

# POLITECNICO DI TORINO

Corso di Laurea Magistrale in  
INGEGNERIA INFORMATICA



**Politecnico  
di Torino**

Tesi di Laurea Magistrale

**Progetto e implementazione di un sistema  
di screen casting basato su Wi-Fi Direct**

Relatore

Prof. Giovanni MALNATI

Candidato

Giovanni SINIGAGLIA

Aprile 2024



## Abstract

Wi-Fi Direct è una tecnologia che consente la connessione diretta tra dispositivi senza necessità di access point o altri elementi infrastrutturali: questo potenzialmente semplifica l'interoperabilità tra dispositivi eterogenei minimizzando il bisogno di configurazione e semplificando le procedure di connessione.

Mentre questa tecnologia è stata abbondantemente utilizzata nel caso di connessioni dirette uno a uno per lo streaming video (Chromecast, Miracast), la stampa di documenti (Bonjour) e la condivisione file (AirDrop), non sembra aver avuto altrettanto successo per la gestione di comunicazioni di gruppo.

Questo lavoro nasce con l'obiettivo di realizzare un sistema di condivisione dello schermo da un dispositivo sorgente a un gruppo di dispositivi destinatari che si trovino nelle vicinanze, in modo semplice e intuitivo per gli utenti. L'assenza di parti terze, quali server o servizi cloud, apre anche a scenari particolari in cui è richiesta confidenzialità della comunicazione.

L'implementazione di quest'idea è stata guidata dall'analisi delle diverse alternative tecnologiche disponibili, al fine di scegliere le soluzioni adeguate per ottenere prestazioni ottimali del sistema, considerando aspetti quali la latenza, la qualità video e l'usabilità.

Fra queste, il linguaggio di programmazione scelto è stato C++, che offre una sintassi di alto livello, prestazioni elevate e un controllo a basso livello sul sistema. Per ottenere uno streaming video di alta qualità e a bassa latenza si sono sfruttate le librerie offerte da FFmpeg, framework multimediale leader nelle operazioni di codifica, decodifica, multiplazione, demultiplazione e filtraggio video. L'integrazione delle tecnologie web con il framework Qt ha permesso di sviluppare un'interfaccia grafica progettata per essere semplice, chiara e immediata all'utente, ottenendo così un design moderno e professionale.

Il risultato ottenuto è un software completamente open source basato su Linux in grado di incontrare le esigenze in ambito lavorativo, di studio o di intrattenimento.



# Sommario

Questo progetto di tesi si pone l'obiettivo di esplorare le potenzialità del Wi-Fi Direct e ottenere un software open-source che possa offrire una soluzione alternativa ai software di videoconferenza nei casi in cui si ha la necessità di condividere lo schermo del proprio laptop su uno o più PC nelle vicinanze. Utilizzando il Wi-Fi Direct questa funzionalità è raggiungibile anche in assenza di connessione Internet. Inoltre, l'assenza di parti terze, quali server o servizi cloud, apre la strada anche a scenari particolari in cui è richiesta confidenzialità della comunicazione.

Il software progettato è strettamente legato al sistema operativo per la necessità di accedere a funzioni di basso livello, come la gestione della rete e l'acquisizione dello schermo, esposte in modo diverso da ogni sistema.

Si è scelto di implementare il progetto in ambiente Linux in quanto si ha una maggiore disponibilità di documentazione e supporto da parte della community open-source. Allo stesso tempo, sviluppando un progetto open-source si ha la possibilità di partecipare attivamente alla community, segnalando o risolvendo bug, migliorando l'affidabilità del software e contribuendo con la pubblicazione del proprio lavoro.

Wi-Fi Direct è una tecnologia che consente la connessione diretta tra dispositivi senza necessità di access point o altri elementi infrastrutturali. Nel contesto delle connessioni peer-to-peer, l'utilizzo del Wi-Fi Direct consente di ottenere prestazioni adeguate in termini di velocità di trasferimento e bassa latenza rispetto ad altre tecnologie come Bluetooth, oltre a una buona affidabilità della connessione e compatibilità tra dispositivi.

L'implementazione di quest'idea è stata guidata dall'analisi delle diverse alternative tecnologiche disponibili, al fine di scegliere le soluzioni adeguate per ottenere prestazioni ottimali del sistema, considerando aspetti quali la latenza, la qualità video e l'usabilità.

Si è sviluppato il software utilizzando il linguaggio di programmazione C++, che offre una sintassi di alto livello, prestazioni elevate e un controllo a basso livello sul sistema.

Per ottenere uno streaming video di alta qualità e a bassa latenza si sono sfruttate le librerie offerte da FFmpeg, framework multimediale leader nelle operazioni di codifica, decodifica, multiplazione, demultiplazione e filtraggio video.

La maggior parte delle distribuzioni Linux attuali utilizzano NetworkManager per la gestione delle connessioni di rete, ma questo servizio non supporta in modo completo le funzionalità P2P e perciò è stato necessario scendere di livello e sfruttare un'interfaccia diretta con wpa\_supplicant.

wpa\_supplicant supporta una *control interface* utilizzabile da programmi esterni che necessitano di interagire con il *daemon* in esecuzione, ma espone anche alcune interfacce sul D-Bus di sistema con metodi, segnali e proprietà che permettono di raggiungere lo stesso obiettivo.

L'integrazione delle tecnologie web con il framework Qt ha permesso di sviluppare un'interfaccia grafica progettata per essere semplice, chiara e immediata all'utente, ottenendo così un design moderno e professionale.

Il flusso operativo dell'applicazione segue a grandi linee la macchina a stati dello standard Wi-Fi Direct.

L'utente che avvia la condivisione dello schermo del proprio dispositivo crea un *P2P Group* di cui diventa il *P2P Group Owner*.

Al contrario, gli utenti che vogliono visualizzare lo streaming avviano una procedura di *Discovery* per trovare i peer che hanno creato un *P2P Group* e partecipare alla sessione come *P2P Client*.

L'architettura del sistema è organizzata a moduli, suddivisi per funzionalità. Il modulo principale gestisce l'interfaccia grafica e l'interazione con i due moduli sottostanti. Il modulo `wpa-supplciant-manager` si occupa della gestione della rete, mentre al modulo `screen-casting-service` è affidato l'interfacciamento con il display server e il processo di streaming video.

Il risultato ottenuto è un software completamente open-source basato su Linux in grado di soddisfare le esigenze in ambito lavorativo, di studio o di intrattenimento.







# Indice

<b>Elenco delle figure</b>	IX
<b>Acronimi</b>	X
<b>1 Introduzione</b>	1
1.1 Tecnologie per la connessione tra dispositivi . . . . .	1
1.2 Confronto tra Wi-Fi Direct e Bluetooth . . . . .	2
1.3 Software e tecnologie disponibili . . . . .	3
1.4 Librerie e Framework . . . . .	5
1.4.1 Gestione flusso video (FFmpeg) . . . . .	6
1.4.2 Gestione delle connessioni di rete (wpa_supplicant) . . . . .	8
1.4.3 Comunicazione tra processi (D-Bus) . . . . .	8
1.4.4 Interfaccia grafica (Qt) . . . . .	9
<b>2 Architettura del sistema</b>	11
2.1 Ambiente di sviluppo . . . . .	11
2.2 Standard Wi-Fi Direct . . . . .	12
2.3 wpa_supplicant . . . . .	18
2.4 Componenti principali . . . . .	22
2.4.1 Interfaccia grafica . . . . .	22
2.4.2 AppProxy . . . . .	23
2.4.3 wpa_supplicant manager . . . . .	23
2.4.4 Screen casting service . . . . .	24
2.5 Flusso operativo . . . . .	26
<b>3 Implementazione</b>	27
3.1 Requisiti e dipendenze . . . . .	27
3.2 Fasi di sviluppo . . . . .	28
3.3 Confronto protocolli per lo streaming e formati video . . . . .	40
3.4 Interfaccia grafica . . . . .	42

<b>4 Conclusioni e sviluppi futuri</b>	<b>49</b>
<b>Bibliografia</b>	<b>51</b>

# Elenco delle figure

1.1	Confronto tra Wi-Fi Direct e Bluetooth . . . . .	3
1.2	AirDrop . . . . .	4
1.3	AirPlay . . . . .	4
1.4	Miracast . . . . .	5
1.5	Flusso operazioni FFmpeg . . . . .	7
2.1	P2P Topology . . . . .	13
2.2	P2P Concurrent device . . . . .	13
2.3	P2P Device Discovery procedure . . . . .	14
2.4	P2P Service Discovery procedure . . . . .	15
2.5	P2P State Machine . . . . .	18
2.6	wpa_supplicant modules . . . . .	19
2.7	P2P module within wpa_supplicant . . . . .	20
2.8	P2P module state machine . . . . .	21
2.9	Architettura progetto . . . . .	22
2.10	Flusso transcodifica video . . . . .	25
2.11	Diagramma flusso operativo . . . . .	26
3.1	Screenshot D-Feet . . . . .	31
3.2	Sequence diagram - P2P Create Group . . . . .	37
3.3	Sequence diagram - P2P GO Negotiation . . . . .	37
3.4	Wi-Fi Wonders - Schermata iniziale . . . . .	42
3.5	Wi-Fi Wonders - Home . . . . .	43
3.6	Wi-Fi Wonders - Share screen . . . . .	43
3.7	Wi-Fi Wonders - View screen . . . . .	43
3.8	Wi-Fi Wonders - Selezione schermo . . . . .	44
3.9	Wi-Fi Wonders - Condivisione avviata . . . . .	44
3.10	Wi-Fi Wonders - Ricerca dispositivi . . . . .	45
3.11	Wi-Fi Wonders - Client connesso . . . . .	45
3.12	Wi-Fi Wonders - Visualizzazione streaming . . . . .	46
3.13	Wi-Fi Wonders - Visuale Group Owner . . . . .	46

# Acronimi

## **I/O**

Input/Output

## **D-Bus**

Desktop-Bus

## **IPC**

Inter-process communication

## **MOC**

Meta-Object Compiler

## **RCC**

Resource Compiler

## **AP**

Access Point

## **P2P**

Peer-to-Peer

## **GO**

Group Owner

## **DHCP**

Dynamic Host Configuration Protocol

## **WPS**

Wi-Fi Protected Setup

**WSC**

Wi-Fi Simple Configuration

**WPA-2**

Wi-Fi Protected Access II

**AES**

Advanced Encryption Standard

**CCMP**

Counter-Mode/CBC-Mac Protocol

**PSK**

Pre-Shared Key

**SSID**

Service Set Identifier

**RTMP**

Real-Time Messaging Protocol

**FLV**

Flash Video

**UDP**

User Datagram Protocol

**TCP**

Transmission Control Protocol

**MPEG**

Moving Picture Experts Group

**MPEG-TS**

MPEG transport stream



# Capitolo 1

## Introduzione

Con il crescente utilizzo del digitale, la condivisione dei dati e delle risorse multimediali sta diventando un fenomeno sempre più diffuso. I dispositivi, ormai sempre connessi in rete, consentono di trasferire in modo semplice e veloce documenti, immagini, video. Il software, spesso disponibile in modo gratuito, consente ad un vasto pubblico di usufruire di funzionalità quali videoconferenze, controllo remoto e condivisione di file. Inoltre, la comunicazione point-to-point sta diventando sempre più comune, consentendo ai dispositivi di stabilire connessioni dirette tra loro senza passare attraverso un server centrale. Ciò permette trasferimenti di dati più veloci e sicuri, rendendo la condivisione di risorse ancora più efficiente.

Questo progetto di tesi nasce con il nome “*Wi-Fi Wonders*” e si pone l’obiettivo di esplorare le potenzialità del Wi-Fi, in particolare del Wi-Fi Direct, e ottenere un software open-source che possa offrire soluzioni innovative e, in un certo senso, “straordinarie” o “meravigliose”.

### 1.1 Tecnologie per la connessione tra dispositivi

Le tecnologie disponibili per la connessione tra dispositivi sono molteplici. Ogni soluzione offre caratteristiche distintive e va scelta in base alle necessità del progetto.

La **connessione cablata**, in particolare la tecnologia Ethernet [1], molto diffusa nelle reti locali aziendali, offre ottime prestazioni in termini di affidabilità e velocità di trasferimento dati. Il grosso limite di questa tecnologia sta proprio nell’utilizzo di cavi fisici e quindi non la rende adatta a un utilizzo con dispositivi mobili come laptop o smartphone.

Il **Wi-Fi** [2] risolve questo limite offrendo una connessione senza fili, mantenendo una velocità di trasferimento elevata, soprattutto nelle ultime versioni, e consentendo una connessione comoda e versatile tra dispositivi all’interno di una rete locale.

Questa tecnologia è ampiamente utilizzata sia in ambiente domestico che in ambiente lavorativo, ma anche in luoghi pubblici per l'accesso ad Internet e la condivisione di risorse.

Successivamente alla definizione dello standard Wi-Fi, la Wi-Fi Alliance ha sviluppato lo standard **Wi-Fi Direct** [3], basato sullo standard Wi-Fi, per le connessioni wireless peer-to-peer. Questa tecnologia permette ai dispositivi di stabilire connessioni dirette tra di loro, senza la necessità di un access point Wi-Fi tradizionale o altre tecnologie di rete.

Un altro standard che permette di creare connessioni peer-to-peer senza fili è il **Bluetooth** [4]. Questa tecnologia offre una banda e una portata inferiori rispetto al Wi-Fi, ma consuma meno energia. È la soluzione ideale per dispositivi mobili e wearable, come altoparlanti, cuffie e sistemi di infotainment per auto.

Le **reti mobili**, basate su tecnologie come 3G, 4G LTE e 5G, consentono la connessione a Internet e la comunicazione tra dispositivi ovunque ci sia copertura di rete cellulare. Questa opzione è cruciale per mantenere la connettività durante gli spostamenti e in luoghi dove il Wi-Fi non è disponibile, ma comporta costi relativi agli abbonamenti offerti dagli operatori e non consente di condividere risorse se non attraverso un server o un servizio cloud.

## 1.2 Confronto tra Wi-Fi Direct e Bluetooth

Nel contesto delle comunicazioni wireless peer-to-peer, Wi-Fi Direct e Bluetooth emergono come due tecnologie prevalenti che consentono lo scambio di dati tra dispositivi senza la necessità di un punto di accesso esterno. Tuttavia, presentano differenze sostanziali in termini di prestazioni, copertura, velocità di trasferimento dati, consumi energetici e compatibilità con dispositivi. [5]

Wi-Fi Direct, essendo basato sullo standard Wi-Fi, permette di raggiungere velocità di trasferimento dei dati superiori rispetto al Bluetooth. Wi-Fi Direct, infatti, offre una larghezza di banda fino a 250 Mbps, adatta a trasferire grandi quantità di dati in tempi rapidi, mentre Bluetooth permette di arrivare a una velocità massima di circa 4 Mbps nella versione 4.0.

Per quanto riguarda la portata, il Wi-Fi Direct permette di raggiungere dispositivi a una distanza maggiore rispetto al Bluetooth. Tipicamente, la portata operativa di Bluetooth si aggira intorno ai 10 metri, anche se questa può variare in base all'ambiente, mentre Wi-Fi Direct, basato sullo standard Wi-Fi, può arrivare fino a circa 200 metri all'aperto, che si riducono, però, a circa 20 metri all'interno di un edificio per via dell'attenuazione data dai muri.

D'altro canto, in termini di consumo energetico Bluetooth è preferibile rispetto a Wi-Fi Direct. Infatti, Bluetooth è stato progettato per avere un basso consumo



di energia e questo lo rende perfetto per dispositivi indossabili o sensori. Wi-Fi Direct, invece, risulta più energivoro per via della velocità di trasmissione dei dati e delle capacità più elevate.

La tecnologia più adatta in questo progetto è, quindi, il Wi-Fi Direct perché, sebbene la configurazione e l'utilizzo possano richiedere maggiori sforzi, risulta particolarmente adeguato per applicazioni che richiedono trasferimenti di dati ad alta velocità, come lo streaming video, e nel contesto applicativo considerato l'utilizzo maggiore di energia non comporta particolari problemi.









		 Bluetooth
 <b>Velocità</b>		
 <b>Portata</b>		
 <b>Consumo energetico</b>		

Figura 1.1: Confronto tra Wi-Fi Direct e Bluetooth

## 1.3 Software e tecnologie disponibili

L'idea del progetto è stata sviluppata a partire da un'analisi del software disponibile sul mercato, concentrandosi in particolare sulle caratteristiche offerte dalle applicazioni di videoconferenza (ad esempio Zoom, Google Meet, Microsoft Teams, Skype e altri), controllo remoto (TeamViewer, AnyDesk, ...) e di condivisione file nelle vicinanze (Apple AirDrop, Samsung Quick Share).

I software di videoconferenza, sempre più diffusi, integrano tutti al loro interno le funzionalità di condivisione dello schermo. In questo progetto si è voluta integrare questa funzione, escludendo le altre che risulterebbero superflue in un contesto in cui gli utenti si trovano in prossimità, come ad esempio l'acquisizione video della webcam e dell'audio.

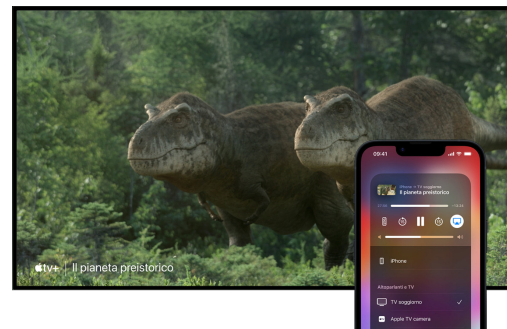
Una funzionalità aggiuntiva che potrebbe tornare utile in questo contesto è il controllo remoto, cioè la possibilità di controllare, muovendo il mouse o digitando sulla tastiera, il computer della persona che ha avviato la condivisione. Questa funzionalità non è stata inclusa nella prima versione del progetto, ma potrebbe essere sviluppata in futuro.

Il protocollo di scoperta dei servizi Bonjour [6] consente ai dispositivi di individuare automaticamente i servizi disponibili su una rete locale senza la necessità di configurazione manuale. Viene utilizzato per la scoperta di servizi di stampa di documenti, condivisione di file e media. Sebbene questo protocollo non sia necessario nell'implementazione della soluzione, è stato tenuto in considerazione come idea durante la fase di progettazione.

È stato tratto grande spunto dal software di condivisione file nelle vicinanze: l'obiettivo del progetto è ottenere l'immediatezza e la semplicità che in particolare AirDrop [7] è in grado di dare. AirDrop è una funzionalità integrata nei dispositivi Apple che permette di trasferire in modo rapido file, foto, video e altri contenuti tra dispositivi Apple compatibili. Utilizza connessioni Wi-Fi e Bluetooth per creare una rete ad hoc tra i dispositivi, consentendo loro di scambiarsi file senza la necessità di connessioni Internet o cavi.



**Figura 1.2:** AirDrop



**Figura 1.3:** AirPlay

Un'altra tecnologia Apple, forse meno conosciuta, è AirPlay [8] che consente lo streaming wireless di contenuti audio, video e foto da un dispositivo Apple a un altro dispositivo compatibile. Può essere utilizzato per riprodurre musica su

sistemi audio wireless o condividere il proprio schermo da un dispositivo iOS o macOS su un display o un proiettore.

Un'alternativa concorrente di Apple è Chromecast [9], un dongle che si collega alla porta HDMI del televisore e permette di duplicare il contenuto del proprio desktop o del dispositivo mobile sulla TV. Alcuni dispositivi moderni integrano Chromecast al proprio interno e quindi non richiedono l'acquisto del dongle HDMI esterno.

Wi-Fi Alliance ha creato uno standard denominato Miracast [10] per la trasmissione di video e audio direttamente dai dispositivi (laptop, tablet, smartphone) ai display (TV, monitor, proiettori). Miracast può connettere due dispositivi utilizzando l'infrastruttura di rete o Wi-Fi Direct e può essere definito come "HDMI over Wi-Fi". Questo standard consente di duplicare lo schermo del dispositivo sorgente sul dispositivo di visualizzazione senza la necessità di cavi. Questo si avvicina molto all'obiettivo del progetto, ma è specifico per una connessione uno-a-uno tra dispositivo trasmittente a display.



**Figura 1.4:** Miracast

Il progetto unisce l'idea di Miracast alla condivisione schermo tipica delle videoconferenze. Lo schermo di chi trasmette può essere visto da più utenti connessi utilizzando il Wi-Fi Direct. L'utilizzo del Wi-Fi Direct aggiunge la possibilità di avviare una condivisione senza connessione Internet attiva e in questo modo si evita anche l'interazione con parti terze, quali server o servizi cloud, come avviene nei più diffusi software di videoconferenza.

## 1.4 Librerie e Framework

Per l'implementazione del progetto si sono utilizzate librerie e framework disponibili online che hanno permesso di semplificare e ottimizzare il lavoro, garantendo efficienza e qualità del software sviluppato.

In questa sezione, verranno esaminati i principali strumenti adottati, evidenziando le loro caratteristiche principali e il ruolo all'interno del progetto.

### 1.4.1 Gestione flusso video (FFmpeg)

FFmpeg [11] è una soluzione open-source che fornisce una suite di strumenti per la manipolazione, la conversione e la riproduzione di file multimediali, come video e audio. È estremamente potente e versatile e può essere utilizzato da riga di comando per eseguire una vasta gamma di operazioni, come la conversione di formati video e audio, l'editing di video, l'estrazione di frame da video, la creazione di anteprime e molto altro ancora.

FFmpeg fornisce anche un set di librerie scritte in linguaggio C che permette l'integrazione delle funzionalità multimediali in applicazioni e progetti software. Queste librerie offrono metodi per manipolare flussi multimediali in modo programmatico.

Di seguito si riporta una descrizione delle librerie FFmpeg.

- **libavcodec**

È il cuore di FFmpeg, gestisce l'encoding e il decoding dei codec audio e video. Ogni formato multimediale utilizza codec specifici per comprimere e decomprimere i dati audio e video. Libavcodec fornisce un'ampia gamma di codec implementati per soddisfare le esigenze di diverse piattaforme e applicazioni. Supporta codec comuni come H.264, AAC, MP3 e molti altri, consentendo di creare applicazioni che possono manipolare praticamente qualsiasi tipo di file multimediale.

- **libavfilter**

Offre funzionalità avanzate di editing dei frame video basate su grafi. È possibile creare pipeline di filtri e effetti per manipolare i frame video in modi complessi. Ad esempio, è possibile applicare filtri di nitidezza, sfocatura, regolazione del colore e molti altri effetti visivi. L'utilizzo di un modello basato su grafi offre flessibilità e potenza nel design delle trasformazioni video.

- **libavformat**

Gestisce l'I/O dei file multimediali, compresa la lettura e la scrittura di formati di file audio e video. Supporta una vasta gamma di formati, tra cui i più comuni come MP4, AVI, MOV, MP3 e molti altri. Inoltre, gestisce anche il muxing e il demuxing, ovvero la combinazione e la separazione di flussi audio e video all'interno di un unico file multimediale.

- **libavdevice**

Fornisce supporto per l'I/O di dispositivi speciali come webcam, dispositivi di acquisizione video, sintonizzatori TV e altri dispositivi multimediali. È utile quando si desidera catturare video o audio direttamente da una sorgente esterna, anziché da un file multimediale già esistente.

- **libavutil**

Fornisce una vasta gamma di funzioni di utilità comuni utilizzate da altre parti di FFmpeg. Include funzionalità per la gestione della memoria, la manipolazione delle stringhe, le operazioni matematiche e altro ancora. Queste funzionalità sono essenziali per lo sviluppo di applicazioni multimediali robuste e efficienti.

- **libswresample**

Gestisce la resampling dell'audio, la conversione dei formati audio e il mixing. È utile quando si lavora con flussi audio e si desidera adattarli a diverse specifiche, ad esempio cambiare la frequenza di campionamento o il formato audio.

- **libpostproc**

Fornisce funzionalità di post-processing per i frame video. Può essere utilizzata per applicare filtri e effetti aggiuntivi dopo l'encoding o il decoding, come ad esempio la riduzione del rumore, il miglioramento della nitidezza o l'aggiunta di effetti speciali.

- **libswscale**

Gestisce la conversione dei colori e il ridimensionamento dei frame video. È utile per adattare i frame video a diverse risoluzioni e formati di colore, consentendo la manipolazione e la visualizzazione video in modi diversi a seconda delle esigenze dell'applicazione.

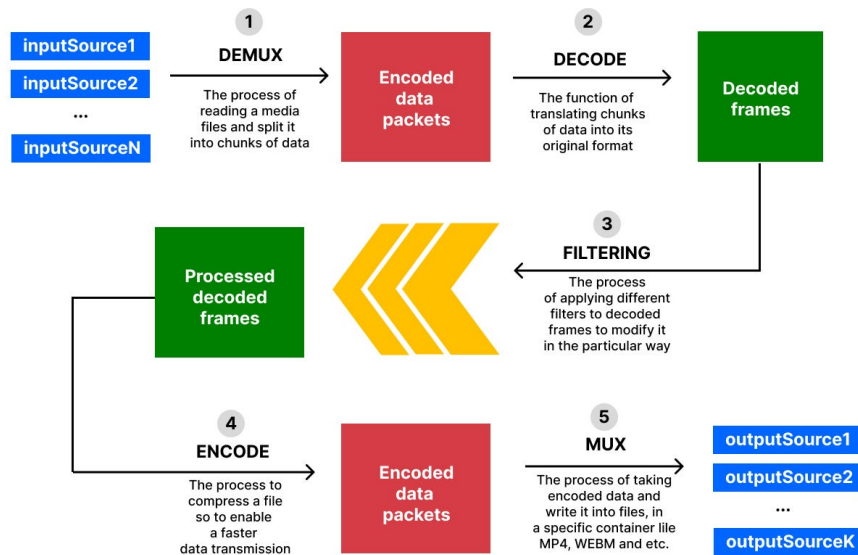


Figura 1.5: Flusso operazioni FFmpeg

La sequenza di operazioni per la gestione di un flusso multimediale con FFmpeg è composta da diverse fasi, che includono la lettura del flusso in ingresso, il demuxing, il decoding, l'eventuale processing, l'encoding, il muxing e infine la scrittura del flusso in uscita. [12]

### 1.4.2 Gestione delle connessioni di rete (wpa\_supplicant)

La maggior parte delle distribuzioni Linux attuali utilizzano NetworkManager [13] per creare e gestire le connessioni di rete. NetworkManager viene eseguito come servizio systemd ed è abilitato di default. NetworkManager funziona con D-Bus per rilevare e configurare le interfacce di rete non appena vengono collegate al computer Linux.

NetworkManager utilizza wpa\_supplicant internamente per ottenere le funzionalità wireless, come riportato anche nell'analisi condotta da Vipin e Srikanth [14].

È stato necessario utilizzare direttamente wpa\_supplicant in quanto NetworkManager non supporta in modo completo le funzionalità di Wi-Fi Direct, come anche riscontrato da López et al. [15].

wpa\_supplicant [16] è un Supplicant WPA multipiattaforma con supporto per WPA, WPA2 (IEEE 802.11i / RSN) e WPA3. È adatto sia per computer desktop/portatili che per sistemi embedded. Il Supplicant è il componente IEEE 802.1X/WPA utilizzato nei client, responsabile della gestione delle credenziali di accesso e dell'handshake di autenticazione necessario per stabilire una connessione a una rete wireless.

wpa\_supplicant supporta un'interfaccia a riga di comando (wpa\_cli) e un'interfaccia grafica (wpa\_gui), inclusi nel software wpa\_supplicant. Entrambe utilizzano una control interface (wpa\_ctrl), libreria C che può essere sfruttata da programmi esterni per controllare le operazioni del demone wpa\_supplicant e per ottenere informazioni sullo stato e notifiche di eventi.

### 1.4.3 Comunicazione tra processi (D-Bus)

D-Bus (Desktop-Bus) [17] è un sistema di comunicazione tra processi che offre alle applicazioni un modo semplice per scambiare messaggi tra loro.

Entrambi wpa\_supplicant e NetworkManager utilizzano D-Bus come mezzo di comunicazione per interagire con il sistema operativo e con altre applicazioni sul sistema Linux, esponendo i propri oggetti, ciascuno dei quali può avere una o più interfacce. Le interfacce definiscono i metodi, le proprietà e i segnali che un oggetto supporta.

D-Bus fornisce un bus di sistema (system bus) e un bus di sessione (session bus). Il primo è usato per eventi quali “aggiunta di un nuovo dispositivo hardware” o “modifica della coda della stampante”, mentre il secondo per esigenze generali di IPC tra le applicazioni utente.

Inoltre, include un meccanismo di introspezione standard per l’interrogazione in tempo reale delle interfacce degli oggetti al fine di scoprire in modo dinamico le capacità e le funzionalità disponibili su un bus D-Bus senza richiedere conoscenze predefinite sui servizi o sulle interfacce.

Uno strumento particolarmente utile per ispezionare e debuggare facilmente il sistema del D-Bus è D-Feet. D-Feet [18] è un software open-source che può essere utilizzato per ispezionare le interfacce D-Bus dei programmi in esecuzione e invocare metodi su tali interfacce oppure visualizzare i valori delle proprietà.

Infine, `dbus-cxx` [19] è una libreria C++ che implementa il protocollo D-Bus, senza appoggiarsi all’implementazione C di `libdbus`. Lo scopo è fornire un’interfaccia C++-like e risolvere i problemi di multithreading di `libdbus`. Per fornire un’interfaccia object oriented al D-Bus, `dbus-cxx` fa uso della libreria `libsigc++`. Inoltre, insieme alla libreria, è incluso il tool `dbus-cxx-xml2cpp` per generare interfacce proxy e adapter a partire da documenti XML di introspezione del D-Bus.

#### 1.4.4 Interfaccia grafica (Qt)

Qt [20] è un framework multipiattaforma per lo sviluppo di applicazioni software multipiattaforma, scritto in C++. Offre una vasta gamma di strumenti e librerie, organizzati in moduli riconducibili a due categorie: Qt Essentials e Qt Add-Ons.

Qt utilizza un’organizzazione degli oggetti ad albero, con la maggior parte delle classi che derivano da `QObject`. Inoltre, per supportare funzionalità non presenti nativamente in C++, come signal e slot, introspezione e chiamate di funzioni asincrone, Qt fa uso del MOC (compilatore di metaoggetti), il quale interpreta specifiche macro nel codice C++ e genera del codice aggiuntivo che abilita le funzionalità avanzate di Qt. Un altro componente utilizzato è l’RCC (compilatore di risorse), con lo scopo di incorporare risorse binarie all’interno dell’eseguibile.

Le principali tecnologie di interfacce grafiche del framework Qt sono Qt Quick [21] e Qt Widgets [22]. Le interfacce Qt Quick sono fluide e dinamiche e si adattano meglio alle interfacce touch, mentre Qt Widgets serve per creare applicazioni desktop complesse [23]. È possibile creare interfacce Qt Quick e Qt Widgets che si integrino bene con la piattaforma di destinazione, offrendo un aspetto nativo su Windows, Linux e macOS.

I widget sono gli elementi principali per la creazione di interfacce utente in Qt. I widget possono visualizzare dati e informazioni di stato, ricevere input dall’utente

e fornire un contenitore per altri widget che devono essere raggruppati. Un widget che non è incorporato in un widget padre è chiamato finestra.

La classe `QWidget` consente di eseguire il rendering sullo schermo e di gestire gli eventi di input dell'utente. Tutti gli elementi dell'interfaccia utente forniti da Qt sono sottoclassi di `QWidget` o sono utilizzati in combinazione con una sottoclasse di `QWidget`. La creazione di widget personalizzati si effettua creando sottoclassi di `QWidget` o di una sottoclasse adatta e reimplementando gli *event handler* virtuali.



## Capitolo 2

# Architettura del sistema

In questo capitolo verrà esaminata l'architettura del progetto, delineando le sue componenti principali, le loro interazioni e le decisioni progettuali sottese. L'obiettivo è quello di rendere chiaro il funzionamento e la struttura dell'applicazione, offrendo una visione dettagliata delle sue fondamentali architetture e delle scelte progettuali che ne hanno plasmato l'evoluzione.

Inizieremo con una descrizione generale dell'architettura complessiva, evidenziando le sue caratteristiche fondamentali e il contesto in cui si inserisce. Successivamente, approfondiremo ciascun componente architettonico, analizzandone le funzionalità, le responsabilità e le interazioni con le altre parti del sistema.

### 2.1 Ambiente di sviluppo

Il progetto è stato sviluppato in ambiente Linux, in particolare su Ubuntu 22.04 LTS Desktop.

Linux [24] è una famiglia di sistemi operativi open-source Unix-like, basati sul kernel Linux. Viene tipicamente fornito come distribuzione (distro) Linux, che include il kernel e il software di sistema con le librerie di supporto, molte delle quali sono fornite dal GNU Project. Le distribuzioni Linux più diffuse sono Debian, Fedora Linux, Arch Linux e Ubuntu.

Ubuntu [25] è una distribuzione Linux basata su Debian e composta principalmente da software libero e open-source. È rilasciato ufficialmente in più edizioni: Desktop, Server e Core per dispositivi IoT e robot. Ubuntu Desktop è noto per la sua facilità d'uso e accessibilità.

Il linguaggio di programmazione (C++) e la maggior parte delle librerie utilizzate sono progettati per essere cross-platform, consentendo quindi una potenziale compatibilità del software con diversi sistemi operativi. Tuttavia, alcune operazioni

richiedono l'accesso diretto alle risorse del sistema operativo, rendendo l'applicazione dipendente dalla piattaforma. Per rendere il software veramente multiplatforma, sarebbe necessario adattare i segmenti di codice specifici per Linux anche per gli altri sistemi operativi. Questo potrebbe essere ottenuto attraverso metodi specifici per riconoscere il sistema operativo in cui l'applicazione è in esecuzione, oppure creando versioni diverse del software, ognuna compilata per la specifica piattaforma.

In particolare, la cattura dello schermo è dipendente anche dal sistema di finestre attivo sul sistema Linux. Il progetto è basato sull'X Window System (X11) [26] e non è compatibile con Wayland [27]. Wayland è stato sviluppato con l'obiettivo di sostituire X11 con un sistema di finestre sicuro, più semplice e più efficiente per Linux e altri sistemi operativi Unix-like. Tuttavia, considerando che molte applicazioni e ambienti desktop sono ancora ottimizzati per X11, si è deciso di fornire il supporto a X11 nella prima versione del software anziché a Wayland.

## 2.2 Standard Wi-Fi Direct

In «Device-to-device communications with Wi-Fi Direct: overview and experimentation» [28] è presente un'ampia panoramica delle reti Wi-Fi Direct. Questo articolo è stato utile per comprendere il funzionamento della tecnologia e applicarla per l'obiettivo del progetto. Anche il documento *Wi-Fi Direct<sup>®</sup> Specification v1.9* [29] ha fornito informazioni dettagliate e illustrazioni esaustive, riportate di seguito.

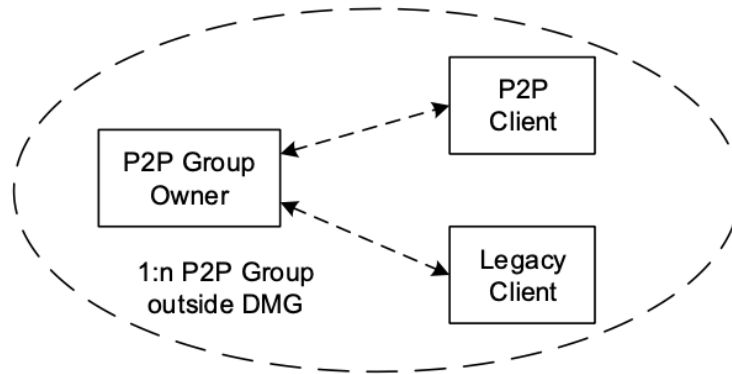
Nelle reti Wi-Fi tradizionali, i dispositivi client rilevano e si connettono alle reti wireless create e annunciate dagli AP (Access Point). In questo modo, un dispositivo si comporta in modo inequivocabile come AP o come client, ognuno dei quali comporta una serie di funzionalità distinte.

Con il Wi-Fi Direct, invece, questi ruoli sono dinamici, e quindi un dispositivo certificato Wi-Fi Direct deve implementare sia il ruolo di client che quello di AP (talvolta indicato come Soft-AP).

I dispositivi Wi-Fi Direct, noti anche come dispositivi P2P (Peer-to-Peer), comunicano creando dei P2P Group. I gruppi P2P sono equivalenti alle reti Wi-Fi tradizionali in termini di funzionalità. Il dispositivo che funge da Access Point (AP) all'interno del P2P Group è chiamato P2P GO (Group Owner), mentre i dispositivi che agiscono come client sono chiamati P2P Client.

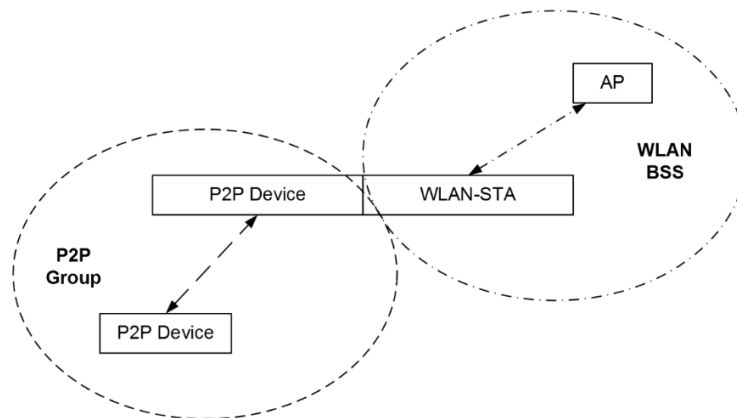
Quando due dispositivi P2P si rilevano reciprocamente, devono negoziare i loro ruoli (P2P Client e P2P GO) per stabilire un P2P Group. Una volta che il gruppo P2P è stabilito, altri client P2P possono unirsi ad esso, proprio come in una rete Wi-Fi tradizionale. I client legacy compatibili possono comunicare con il P2P GO, anche se non fanno formalmente parte del gruppo P2P perché vedono semplicemente il P2P GO come un AP tradizionale.

In Figura 2.1 è mostrato un esempio di topologia di rete P2P con un P2P GO, un P2P Client e un dispositivo legacy.



**Figura 2.1:** P2P Topology

Sfruttando frequenze diverse o tecniche di virtualizzazione per la condivisione del canale, un dispositivo può essere connesso contemporaneamente a una rete Wi-Fi tradizionale e a un gruppo Wi-Fi Direct, come illustrato in Figura 2.2, oppure a più di un gruppo P2P. Questa funzionalità, però, non è disponibile su tutti i dispositivi.



**Figura 2.2:** P2P Concurrent device

Esistono tre diverse modalità di creazione di un gruppo P2P (*Group Formation*). Il caso più complesso, descritto successivamente come caso *standard*, è quando i due dispositivi devono negoziare il ruolo di P2P GO, mentre i casi indicati come *autonomi* e *persistenti* si possono considerare scenari semplificati.

La prima fase, inclusa in tutti i casi citati sopra, è quella di P2P Device Discovery. Il meccanismo principale è quello di “active scanning” in cui i dispositivi usano i frame *probe request* e *probe response* per scoprirsi a vicenda. La procedura è ben descritta nel blog di Praneeth [30] ed è illustrata nella Figura 2.3, disponibile nel documento *Wi-Fi Direct® Specification v1.9*.

Una volta che il gruppo è stato formato, il P2P GO trasmette il *beacon*, consentendo così la scoperta anche tramite quest’ultimo. I dispositivi P2P non rispondono alle *probe request* a meno che non siano P2P GO o che si trovino nello stato di ascolto. Inoltre, il dispositivo P2P non emette un *beacon* a meno che non assuma il ruolo di P2P Group Owner.

Il P2P GO è anche tenuto a eseguire un server DHCP per fornire agli utenti P2P gli indirizzi IP. Inoltre, è bene specificare che le specifiche di Wi-Fi Direct non consentono il trasferimento del ruolo di P2P GO all’interno di un gruppo P2P. Quindi, se il P2P GO lascia il gruppo P2P, il gruppo viene sciolto e deve essere ristabilito utilizzando una delle procedure specificate.

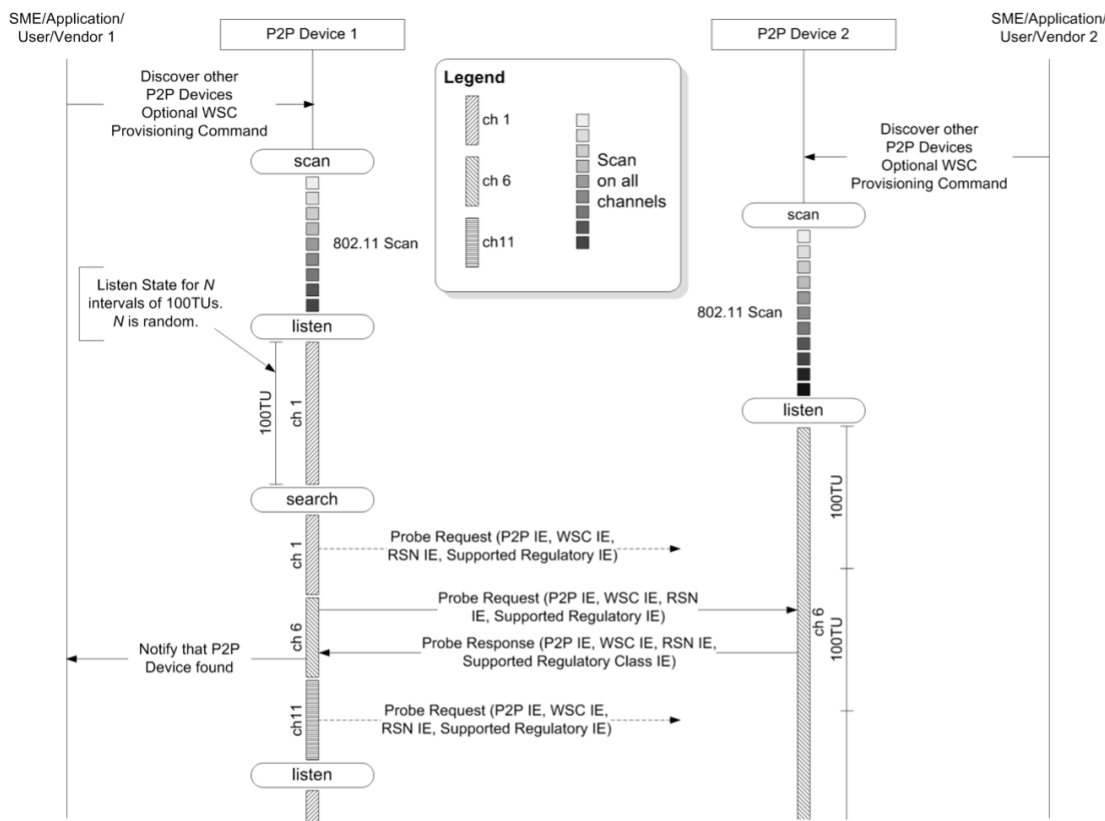


Figura 2.3: P2P Device Discovery procedure

La procedura di P2P Device Discovery comprende tre fasi principali:

- **Scan:** i dispositivi P2P cercano attivamente altri dispositivi P2P o gruppi P2P, per individuare il miglior canale operativo potenziale per creare un gruppo P2P, utilizzando il processo di scansione definito in IEEE Std 802.11-2012.
- **Listen:** i dispositivi diventano individuabili e rispondono ai *Probe Request* frame con *Probe Response* frame.
- **Search:** i dispositivi trasmettono *Probe Request* frame su tutti i canali disponibili per scoprire gli altri dispositivi disponibili.

Le fasi di Listen e Search si alternano in modo iterativo, cambiando randomicamente il canale di ascolto per favorire la convergenza. Queste due fasi si possono racchiudere in una fase definita **Find**.

Dopo una procedura di P2P Device Discovery andata a buon fine, prima della formazione del gruppo, è possibile eseguire opzionalmente la procedura di P2P Service Discovery. Anche questa procedura è ben descritta nel blog di Praneeth [31].

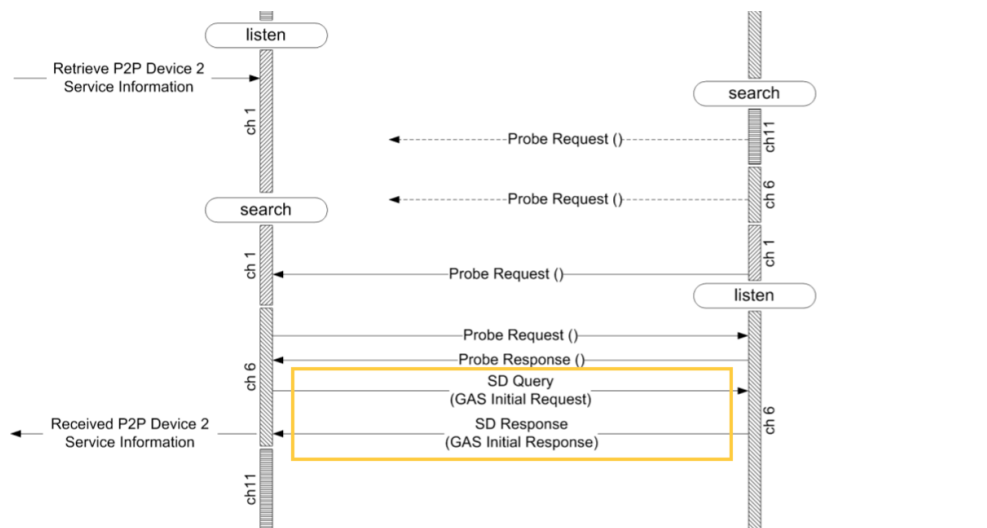


Figura 2.4: P2P Service Discovery procedure

Questa procedura può essere utilizzata per determinare informazioni di compatibilità sui servizi offerti da un dispositivo P2P. Questo protocollo è estensibile e flessibile, in modo da consentire l'utilizzo di diversi tipi di protocolli di annuncio di servizi di livello superiore, come Bonjour e UPnP.

Nel caso di *Group Formation* standard, una volta che i due dispositivi P2P si sono trovati, iniziano la fase di GO Negotiation. Questa fase è implementata utilizzando un handshake a tre vie, ovvero *GO Negotiation Request/Response/Confirmation*, con cui i due dispositivi si accordano su quale dispositivo agirà come P2P GO e sul canale in cui il gruppo opererà, che può essere nelle bande a 2.4 GHz o 5 GHz.

Per concordare il dispositivo che fungerà da P2P GO, i dispositivi P2P inviano un parametro numerico (*GO Intent value*) all'interno dell'handshake a tre vie. Il dispositivo che dichiara il valore più alto diventa il P2P GO. Quando due dispositivi dichiarano lo stesso GO Intent, il Group Owner viene eletto casualmente.

La fase successiva è l'instaurazione di una comunicazione sicura tramite WPS (Wi-Fi Protected Setup), che indichiamo come fase WPS Provisioning, e infine uno scambio DHCP per impostare la configurazione IP.

Nel caso di *Autonomous Group Formation*, invece, un dispositivo P2P crea autonomamente un gruppo P2P, in cui diventa immediatamente il GO P2P, scegliendo un canale e iniziando a fare beacon.

Gli altri dispositivi possono scoprire il gruppo creato utilizzando i tradizionali meccanismi di scansione, e quindi procedere direttamente con le fasi di Provisioning WPS e di Configurazione dell'indirizzo.

Rispetto al caso precedente, quindi, la fase di scoperta è semplificata in questo caso, poiché il dispositivo che crea il gruppo non alterna gli stati di Listen e Search e non è necessaria la fase di negoziazione del Group Owner.

Il caso di *Persistent Group Formation* si differenzia da quello standard perché durante il processo di formazione, i dispositivi P2P possono dichiarare un gruppo come persistente, utilizzando un flag nell'attributo P2P Capabilities presente nei frame *Beacon*, *Probe Responses* and *GO negotiation*.

In questo modo, i dispositivi che formano il gruppo memorizzano le credenziali di rete e i ruoli P2P GO e Client assegnati per le successive reinstanziazioni del gruppo P2P.

In particolare, dopo la fase di scoperta, se un dispositivo P2P riconosce di aver formato in passato un gruppo persistente con il peer corrispondente, uno qualsiasi dei due dispositivi P2P può utilizzare la procedura di invito (un handshake a due vie) per reinstanziare rapidamente il gruppo.

Per quanto riguarda la sicurezza, la procedura WSC (Wi-Fi Simple Configuration) inizia dopo la negoziazione dei ruoli, se prevista, o dopo la fase di scoperta.

WSC è un meccanismo utilizzato per il WPS (Wi-Fi Protected Setup), che consente di stabilire una connessione sicura digitando un PIN mostrato sull'altro peer o premendo un bottone, tipicamente virtuale, nei due dispositivi P2P.

WPS è basato su WPA-2 (Wi-Fi Protected Access II) che utilizza la cifratura AES (Advanced Encryption Standard)-CCMP (Counter-Mode/CBC-Mac Protocol) e una PSK (Pre-Shared Key) per l'autenticazione reciproca. Nel caso di *Persistent Group Formation* non è necessario effettuare lo scambio delle chiavi e si può effettuare direttamente l'autenticazione.

Completata l'autenticazione, il GO (Group Owner) assegna un indirizzo al dispositivo P2P appena connesso, utilizzando il DHCP (Dynamic Host Configuration Protocol). A questo punto inizia la fase chiamata *operativa*, durante la quale i dispositivi possono scambiarsi dati direttamente tra loro.

L'SSID (Service Set Identifier), cioè il nome della rete Wi-Fi, di qualsiasi P2P Group inizierà sempre con "DIRECT-" seguito da due caratteri casuali alfanumerici. È possibile anche aggiungere un suffisso personalizzato per rendere la rete più riconoscibile.

La procedura di P2P Invitation offre un metodo alternativo a quelli descritti in precedenza per la formazione di un gruppo e può essere utilizzata nei seguenti casi:

- Il GO invita un dispositivo P2P a diventare client del suo gruppo P2P
- Un client P2P invita un altro dispositivo P2P a unirsi al gruppo P2P di cui il client è membro perché desidera utilizzare un servizio del dispositivo P2P
- Si vuole attivare un gruppo P2P persistente al quale entrambi i dispositivi hanno precedentemente partecipato e uno dei dispositivi è il GO del gruppo P2P persistente

I frame scambiati durante la procedura di P2P Invitation sono *P2P Invitation Request* e *P2P Invitation Response*. Quest'ultimo può essere sia di successo, confermando l'accettazione dell'invito, sia di fallimento, indicando che l'invito è stato respinto o non è stato possibile stabilire la comunicazione P2P.

La disconnessione da un gruppo Wi-Fi Direct può avvenire in diversi modi:

- *Un Client si disconnette da un P2P Group*  
Un Client deve, se possibile, indicare l'intenzione di disconnettersi dal gruppo inviando un frame di dissociazione al GO.
- *Il Group Owner rimuove un Client dal P2P Group*  
Un GO può disconnettere un Client dal gruppo inviando un frame di deautenticazione al Client.
- *Il Group Owner si disconnette da un P2P Group*  
Quando un GO si disconnette dal gruppo la connessione tra tutti i dispositivi connessi a quel gruppo verrà interrotta. Un GO può indicare l'intenzione di terminare una sessione Wi-Fi Direct inviando un frame di deautenticazione a tutti i client connessi.

Nella Figura 2.5 è riportata una macchina a stati completa dello standard Wi-Fi Direct.

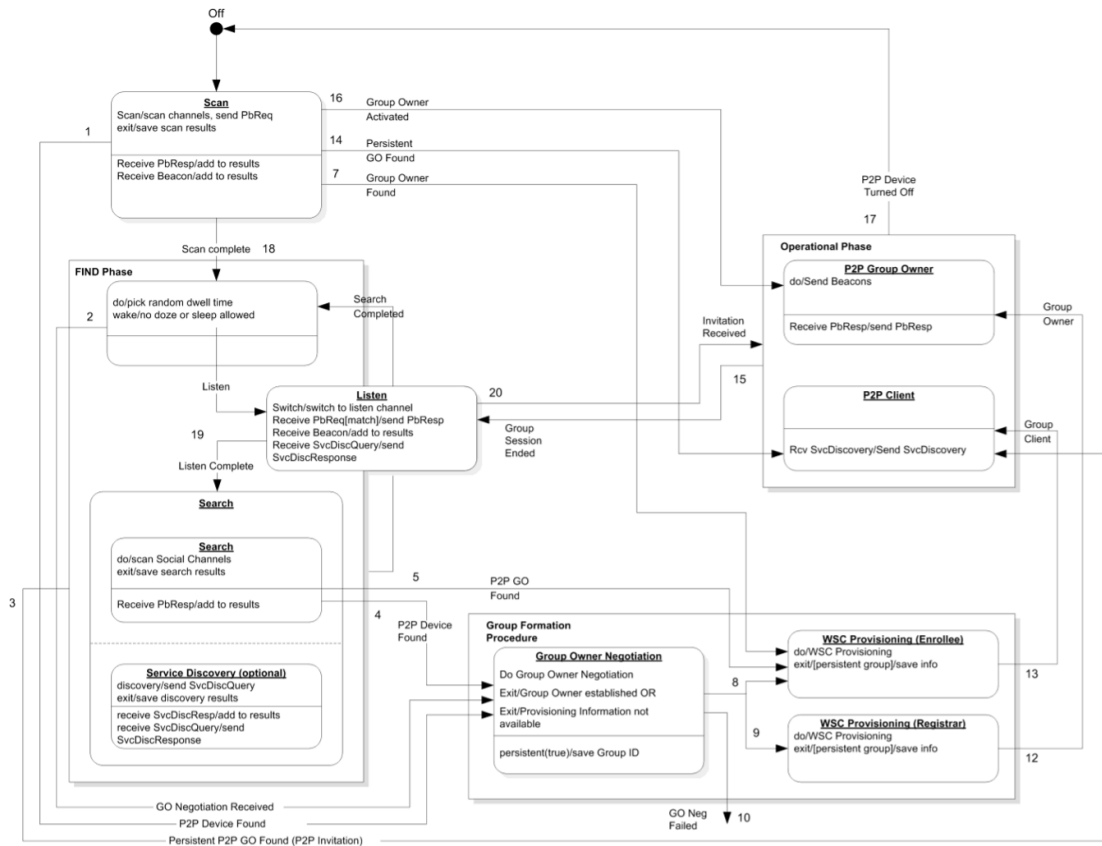


Figura 2.5: P2P State Machine

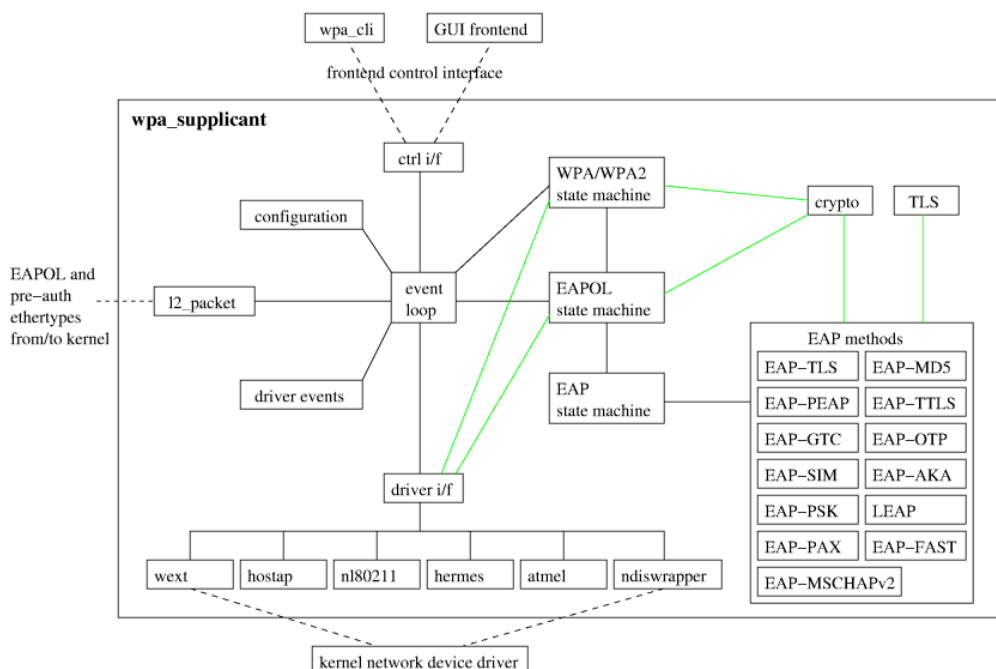
## 2.3 wpa\_supplicant

L'obiettivo del progetto di wpa\_supplicant è quello di utilizzare un codice C portatile e indipendente da hardware, driver e sistema operativo per tutte le funzionalità WPA.

wpa\_supplicant implementa una *control interface* che può essere utilizzata da programmi esterni per controllare le operazioni del demone wpa\_supplicant e per ottenere informazioni sullo stato e notifiche di eventi. Esiste una piccola libreria C che fornisce funzioni di aiuto per facilitare l'uso dell'interfaccia di controllo. Questa libreria può essere utilizzata anche in C++. wpa\_cli and wpa\_gui sono esempi di programmi che utilizzano questa libreria.



La Figura 2.6 illustra l'architettura di wpa\_supplicant. È bene tenere a mente questo schema per comprendere meglio con quali moduli sono coinvolti quando si eseguono le operazioni di rete.



**Figura 2.6:** wpa\_supplicant modules

Le funzionalità Wi-Fi Direct sono implementate a molti livelli nello stack WLAN, dalle operazioni di basso livello del driver alla progettazione di alto livello dell'interfaccia grafica. Le seguenti figure e illustrazioni sono tratte dalla documentazione ufficiale di wpa\_supplicant [32] e del P2P module [33].

Il modulo P2P implementa funzionalità di livello superiore per la gestione dei gruppi P2P. Si occupa di Device Discovery, Service Discovery, Group Owner Negotiation, P2P Invitation. Inoltre, mantiene le informazioni sui dispositivi P2P vicini. La fase di provisioning della formazione dei gruppi è implementata utilizzando il modulo WPS.

La funzionalità P2P interessa molte aree dell'architettura di wpa\_supplicant. Nella figura 2.7 è riportata la posizione dei principali componenti P2P nel caso dell'architettura Linux. Le frecce verdi indicano i percorsi di comunicazione dal modulo P2P alle funzionalità di gestione del livello superiore, fino all'interfaccia grafica che l'utente può utilizzare per gestire le connessioni P2P. Le frecce blu mostrano il percorso seguito per le operazioni di livello inferiore.

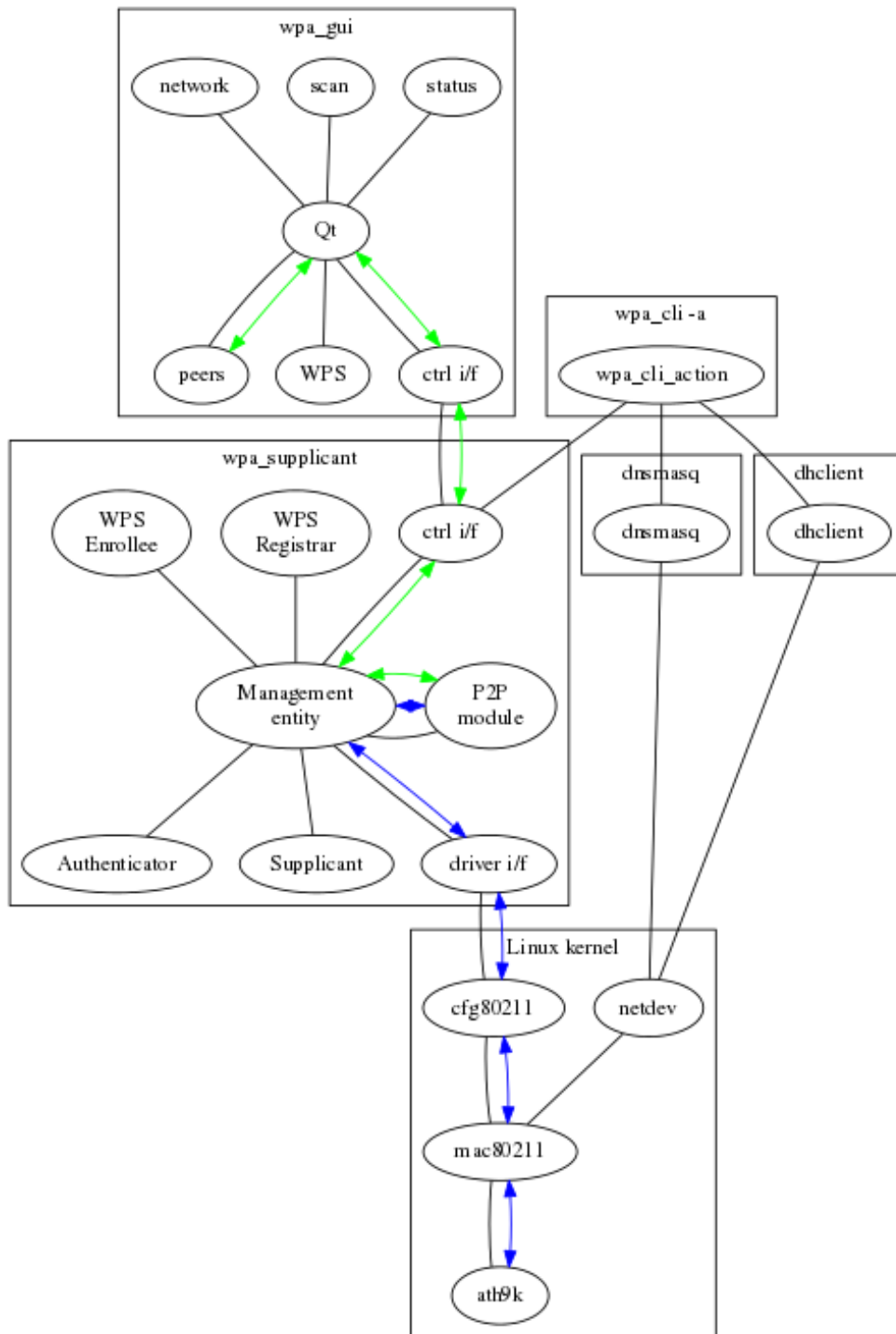
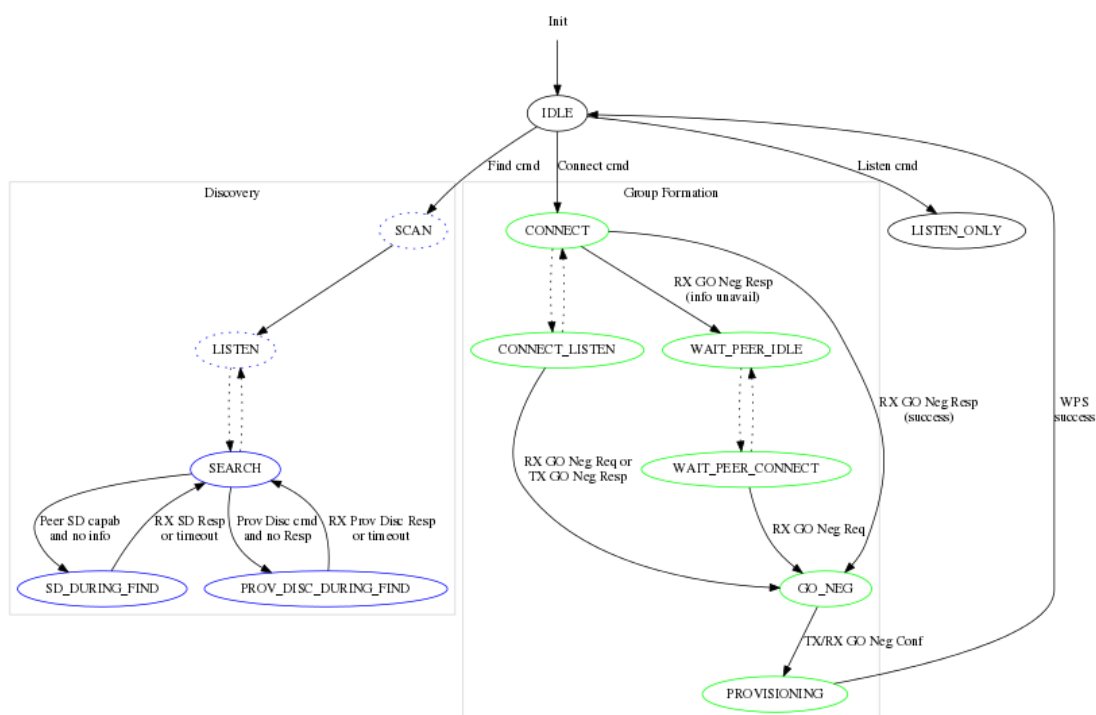


Figura 2.7: P2P module within wpa\_supplicant

Il modulo P2P gestisce la scoperta e la formazione dei gruppi con un'unica macchina a stati, vale a dire che in ogni momento può essere in corso una sola operazione per dispositivo. La Figura 2.8 descrive la macchina a stati P2P. Per maggiore chiarezza, non include le transizioni di stato in caso di timeout delle operazioni verso lo stato IDLE. Gli stati contrassegnati da un'ellisse tratteggiata sono elencati per rispettare la definizione del protocollo per la fase di Device Discovery, ma non sono utilizzati nell'implementazione (lo stato SEARCH è utilizzato per gestire la scansione iniziale e l'alternanza degli stati Listen e Search all'interno di Find).



**Figura 2.8:** P2P module state machine

Un'ulteriore possibilità di interazione con wpa\_supplicant è l'utilizzo delle interfacce D-Bus implementate sul di sistema, le cui API sono riportate sulla documentazione ufficiale [34], insieme a metodi, segnali e proprietà con argomenti, valori restituiti e possibili errori.

## 2.4 Componenti principali

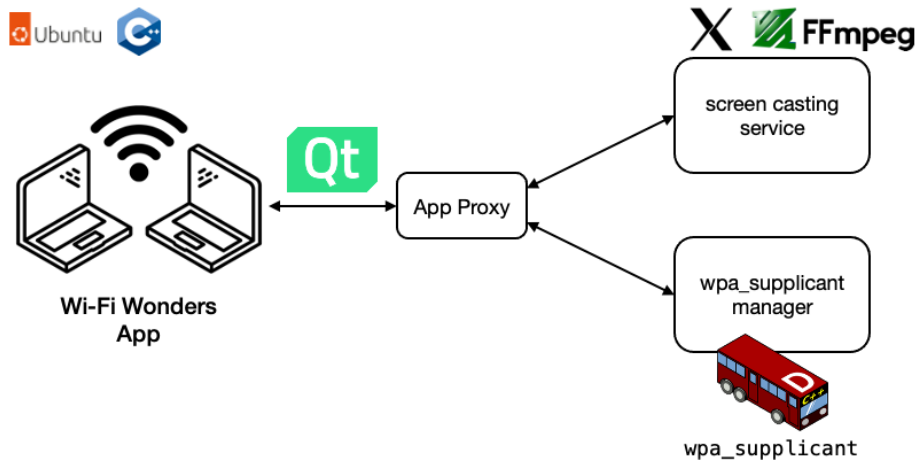


Figura 2.9: Architettura progetto

### 2.4.1 Interfaccia grafica

L'interfaccia grafica dell'applicazione è stata sviluppata con il framework multi-piattaforma Qt, introdotto nella sezione 1.4.4, nella versione 6.2 LTS.

Qt Quick e Qt Widgets, benché permettano di applicare uno stile personalizzato e ottenere un buon risultato in termini di design, presentano una curva di apprendimento impegnativa. Si è quindi utilizzato un approccio multi-linguaggio, costruendo l'interfaccia grafica con l'ausilio delle tecnologie web, già note.

L'applicazione Qt è composta semplicemente da una `QWebEngineView` [35]. Una web view è il componente widget principale del modulo Qt WebEngine [36]. Può essere utilizzata in varie applicazioni per visualizzare contenuti siti web da Internet oppure per impostare direttamente l'HTML. Il nucleo di Qt WebEngine è basato su Chromium Project [37]. Chromium fornisce i propri engine di rete e di grafica ed è sviluppato in stretta collaborazione con i suoi moduli dipendenti.

Per mettere in comunicazione l'applicazione HTML/JavaScript con l'applicazione C++ è possibile utilizzare il modulo Qt WebChannel [38], supportato direttamente da Qt WebEngine. Il modulo fornisce una libreria JavaScript per una perfetta integrazione delle applicazioni C++ con i client HTML/JavaScript. I client devono

utilizzare la libreria JavaScript per accedere ai QObject registrati e serializzati dalle applicazioni host.

### 2.4.2 AppProxy

La classe `AppProxy` è un `QObject` definito durante lo sviluppo del progetto con lo scopo di contenere l'intera logica per la gestione dello stato e per la comunicazione tra i diversi componenti. Infatti, sono memorizzate al suo interno le istanze degli oggetti dei moduli `wpa-supPLICANT-manager` e `screen-casting-service`.

La classe `AppProxy` espone segnali, slot e funzioni marchiate con la macro `Q_INVOKABLE`. Questa macro consente di richiamare una funzione dal codice QML. Nel caso di questo progetto, `Q_INVOKABLE` serve per poter chiamare le funzioni direttamente da Javascript. Infatti, l'oggetto della classe `AppProxy` istanziato nel `main` è registrato nel canale di comunicazione tra Qt e Javascript. Le funzioni `Q_INVOKABLE` richiedono dei parametri dei tipi definiti da Qt (per esempio `QString`, `QVector`, `QMap`) al posto di quelli della *standard library* per poter essere serializzati correttamente nel `QWebChannel`.

All'interno della classe `AppProxy` sono memorizzati anche i parametri di configurazione dell'applicazione, come nome del dispositivo, nome dell'interfaccia e percorso della cartella contenente i file di configurazione.

Inoltre, nella stessa classe `AppProxy` è definita anche una callback che viene passata al modulo `wpa-supPLICANT-manager` per la gestione degli eventi P2P.

### 2.4.3 wpa\_supPLICANT manager

Il modulo `wpa-supPLICANT-manager` espone la classe `WpaSupPLICANTManager`, che gestisce in modo completo la connessione Wi-Fi Direct. All'interno dello stesso modulo è presente anche un file separato denominato `utils.hpp` contenente alcuni metodi di utilità generici e di interfacciamento con il sistema operativo.

La classe `WpaSupPLICANTManager` utilizza altre strutture di supporto definite all'interno dello stesso file. La struct `Peer` rappresenta un dispositivo P2P scoperto, ma non necessariamente connesso. La struct `Group` rappresenta un gruppo P2P a cui il dispositivo partecipa. Contiene informazioni sul gruppo e sui peer connessi. Sono definite anche le enum `P2PEvent` e `P2PEventParamName` utilizzate nei parametri della callback per la gestione degli eventi P2P.

La classe `WpaSupPLICANTManager` sfrutta la libreria `dbus-cxx` e le classi definite attraverso il tool `dbus-cxx-xml2cpp`. Queste classi gestiscono in modo object-oriented i metodi, le proprietà e i segnali delle interfacce degli oggetti del D-Bus.

Per l'interfacciamento con il modulo `wpa_supplicant`, e in particolare con il modulo P2P, si è utilizzato il *system bus* denominato `fi.w1.wpa_supplicant1`. Per la gestione delle interfacce di rete si è utilizzato il D-Bus Object con path `/fi/w1/wpa_supplicant1`. Da questo si può ottenere il path dell'interfaccia di rete vera e propria (solitamente `/fi/w1/wpa_supplicant1/Interfaces/0`). Attraverso le interfacce di questo oggetto è possibile gestire l'interfaccia di rete per quanto riguarda la connessione di rete, ma anche le capacità di P2P e il modulo WPS.

Si noti che la terminologia può risultare fuorviante, in quanto il termine "interfaccia" si può riferire sia alla scheda di rete che al un oggetto del D-Bus. Non è stato semplice trovare una nomenclatura adeguata per identificare l'interfaccia dell'oggetto del D-Bus che fa riferimento all'interfaccia della scheda di rete.

#### 2.4.4 Screen casting service

Il modulo `screen-casting-service`, invece, espone la classe `MonitorService` e la classe `CastingService`: la prima utilizza la libreria X11 per ottenere la lista degli schermi disponibili per la condivisione e la loro anteprima, mentre la seconda sfrutta le librerie del framework FFmpeg per la cattura dello schermo e trasmissione del video.

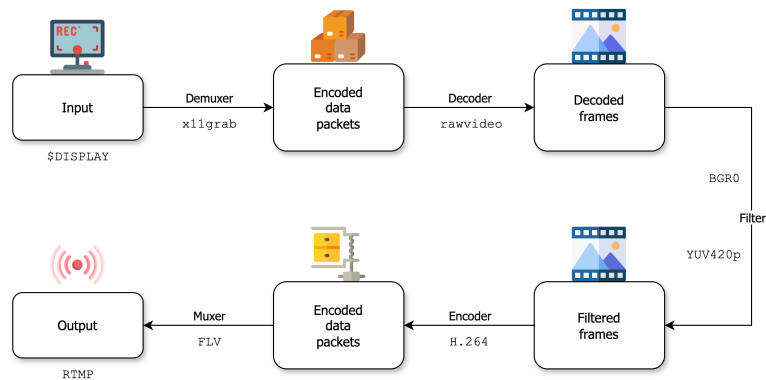
La classe `MonitorService` espone due funzioni statiche: la prima permette di ottenere la lista dei monitor in ambiente X11, mentre la seconda permette di catturare lo screenshot di un'area dello schermo e ritornare la codifica dell'immagine PNG in Base64.

La classe `CastingService`, invece, è stata progettata secondo il pattern Singleton. Questo design pattern garantisce che una classe abbia una sola istanza e fornisce un modo per accedere a questa istanza da qualsiasi punto del programma, semplificando l'accesso e l'utilizzo di oggetti condivisi. Inoltre, è possibile ottenere la creazione dell'istanza in modo sicuro anche in un ambiente multithread.

Al suo interno, la classe `CastingService`, utilizza la classe `LibAvWrapper`, anch'essa progettata secondo il pattern Singleton. Questa classe incapsula al suo interno le librerie C fornite dal framework FFmpeg, permettendo di gestire in maniera più semplice l'inizializzazione, il rilascio delle risorse e il flusso video.

Le librerie FFmpeg utilizzate sono `libavcodec`, `libavdevice`, `libavformat`, `libswscale`, `libavutil`.

La libreria `libavformat`, insieme a `libavcodec` e `libavdevice`, permette di gestire l'acquisizione dello stream video, utilizzando come sorgente il display nel formato `x11grab`, e la scrittura dello stream in formato FLV (Flash Video) codificato in H.264 sul server RTMP (Real-Time Messaging Protocol).



**Figura 2.10:** Flusso transcoding video

Il protocollo RTMP [39] si basa su una connessione TCP ed è noto per offrire bassa latenza nella trasmissione dei dati. Inizialmente era stato sviluppato come protocollo proprietario da Macromedia, ma Adobe, dopo aver acquisito Macromedia, lo ha reso pubblico.

Il formato video più supportato da RTMP è FLV [40]. FLV supporta i codec video H.264, VP6 e H.263. La codifica video H.264 [41] è delle più diffuse e offre un'ottima qualità video con un'efficienza di compressione elevata. Gli altri codec supportati offrono tutti prestazioni inferiori in termini di efficienza di compressione.

Negli ultimi anni, l'uso di RTMP è diminuito in favore di protocolli più recenti come HLS (HTTP Live Streaming) e MPEG-DASH (Dynamic Adaptive Streaming over HTTP). RTMP rimane, però, più adatto per applicazioni che richiedono una risposta in tempo reale perché garantisce una latenza più bassa rispetto a HLS e DASH. Questi ultimi hanno il vantaggio di essere compatibili con le tecnologie web e i video player HTML5, ma a causa della segmentazione del flusso non permettono di raggiungere le stesse prestazioni di RTMP.

È stato preso in considerazione anche lo streaming su UDP utilizzando il formato MPEG-TS, ma sarebbe stato necessario gestire singolarmente lo streaming verso ogni peer collegato, perché utilizzando UDP multicast si ottengono pessime prestazioni.

Lo streaming su protocollo RTMP necessita di un server. Chi trasmette invia lo stream al server che lo redistribuisce a tutti i client che vogliono riceverlo. Il progetto sfrutta un server nginx con l'aggiunta del modulo RTMP.

La libreria `libswscale` è necessaria nella catena di transcoding per convertire il formato dei pixel dei frame video. Infatti, il codec H.264 richiede in input un frame nel sistema di codifica colori YUV (in particolare YUV420p), mentre i frame sorgente sono in formato RGB (più precisamente BGR0).

In conclusione, l'analisi e lo studio condotti sul framework FFmpeg, sia sui tool da linea di comando che sulle librerie libav, hanno permesso di ottenere una condivisione schermo fluida e immediata. In particolare, si è ottenuto un codice per la trasmissione video molto efficiente sfruttando le librerie libav. Per quanto concerne invece la ricezione dello streaming RTMP, è stata adottata una soluzione basata sull'utilizzo del tool ffmpeg, il quale ha dimostrato di offrire una piattaforma stabile e performante per la riproduzione dei flussi audiovisivi in ingresso.

## 2.5 Flusso operativo

In Figura 2.11 è riportato il flusso operativo dell'applicazione Wi-Fi Wonders. Nel prossimo capitolo verrà presentata anche l'interfaccia grafica seguendo il flusso qui rappresentato.

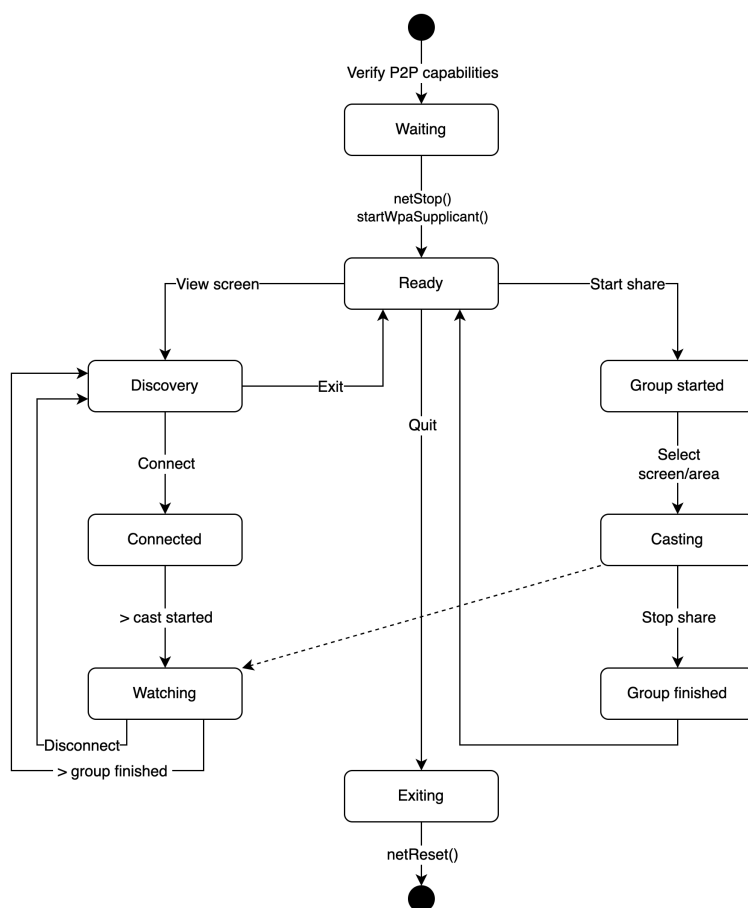


Figura 2.11: Diagramma flusso operativo



# Capitolo 3

## Implementazione

In questo capitolo verranno descritti i processi, le metodologie e le decisioni tecniche adottate per sviluppare e realizzare il progetto Wi-Fi Wonders.

### 3.1 Requisiti e dipendenze

Il progetto C++ è organizzato in moduli, la cui compilazione è gestita con CMake [42]. Il modulo principale che contiene il `main.cpp`, la classe `AppProxy` e la cartella `html-ui` contenente l'interfaccia grafica. Questo modulo si collega a due sotto-moduli `wpa-supPLICANT-manager` e `screen-casting-service`.

I due sotto-moduli sono compilati come librerie statiche che vengono quindi linkate staticamente durante la fase di compilazione del progetto.

Il requisito fondamentale è che il software sia eseguito su Linux, utilizzando l'ambiente grafico X11. Alcune distribuzioni Linux attuali eseguono già Wayland come ambiente grafico predefinito, come Ubuntu 22.04 LTS. Pertanto, per eseguire il software è necessario passare al display server X al posto di Wayland. Per farlo in Ubuntu 22.04 LTS è sufficiente modificare una riga del file `/etc/gdm3/custom.conf` da `WaylandEnable=true` a `WaylandEnable=false` e riavviare il sistema.

Il modulo `wpa_supPLICANT` è già presente nella maggior parte delle distribuzioni Linux. Ovviamente, il requisito hardware è che la scheda di rete sia presente e supporti il Wi-Fi Direct. Si noti che non è possibile utilizzare un sistema virtualizzato, anche se la scheda di rete supporta il Wi-Fi Direct. Il software di virtualizzazione infatti “nasconde” la scheda di rete fisica del sistema operativo host, mostrando al sistema operativo guest solo un'interfaccia Ethernet senza capacità wireless.

Per effettuare la trasmissione dello streaming è necessario eseguire un server `nginx` che supporti RTMP.

Su Ubuntu 22.04 LTS è possibile installare il package *libnginx-mod-rtmp*, ma una volta installato è necessario aggiungere al file `/etc/nginx/nginx.conf` il seguente blocco di configurazione, aprire la porta 1935 del firewall, se abilitato, e riavviare il servizio nginx.

```
rtmp {
    server {
        listen 1935;
        chunk_size 4096;
        allow publish 127.0.0.1;
        deny publish all;

        application live {
            live on;
            record off;
        }
    }
}
```

Il P2P Group Owner deve necessariamente eseguire un DHCP server, quindi è necessario che sia installato nel sistema e venga eseguito dal GO quando viene creato un gruppo. In Ubuntu 22.04 LTS è disponibile il package *udhcpd* come implementazione del DHCP server.

Nel file README.md del progetto sono presenti le istruzioni e i comandi shell per l'installazione delle dipendenze. Le principali sono Qt6 base, più i componenti del WebEngine, libav, libxrandr, libfmt e libdbus. Tutti questi moduli sono facilmente installabili dal package manager (apt o simili).

La libreria *dbus-cxx*, invece, deve essere compilata e installata manualmente. Inoltre, *dbus-cxx* richiede *libsigc++* come dipendenza ed è quindi necessario compilare e installare *libsigc++* prima di *dbus-cxx*.

## 3.2 Fasi di sviluppo

Il progetto Wi-FI Wonders è stato suddiviso in sotto-progetti, corrispondenti ai principali componenti presentati nel Capitolo 2.

Nella prima fase laboratoriale si è sviluppato un programma che effettuasse la cattura dello schermo e la trasmissione dell'immagine a un server sulla rete locale.

Dopo questo primo approccio limitato all'acquisizione di uno screenshot si è iniziata un'analisi del software per la cattura di un video. L'idea di catturare

manualmente frame per frame si è ritenuta poco efficiente. Si è quindi iniziato a studiare FFmpeg e le capacità offerte da questo framework.

Il tool da linea di comando `ffmpeg` permette di leggere un'ampia varietà di ingressi, compresi i dispositivi di acquisizione/registrazione dal vivo, di filtrarli e di transcodificarli in una vasta gamma di formati di uscita. Si è trovata una documentazione adeguata per questo tool che, unita a molti esempi disponibili online, ha permesso di effettuare molti test di streaming, prima sulla rete locale e in un secondo momento sulla rete P2P.

Il passaggio alle librerie *libav* è stato meno immediato del previsto. Infatti, la documentazione e gli esempi per queste librerie sono più scarsi rispetto al tool da linea di comando. Sono stati di supporto alcuni repository GitHub, in particolare *FFmpeg-libav-tutorial* di Moreira [43], e il libro *Understanding FFmpeg with source code* [44].

Una volta ottenuto un primo risultato di screen casting sulla rete locale, ci si è concentrati sul processo di creazione e gestione della connessione Wi-Fi Direct. In questa fase si è esplorata la gestione della rete in Linux, partendo dal *NetworkManager* e analizzando gli strati sottostanti.

*NetworkManager* è la suite di strumenti standard per la configurazione di rete di Linux. Supporta un'ampia gamma di configurazioni di rete, dai desktop ai server e ai dispositivi mobili, e si integra bene con i più diffusi ambienti desktop e strumenti di gestione della configurazione dei server.

La maggior parte delle distribuzioni Linux consente di controllare *NetworkManager* direttamente dall'interfaccia grafica delle impostazioni di sistema. Esiste anche il tool da linea di comando `nmcli`, il quale permette di svolgere operazioni come visualizzare informazioni sulle interfacce di rete, le connessioni disponibili, attivare/disattivare connessioni e altro.

*NetworkManager* fornisce anche un'interfaccia D-Bus sul bus di sistema. È possibile utilizzare questa interfaccia per interrogare lo stato della rete e i dettagli delle interfacce di rete, come gli indirizzi IP correnti o le opzioni DHCP, e per attivare, disattivare, creare, modificare ed eliminare le connessioni di rete salvate.

Il problema principale di *NetworkManager* è che non supporta in modo completo le connessioni Wi-Fi Direct. Infatti, l'interfaccia grafica delle impostazioni di rete di Ubuntu non riporta nessun comando relativo alle reti P2P. Utilizzando `nmcli` o l'interfaccia D-Bus è possibile eseguire i metodi P2P `StartFind` e `StopFind`, ma non è stato trovato nessun metodo per creare un gruppo P2P.

Si è quindi passati a un livello inferiore, studiando *wpa\_supplicant*. Questo “demone” viene eseguito in background e agisce come componente di backend che controlla la connessione wireless. Per interagire con *wpa\_supplicant* è possibile

utilizzare un tool da linea di comando (*wpa\_cli*) o anche un'interfaccia grafica (*wpa\_gui*).

Si è partiti dall'utilizzo interattivo del tool `wpa_cli` per testare i metodi offerti dall'interfaccia di controllo e comprendere il flusso di esecuzione necessario per ottenere la connessione Wi-Fi Direct. Si è notato che è necessario tenere in considerazione le capacità della scheda di rete. Infatti, una scheda di rete meno recente non crea interfacce virtuali quando un gruppo P2P viene formato, al contrario di altre schede più attuali che permetterebbero anche di mantenere la connessione a una rete Wi-Fi tradizionale quando si partecipa a un gruppo Wi-Fi Direct.

La soluzione più immediata per iniziare a scrivere un programma che interagisse con `wpa_supplicant` poteva sembrare l'utilizzo della control interface tramite la libreria C `wpa_ctrl`. Nella fase di analisi, però, si è notato che l'utilizzo del D-Bus e in particolare della libreria `dbus-cxx` permettevano di semplificare l'interazione con `wpa_supplicant`, in particolare l'utilizzo dei segnali.

Per poter usufruire a pieno delle capacità P2P è necessario avviare un'istanza di `wpa_supplicant` con una configurazione custom. Per fare ciò, però, è necessario prima interrompere l'esecuzione del NetworkManager e dell'istanza di default di `wpa_supplicant`. È necessario quindi eseguire sul terminale i seguenti comandi con i privilegi di amministratore.

```
systemctl stop NetworkManager.service
killall wpa_supplicant
```

```
wpa_supplicant -Dnl80211 -iwlan0 -cwpa_supplicant.conf -B -u
```

I parametri passati all'ultimo comando servono per specificare le opzioni necessarie, in particolare:

- `-D` è seguito dal driver da usare per la gestione delle connessioni Wi-Fi
- `-i` serve per indicare l'interfaccia sulla quale `wpa_supplicant` dovrà operare
- `-c` si usa per specificare il percorso del file di configurazione
- `-B` avvia `wpa_supplicant` in modalità *daemon*
- `-u` abilita la D-Bus control interface

Per ripristinare la configurazione di rete è necessario interrompere l'istanza custom di `wpa_supplicant` e riavviare il NetworkManager

```
killall wpa_supplicant
systemctl restart NetworkManager
```

Il blocco seguente riporta un esempio di file di configurazione di `wpa_supplicant`, tipicamente denominato `wpa_supplicant.conf`.

Il `device_name` verrà impostato dal programma e sarà il nome identificativo del dispositivo.

`device_type=1-0050F204-1` significa che il dispositivo è un Computer/PC. Il codice è nel formato `categ-OUI-subcateg`.

```

1 update_config=1
2 ctrl_interface=/var/run/wpa_supplicant
3 country=IT
4 device_name=<DEVICE_NAME>
5 device_type=1-0050F204-1
6 manufacturer=WiFiWonders
7 config_methods=virtual_push_button

```

D-FeeT è stato uno strumento fondamentale al fine di testare in modo efficiente i metodi delle interfacce D-Bus. Nella Figura 3.1 si può notare la struttura dell'applicazione. Cliccando sugli elementi nella sezione a destra è possibile eseguire i metodi e ottenere i valori delle proprietà. Per mettersi in ascolto dei segnali invece è necessario utilizzare un tool da linea di comando chiamato `dbus-monitor`.

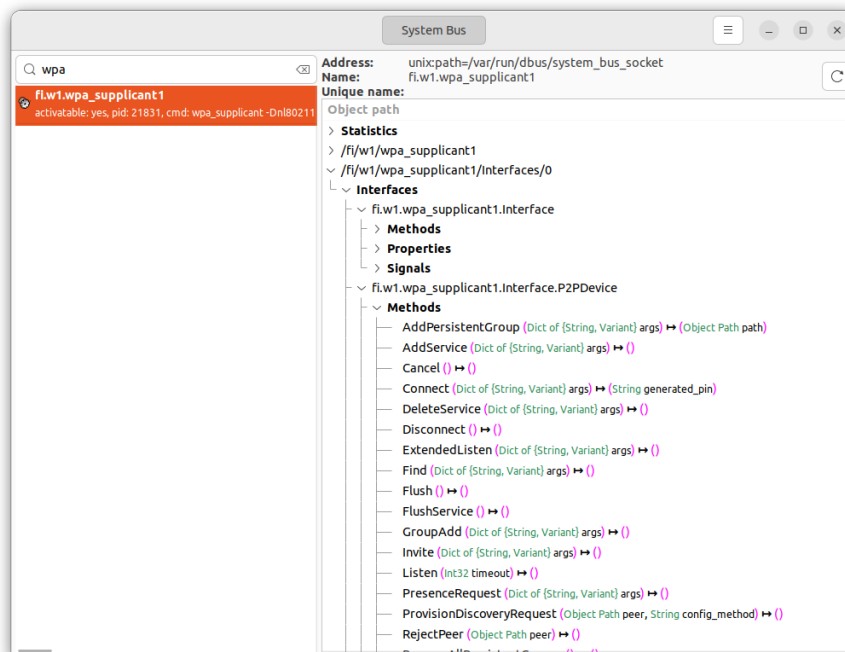


Figura 3.1: Screenshot D-FeeT

Richiamando il metodo `Introspect()` si ottiene un documento XML che descrive la struttura delle interfacce disponibili sul nodo D-Bus identificato dall'Object Path. Aggiungendo gli attributi richiesti e passando il file XML al tool `dbus-cxx-xml2cpp` è possibile generare le *proxy interfaces* del nodo e ottenere, così, classi C++ che permettono un'interazione semplice con il D-Bus e i moduli sottostanti.

Si riporta di seguito un esempio di documento XML e i file `.h` generati con il tool `dbus-cxx-xml2cpp`. Per ciascun documento XML viene generato un `ObjectProxy` per tutti i nodi presenti all'interno del file e un `InterfaceProxy` per ogni interfaccia presente all'interno dei nodi.

fi.w1.wpa\_supplicant1.xml

```

1 <?xml version="1.0"?>
2 <!DOCTYPE node PUBLIC "-//freedesktop//DTD D-BUS Object Introspection
3   1.0//EN"
4   "http://www.freedesktop.org/standards/dbus/1.0/introspect.dtd">
5 <node cppname="WpaSupplicant" gen-namespace="WpaSupplicant" dest="fi.
6   w1.wpa_supplicant1" path="/fi/w1/wpa_supplicant1">
7   <interface name="org.freedesktop.DBus.Introspectable">
8     <method name="Introspect">
9       <arg name="data" type="s" direction="out"/>
10    </method>
11  </interface>
12  <interface name="org.freedesktop.DBus.Properties">
13    <method name="Get">
14      <arg name="interface" type="s" direction="in"/>
15      <arg name="propname" type="s" direction="in"/>
16      <arg name="value" type="v" direction="out"/>
17    </method>
18    <method name="GetAll">
19      <arg name="interface" type="s" direction="in"/>
20      <arg name="props" type="a{sv}" direction="out"/>
21    </method>
22    <method name="Set">
23      <arg name="interface" type="s" direction="in"/>
24      <arg name="propname" type="s" direction="in"/>
25      <arg name="value" type="v" direction="in"/>
26    </method>
27  </interface>
28  <interface name="fi.w1.wpa_supplicant1">
29    <method name="CreateInterface">
30      <arg name="args" type="a{sv}" direction="in"/>
31      <arg name="path" type="o" direction="out"/>
32    </method>
33    <method name="RemoveInterface">
34      <arg name="path" type="o" direction="in"/>
35    </method>
36    <method name="GetInterface">

```

```

35     <arg name="ifname" type="s" direction="in" />
36     <arg name="path" type="o" direction="out" />
37 </method>
38 <method name="ExpectDisconnect" />
39 <signal name="InterfaceAdded">
40     <arg name="path" type="o" />
41     <arg name="properties" type="a{sv}" />
42 </signal>
43 <signal name="InterfaceRemoved">
44     <arg name="path" type="o" />
45 </signal>
46 <signal name="PropertiesChanged">
47     <arg name="properties" type="a{sv}" />
48 </signal>
49 <property name="DebugLevel" type="s" access="readwrite" />
50 <property name="DebugTimestamp" type="b" access="readwrite" />
51 <property name="DebugShowKeys" type="b" access="readwrite" />
52 <property name="Interfaces" type="ao" access="read" />
53 <property name="EapMethods" type="as" access="read" />
54 <property name="Capabilities" type="as" access="read" />
55 <property name="WFDIES" type="ay" access="readwrite" />
56 </interface>
57 <!-- <node name="Interfaces" dest="" /> -->
58 </node>

```

## WpaSupplicantProxy.h (ObjectProxy)

```

1 #ifndef WPASUPPLICANTPROXY_H
2 #define WPASUPPLICANTPROXY_H
3
4 #include <dbus-cxx.h>
5 #include <memory>
6 #include <stdint.h>
7 #include <string>
8 #include "fi_w1_wpa_supplicant1Proxy.h"
9 namespace WpaSupplicant {
10 class WpaSupplicantProxy
11 : public DBus::ObjectProxy {
12 public:
13 WpaSupplicantProxy(std::shared_ptr<DBus::Connection> conn, std::
    string dest = "fi.w1.wpa_supplicant1", std::string path = "/fi/w1/
    wpa_supplicant1" );
14 public:
15     static std::shared_ptr<WpaSupplicantProxy> create(std::shared_ptr
    <DBus::Connection> conn, std::string dest = "fi.w1.wpa_supplicant1
    ", std::string path = "/fi/w1/wpa_supplicant1", DBus::
    ThreadForCalling signalCallingThread = DBus::ThreadForCalling::
    DispatcherThread );

```

```

16     std::shared_ptr<WpaSupplicant::fi_w1_wpa_supplicant1Proxy>
17     getfi_w1_wpa_supplicant1Interface( );
18 protected:
19     std::shared_ptr<WpaSupplicant::fi_w1_wpa_supplicant1Proxy>
20     m_fi_w1_wpa_supplicant1Proxy;
21 };
22 } /* namespace WpaSupplicant */
23 #endif /* WPASUPPLICANTPROXY_H */

```

## fi\_w1\_wpa\_supplicant1Proxy.h (InterfaceProxy)

```

1 #ifndef FI_W__WPA_SUPPLICANT_PROXY_H
2 #define FI_W__WPA_SUPPLICANT_PROXY_H
3
4 #include <dbus-cxx.h>
5 #include <memory>
6 #include <stdint.h>
7 #include <string>
8 namespace WpaSupplicant {
9 class fi_w1_wpa_supplicant1Proxy
10 : public DBus::InterfaceProxy {
11 protected:
12 fi_w1_wpa_supplicant1Proxy(std::string name );
13 public:
14     static std::shared_ptr<WpaSupplicant::fi_w1_wpa_supplicant1Proxy>
15     create(std::string name = "fi.w1.wpa_supplicant1" );
16     DBus::Path CreateInterface(std::map<std::string,DBus::Variant>
17     args );
18     void RemoveInterface(DBus::Path path );
19     DBus::Path GetInterface(std::string ifname );
20     void ExpectDisconnect( );
21     std::shared_ptr<DBus::SignalProxy<void(DBus::Path, std::map<std::
22     string,DBus::Variant)>>> signal_InterfaceAdded( );
23     std::shared_ptr<DBus::SignalProxy<void(DBus::Path)>>>
24     signal_InterfaceRemoved( );
25     std::shared_ptr<DBus::SignalProxy<void(std::map<std::string,DBus
26     ::Variant)>>> signal_PropertiesChanged( );
27 private:
28     void setDebugLevel(std::string value );
29     std::string DebugLevel( );
30     std::shared_ptr<DBus::PropertyProxy<std::string>>
31     getProperty_DebugLevel( );
32     void setDebugTimestamp(bool value );
33     bool DebugTimestamp( );
34     std::shared_ptr<DBus::PropertyProxy<bool>>
35     getProperty_DebugTimestamp( );
36     void setDebugShowKeys(bool value );
37     bool DebugShowKeys( );

```



```

31     std::shared_ptr<DBus::PropertyProxy<bool>>
getProperty_DebugShowKeys( );
32     void setInterfaces(std::vector<DBus::Path> value );
33     std::vector<DBus::Path> Interfaces( );
34     std::shared_ptr<DBus::PropertyProxy<std::vector<DBus::Path>>>
getProperty_Interfaces( );
35     void setEapMethods(std::vector<std::string> value );
36     std::vector<std::string> EapMethods( );
37     std::shared_ptr<DBus::PropertyProxy<std::vector<std::string>>>
getProperty_EapMethods( );
38     void setCapabilities(std::vector<std::string> value );
39     std::vector<std::string> Capabilities( );
40     std::shared_ptr<DBus::PropertyProxy<std::vector<std::string>>>
getProperty_Capabilities( );
41     void setWFDIEs(std::vector<uint8_t> value );
42     std::vector<uint8_t> WFDIEs( );
43     std::shared_ptr<DBus::PropertyProxy<std::vector<uint8_t>>>
getProperty_WFDIEs( );
44 public:
45     std::shared_ptr<DBus::PropertyProxy<std::string>>
m_property_DebugLevel;
46     std::shared_ptr<DBus::PropertyProxy<bool>>
m_property_DebugTimestamp;
47     std::shared_ptr<DBus::PropertyProxy<bool>>
m_property_DebugShowKeys;
48     std::shared_ptr<DBus::PropertyProxy<std::vector<DBus::Path>>>
m_property_Interfaces;
49     std::shared_ptr<DBus::PropertyProxy<std::vector<std::string>>>
m_property_EapMethods;
50     std::shared_ptr<DBus::PropertyProxy<std::vector<std::string>>>
m_property_Capabilities;
51     std::shared_ptr<DBus::PropertyProxy<std::vector<uint8_t>>>
m_property_WFDIEs;
52 protected:
53     std::shared_ptr<DBus::MethodProxy<DBus::Path(std::map<std::string
,DBus::Variant>>> m_method_CreateInterface;
54     std::shared_ptr<DBus::MethodProxy<void(DBus::Path)>>
m_method_RemoveInterface;
55     std::shared_ptr<DBus::MethodProxy<DBus::Path(std::string)>>
m_method_GetInterface;
56     std::shared_ptr<DBus::MethodProxy<void()>>
m_method_ExpectDisconnect;
57     std::shared_ptr<DBus::SignalProxy<void(DBus::Path, std::map<std::
string,DBus::Variant>>> m_signalproxy_InterfaceAdded;
58     std::shared_ptr<DBus::SignalProxy<void(DBus::Path)>>
m_signalproxy_InterfaceRemoved;
59     std::shared_ptr<DBus::SignalProxy<void(std::map<std::string,DBus
::Variant>>> m_signalproxy_PropertiesChanged;
60 };

```

```
61 } /* namespace WpaSupplicant */  
62 #endif /* FI_W__WPA_SUPPLICANT_PROXY_H */
```

Durante lo sviluppo del modulo `wpa-supPLICANT-manager` si sono avuti problemi nelle operazioni di *demarshaling* di alcune proprietà particolarmente complesse e articolate dell'interfaccia `wpa_supplicant`. Il *demarshaling* è il processo di conversione dei dati serializzati, in questo caso provenienti dal D-Bus, in una struttura dati utilizzabile da un programma. Dopo un'attenta analisi ci si è resi conto che il problema era interno alla libreria D-Bus, perciò si è aperta un'*issue* sul repository GitHub che è stato prontamente risolto da uno dei *contributors* della libreria.

La segnalazione di un bug in un progetto open-source è un'azione di fondamentale importanza in quanto consente agli sviluppatori del progetto di venire a conoscenza del problema e di lavorare per risolverlo, contribuendo al miglioramento continuo del progetto, rendendolo più affidabile e funzionale. Il contributo alla comunità è utile anche quando non si è in grado di risolvere il problema perché fornire informazioni dettagliate e pertinenti sul bug può aiutare gli sviluppatori a risolverlo più rapidamente.

La prima versione di `wifi-direct-manager` permetteva di creare un gruppo, avviare la ricerca, connettersi a un peer trovato e disconnettersi da un gruppo da linea di comando.

Nel file `wpa_supplicant.conf` è stato aggiunto il campo `manufacturer` per filtrare i dispositivi che non eseguono il software sviluppato. Il filtro sarebbe stato implementato più correttamente attraverso la procedura di Service Discovery, ma l'interfaccia D-Bus di `wpa_supplicant` non permette di gestire in modo completo i servizi P2P. Infatti, analizzando il codice della libreria si è scoperto che supporta solo il servizio UPNP.

Per creare una connessione Wi-Fi Direct erano possibili due alternative illustrate in Figura 3.2 e in Figura 3.3. Le azioni eseguite dall'utente sono indicate con una sottolineatura, le operazioni automatiche sono in corsivo, mentre gli eventi sono indicati tra parentesi angolari.

Confrontando le due soluzioni si è notato che il flusso illustrato in Figura 3.2 permetteva di stabilire la connessione con più naturalezza e semplicità. Infatti, l'operazione di P2P GO Negotiation non sempre andava a buon fine, il gruppo non si riusciva a formare e i due dispositivi rimanevano in una situazione di stallo. Si è quindi deciso che l'applicazione Wi-Fi Wonders avrebbe offerto come unica possibilità quella di connettersi a un gruppo già esistente, evitando la procedura di GO Negotiation. L'utente che vuole condividere lo schermo del proprio dispositivo crea il gruppo Wi-Fi Direct e chi vuole partecipare alla condivisione effettua il join al gruppo già creato.

Device A (GO)	Device B (Client)
<u>Start group</u> <i>Start DHCP server</i>	<u>Start find</u>
	<DEVICE FOUND>
	<u>Connect</u>
<DEVICE FOUND>	
<PROV DISC REQ>	
<i>Start WPS</i>	
<PEER JOINED>	<GROUP STARTED>
	<i>Start DHCP client</i>
...	...
<u>Remove group</u>	or <u>Remove group</u>
<GROUP FINISHED> <PEER DISCONNECTED>	or <GROUP FINISHED>

**Figura 3.2:** Sequence diagram - P2P Create Group

Device A (GO)	Device B (Client)
<u>Start find</u>	<u>Start find</u>
<DEVICE FOUND>	<DEVICE FOUND>
<u>Connect</u>	<u>Connect</u>
<P2P GO Neg Req>	<P2P GO Neg Req>
<P2P GO Neg Success>	<P2P GO Neg Success>
<GROUP STARTED> (GO)	<GROUP STARTED> (Client)
<i>Start DHCP server</i>	<i>Start DHCP client</i>
...	...
<u>Remove group</u>	or <u>Remove group</u>
<GROUP FINISHED> <PEER DISCONNECTED>	or <GROUP FINISHED>

**Figura 3.3:** Sequence diagram - P2P GO Negotiation

Ottenuti i due moduli del progetto ci si è dedicati alla progettazione e all'implementazione dell'interfaccia grafica. Il framework Qt offre i componenti grafici QtWidgets e QtQuick. Si sono create interfacce di test con entrambe le tecnologie, ma non si sono ottenuti risultati adeguati in termini di design nei tempi prestabiliti. Inoltre l'IDE Qt Creator è risultato poco intuitivo e non ha offerto particolare supporto per la progettazione dell'interfaccia utente.

Si è scelto, quindi, di costruire l'interfaccia grafica basandosi sulle tecnologie web già note (HTML, CSS, Javascript, Bootstrap) e di implementare una classe AppProxy che permettesse di mettere in comunicazione l'interfaccia grafica con il resto del codice C++.

main.cpp

```
1 #include <QApplication>
2 #include <QWebEngineView>
3 #include <QtWebChannel/QtWebChannel>
4 #include "app_proxy.h"
5
6
7 int main(int argc, char **argv) {
8
9     QApplication app(argc, argv);
10    QWebEngineView view;
11
12    view.setWindowTitle("Wi-Fi Wonders");
13    view.setFixedSize(900, 600);
14
15    AppProxy app_proxy;
16    QWebChannel channel;
17    channel.registerObject(QStringLiteral("app_proxy"), &app_proxy);
18    view.page()->setWebChannel(&channel);
19
20    view.load(QUrl("qrc:/html-ui/index.html"));
21    view.show();
22
23    return QApplication::exec();
24 }
```

Come si può notare dal codice sorgente del main l'unico componente della QApplication è la QWebEngineView. Si è deciso di non permettere il ridimensionamento dell'applicazione per rendere l'interfaccia più standard e uniforme.

Per permettere lo scambio di informazioni tra l'applicazione C++ e l'interfaccia HTML/Javascript si è utilizzato un QWebChannel. Su questo canale viene registrato l'oggetto AppProxy che permette al codice Javascript di richiamare i metodi e connettersi ai segnali dell'oggetto registrato.

Nel file HTML è necessario utilizzare e configurare le API JavaScript fornite da `qwebchannel.js`. Per i client eseguiti all'interno di Qt WebEngine, è possibile caricare il file tramite `qrc:///qtwebchannel/qwebchannel.js`.

L'interfaccia web è stata integrata nell'eseguibile utilizzando il Qt resource system. Questo meccanismo permette di incorporare all'interno dell'eseguibile dell'applicazione un certo set di file (tipicamente icone, file di traduzione, immagini).

Una nota importante riguarda un conflitto tra le keyword `signals`, `slots` ed `emit` definite da Qt e quelle definite dalla libreria `sigc++`. Come riportato nel seguente blocco di codice è stato necessario rimuovere le definizioni delle keyword di Qt prima di includere il codice che usa al suo interno `dbus-cxx` che a sua volta usa `sigc++`. Effettuando questa operazione si è riusciti a risolvere il problema.

#### app\_proxy.h

```

4 #include <QObject>
5 #include <QMap>
6 #include <QString>
7
8 #undef slots
9 #undef signals
10 #undef emit
11
12 #include "wpa-supPLICANT-manager/ utils .hpp"
13 #include "wpa-supPLICANT-manager/wpa_supPLICANT_manager .h"
14 #include "wpa-supPLICANT-manager/P2PEvent .h"
15
16 #include "screen-casting-service/monitor_service .h"
17 #include "screen-casting-service/casting_service .h"
18
19 #define slots Q_SLOTS
20 #define signals Q_SIGNALS
21 #define emit

```

L'integrazione tra il modulo `wpa-supPLICANT-manager` e l'interfaccia grafica ha richiesto la risoluzione di un problema riguardo il processamento degli eventi. Infatti, al verificarsi di un evento P2P sarebbe stato necessario emettere un segnale Qt. Questo però non era possibile, dato che `wpa-supPLICANT-manager` non contiene nessun riferimento all'oggetto `AppProxy`. Si è quindi definita nella classe `AppProxy` una funzione che per ogni `P2PEvent` emette un `Q_signal`. Il costruttore della classe `WpaSupPLICANTManager` accetta opzionalmente una funzione che, se definita, viene invocata ogni volta che si verifica un evento di rete. In questo modo la funzione definita in `AppProxy` diventa una callback invocata dalla classe `WpaSupPLICANTManager` che permette di nascondere al modulo `wpa-supPLICANT-manager` i segnali di Qt, mantenendo la natura asincrona degli eventi.

Nelle fasi di sviluppo dell'applicazione *Wi-Fi Wonders* completa si è integrato in primo luogo il modulo `wpa-supPLICANT-manager`, avviando la condivisione dello schermo lanciando un processo `ffmpeg`. Una volta ottenuto un primo risultato funzionante si è integrato il modulo `screen-casting-service` che sfruttando le librerie `libav` ha permesso di ottenere ottime prestazioni per la trasmissione dello streaming.

Allo stesso modo, la riproduzione dello streaming è stata implementata in prima battuta lanciando un processo `ffplay`. Nel tentativo di passare alle librerie `libav` anche per la visualizzazione, però, si è notato che le performance e l'esperienza utente non miglioravano significativamente e perciò si è deciso di mantenere la gestione del processo `ffplay`.

Infatti, la scelta del protocollo RTMP, rispetto ad altri protocolli, non ha permesso di integrare lo streaming all'interno della pagina HTML. I flussi RTMP non sono più riproducibili all'interno delle pagine web da quando è stato interrotto il supporto di Adobe Flash Player.

La necessità di dover eseguire comandi di sistema per interrompere e riavviare il servizio del NetworkManager, creare un'istanza personalizzata di `wpa_supplicant` e interagire con essa attraverso il D-Bus di sistema ha costretto l'applicazione a richiedere i privilegi di amministratore per poter funzionare correttamente. Infatti, se si esegue l'applicazione come utente standard non è possibile eseguire le operazioni necessarie per la creazione della rete Wi-Fi Direct. L'utilizzo del NetworkManager per la gestione della rete avrebbe risolto questo problema, ma esso stesso non supporta in modo completo il P2P.

### 3.3 Confronto protocolli per lo streaming e formati video

Mentre per il codec si è subito trovata una risposta chiara su quale fosse la migliore scelta (H.264), non è stato lo stesso per il protocollo di trasmissione e il formato dello stream.

Per chiarezza si riportano i ruoli di protocollo, formato e codec nel contesto di uno streaming video.

Il **protocollo** di trasmissione determina come i dati video vengono trasmessi attraverso la rete. Deve gestire l'invio dei dati in modo efficiente, garantendo al contempo una riproduzione fluida e senza interruzioni. Alcuni esempi di protocolli comunemente utilizzati per lo streaming video sono HTTP, RTMP, RTSP e HLS.

Il **formato** video definisce la struttura dei dati video che viene trasmessa attraverso la rete. Include informazioni come risoluzione, frame rate, codec audio e

video utilizzati e altri parametri. I formati video devono essere compatibili con il protocollo di trasmissione utilizzato. Alcuni formati video popolari per lo streaming comprendono MPEG-4 Part 14 (MP4), FLV, WebM e MPEG-TS.

Il **codec** gestisce la compressione dei dati video durante lo streaming. Nei flussi video in tempo reale, è fondamentale utilizzare codec efficienti che possano comprimere i dati in modo da ridurre la larghezza di banda richiesta senza compromettere eccessivamente la qualità dell'immagine. Alcuni codec video comuni per lo streaming includono H.264, H.265, VP9 e AV1.

Si è partiti dalla guida allo streaming disponibile sul sito FFmpeg [45] per analizzare le soluzioni proposte.

In prima istanza si è pensato che il protocollo UDP (User Datagram Protocol) potesse essere il protocollo migliore, in quanto è un protocollo senza connessione che offre maggiore velocità e efficienza rispetto a TCP (Transmission Control Protocol) perché non ha il carico aggiuntivo di garantire la consegna dei dati in ordine e senza errori. Questo lo renderebbe adatto per applicazioni in tempo reale, come lo streaming video o audio, dove la velocità è fondamentale e la perdita di alcuni pacchetti dati occasionali può essere tollerata.

Il formato più adatto per lo streaming su UDP è MPEG-TS (MPEG transport stream). Questo formato è uno standard utilizzato per la trasmissione digitale di video, audio e altri dati. Si basa su una struttura a pacchetti che consente una trasmissione efficiente e flessibile ed è compatibile con i codec di compressione video MPEG-2, H.264 e H.265.

Inoltre, con il protocollo UDP è possibile sfruttare il multicast, che consente di inviare dati da un mittente a più destinatari contemporaneamente. Questo sarebbe vantaggioso perché permetterebbe di ottimizzare il traffico. Si è notato però che calano drasticamente le prestazioni dell'intera rete Wi-Fi, fino al completo intasamento della rete. Infatti, l'Access Point è obbligato a ridurre la velocità di trasmissione al proprio basic rate, ovvero al limite inferiore garantito da tutti i dispositivi connessi (1 Mbps), senza consentire alcuna forma di controllo sull'avvenuta ricezione. Questo determina un tasso estremamente elevato di pacchetti corrotti che vengono scartati alla ricezione, una riduzione della banda e un aumento della latenza. Il problema del multicast su reti Wi-Fi è spiegato nell'articolo di Wiens [46]. La soluzione dell'UDP multicast, quindi, non può essere considerata valida.

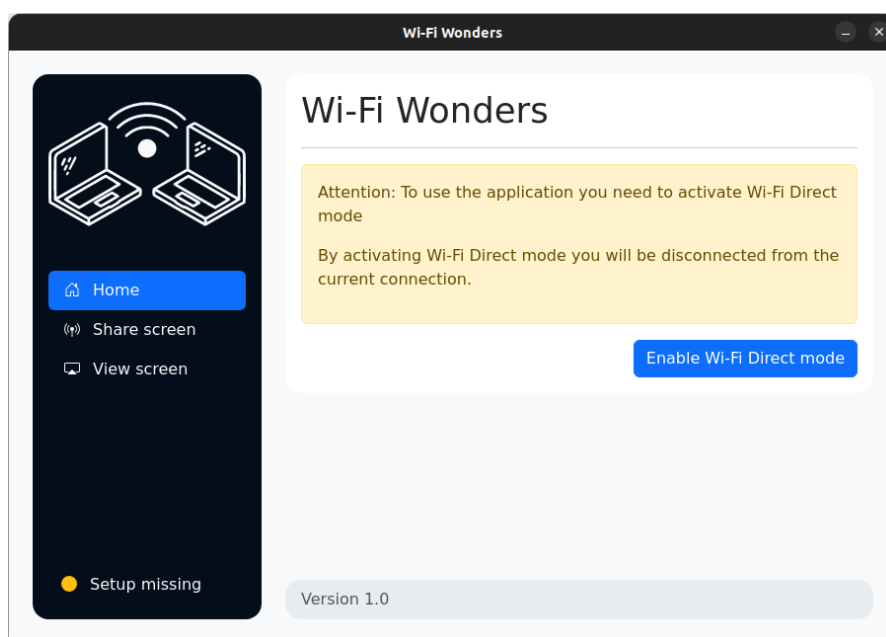
Si sono analizzati anche i protocolli HLS, MPEG-DASH, RTP e WebRTC. Queste soluzioni però non permettevano di raggiungere la latenza minima richiesta oppure necessitavano di configurazioni complesse, non adeguate all'applicazione che si stava sviluppando.

Il protocollo scelto è stato RTMP (Real-Time Messaging Protocol) che offre il vantaggio di ottenere uno streaming in tempo reale affidabile, oltre a una configurazione semplice. Il formato video più comunemente associato al protocollo RTMP è FLV (Flash Video). Questo formato ha soddisfatto i requisiti del progetto e perciò è stato scelto e utilizzato nel processo di screen casting. Inoltre, FLV supporta la codifica H.264, già considerata la scelta migliore per la compressione del video.

### 3.4 Interfaccia grafica

L'applicazione presenta un'interfaccia grafica moderna con un pannello laterale che include il logo e i bottoni per le tre schermate principali.

All'avvio viene chiesto all'utente di abilitare la modalità Wi-Fi Direct avvisandolo che verrà disconnesso dalla rete.



**Figura 3.4:** Wi-Fi Wonders - Schermata iniziale

Una volta inizializzata l'istanza custom di wpa\_supplicant l'utente può scegliere l'operazione da svolgere.



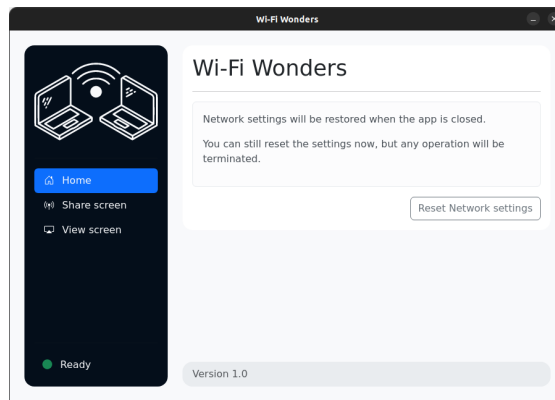


Figura 3.5: Wi-Fi Wonders - Home

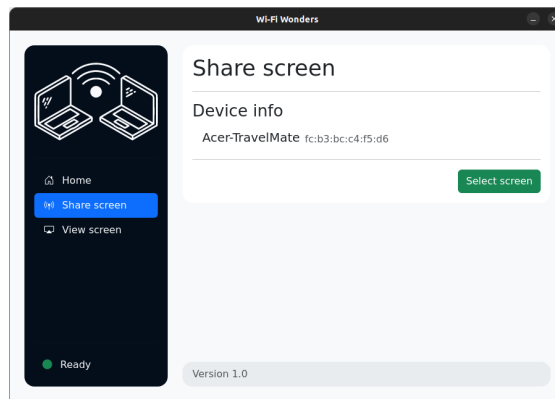


Figura 3.6: Wi-Fi Wonders - Share screen

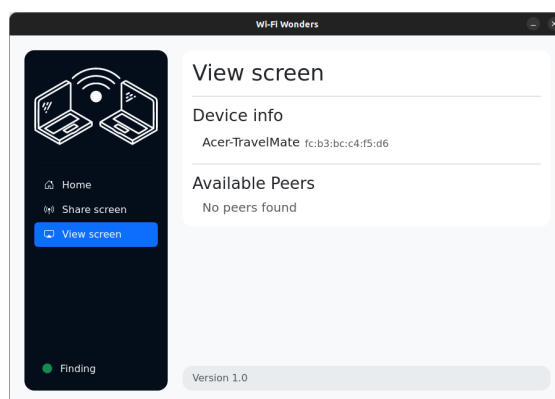


Figura 3.7: Wi-Fi Wonders - View screen

Per condividere lo schermo l'utente clicca sul bottone "Select screen" per selezionare lo schermo da condividere. Viene mostrata l'anteprima di ogni display connesso e dopo aver selezionato lo schermo desiderato è possibile avviare la condivisione.

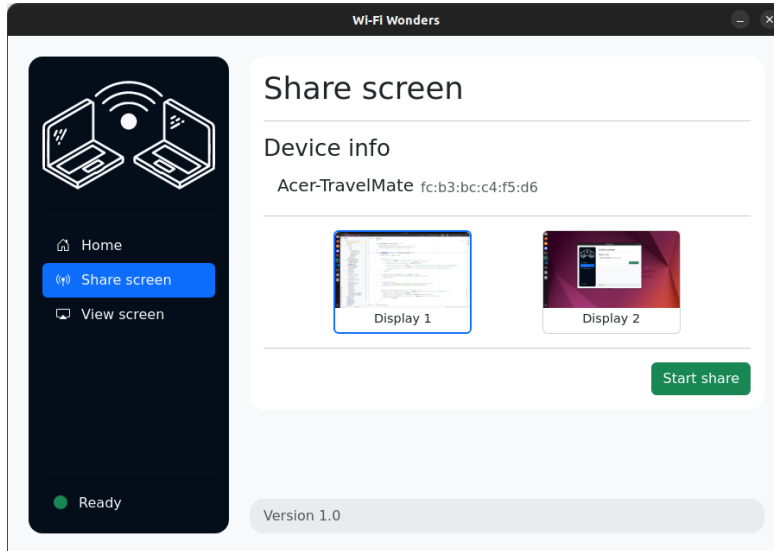


Figura 3.8: Wi-Fi Wonders - Selezione schermo

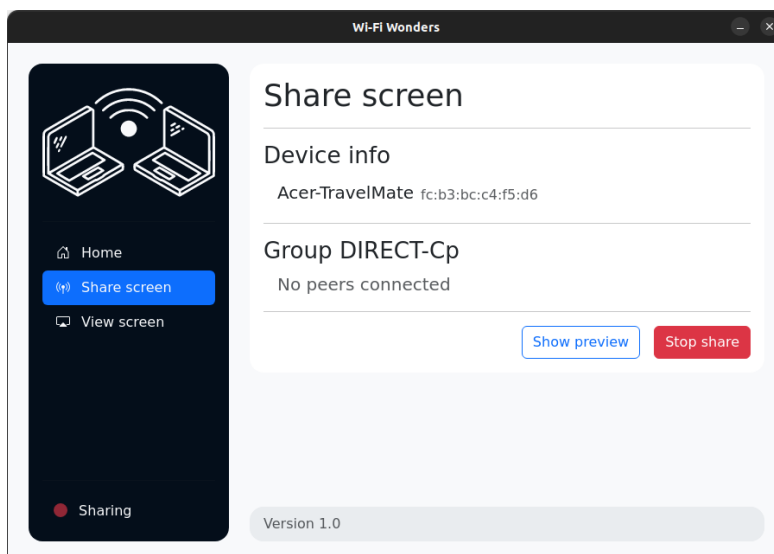


Figura 3.9: Wi-Fi Wonders - Condivisione avviata

A questo punto è stato creato anche il gruppo P2P a cui gli altri utenti possono partecipare per visualizzare lo schermo condiviso.

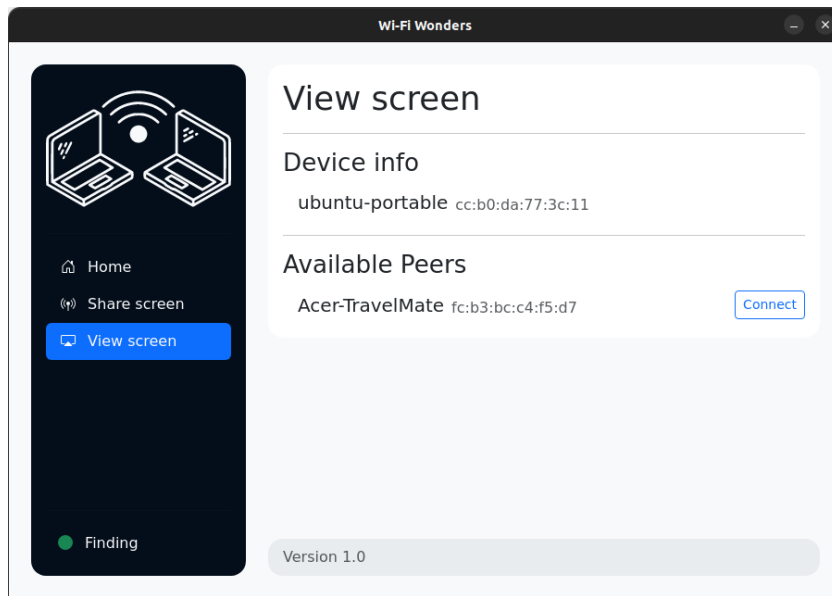


Figura 3.10: Wi-Fi Wonders - Ricerca dispositivi

Quindi il peer che ha avviato la condivisione compare nella lista dei peer disponibili e l'utente che vuole visualizzare lo streaming si connette al peer trovato.

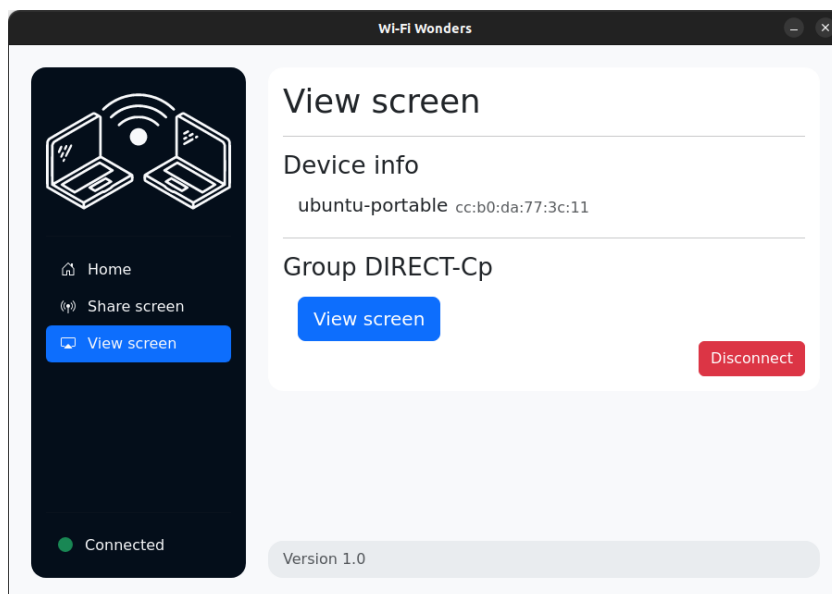


Figura 3.11: Wi-Fi Wonders - Client connesso

## Implementazione

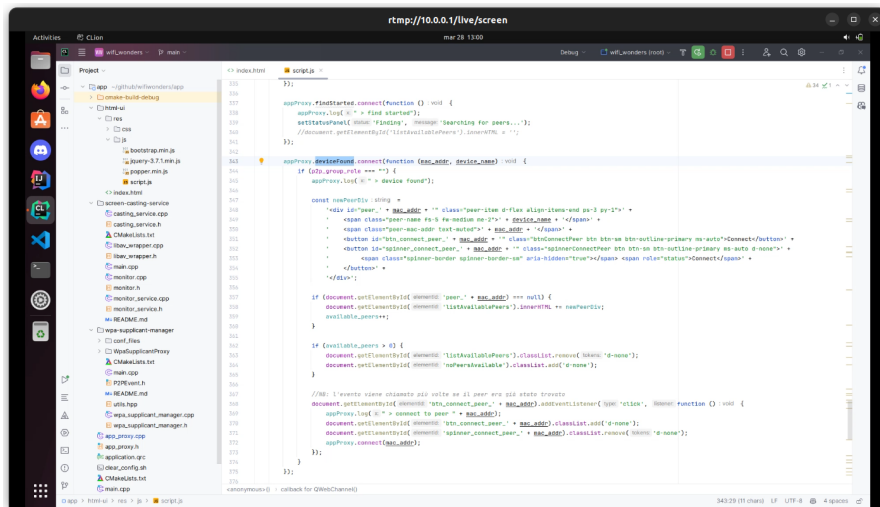


Figura 3.12: Wi-Fi Wonders - Visualizzazione streaming

Una volta stabilita la connessione il P2P Client può visualizzare lo streaming in una nuova finestra o anche a schermo intero. Dall'altro lato il P2P Group Owner vede il nuovo peer connesso e può eventualmente rimuoverlo dal gruppo.

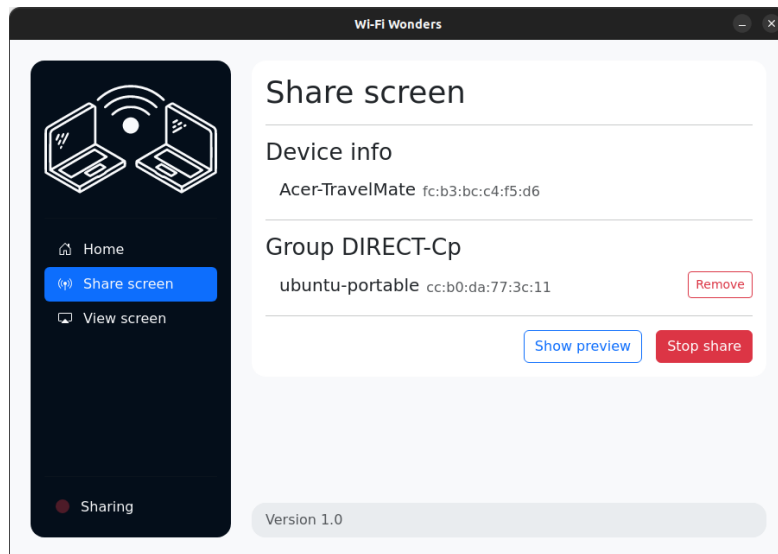


Figura 3.13: Wi-Fi Wonders - Visuale Group Owner

In un qualsiasi momento un client può disconnettersi dal gruppo e in questo caso la condivisione continua senza interruzioni per gli altri eventuali client connessi. Se, invece, il Group Owner interrompe la condivisione il gruppo termina per tutti i partecipanti e la finestra dove è visualizzato lo streaming viene chiusa.

A questo punto si torna allo stato iniziale, dopo l'attivazione della modalità Wi-Fi Direct, e si può continuare a utilizzare l'app come descritto sopra. Quando si chiude il programma resetta in automatico le impostazioni di rete.



## Capitolo 4

# Conclusioni e sviluppi futuri

Parte fondamentale di questo percorso è stata l'analisi delle tecnologie disponibili allo stato dell'arte attuale, ricercando la soluzione migliore in termini di efficienza, prestazioni e robustezza.

La documentazione ufficiale spesso scarsa ha richiesto una ricerca approfondita di fonti alternative utili allo studio delle tecnologie prese in considerazione e di esempi di codice che permettessero uno sviluppo più efficace.

Al termine dello sviluppo l'applicazione rispetta gli obiettivi prefissati. Offre un sistema di screen casting semplice per l'utente con un'interfaccia professionale che permette di condividere lo schermo a più dispositivi nelle vicinanze utilizzando lo standard Wi-Fi Direct, ottenendo ottimi risultati in termini di latenza dello streaming e prestazioni.

Da questa prima versione del progetto si potrebbe iniziare uno sviluppo professionale del software per aggiungere nuove feature e migliorare quelle già presenti.

Durante lo sviluppo sono state pensate e raccolte molte idee di funzionalità aggiuntive quali l'integrazione di una chat, un file sharing o la possibilità di controllare il desktop remoto.

Inoltre, il progetto potrebbe acquisire grande valore aggiungendo il supporto multi-piattaforma e quindi rendendo l'applicazione compatibile anche con gli altri sistemi operativi, Windows e MacOS, ma ancor di più se si riuscisse a sviluppare anche una versione mobile per smartphone e tablet, sia Android che Apple.

Un primo passo verso questa direzione potrebbe essere aggiungere il supporto a Wayland e gestire in maniera più user-friendly i permessi richiesti dall'interfaccia con la rete.

È possibile trovare il software realizzato per questo lavoro di tesi sul profilo GitHub personale al seguente link: <https://github.com/giovasini>



# Bibliografia

- [1] *Ethernet*. URL: <https://ethernetalliance.org> (cit. a p. 1).
- [2] *Wi-Fi*. URL: <https://www.wi-fi.org> (cit. a p. 1).
- [3] *Wi-Fi Direct*. URL: <https://www.wi-fi.org/discover-wi-fi/wi-fi-direct> (cit. a p. 2).
- [4] *Bluetooth*. URL: <https://www.bluetooth.com> (cit. a p. 2).
- [5] *Bluetooth vs Wi-Fi Direct*. URL: <https://www.dignited.com/23330/bluetooth-5-vs-wifi-direct-which-is-the-best-for-sharing-files-between-smartphones/> (cit. a p. 2).
- [6] *Bonjour*. URL: <https://developer.apple.com/library/archive/documentation/Cocoa/Conceptual/NetServices/Introduction.html> (cit. a p. 4).
- [7] *AirDrop*. URL: <https://en.wikipedia.org/wiki/AirDrop> (cit. a p. 4).
- [8] *AirPlay*. URL: <https://en.wikipedia.org/wiki/AirPlay> (cit. a p. 4).
- [9] *Chromecast*. URL: <https://en.wikipedia.org/wiki/Chromecast> (cit. a p. 5).
- [10] *Miracast*. URL: <https://www.wi-fi.org/discover-wi-fi/miracast> (cit. a p. 5).
- [11] *FFmpeg*. URL: <https://ffmpeg.org/> (cit. a p. 6).
- [12] *How to process video with FFmpeg*. URL: <https://medium.com/numatic-ventures/how-to-process-video-with-ffmpeg-framework-syntax-from-zero-to-hero-81812e8e8785> (cit. a p. 8).
- [13] *NetworkManager*. URL: <https://networkmanager.dev/> (cit. a p. 8).
- [14] M Vipin e S Srikanth. «Analysis of open source drivers for IEEE 802.11 WLANs». In: *2010 International Conference on Wireless Communication and Sensor Computing (ICWCSC)*. IEEE. 2010, pp. 1–5. URL: <https://ieeexplore.ieee.org/abstract/document/5415877> (cit. a p. 8).

- 
- [15] Andrés Marín López, Florina Almenáres Mendoza, Patricia Arias Cabarcos e Daniel Díaz Sánchez. «Wi-Fi Direct: Lessons learned». In: *2016 Mediterranean Ad Hoc Networking Workshop (Med-Hoc-Net)*. IEEE. 2016, pp. 1–8. URL: <https://ieeexplore.ieee.org/abstract/document/7528493> (cit. a p. 8).
- [16] *wpa\_supplicant*. URL: [https://w1.fi/wpa\\_supplicant/](https://w1.fi/wpa_supplicant/) (cit. a p. 8).
- [17] *D-Bus*. URL: <https://www.freedesktop.org/wiki/Software/dbus/> (cit. a p. 8).
- [18] *D-Feet*. URL: <https://wiki.gnome.org/Apps/DFeet> (cit. a p. 9).
- [19] *dbus-cxx library*. URL: <https://dbus-cxx.github.io/> (cit. a p. 9).
- [20] *Qt*. URL: [https://wiki.qt.io/About\\_Qt](https://wiki.qt.io/About_Qt) (cit. a p. 9).
- [21] *Qt Quick*. URL: <https://doc.qt.io/qt-6.2/qtquick-index.html> (cit. a p. 9).
- [22] *Qt Widgets*. URL: <https://doc.qt.io/qt-6.2/qtwidgets-index.html> (cit. a p. 9).
- [23] *Qt User Interfaces*. URL: <https://doc.qt.io/qt-6.2/topics-ui.html> (cit. a p. 9).
- [24] *Linux*. URL: <https://en.wikipedia.org/wiki/Linux> (cit. a p. 11).
- [25] *Ubuntu*. URL: <https://en.wikipedia.org/wiki/Ubuntu> (cit. a p. 11).
- [26] *X11*. URL: [https://en.wikipedia.org/wiki/X\\_Window\\_System](https://en.wikipedia.org/wiki/X_Window_System) (cit. a p. 12).
- [27] *Wayland*. URL: [https://en.wikipedia.org/wiki/Wayland\\_\(protocol\)](https://en.wikipedia.org/wiki/Wayland_(protocol)) (cit. a p. 12).
- [28] Daniel Camps-Mur, Andres Garcia-Saavedra e Pablo Serrano. «Device-to-device communications with Wi-Fi Direct: overview and experimentation». In: *IEEE wireless communications* 20.3 (2013), pp. 96–104. URL: <https://ieeexplore.ieee.org/abstract/document/6549288> (cit. a p. 12).
- [29] Wi-Fi Alliance. *Wi-Fi Direct<sup>®</sup> Specification v1.9*. Rapp. tecn. 2021. URL: [https://www.wi-fi.org/system/files/Wi-Fi\\_Direct\\_Specification\\_v1.9.pdf](https://www.wi-fi.org/system/files/Wi-Fi_Direct_Specification_v1.9.pdf) (cit. alle pp. 12, 14).
- [30] Kemparaj Praneeth. *P2P Device Discovery*. URL: <https://praneethwifi.in/2019/10/26/p2p-device-discovery/> (cit. a p. 14).
- [31] Kemparaj Praneeth. *P2P Service Discovery*. URL: <https://praneethwifi.in/2019/10/27/p2p-service-discovery/> (cit. a p. 15).
- [32] *wpa\_supplicant developers' documentation*. URL: [https://w1.fi/wpa\\_supplicant/devel/index.html](https://w1.fi/wpa_supplicant/devel/index.html) (cit. a p. 19).

- [33] *wpa\_supplicant P2P module*. URL: [https://w1.fi/wpa\\_supplicant/development/p2p.html](https://w1.fi/wpa_supplicant/development/p2p.html) (cit. a p. 19).
- [34] *wpa\_supplicant D-Bus API*. URL: [https://w1.fi/wpa\\_supplicant/development/dbus.html](https://w1.fi/wpa_supplicant/development/dbus.html) (cit. a p. 21).
- [35] *QWebEngineView Class*. URL: <https://doc.qt.io/qt-6.2/qwebengineview.html> (cit. a p. 22).
- [36] *Qt WebEngine Overview*. URL: <https://doc.qt.io/qt-6.2/qtwebengine-overview.html> (cit. a p. 22).
- [37] *Chromium Project*. URL: <http://www.chromium.org/> (cit. a p. 22).
- [38] *Qt WebChannel*. URL: <https://doc.qt.io/qt-6.2/qtwebchannel-index.html> (cit. a p. 22).
- [39] *RTMP*. URL: [https://en.wikipedia.org/wiki/Real-Time\\_Messaging\\_Protocol](https://en.wikipedia.org/wiki/Real-Time_Messaging_Protocol) (cit. a p. 25).
- [40] *FLV*. URL: [https://en.wikipedia.org/wiki/Flash\\_Video](https://en.wikipedia.org/wiki/Flash_Video) (cit. a p. 25).
- [41] *H.264*. URL: [https://en.wikipedia.org/wiki/Advanced\\_Video\\_Coding](https://en.wikipedia.org/wiki/Advanced_Video_Coding) (cit. a p. 25).
- [42] *CMake*. URL: <https://cmake.org/> (cit. a p. 27).
- [43] Leandro Moreira. *FFmpeg-libav-tutorial*. URL: <https://github.com/leandromoreira/ffmpeg-libav-tutorial> (cit. a p. 29).
- [44] Akin Yaprakgöl. *Understanding FFmpeg with source code*. 2021 (cit. a p. 29).
- [45] *FFmpeg Streaming guide*. URL: [http://trac.ffmpeg.org/wiki/Streaming\\_Guide](http://trac.ffmpeg.org/wiki/Streaming_Guide) (cit. a p. 41).
- [46] Nathan Wiens. *Multicast over Wireless*. URL: <https://wirelesslywired.com/2019/05/02/multicast-over-wireless/> (cit. a p. 41).