessage() throws IOException {
2.          dataOutputStream = new DataOutputStream(socket.getOutputStream());
3.          System.out.println("writeUTF");
4.          dataOutputStream.writeUTF(message + '\n');
5.          System.out.println("message == " + message);
6.          dataOutputStream.flush();
7.          System.out.println("flush");
8.  }
```

Figura 5: viene mostrata la fase di scrittura sul socket che avviene all'interno della funzione serverSendMessage(). Quando l'host necessita di inviare un messaggio al client, crea un oggetto

ServerSocketThreadSender ed esegue tale funzione in un thread in background. Il messaggio viene scritto sull'outputStream del socket attualmente in funzione, e reso visibile a all'oggetto ClientSocketListener.

Eseguendo la seconda, invece, ogni 500 millisecondi (scelta di tempo atta a minimizzare interferenze con altre parti di codice) viene controllata l'eventuale presenza nell'inputStream di messaggi, spediti dall'altro capo del socket, come visibile in Figura 6. Se è presente del contenuto, esso viene estratto e, dopo un'analisi dell'informazione, viene proiettato nello spazio dedicato alla ricezione dei messaggi.

```
1.      private void listen() throws IOException, InterruptedException {
2.          while(!interrupted) {
3.              Thread.sleep(500);
4.              while(!MainActivity.socket.isConnected()){
5.                  //System.out.println("dentro isconnected while");
6.              }
7.              try {
8.                  inputstream = MainActivity.socket.getInputStream();
9.              } catch (SocketException e) {
10.                 e.printStackTrace();
11.             }
12.             System.out.println("ClientSocketListener: step 1: numero bytes da
    leggere inputstream == " + inputstream.available());
13.             String line = "";
14.             bufferedReader = new BufferedReader(new InputStreamReader(in-
    putstream));
15.             if(inputstream.available()>2) {
16.                 System.out.println("readline");
17.                 line = bufferedReader.readLine();
18.                 System.out.println("readline finished");
19.                 System.out.println("line == " + line);
20.                 if(line.length()>2) {
21.                     line = line.substring(2);
22.                 }
23.             }
24.             if (!line.equals("")) {
25.                 listener.onUpdate(line);
26.             }
27.         }
28.         interrupted = false;
29.     }
```

Figura 6: viene mostrata la funzione listen() all'interno della classe ClientSocketListener. Nel momento in cui il client necessita di inviare un messaggio, viene creato (o svrascritto se ne era già presente uno) un socket che viene accettato dal server. Succesivamente viene creato un oggetto ClientSocketListener e, in un thread in background, viene eseguita la funzione listen(). Essendo il codice all'interno di un ciclo while, finché interrupted non diventa true, ogni 500 ms viene controllata la presenza in inputStream di un eventuale messaggio.

Viene quindi creata una connessione P2P completa, basandosi sulla seguente idea: l'invio dei messaggi inizia sempre dal client che crea un socket di connessione al server per inviare un messaggio. Tale socket, accettato dal server, rimane aperto e connesso fino all'invio del prossimo messaggio da parte del client; a quel punto verrà chiuso e subito sovrascritto da un nuovo socket. Fra la connessione e la distruzione del socket, in ogni momento, anche il server può scrivere sul socket esistente inviando messaggi al client.

Il pulsante "configure" non ha più la stessa funzionalità del progetto originale. Infatti, la richiesta di informazioni sullo stato della connessione e la decisione su quale dispositivo sia owner viene eseguita in precedenza premendo sul pulsante "connect".

Pertanto, la configurazione diventa necessaria solo per i dispositivi client e consiste nell'invio di un messaggio predefinito (scelto essere *~~connectionMessage~~*) che, da quel momento in poi, permetterà anche al server di inviare messaggi al client grazie al socket appena creato. Anche la funzionalità del pulsante "server start", ovvero l'esecuzione in background di un oggetto ServerSocketThread, è stata spostata all'interno del codice che viene eseguito premendo sul pulsante "connect".

Il messaggio ricevuto dall'utente è colorato in rosso ed è visibile, come si può vedere in Figura 7 e in Figura 8, nel seguente formato:
nome_mittente>> contenuto_messaggio

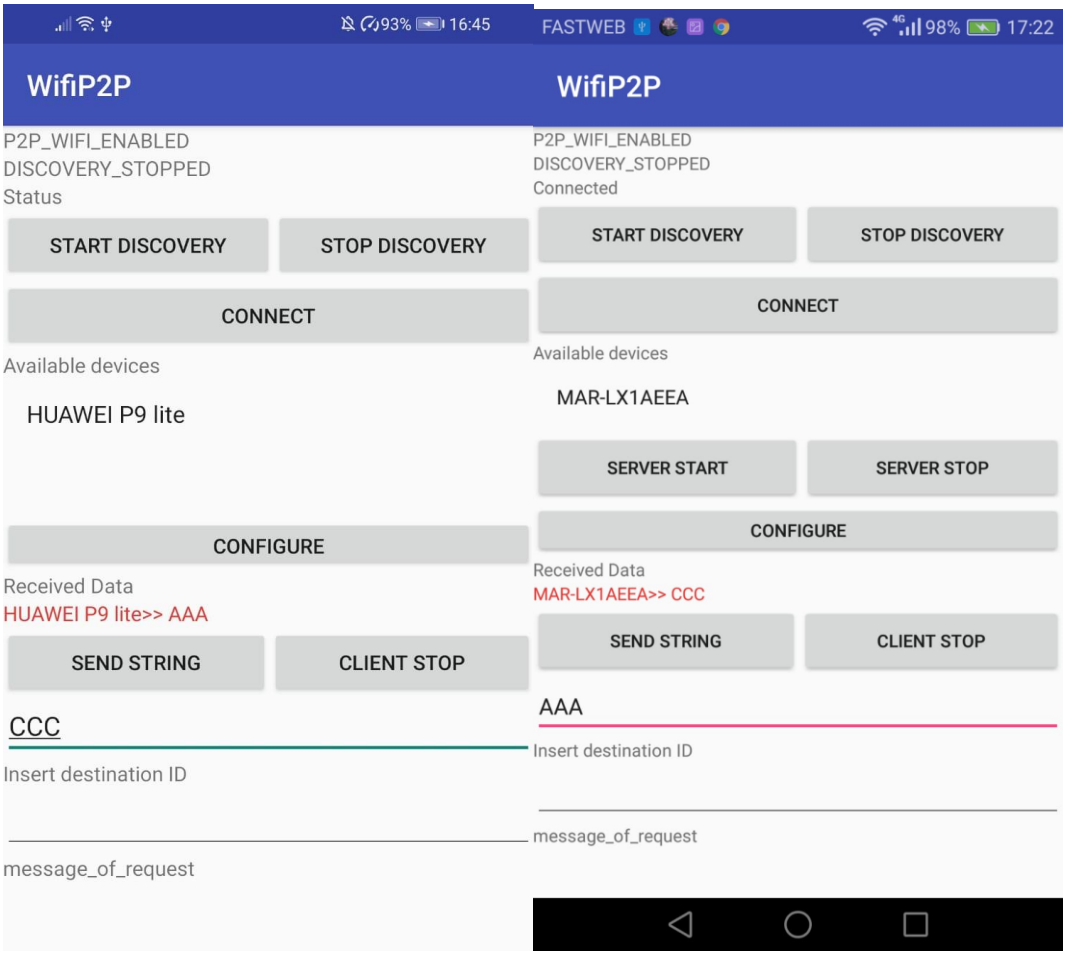


Figura 7 (sinistra) e Figura 8 (destra): mostrano rispettivamente i dispositivi client e host, capaci di ricevere e inviare messaggi, tramite WiFi Direct, in una connessione peer to peer completa.

4.2 Tipologia di messaggi

Dall'invio alla ricezione, il messaggio originale, scritto dall'utente, può subire delle modifiche e successivamente venire analizzato per ricavare informazioni aggiuntive. Infatti, il messaggio digitato dall'utente, se destinato a un dispositivo direttamente connesso, rimane invariato. Invece, se il messaggio non ha come destinatario un dispositivo adiacente, viene definito *request*: in tal caso il messaggio originale, *data1*, viene rielaborato come segue:

data = "$$" + nameOrigin + "||" + nameDestination + "::" + visitedDevices + "~~" + data1 + '\n';

Viene ora mostrato un esempio di messaggio *request* rielaborato:
$$Huawei p10 lite||Simo_Roz::Huawei p10 lite++device1++device2++device3~~Hi, how are you?

Pertanto, *data*, il messaggio rielaborato, è il risultato della concatenazione di stringhe e caratteri separatori:

"$$": due caratteri separatori che indicheranno l'inizio di un messaggio *request* (che il ricevente dovrà trattare diversamente, perchè destinato a essere inoltrato).

nameOrigin: nome del dispositivo che invia il messaggio. Tale nome verrà poi proiettato sul dispositivo del destinatario per indicare il mittente.

"||": due caratteri separatori che indicano il termine del nome dell'origine e l'inizio del nome di destinazione.

nameDestination: nome del dispositivo destinatario finale del messaggio. Tale nome, nel progetto, non è univoco e pertanto non ammissibile in un'applicazione commerciale. Si veda il paragrafo riguardante la scelta dell'identificativo per una scelta opportuna da sostituire al nome.

"::": due caratteri separatori che indicano il termine del nome di destinazione e l'inizio dell'elenco dei dispositivi visitati prima di raggiungere la destinazione.

visitedDevices: è una lista, costituita da una stringa che a ogni inoltro di messaggio si allunga, includendo il nuovo dispositivo da cui è transitata l'informazione. In questo progetto, poiché vi è solo un intermediario, ovvero l'host, la lista conterà alla fine tre dispositivi: quello d'origine, quello dell'unico intermediario e quello del client destinatario. Il nome di ogni dispositivo è separato dal successivo da due caratteri separatori: "++".

"~~": due caratteri separatori che indicano il termine dell'elenco dei dispositivi visitati e l'inizio del contenuto del messaggio. Il messaggio è infine seguito dal carattere di fine riga: '\n'.

4.3 Creazione connessione a tre dispositivi

Come accennato in precedenza, sono state definite due modalità con cui un messaggio viene inviato:

- La prima, più semplice, permette di inviare un messaggio da un client a un server o viceversa. Il destinatario finale è il dispositivo a cui il mittente è direttamente connesso.
- La seconda permette la connessione fra tre dispositivi. E' stata definita *request* in quanto consiste nell'invio di un messaggio preceduto dai due caratteri speciali "$$" che indicano che il destinatario finale del messaggio non è direttamente raggiungibile e pertanto l'host avrà il compito di inoltro.

Per procedere a realizzare una connessione a tre dispositivi, si crea prima una connessione P2P fra l'host e i due dispositivi che devono scambiarsi i messaggi, come in Figura 10, in Figura 11 e in Figura 12.

Un utente che voglia inviare un messaggio *request* deve poi procedere come segue: digita il nome del dispositivo senza spazi finali (nel progetto deve quindi essere conosciuta a priori la denominazione del telefono) nella casella di testo sopra a "message_of_request". Digita inoltre il contenuto del messaggio nella solita casella di testo, sotto al pulsante "send string".

Successivamente, inviando il messaggio, viene rilevata la scrittura di un destinatario da parte dell'utente. Pertanto, viene prima cercato, all'interno della lista dei dispositivi rilevati, se è presente quello richiesto dall'utente. Se così fosse, il messaggio verrebbe direttamente inviato al dispositivo. Altrimenti, poichè il dispositivo destinatario non è nel raggio di estensione di WiFi Direct, viene inviato all'host un messaggio *request*. Si veda la Figura 9. Il group owner, riconosciuto il messaggio request, ne proietta il contenuto integrale all'interno della casella "message_of_request". A questo punto il vero destinatario del messaggio potrà accedere al contenuto del messaggio: potrà quindi inviare un messaggio predefinito (che, essendo sempre *~~connectionMessage~~*, può essere inviato premendo nuovamente sul pulsante "configure" del dispositivo destinatario) ottenendo come risposta il messaggio integrale, come si vede in Figura 13, in Figura 14 e in Figura 15. Infatti, alla ricezione del messaggio standard, il server, in ascolto tramite l'oggetto ServerSocketThread, risponde automaticamente inoltrando il messaggio richiesto.

Infine, ricavando il nome del dispositivo originario dal messaggio integrale, verrà proiettato al destinatario il seguente:

nome_mittente>> contenuto_messaggio

La stringa *nome_mittente* indica il nome del dispositivo originario che ha inviato il messaggio.

```
1.                 requestDestination = EditIDDestination.getText().toString();
2.                 String dataToSend = editTextTextInput.getText().toString();
3.                 if(device == null && requestDestination.equals("")) {
   //Didn't select anything, didn't write anything
4.                     Toast.makeText(MainActivity.this,"Please discover and select
   a device OR write the destination below",Toast.LENGTH_SHORT).show();
5.                     return;
6.                 }
7.                 else if(!requestDestination.equals("")) {  //Didn't select a de-
   vice, wrote destination
8.                     //ottenere la lista di dispositivi e connettersi a tutti
9.                     boolean found = false;
10.                    for(int i = 0; i < deviceListItems.length; i++) {
11.                        if(deviceListItems[i].deviceName.equals(requestDestina-
   tion)){      //what was written in the request was, in reality, the name of a de-
   vice present in the list
12.                            device = deviceListItems[i];
13.                            System.out.println("mi connetto a un dispositivo " +
   device.deviceName);
14.                            connect(device);
   //connect to the device, then send the message
15.                            isRequest = false;
16.                            found = true;
17.                            if (!dataToSend.equals("")) {
18.                                if (!IS_OWNER) {
19.                                    closeSocketAndInterrupt();
20.                                    ClientSocket clientSocket = new Clien-
   tSocket(MainActivity.this, this, dataToSend, isRequest, null, null, null, clien-
   tSocketListener);
21.                                    clientSocket.executeOnExecutor(Async-
   Task.THREAD_POOL_EXECUTOR);
22.                                    startClientSocketListener();
23.                                } else {//IS_OWNER
24.                                    ServerSocketThreadSender serverSocketThread-
   Sender = new ServerSocketThreadSender(MainActivity.this, this, serverSock-
   etThread.client, dataToSend, isRequest, null, null, null);
```

```
25.                                     serverSocketThreadSender.executeOnExecu-
tor(AsyncTask.THREAD_POOL_EXECUTOR);
26.                                 }
27.                             }
28.                             break;
29.                         }
30.                     }
31.                     if(!found) {
32.                         Toast.makeText(MainActivity.this,"Request is being for-
warded to all available devices 10",Toast.LENGTH_SHORT).show();
33.                         System.out.println("Request is being forwarded to all
available devices 1");
34.                         for (int i = 0; i < deviceListItems.length; i++) {
35.
36.                             if(deviceListItems[i].deviceName.equals("WAS-LX1A")
|| deviceListItems[i].deviceName.equals("HUAWEI P10 lite")) {
37.                                 System.out.println("SALTO HUAWEI P10 LITE");
38.                                 continue;
39.                             }
40.                             device = deviceListItems[i];
41.                             System.out.println("mi connetto a una caterva di di-
spositivi!!!!!!!!! " + device.deviceName);
42.                             System.out.println("******* 4");
43.                             connect(device);
//connect to each device, then send the message to all of them
44.                             isRequest = true;
45.                             String firstDeviceName = Set-
tings.Global.getString(getContentResolver(), Settings.Global.DEVICE_NAME);
46.                             seenDevices = firstDeviceName;
47.                             seenDevicesUpdate();
48.                             if (!dataToSend.equals("")) {
49.                                 if (!IS_OWNER) {
50.                                     System.out.println("mi connetto a " + de-
vice.deviceName);
51.                                     closeSocketAndInterrupt();
52.                                     ClientSocket clientSocket = new Clien-
tSocket(MainActivity.this, this, dataToSend, isRequest, requestDestination,
firstDeviceName, seenDevices, clientSocketListener);
53.                                     clientSocket.executeOnExecutor(Async-
Task.THREAD_POOL_EXECUTOR);
54.                                     startClientSocketListener();
55.                                 } else {//IS_OWNER
```

```
56.                                     System.out.println("mi connetto a " + de-
   vice.deviceName);
57.                                     ServerSocketThreadSender serverSocketThread-
   Sender = new ServerSocketThreadSender(MainActivity.this, this, serverSock-
   etThread.client, dataToSend, isRequest, firstDeviceName, requestDestination,
   seenDevices);
58.                                     serverSocketThreadSender.executeOnExecu-
   tor(AsyncTask.THREAD_POOL_EXECUTOR);
59.                                 }
60.                             }
61.                         }
62.                     }
63.                 }
64.                 else {                                                  //Selected
   a device, didn't write anything
65.                     System.out.println("******* 5 GIUSTO");
66.                     isRequest = false;
67.                     if (!dataToSend.equals("")) {
68.                         if (!IS_OWNER) {
69.                             closeSocketAndInterrupt();
70.                             clientSocket = new ClientSocket(MainActivity.this,
   this, dataToSend, isRequest, null, null, null, clientSocketListener);
71.                             clientSocket.executeOnExecutor(Async-
   Task.THREAD_POOL_EXECUTOR);
72.                             startClientSocketListener();
73.                         }
74.                         else{//IS_OWNER
75.                             dataToSend = editTextTextInput.getText().toString();
76.                             ServerSocketThreadSender serverSocketThreadSender =
   new ServerSocketThreadSender(MainActivity.this, this, serverSocketThread.client,
   dataToSend, isRequest, null, null, null);
77.                             serverSocketThreadSender.executeOnExecutor(Async-
   Task.THREAD_POOL_EXECUTOR);
78.                         }
79.                     }
80.                 }
```

Figura 9: mostrano il codice che viene eseguito quando l'utente preme sul pulsante "send string". Si distingue il caso in cui il nome del dispositivo scritto dall'utente come destinazione è in realtà presente nella lista dei dispositivi adiacenti; il caso in cui, non essendo presente, viene inviato un messaggio di request a un dispositivo adiacente e infine il caso in cui non è stato

scritto il nome di alcun dispositivo e, pertanto, viene inviato un messaggio semplice al dispositivo selezionato dall'utente fra quelli nell'apposita lista.

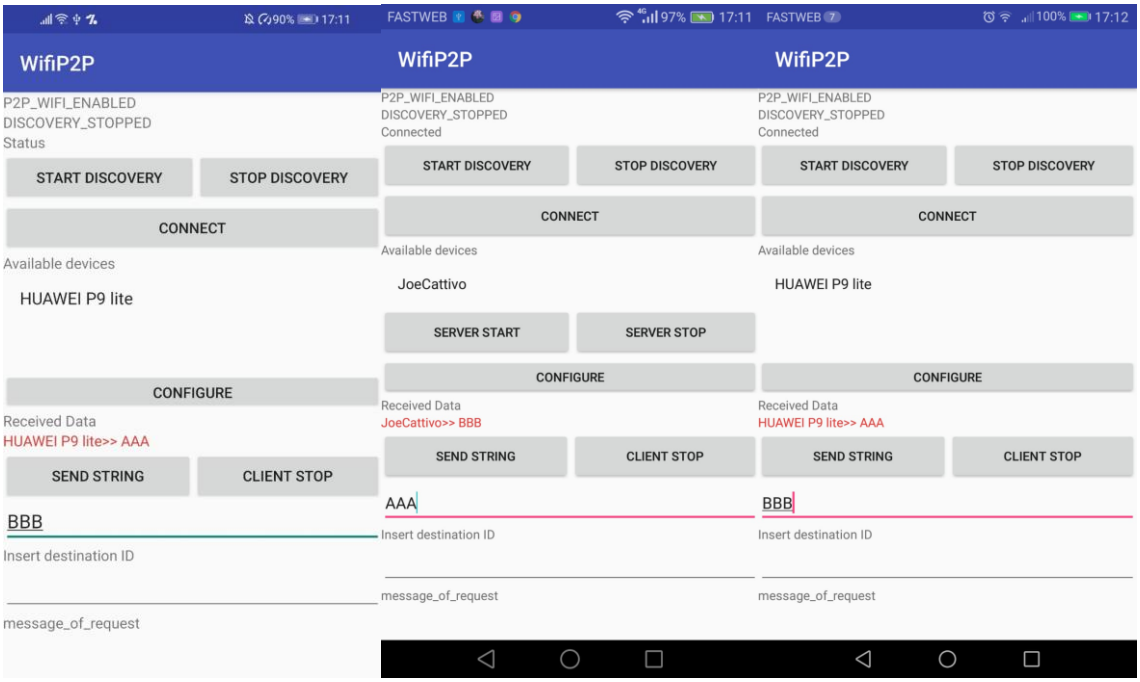
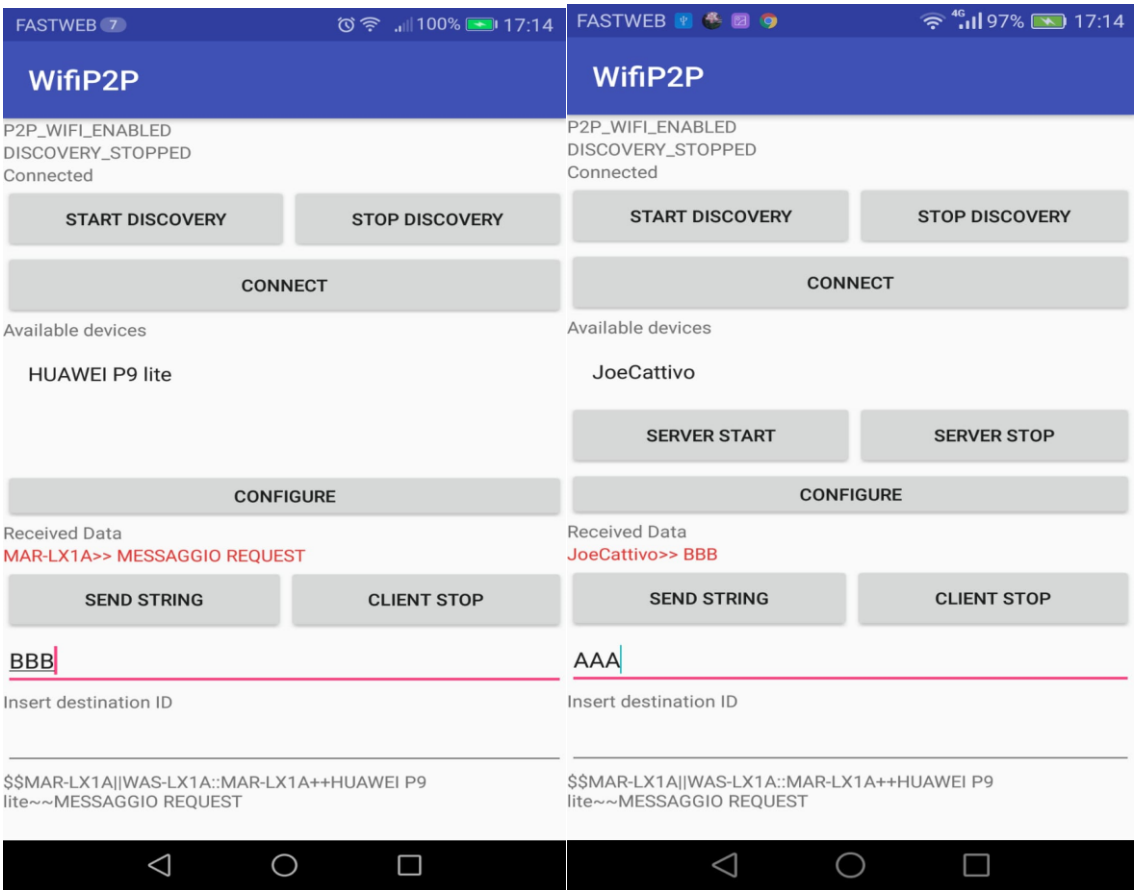


Figura 10 (sinistra) e Figura 11 (centro) e Figura 12 (destra): per procedere a realizzare una connessione a tre dispositivi, si crea prima una connessione P2P fra quello centrale e i due a destra e a sinistra.


FASTWEB 7
100% 17:14
WifiP2P
P2P_WIFI_ENABLED
DISCOVERY_STOPPED
Connected
START DISCOVERY
STOP DISCOVERY
CONNECT
Available devices
HUAWEI P9 lite
CONFIGURE
Received Data
MAR-LX1A>> MESSAGGIO REQUEST
SEND STRING
CLIENT STOP
BBB
Insert destination ID
$$MAR-LX1A||WAS-LX1A::MAR-LX1A++HUAWEI P9 lite~~MESSAGGIO REQUEST
FASTWEB
97% 17:14
WifiP2P
P2P_WIFI_ENABLED
DISCOVERY_STOPPED
Connected
START DISCOVERY
STOP DISCOVERY
CONNECT
Available devices
JoeCattivo
SERVER START
SERVER STOP
CONFIGURE
Received Data
JoeCattivo>> BBB
SEND STRING
CLIENT STOP
AAA
Insert destination ID
$$MAR-LX1A||WAS-LX1A::MAR-LX1A++HUAWEI P9 lite~~MESSAGGIO REQUEST

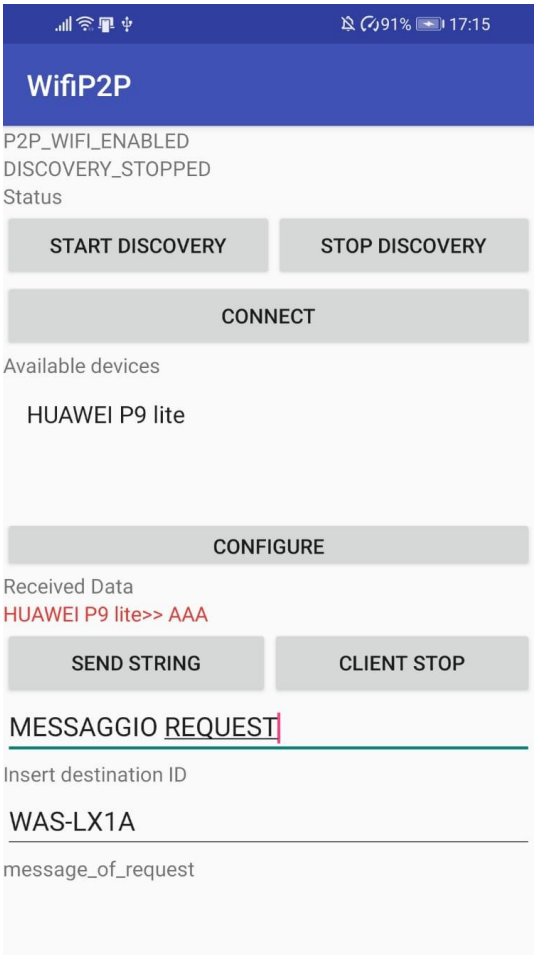


Figura 13 (sinistra) e Figura 14 (centro) e Figura 15 (in basso): l'utente scrive il nome del dispositivo a cui inviare il messaggio e il contenuto dello stesso. Dopo aver premuto sul pulsante "send string", il messaggio rielaborato compare scritto, nello spazio dedicato alle request, all'interno del dispositivo centrale. Infine, quando l'utente destinatario (sinistra) preme sul pulsante configure, ottiene in risposta il messaggio opportuno, preceduto dal mittente reale.

4.4 Test, risultati e considerazioni

Alla prima installazione, prima di utilizzare l'applicazione, per dispositivi muniti di sistema operativo Android 10 o più recente, è necessario accedere alle impostazioni del dispositivo e fornire manualmente il permesso alla locazione [5].

Impostazioni → App → selezionare l'applicazione WifiP2P → Autorizzazioni → consentire l'autorizzazione alla posizione (si veda Figura 15 e Figura 16).

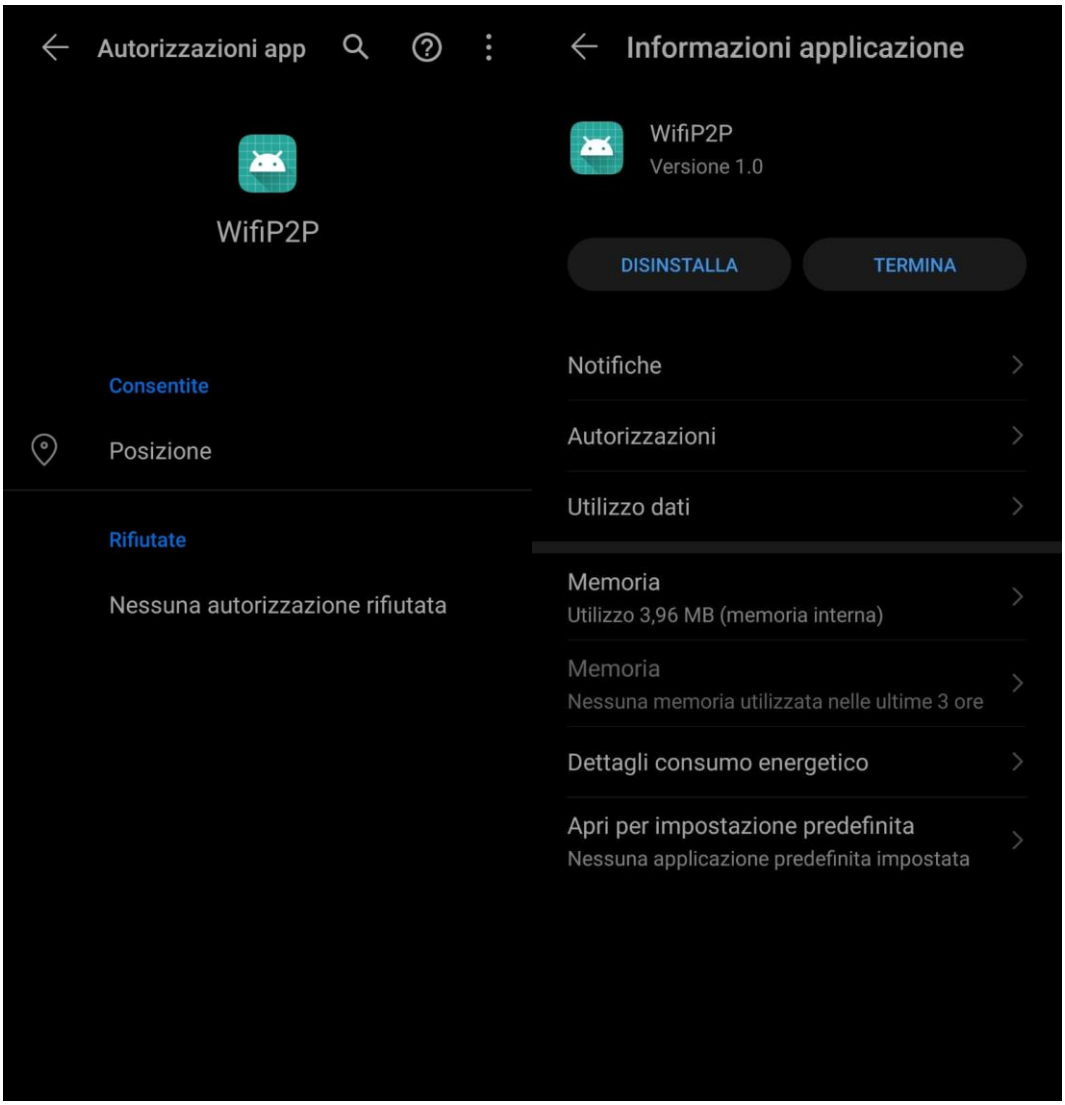


Figura 15 (sinistra) e Figura 16 (destra): mostrano l'autorizzazione alla posizione che è stata concessa dall'utente all'applicazione WifiP2P accedendo al menù delle impostazioni.

Nella fase di test, si sono utilizzati telefoni cellulari Android reali, senza fare ricorso ai simulatori virtuali messi a disposizione dall'ambiente di sviluppo di Android Studio. Infatti, essi non permettono di rilevare dispositivi adiacenti tramite WiFi Direct a causa di firewall che impediscono di fatto la connessione con altri dispositivi.
I dispositivi utilizzati sono:

- HUAWEI P10 lite (MAR-LX1A con nome personalizzato Sim_Roz)
- HUAWEI P10 lite (WAS-LX1A)
- HUAWEI P9 lite (HUAWEI VNS-L21)

Per testare con maggiore facilità si sono utilizzati i tre dispositivi uno a fianco all'altro. Si è simulato il distanziamento fra gli stessi permettendo solo a un dispositivo, il group owner, di interagire con entrambi.

Il programma riscontra alcuni problemi di funzionamento spesso legati a una connessione non sempre affidabile. Talvolta, ciò porta a non riuscire a connettersi subito e a riprovare alcune volte prima di potersi scambiare messaggi. Principalmente, la poca affidabilità del sistema si riscontra, seppur non sempre, con la funzione "connect". Talvolta, infatti, la connessione del socket fra client e server fallisce, impedendo l'invio del primo messaggio.

L'affidabilità nell'instaurazione della connessione è leggermente migliorata una volta aumentato il parametro di timeout, che corrisponde al tempo massimo che viene concesso alla funzione per realizzare la connessione. Tale problema si riscontra quasi solo in fase di configurazione, infatti, una volta creato il primo socket, la successiva sovrascrittura dello stesso non desta problematiche. Inoltre, dato che, come detto, la chiusura del socket è subito seguita da una sovrascrittura dello stesso, quasi mai si verificano problemi di concorrenza (ad esempio che il server scriva su un socket che non è ancora stato creato. Tuttavia, per un'applicazione commerciale, bisognerebbe ovviamente tenere conto della problematica, provvedendo a renderla del tutto impossibile). Nei rari casi in cui ciò si verifica, si è scelto di fare in modo che il client riceverà un messaggio vuoto e, pertanto, il server dovrà rinviare il messaggio.

Capitolo 5

Conclusioni

Il progetto realizzato, seppur con dei difetti di connettività, permette di scambiare informazioni fra tre dispositivi diversi, avendo il mittente e il destinatario abbastanza distanti da non rilevarsi a vicenda. Il tutto viene implementato tramite WiFi Direct in un'applicazione Android. Tuttavia, l'API di Android non permette la creazione di una rete di comunicazione *ad hoc* ma, al contrario, si fonda sulla presenza di un host, ovvero il group owner, e uno o più client. Con tale consapevolezza, come ultimo tentativo, si è valutato di aggirare il problema, legato al fatto di avere un solo group owner (e quindi un solo indirizzo IP di riferimento), creando connessioni successive per permettere l'inoltro dei messaggi. In tal modo, una volta giunto il messaggio a un nodo mediatore, la connessione sarebbe stata interrotta e ne sarebbe stata instaurata un'altra, continuando a inoltrare il messaggio fino al raggiungimento del destinatario. Tuttavia, anche questa opzione è stata scartata sia perché molto spesso sarebbe stato necessario l'intervento dell'utente per accettare la connessione sia perché il sistema sarebbe entrato frequentemente in stato *busy*. Risultano pertanto chiare le difficoltà nel realizzare una rete di comunicazione non tradizionale, come quella Mesh.

Infatti, il progetto realizzato non ne crea effettivamente una ma si limita a estendere la connettività di WiFi Direct, dando comunque origine a una rete più ampia della semplice connessione fra due dispositivi. Sarebbe necessaria un'architettura apposita, diversa da quella attuale di Android, che offra funzionalità integrate per una facile ed efficiente gestione della struttura Mesh.

Bibliografia

[1] C. de Lima et al., “6G white paper on localization and sensing,” 2020. [Online]. Available: http://jultika.oulu.fi/Record/isbn978-952-62-2674-3, accessed: Feb 15, 2021

[2] J. A. del Peral-Rosado, R. Raulefs, J. A. López-Salcedo, and G. Seco-Granados, “Survey of cellular mobile radio localization methods: From 1G to 5G,” IEEE Commun. Surveys & Tutorials, vol. 20, no. 2, pp. 1124–1148, 2018.

[3] K. Huang and H. Wang, “Identifying the fake base station: A location based approach,”

[4] S. Tomasin, M. Centenaro, G. Seco-Granados, S. Roth, and A. Sezgin, “Location-Privacy Leakage and Integrated Solutions for 5G Cellular Networks and Beyond,” *Sensors*, vol. 21, no. 15, p. 5176, Jul. 2021.

IEEE Commun. Letters, vol. 22, no. 8, pp. 1604–1607, 2018.

[5] “Privacy changes in Android 10.” [Online]. Available: https://developer.android.com/about/versions/10/privacy/changes, accessed: Feb 15, 2021