Alma Mater Studiorum - Università di Bologna

Scuola di Scienze
Corso di Laurea in Informatica

Progettazione ed implementazione di un'applicazione per la condivisione di stream MIDI

Relatore:
Chiar.mo Prof.
Marco Di Felice

Correlatore:
Dott. Luca Sciullo

Presentata da:
Marius Sili

Sessione II
Anno Accademico 2019-2020

Sommario

La musica e i metodi attraverso i quali essa viene creata, condivisa e riprodotta hanno sempre avuto importanza per le persone. Nonostante esistano una moltitudine di servizi che permettono lo streaming di prodotti finiti, al giorno d'oggi risulta particolarmente difficile riuscire a suonare e creare brani musicali a distanza, in tempo reale. Non vi è alcun servizio che permetta una gestione semplice e funzionante di tale processo. L'obiettivo della ricerca e del progetto qui presentati è quello di fornire un modo intuitivo in cui un qualsiasi utente provvisto di strumento musicale MIDI e di dispositivi ormai largamente diffusi, come uno smartphone o un tablet, possa effettuare una "Jam Session", cioè una sessione in cui suonare in tempo reale insieme ad altre persone collegate in remoto. I concetti che vengono affrontati sono legati al mondo dell'Internet of Things e della Networked Music Performance. Questa ricerca può portare grandi benefici al processo creativo di brani musicali, alle esibizioni "dal vivo" e alla didattica musicale.

Assieme al progetto, viene descritta una implementazione e uno studio di fattibilità. Si vedrà come il sistema proposto possa essere a tutti gli effetti utilizzato ed eventualmente ampliato in futuro.

Indice

Capitolo 1

Introduzione

Sin dalla preistoria la musica ha suscitato l'interesse delle persone. C'è un qualcosa di "magico" all'interno di essa che stimola l'ascoltatore a creare immagini e sensazioni che per altre vie sarebbe difficile se non impossibile evocare. Sicuramente vi è un grado di soggettività notevole dietro all'interpretazione della musica, ma è innegabile il contributo del tempo in tutto ciò: l'evoluzione della cultura musicale, i cambiamenti all'interno della società e la creazione di nuovi strumenti musicali e di nuove opzioni di creare, comporre e produrre. Le differenze tra il modo in cui Debussy racconta il mare e come lo potrebbe fare un (non così) allegro menestrello del Medioevo o magari un produttore di EDM al giorno d'oggi sono date in gran parte da fattori tecnologici, oltreché culturali e sociali. E si potrebbe supporre che in certi casi sia stata proprio l'evoluzione tecnologica a modificare gli altri due fattori sia per quanto riguarda le modalità di ascolto che quelle di composizione.

L'invenzione e la diffusione del grammofono e poi del giradischi ha portato la musica nelle case delle persone. Ciò che prima si poteva ascoltare soltanto all'interno di una sala teatrale diventa riproducibile in un ambiente più intimo. La diffusione della radio ha successivamente modificato ulteriormente le modalità di ascolto e di creazione. Si inizia ad avere la necessità di soddisfare i gusti di una molteplicità di persone, di ceti sociali diversi. Ciò porta a modifiche strutturali e temporali all'interno dei brani musicali. Diventa sempre più importante il concetto di popolarità del brano il quale si spera che venga riprodotto più spesso possibile dalle emittenti radiofoniche. I servizi di streaming e le piattaforme di condivisione video odierne hanno cambiato ulteriormente le carte in gioco. È sì importante la popolarità, ma diventa sempre più facile per un artista trovare una propria nicchia che gli permetta di produrre melodie meno familiari, ma comunque apprezzabili da determinate persone.

Le tecnologie che abbiamo a disposizione arricchiscono le possibilità di creare brani, nonché di collaborare nel processo. La collaborazione è un fattore chiave (solitamente Do maggiore) nella musica, in quanto permette la fusione di idee e conoscenze musicali e tecnologiche. La collaborazione è un processo creativo e più possibilità hanno le persone di partecipare al processo, più variopinto diventerà il risultato finale. Un passo incredibile fu fatto negli anni '80 con la definizione dello standard MIDI il quale ha reso possibile l'interfacciamento di dispositivi eterogenei, di marche diverse. RTP-MIDI è un ulteriore passo avanti che permette la collabo-

razione di più utenti nella stessa rete locale. Tuttavia c'è un problema, al giorno d'oggi è incredibilmente macchinoso e lento collaborare in remoto. Per collaborare a distanza, due persone devono avere conoscenze informatiche che vanno a rovinare l'appeal di tale processo. Sarebbe incredibile poter permettere a due utenti di iniziare a suonare insieme, in tempo reale, facendo un semplice tap sul proprio smartphone, pur essendo a kilometri di distanza! A volte per cause maggiori diventa l'unica soluzione viabile ed è giusto creare questa possibilità. Questo è il problema che si pone il campo di ricerca della Networked Music Performance ed è su questo che si concentra la ricerca e il progetto descritti all'interno di questa tesi. Il proseguimento della ricerca in questo ambito può portare migliorie non soltanto al processo creativo, ma anche alle esibizioni "dal vivo" e all'istruzione musicale.

1.1 Progetto

L'obiettivo principale di questo studio è quello di permettere a qualsiasi utente in possesso di uno smartphone o di un tablet di collegarsi con un certo grado di facilità ad una sessione remota nella quale suonare in tempo reale insieme ad altri utenti. Gli utenti devono essere provvisti di strumento musicale MIDI, ma si dovrebbe permettere anche l'utilizzo di strumenti sullo schermo del dispositivo mobile in una eventuale implementazione futura.

Prima di fare ciò, si analizzeranno vari protocolli da utilizzare nella spedizione in tempo reale di pacchetti MIDI. Si sceglierà quello più adatto e a quel punto verrà sviluppato un meccanismo attraverso il quale i vari dispositivi mobile possano effettuare il collegamento tra di loro. Ciò che risulterà sarà una combinazione di architettura client-server per il tracciamento delle varie connessioni, mista al peer-to-peer per la spedizione veloce dei segnali musicali. Il progetto presentato sarà anche in grado di effettuare la generazione di segnali audio e gestire dei sintetizzatori virtuali per tutti gli utenti partecipanti alla sessione. Verrà infine descritta l'implementazione proposta, che vede l'utilizzo dell'ambiente iOS in quanto provvisto di molti tool per la gestione del MIDI. L'architettura del progetto proposto è tuttavia indipendente dalla piattaforma sulla quale viene implementato, a patto che vengano supportati i protocolli necessari i quali sono comunque standard.

Oltre alla definizione dell'architettura e all'implementazione, viene proposto uno studio di fattibilità e una discussione sui possibili sviluppi della ricerca in questo ambito la quale può avere un impatto globale.

Il resto della tesi è così suddiviso:

- Nel Capitolo 2 viene presentato lo stato attuale della ricerca nell'ambito della condivisione dei segnali MIDI, dei suoi sviluppi e delle difficoltà da affrontare;
- Nel Capitolo 3 vine effettuata l'illustrazione dettagliata dell'architettura e funzionalità del progetto proposto;
- Nel Capitolo 4 si descrive l'implementazione proposta;
- Nel Capitolo 5 si spiega il processo di validazione del progetto;
- Il Capitolo 6 presenta la conclusione e una discussione sui possibili sviluppi del progetto, anche in relazione con le ricerche e tecnologie odierne.

Capitolo 2

Stato dell'arte

In questo capitolo si vuole illustrare lo stato attuale delle tecnologie e protocolli tramite i quali sono state effettuate le prove e l'implementazione finale.

Siccome la ricerca si basa su segnali musicali, viene preso in considerazione lo standard MIDI. All'interno di questo capitolo viene fatta una descrizione di questo standard che è stato rivoluzionario per il mondo della musica.

Viene proposta una descrizione dell'IoT (Internet of Things), assieme ad un confronto tra i principali protocolli di rete (TCP, UDP e RTP-MIDI) e richiami a ricerche precedenti, con un focus sull'interconnessione di strumenti musicali tramite reti fisiche e virtuali [11]. Discuteremo inoltre il concetto di Intenet of Musical Things (IoMusT) [15] e i progressi e problemi attuali della ricerca in questo ambito.

Figura 2.1: Controller MIDI a due ottave, popolare grazie alla sua portabilità (https://en.wikipedia.org/wiki/MIDI/media/File:Remote_25.jpg)

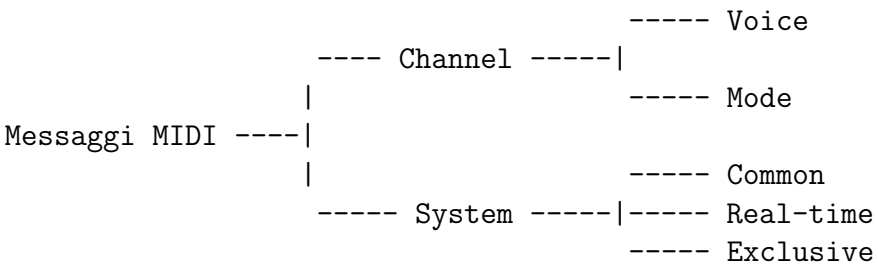
2.1 MIDI

Musical Instrument Digital Interface[4] (MIDI) è uno standard concordato nel 1982 che descrive un protocollo di comunicazione, un'interfaccia digitale e dei connettori elettrici che favoriscono la connessione e comunicazione tra strumenti musicali elettronici, computer e ulteriori dispositivi dedicati alla registrazione e editing musicale. Prima dell'avvento di tale standard era impossibile assicurare la comunicazione corretta tra dispositivi di produttori diversi che non venivano resi compatibili tra di loro.

2.1.1 Comunicazione

I segnali MIDI vengono trasmessi tramite un apposito cavo attraverso il quale possono passare fino a 16 canali di informazione, ognuno dei quali può essere indirizzato verso uno specifico dispositivo. Al giorno d'oggi è possibile effettuare il collegamento MIDI anche tramite cavo USB e tecnologie wireless come il Bluetooth se si fa uso dello standard RTP-MIDI che verrà descritto nella sezione 2.3.3. I messaggi vengono passati sotto forma di pacchetti e sono divisi in due tipi principali:

- **Channel** - Messaggi indirizzati ad uno dei 16 canali. I più importanti in questa categoria sono "noteon" e "noteoff" che denotano rispettivamente la pressione (trigger) e il rilascio (release) di una nota;
- **System** - Messaggi indirizzati all'intero sistema.



2.1.2 Evento

Il MIDI permette il trasporto dei seguenti eventi:

- **Nota** - Denotata dalla notazione, altezza (pitch) e dinamica (velocity);
- **Vibrato** - Cambiamento pulsante, regolare dell'altezza di una nota;
- **Panning** - Spostamento dell'effetto stereo più a sinistra o più a destra;
- **Clock** - Utile per definire il tempo.

Quando uno strumento MIDI viene suonato, qualsiasi azione fisica su di esso viene tradotta in segnale MIDI, generando quindi dati del tipo appena descritto. Per tradurre i dati in suoni si fa utilizzo dei sound module, i quali contengono dei suoni musicali sintetizzati (pianoforte, flauto traverso, chitarra ecc.) che vengono successivamente riprodotti tramite un amplificatore. Ovviamente un computer può eseguire tutti i ruoli se provvisto di librerie che astraggono le funzionalità di tali dispositivi.

2.1.3 Pacchetto MIDI

Un pacchetto MIDI è composto da tre byte:

- Uno **Status Byte** - Il primo bit ha sempre valore 1, i seguenti tre identificano il tipo di messaggio e i restanti quattro il canale sul quale deve essere trasmesso il pacchetto;
- Due **Data Byte** - Il primo bit di ciascuno di essi ha sempre valore 0 e i restanti denotano il valore del dato trasmesso (e.g. il numero della nota).

Figura 2.2: Pacchetto MIDI

2.1.4 File MIDI

Lo standard definisce inoltre un formato per i file. I vantaggi principali sono la facilità di modifica, la grandezza ridotta e il suo essere agnostico nei confronti dello strumento che viene sintetizzato. I file MIDI non contengono dati audio, ma istruzioni che i dispositivi compatibili interpretano generando segnali audio.

2.2 Internet of Things

Se con Internet identifichiamo l'interconnessione di computer e di reti di computer, con Internet of Things (IoT) si astrae ulteriormente e si inizia a parlare di interconnessione di oggetti ("things") di natura più semplice e con uno spettro di utilizzo più specifico. Per fare un esempio ormai molto conosciuto possiamo parlare di domotica dove i dispositivi di utilizzo più comune come la televisione, la lavatrice, il termostato ecc. sono dotati di una scheda di rete che permette a loro di effettuare collegamenti via Internet e comunicare sia tra di loro, sia con dei servizi esterni, esponendo delle API pubbliche attraverso le quali diventa possibile controllarli esternamente.

La definizione di IoT si è sempre evoluta con il subentro di nuove tecnologie e nuovi paradigmi e protocolli di comunicazione. I settori che hanno contribuito maggiormente alla nascita di questo concetto sono i sistemi embedded, le Wireless Sensor Networks (WSN) e l'automazione. L'armonia tra questi settori che al giorno d'oggi si presenta nell'IoT è frutto della costante ricerca in ambito accademico e industriale.

La diffusione esponenziale che l'IoT ha avuto in questi anni alimenta tuttavia preoccupazioni riguardanti la sicurezza e la privacy, fattori che sono spesso trascurati.

2.2.1 Dispositivi

Quando si parla di "Things", si denotano dispositivi generalmente suddivisi in due categorie principali: sensori ed attuatori.[8]

Sensori

Sono quei dispositivi dedicati alla ricezione dei fenomeni che occorrono nell'ambiente a loro circostante. Per ogni evento ricevuto forniscono dei dati in output. Siccome molti sensori ricevono segnali analogici e restituiscono segnali in formato digitale, sono spesso dotati di convertitori analogico-digitale. I sensori sono un elemento cruciale nell'IoT in quanto forniscono l'interfaccia tra il mondo fisico e quello digitale.

I sensori possono essere molto semplici oppure più complessi, con la possibilità di filtrare i dati e notificare il gateway in condizioni molto specifiche. Siccome può essere richiesta della logica di programmazione, spesso i dispositivi sono composti da tre parti: sensori, microcontrollori e scheda di rete per spedire i dati elaborati ad un determinato gateway.

In base alle caratteristiche dell'ambiente circostante che devono captare, i sensori sono suddivisi in varie categorie, tra le quali: temperatura, pressione, flusso di fluidi, livello di fluidi, sensori di immagini, rumore, livello di inquinamento dell'aria, prossimità, infrarossi, umidità, velocità.

Attuatori

Un attuatore è un dispositivo che prende in input dei dati e in base ai loro valori esegue delle determinate azioni fisiche. Solitamente i dati che essi ricevono derivano da elaborazioni più o meno complesse effettuate sui dati fisici captati dai sensori. Gli attuatori sono necessari all'automazione all'interno dell'IoT in quanto sono proprio quei dispositivi che effettuano le azioni di nostro interesse in risposta ad un determinato fenomeno. Un esempio potrebbe essere una valvola termostatica motorizzata che attiva il flusso d'acqua all'interno di un termosifone quando la temperatura dell'ambiente scende sotto un determinato valore pre-impostato e blocca il flusso quando la temperatura raggiunge il valore desiderato.

Esistono vari tipi di attuatori:

- **Elettrici** - Motori che convertono l'energia elettrica in movimento meccanico;
- **Meccanici Lineari** - Convertono il movimento rotatorio in movimento lineare;
- **Idraulici** - Dispositivi con parti meccaniche che vengono applicati alle valvole idrauliche. Il loro design è basato sulla legge di Pascal;
- **Pneumatici** - Simili agli attuatori idraulici, ma operano sulle valvole di gas;
- **Manuali** - Dispositivi che richiedono lo sforzo umano per poter essere utilizzati. Possono essere utili per ragioni di sicurezza.

Controllo

Il monitoraggio e il controllo dei dispositivi IoT può essere effettuato in due modi: locale o globale.[8] Il primo favorisce l'utilizzo di una moltitudine di controllori locali "intelligenti" che possono effettuare elaborazioni più complesse in base ai dati ambientali ricavati. Il secondo approccio sposta totalmente il controllo e l'elaborazione dei dati su un host in rete che riceve i dati di tutti i sensori collegati in remoto.

Spesso le soluzioni adottate sono un ibrido tra le due filosofie, basate sul prodotto finale che si vuole implementare e distribuire.

2.2.2 Applicazioni

Al giorno d'oggi l'IoT riscontra applicazioni in una moltitudine di ambiti che sono spesso raccolti nelle seguenti categorie:[6]

- **Applicazioni per consumatori** - Presentano ecosistemi creati appositamente per gli utenti finali. L'ambito della domotica, come già descritto appartiene a questa categoria e vede grosse aziende come Amazon, Google e Apple che concorrono per proporre e vendere agli utenti la propria soluzione. Un altro ambito in cui si sta avendo un rapido sviluppo è la medicina, dove vengono principalmente proposti metodi per ricavare dati e controllare sensori wearable, automatizzando la creazione e l'analisi di dati statistici legati alla salute;

- **Applicazioni commerciali** - Nell'ambito manifatturiero si trattano principalmente meccanismi che automatizzano processi di produzione e controllo. Nell'ambito agricolo vengono proposte implementazioni che fanno utilizzo di sensori per collezionare dati relativi a umidità, temperatura, probabilità di pioggia ecc. e controllarli laddove possibile, per esempio all'interno di una serra;

- **Applicazioni infrastrutturali** - Attività di monitoraggio e controllo di infrastrutture urbane e rurali come semafori, ponti ecc. appartengono a questa categoria. Implementazioni in questi ambiti contribuiscono inoltre a ricavare dati statistici molto utili all'industria edilizia.

2.2.3 IoT e MIDI

Per quanto riguarda l'ambito musicale, l'IoT diventa molto interessante nella gestione dei dispositivi MIDI collegati in remoto. Implementazioni odierne si possono incontrare in applicazioni come Garage Band di Apple, che danno la possibilità agli utenti di suonare insieme nella stessa rete locale.

Una interessante applicazione dell'IoT è stata fatta nella implementazione di un "palcoscenico virtuale": un'applicazione grafica per dispositivi desktop che facilita la gestione della connessione dei dispositivi MIDI in rete locale. Per permettere ciò, è stato implementato uno switch virtuale[11] che configura e gestisce le connessioni di nodi all'interno di una rete e permette lo scambio di messaggi MIDI tra di essi. Il suo funzionamento è complementato da un nodo intermedio che ha il compito di riprodurre, leggere o inoltrare il traffico. L'implementazione è composta da dispositivi hardware che richiedono un certo grado di conoscenze informatiche per poter essere utilizzati in maniera ideale.

Questo meccanismo, pur essendo fatto a regola d'arte, richiede abilità in ambito di reti e IoT da parte dell'utente finale. Non sarebbe una soluzione ideale quella di chiedere ad ogni utente di acquistare e configurare degli appositi microcontrollori hardware. La soluzione che verrà proposta nel capitolo seguente utilizzerà un ibrido tra l'architettura client-server e peer-to-peer che vede l'uso degli smartphone, ormai largamente diffusi, per astrarre i comportamenti di una moltitudine di dispositivi, facilitando in questo modo l'usabilità del servizio.

La strada che si percorre nella ricerca e progettazione attuale consiste nell'analizzare i due protocolli di trasporto principali (TCP e UDP) e metterli a confronto con RTP-MIDI al fine di determinare il candidato ideale per l'implementazione finale. Si vuole inoltre evitare la presenza di dispositivi hardware che un utente non possiede già di default.

2.2.4 Internet of Musical Things

È stata presentata una visione sull'emergente campo di ricerca riguardante l'Internet of Musical Things (IoMusT) [15], il quale trova origine in molte delle ricerche attualmente in corso nei seguenti ambiti:

- **Internet of Things**
- **Networked music performance** - Si intende un insieme di interazioni in real-time (esibizioni, ripetizioni, lezioni di musica ecc.) che avvengono su una rete, permettendo ad utenti geograficamente distanti di suonare come se si trovassero nella stessa stanza;
- **Ubiquitous music** - Un ambito di ricerca interdisciplinare che combina la musica insieme all'informatica, l'educazione, la psicologia e l'ingegneria;
- **Artificial Intelligence** - Ambito di studio che si occupa della ricerca e sviluppo di tecniche e tecnologie per il miglioramento dell'intelligenza delle macchine;
- **Human-computer interaction** - Ambito di studio che si focalizza sulle interfacce che mettono in relazione i computer con le persone;
- **Participatory art** - Un approccio all'arte che fa contribuire il pubblico nel processo creativo.

Con il termine IoMusT si intendono le reti di dispositivi dedicati a fini musicali. IoMusT permette diverse forme di interconnessione tra artisti, insegnanti, ingegneri del suono e ascoltatori sia in reti locali, che in remoto. La visione di una tecnologia simile offre molte opportunità, ma allo stesso tempo deve affrontare molte difficoltà sia tecniche che non per potersi realizzare, difficoltà che si spera possano essere sormontate tramite l'unione della ricerca accademica e quella industriale.

Una delle sfide più grandi dal punto di vista ingegneristico consiste nella trasmissione a bassa latenza di stream di dati audio ad alta qualità su reti wireless e wired. Questa sfida deve essere affrontata tramite lo sviluppo di nuovi metodi che garantiscano bassa latenza e intervalli di ricezione stabili. Quest'ultima è difficile da garantire in quanto la trasmissione di messaggi sia via cavo che tramite canali wireless è sottoposta ad interferenze di natura aleatoria e a traffico in background altrettanto aleatorio, che danno vita a delay random tra la spedizione e la ricezione del messaggio che sono solitamente difficili da controllare. Per risolvere questi problemi l'articolo afferma la necessità di nuovi metodi di gestione del traffico ai livelli fisico, MAC e routing e propone l'applicazione a questi livelli di un'estensione della teoria dell'ottimizzazione che faccia uso di algoritmi di computazione veloci che operino in real-time. Questi metodi dovranno operare sia nel contesto delle tecnologie 5G che delle WLAN.

Il successo di IoMusT dipende fortemente dalle attività di standardizzazione, le quali sono al momento irrealizzate. L'interoperabilità tra sistemi e la risoluzione dei problemi relativi alla sicurezza e alla privacy devono essere realizzate tramite la definizione di standard per protocolli, interfacce e formati. Inoltre, vi deve essere una ricerca riguardante sistemi che supportino nuovi paradigmi di interazione. Altri campi che possono avere un grande ruolo nello sviluppo della ricerca sono:

- **Multimodal machine learning** - Un ambito di ricerca interdisciplinare che vive nel contesto dell'AI e si occupa dell'inclusione e implementazione di modalità di comunicazione multiple, tra le quali quella linguistica e acustica;
- **Semantic audio** - L'estrazione di simboli con un determinato significato da uno stream audio. Un esempio di applicazione è lo Speech Recognition.

L'IoMusT impone una ripensamento delle pratiche di composizione musicale, che dovrà considerare la natura distribuita dei musicisti e spettatori all'interno dell'ecosistema e la multi-modalità del contenuto musicale che si trasmette. Secondo i ricercatori, la visione proposta ha la possibilità di avere un impatto incredibile sui modi nei quali si ascolta, si compone e si impara la musica.

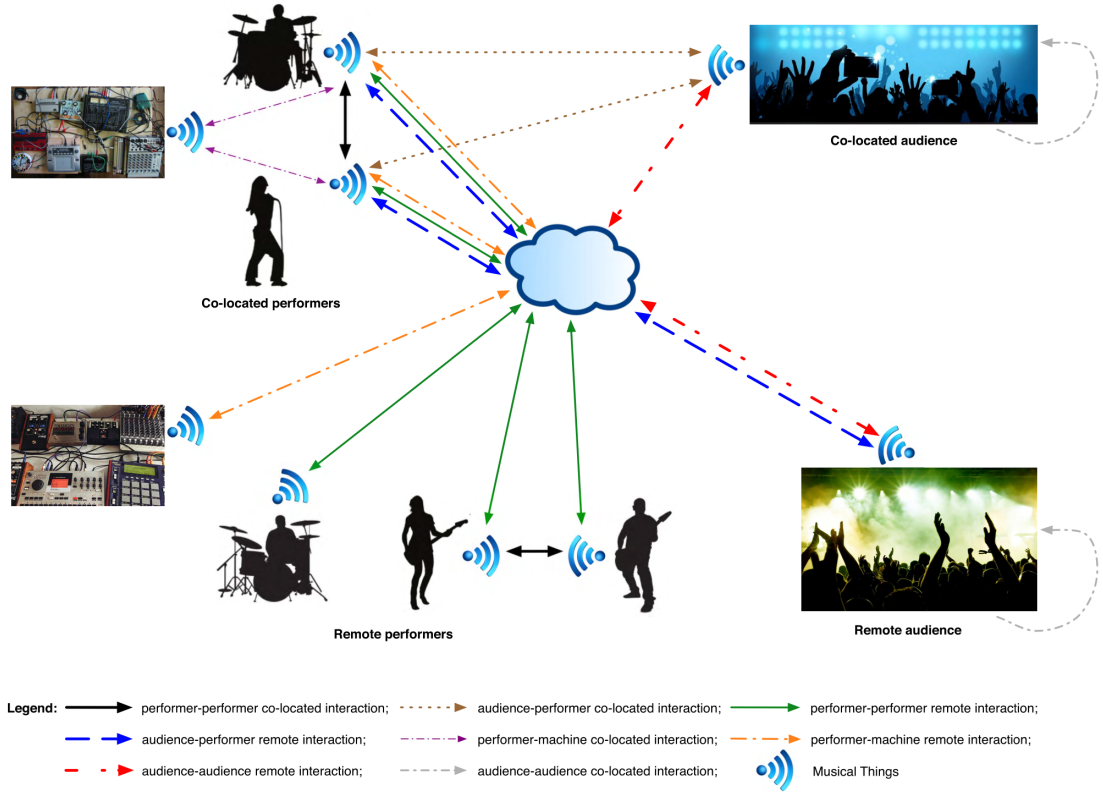


Figura 2.3: Possibilità di interazione descritte nell'articolo [15]

2.3 Protocolli di rete

In questo paragrafo vengono illustrati i principali protocolli di rete assieme ai risultati rilevati tramite delle prove dedite a determinare quale dei tre sia il più adatto al trasporto dei dati MIDI. Il protocollo ideale deve essere abbastanza veloce nella trasmissione dei dati, al fine di non rovinare l'esperienza degli utenti impegnati a suonare nella sessione, sempre per lo stesso motivo è inoltre necessario assicurare la ricezione dei pacchetti da parte di tutti i partecipanti.

Analizzeremo prima i principali protocolli di trasporto (UDP e TCP) e successivamente lo standard RTP-MIDI, definito espressamente per la trasmissione di pacchetti MIDI in real time attraverso Ethernet e reti wireless.

2.3.1 UDP

User Datagram Protocol (UDP) è un protocollo di comunicazione di livello trasporto definito nel 1980 all'interno di RFC 768[7]. L'obiettivo di UDP è quello di permettere alle applicazioni di trasmettere dati sotto forma di datagram ad altri dispositivi connessi attraverso una rete IP. UDP non assicura la ricezione del messaggio da parte del destinatario, non si occupa di gestire l'ordine di arrivo dei pacchetti e non implementa un controllo di congestione, tuttavia la trasmissione e ricezione di un dato è molto veloce proprio grazie all'assenza di questi meccanismi e all'overhead minimo del datagram. UDP è connectionless: un messaggio può essere spedito da un nodo sulla rete ad un altro senza aver prima eseguito un handshake.

È proprio la velocità di questo protocollo il fattore che potrebbe renderlo un buon candidato alla trasmissione di segnali MIDI. Dai test effettuati, che verranno illustrati di seguito, si nota che UDP sarebbe davvero ottimo non fosse per la perdita di messaggi che si vuole assolutamente evitare quando si suona. Inoltre, il fatto che l'arrivo ordinato delle note non sia assicurato è inaccettabile.

2.3.2 TCP

Transmission Control Protocol[1] (TCP) è un protocollo di comunicazione di livello trasporto definito per la prima volta nel 1974 ed è il protocollo sul quale si appoggia l'intero world wide web. Le sue caratteristiche principali sono:

- **Connection-orientedness** - Due applicazioni in rete devono aver effettuato un handshake prima di iniziare a scambiarsi dati;
- **Reliableness** - Il destinatario effettua l'acknowledgement del messaggio ricevuto al fine di informare il mittente riguardo la ricezione;
- **Ordinamento** - L'arrivo ordinato dei pacchetti al destinatario è assicurato;
- **Controllo errori**

Prendendo in considerazione questi fattori, il TCP risulterebbe perfetto per la trasmissione di segnali musicali, tuttavia risulta essere notevolmente più lento di UDP proprio a causa di questi meccanismi. Dai test effettuati si è infatti giunti alla conclusione che i tempi di arrivo siano inaccettabili.

2.3.3 RTP

Real-time Transport Protocol[10] (RTP) è un protocollo di rete di livello application che nasce con l'obiettivo principale di gestire streaming di dati audio e video nelle reti IP. Nella maggior parte delle implementazioni, si appoggia sul protocollo UDP.

RTP implementa meccanismi di rilevamento e rimedio alla perdita dei pacchetti e al loro arrivo disordinato, fenomeni comuni nella comunicazione via UDP. Risulta quindi essere un protocollo veloce, ma allo stesso tempo fornisce tutto il controllo necessario nella gestione di streaming di file multimediali, dove la mancanza dell'arrivo di un pacchetto può essere occultata tramite algoritmi di error concealment.

RTP-MIDI

RTP-MIDI[3] (conosciuto anche con il nome di AppleMIDI) è un protocollo per il trasporto dei messaggi MIDI all'interno di pacchetti RTP. I vantaggi di RTP-MIDI sono la gestione delle sessioni via rete, la rilevazione e rigenerazione automatica dei pacchetti perduti e la sincronizzazione dei dispositivi.

I driver RTP-MIDI sono preinstallati negli odierni sistemi MacOS e sono esposti tramite interfaccia grafica all'interno della quale è possibile collegarsi e gestire le sessioni sia in rete locale, che remota. Implementazioni di RTP-MIDI sono disponibili sia per sistemi Windows che Linux. Il discovery di dispositivi su rete locale è pressoché automatico, tuttavia per effettuare un collegamento via Internet è necessario aver impostato il port forwarding dal proprio router di rete. Le porte di interesse sono:

- **5004** - Per iniziare/concludere una sessione;
- **5005** - Sulla quale si ricevono i pacchetti MIDI.

In ambiente iOS, le sessioni possono essere gestite tramite la libreria CoreMIDI, che espone delle API per Swift e Objective-C. Questa libreria è stata utilizzata all'interno del progetto proposto in questa tesi.

2.4 Confronto

Per capire quale dei protocolli precedentemente descritti sia più adatto all'implementazione del progetto, sono stati eseguiti dei test su rete locale, in ambiente browser e Node.js, uno per ogni protocollo da analizzare:

- **UDP** - Client e server Node.js che comunicano utilizzando il modulo `dgram`[5] dalla libreria ufficiale di Node.js.;
- **TCP** - Client browser e server Node.js che comunicano utilizzando la libreria Socket.io[12] che a sua volta fa utilizzo delle WebSocket;
- **RTP-MIDI** - Due client browser che comunicano tra di loro tramite una connessione effettuata utilizzando la libreria WebMIDI[16], su una sessione creata con l'utilizzo di "Audio MIDI Setup" su MacOS.

In ognuno di questi test sono stati spediti 1000 pacchetti MIDI via il protocollo scelto e sono stati registrati i tempi di spedizione e arrivo per ogni pacchetto. Nel caso dell'UDP non sono stati considerati i tempi dei pacchetti non arrivati. Sia nel caso UDP che nel caso TCP il client spedisce un pacchetto e il server lo rimanda indietro, il tempo registrato equivale quindi al tempo in cui il pacchetto ritorna al client.

Di seguito sono proposti due grafici che confrontano i tempi nei tre casi.

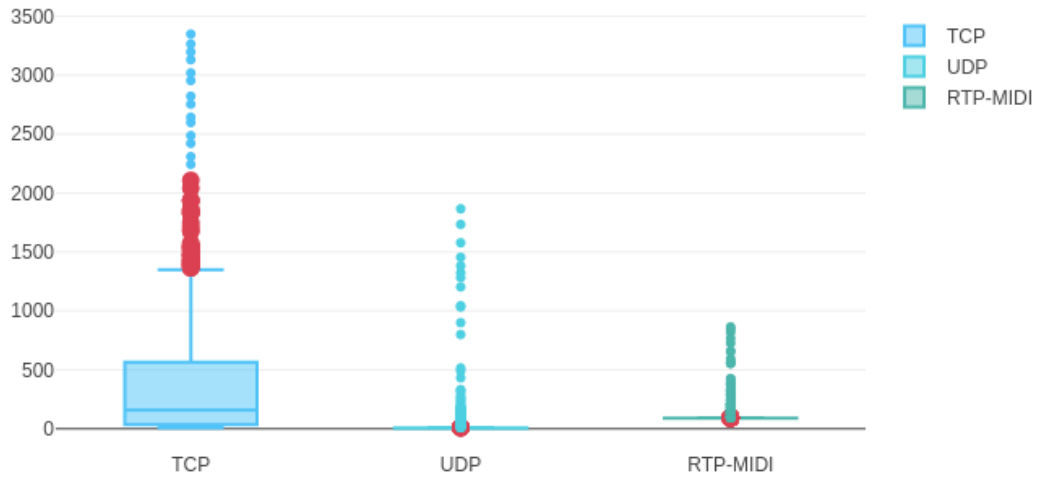


Figura 2.4: Concentrazione dei delay

Dai boxplot nella figura 2.4 si nota che le differenze tra i valori temporali sono molto più ridotte nei casi UDP e RTP-MIDI, TCP risulta invece inaccettabile per i nostri scopi, mostrando inoltre più outlier (identificati dai punti rossi) rispetto agli altri due protocolli.

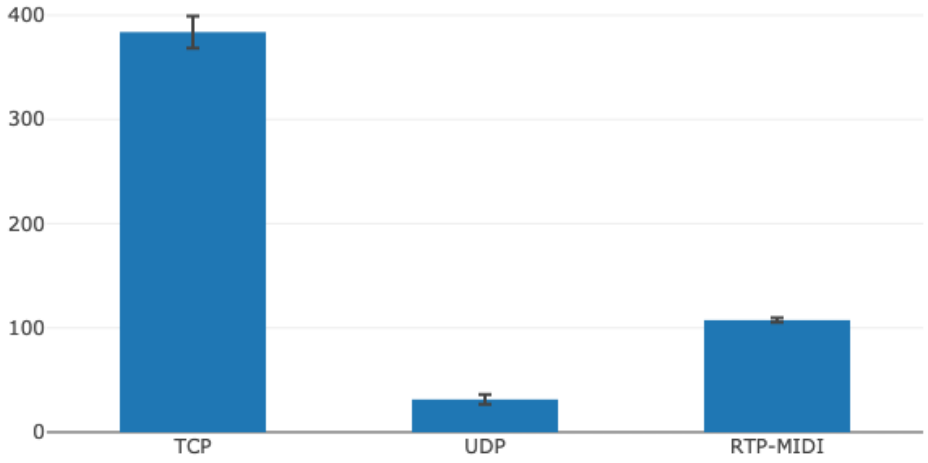


Figura 2.5: Media dei delay

Protocollo	Media [ms]	Errore [ms]	PDR [%]
TCP	383.8	15.3	100
UDP	31.4	4.7	96.7
RTP-MIDI	107.5	2.3	100

Il grafico a barre in figura 2.5 e la tabella sovrastante mettono a confronto la media dei tempi di ricezione dei pacchetti MIDI nei tre casi, compresi gli intervalli di confidenza relativi. Si nota immediatamente che il protocollo TCP è molto più lento delle altre due varianti e sarebbe più difficile suonare uno strumento musicale con un delay così alto. Il protocollo UDP è il più veloce tra i tre, tuttavia sono stati persi 33 pacchetti sui 1000 spediti, fattore che lo rende non affidabile per i nostri scopi. RTP-MIDI d'altro canto, considerando la sua affidabilità, velocità e capacità di gestire sessioni è il protocollo ideale da utilizzare nell'implementazione.

Capitolo 3

Bander

In questo capitolo si descrive l'architettura del progetto, i dispositivi, i loro ruoli e l'interazione che questi hanno tra di loro. L'obiettivo che si vuole raggiungere consiste nel dare la possibilità a qualsiasi utente in possesso di uno smartphone, uno strumento musicale MIDI e una connessione ad internet di poter suonare insieme ad altre persone a distanza. Denotiamo innanzitutto con l'espressione "Jam Session" la sessione che vedrà un utente coinvolto nello suonare e creare musica in remoto insieme ad altri. Effettuare una Jam Session con le tecnologie che oggi sono a disposizione risulta molto macchinoso e complicato. Si vuole ridurre al minimo questa complessità e dare la possibilità al musicista di occuparsi soltanto del processo creativo, con l'intento futuro di avere una vera e propria workstation dove egli possa collaborare in remoto.

Il traffico MIDI condiviso deve essere il più possibile affidabile, ma allo stesso tempo deve arrivare a destinazione in tempi brevi. Nel capitolo precedente abbiamo già illustrato i tre protocolli che sono stati considerati e i criteri secondo i quali sono stati valutati. Il protocollo migliore è risultato essere RTP-MIDI, grazie alla sua velocità di trasmissione e alla sua affidabilità. La differenza nei vari delay calcolati è inoltre poco evidente.

3.1 Architettura

Le capacità odierne dei dispositivi mobile rendono possibile l'utilizzo di una tradizionale architettura client server, mista a un'architettura peer-to-peer che vede coinvolte soltanto le applicazioni client.

Il client consiste in un'applicazione mobile che può quindi essere eseguita su uno smartphone o un tablet collegati ad Internet tramite una rete Wi-Fi, dispositivi dei quali la maggior parte delle persone è ormai in possesso. La potenza odierna dello smartphone fa scomparire la necessità di dover utilizzare dispositivi intermedi con un unico compito, per esempio quelli dedicati esclusivamente alla gestione del traffico MIDI.

Il server tiene traccia delle band, dei loro membri e delle informazioni relative ad una sessione in cui questi ultimi suonano insieme. Per iniziare una Jam Session il client segnala il suo intento al server che lo registra e notifica tutti gli altri client in ascolto, i quali potranno quindi collegarsi direttamente al nuovo arrivato.

Le applicazioni client sono le entità che effettuano a tutti gli effetti il collegamento alla sessione e lo scambio di dati musicali utilizzando il protocollo RTP-MIDI, ma hanno necessità del server per avere conoscenza degli host remoti ai quali collegarsi. Devono inoltre essere autenticati ed identificati affinché possano effettivamente collegarsi insieme ad altri utenti in una sessione.

Nei prossimi paragrafi verranno illustrati più nel dettaglio i ruoli del server e dei client.

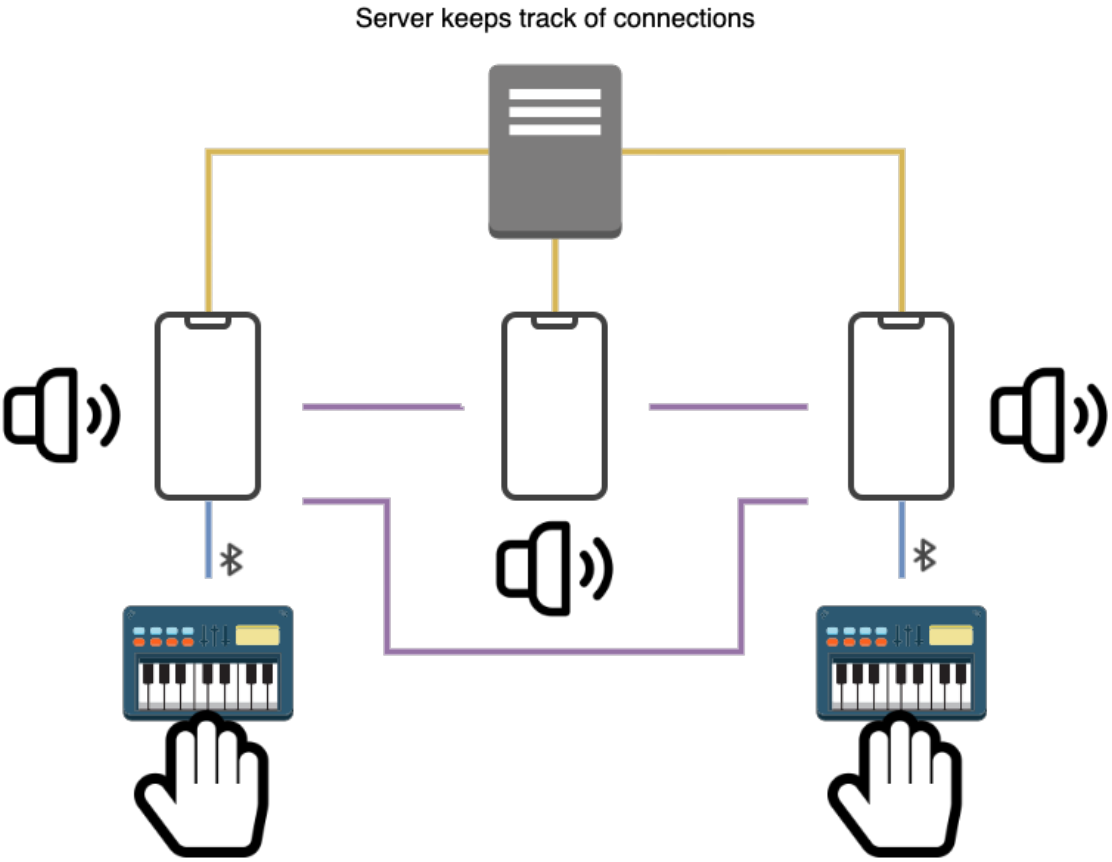


Figura 3.1: Comportamento dei componenti

La figura 3.1 descrive il collegamento tra i principali componenti e il loro comportamento. Le linee denotano i collegamenti secondo il seguente criterio:

- **Giallo** - Collegamento persistente al server;
- **Blu** - Collegamento Bluetooth all'eventuale strumento musicale MIDI;
- **Viola** - Collegamento che i client effettuano tra di loro sulla sessione RTP-MIDI per scambiarsi gli eventi MIDI.

Appena un utente provvisto di uno strumento collegato suona una nota, l'evento corrispondente ad essa viene tradotto in segnale audio e riprodotto sul dispositivo locale. In contemporanea, l'evento MIDI viene condiviso direttamente con gli altri dispositivi collegati alla sessione RTP-MIDI, i quali riprodurranno a loro volta la nota sotto forma di segnale audio.

3.2 Server

Il server consiste in un'applicazione che tiene traccia dei seguenti fattori:

- Autenticazione degli utenti;
- Verifica dell'appartenenza ad un gruppo di utenti che hanno il permesso di suonare insieme, che chiameremo "band";
- Identificazione di tutti i client attualmente connessi alla Jam Session e notifica per ogni nuovo client connesso o disconnesso.

Si noti che qualsiasi tipo di comunicazione che il client effettua col server avviene utilizzando protocolli applicativi che si appoggiano su TCP, in quanto il server stesso non è impegnato nella gestione del traffico di messaggi MIDI. I client devono poter comunicare direttamente tra di loro nella condivisione del traffico MIDI, al fine di ridurre i tempi di ricezione dei messaggi.

3.2.1 Autenticazione

L'utente deve essere autenticato prima di collegarsi ad una Jam Session, in quanto vi è necessità di effettuare un'analisi dei suoi permessi e delle sue capacità di collegarsi ad altri utenti registrati al servizio. Il server dà la possibilità di effettuare una registrazione e di accedere tramite un identificativo ed una password. Accedendo, l'utente diventa il possessore di un token firmato che lo identifica all'interno del servizio ed è in grado di confermare, nonché denotare facilmente la sua identità.

3.2.2 Band e membri

Si tiene traccia dei gruppi di persone che possono suonare insieme. Le band sono composte da membri i quali possono essere considerati a tutti gli effetti gli utenti registrati. Un utente può appartenere a più band. Due utenti possono suonare insieme se e soltanto se sono membri di una stessa band.

Viene gestita la possibilità di aggiungere un utente come membro di una band tramite le seguenti modalità:

- **Invito** diretto all'utente interessato, il quale potrà accettare o rifiutare. Nel caso dell'accettazione, l'utente diventa automaticamente membro della band e può suonare insieme agli altri;
- **Offerta** pubblica alla quale più utenti possono candidarsi, ma soltanto uno dei candidati verrà scelto dalla band per diventare il nuovo membro e poter così suonare in remoto.

Viene inoltre data la possibilità al creatore di una band di eliminare i membri o modificare i loro permessi.

Per memorizzare gli utenti registrati, lo stato di una band e dei suoi membri, il server salva i dati all'interno di un database relazionale persistente. Le modalità in cui vengono definite e gestite le varie entità vengono descritte nel capitolo successivo.

3.2.3 Jam Session

Come già precedentemente affermato, con "Jam Session" identifichiamo la sessione che vede impegnati più utenti nell'atto di suonare insieme. Per mostrare il suo intento nell'effettuare il collegamento ai client degli altri membri nella sessione, il client appena collegato notifica il server fornendogli automaticamente i seguenti dati:

- Token firmato che identifichi l'utente;
- Identificativo della band alla quale l'utente desidera effettuare il collegamento;
- Indirizzo IP pubblico dell'utente e porta del servizio RTP-MIDI (5004);
- Nome della sessione RTP-MIDI locale dell'utente collegato.

Il server verifica innanzitutto la validità dell'utente e accerta la sua esistenza all'interno del database, dopodiché si assicura che egli sia a tutti gli effetti membro della band alla quale richiede di collegarsi. Nel caso negativo, l'utente viene notificato con un messaggio di errore, mentre nel caso positivo i dati che egli ha fornito vengono salvati in un database key-value in memory contenente i dati di tutti gli utenti impegnati nella sessione. Il server ricava poi i dati aggiornati e li condivide con tutti i client partecipanti alla Jam Session tramite un evento di notifica. I client si occuperanno poi di effettuare il collegamento tra di loro nelle modalità descritte di seguito.

3.3 Client

Il software che viene eseguito sui dispositivi mobile è il lato più importante del progetto ed astrae il funzionamento di vari dispositivi, prendendosi i seguenti compiti:

- Collegamento ad uno strumento musicale con interfaccia MIDI e gestione dei comandi da esso generati;
- Mantenimento di una sessione con il server per avere informazioni sugli altri client;
- Gestione di una sessione RTP-MIDI peer-to-peer con gli altri client che permette la generazione e instradamento di segnali MIDI tra di essi;
- Generazione e riproduzione di segnali audio a partire dai segnali MIDI ricevuti sia dallo strumento collegato che dagli utenti collegati in remoto.

L'utilizzatore interagisce con l'applicazione attraverso una semplice interfaccia grafica che si occupa di informarlo dello stato della sessione alla quale è collegato e degli strumenti MIDI che ha a disposizione. Il fatto che non vengano utilizzati dispositivi ad-hoc per effettuare questo genere di operazioni è un punto di forza per quanto riguarda la semplicità di utilizzo. Due utenti che vogliono suonare in remoto non devono avere conoscenze in ambito IoT o acquistare appositi controllori fisici per la gestione del traffico MIDI.

Il progetto permette anche l'implementazione futura di strumenti musicali grafici, potendo così favorire l'utilizzo del servizio anche senza strumento musicale MIDI.

3.3.1 Strumento musicale

L'applicazione tiene traccia degli strumenti MIDI collegati al dispositivo. L'utente ha in questo modo la possibilità di consultare la lista dei dispositivi dall'interfaccia grafica e selezionare quello che preferisce. A quel punto l'applicazione si occupa di assegnare un listener agli eventi generati dallo strumento specificato. Questo listener andrà ad eseguire le operazioni di generazione di segnali audio e la condivisione del messaggio dell'evento ricevuto con gli altri partecipanti alla sessione.

3.3.2 Collegamento al server

Un client che vuole unirsi ad una Jam Session effettua la richiesta di connessione al server fornendogli in automatico i dati necessari. In caso di connessione avvenuta con successo, il client ottiene la lista con i dati di tutti i dispositivi (identificativi, nomi delle sessioni RTP-MIDI, porte, indirizzi IP pubblici) ed inizia ad effettuare le procedure di collegamento alla sessione RTP-MIDI descritte nel paragrafo seguente. Appena un client si disconnette dal server volontariamente o involontariamente, i suoi dati vengono rimossi dalla lista dei partecipanti alla sessione contenuta nel database key-value e gli altri client vengono notificati dal server.

La connessione client-server è persistente. Un server può quindi notificare un client senza che quest'ultimo effettui esplicitamente richieste di aggiornamento.

3.3.3 Sessione RTP-MIDI

Quando un utente mostra il suo intento di collegarsi ad una Jam Session, l'applicazione genera automaticamente una sessione RTP-MIDI con tutti i dati necessari. I dati vengono poi condivisi con il server tramite le modalità precedentemente descritte. Le sessioni RTP-MIDI sono identificate dai seguenti dati: nome locale della sessione, nome bonjour della sessione, indirizzo IP dell'host, porta sulla quale iniziare o concludere la connessione.

Una sessione RTP-MIDI contiene collegamenti ad una molteplicità di host in rete. La sessione appena inizializzata non ha nessun host tra i suoi collegamenti. Appena il client viene notificato da parte del server con i dati di tutti gli utenti collegati, viene automaticamente creata una connessione a tutti quei client in base al loro Nome bonjour, indirizzo IP pubblico e porta (che è praticamente sempre 5004). La ricezione dei dati MIDI provenienti dall'esterno viene in questo modo fatta operando sugli endpoint della sessione, i quali sono di due tipi:

- **Source** - Mostra la sessione come generatore di input di eventi MIDI. A questo endpoint possono essere assegnati dei listener nell'esatto modo in cui vengono assegnati ad uno strumento MIDI;
- **Destination** - Mostra la sessione come ricevitore di eventi in output. Questo endpoint può essere utilizzato per spedire gli eventi in multicast a tutti gli host partecipanti alla sessione.

Ogni evento MIDI proveniente dalla sessione viene quindi interpretato e riprodotto localmente sotto forma di segnali audio nelle modalità descritte nel paragrafo

successivo. Gli eventi generati dallo strumento MIDI locale, oltre ad essere riprodotti localmente, vengono spediti sull'endpoint destination della sessione RTP-MIDI, la quale si occuperà di inoltrarli automaticamente a tutti gli host collegati.

Quando un utente si disconnette dalla Jam Session, i dati relativi alla connessione verso il suo host vengono eliminati dalle sessioni RTP-MIDI di tutti gli altri client.

3.3.4 Generazione di segnali audio

L'applicazione effettua la generazione di segnali audio provenienti da più sorgenti. Per poterlo fare astrae i seguenti dispositivi:

- **Engine** - Il motore che fornisce tutti i tool necessari per astrarre dispositivi di gestione dell'audio. L'engine viene inizializzato appena un utente si unisce alla Jam Session. L'output dell'engine è automaticamente riprodotto dal dispositivo mobile;

- **Sintetizzatore polifonico** - Dispositivo che converte i segnali MIDI in segnali audio. Al momento vengono utilizzati due sintetizzatori, uno per la riproduzione degli eventi provenienti dallo strumento musicale locale e l'altro per gli eventi provenienti dalla sessione RTP-MIDI. La ricezione di un evento di tipo "noteon" da parte di uno dei due dispositivi riproduce sull'opportuno sintetizzatore l'audio corrispondente alla nota specificata. La ricezione di un evento di tipo "noteoff" ferma invece la riproduzione della nota sul sintetizzatore opportuno. Avendo uno strumento polifonico, è possibile riprodurre più di una nota nello stesso istante, ciò permette all'utente di suonare accordi e preserva la dinamica del brano che viene suonato.

- **Mixer** - Il mixer combina l'audio generato da più fonti. Viene impostato come output dell'engine. Senza l'utilizzo del mixer sarebbe impossibile riprodurre istantaneamente i segnali audio di entrambi i sintetizzatori che vengono utilizzati.

In uno sviluppo futuro del progetto sarebbe interessante e possibile utilizzare astrazioni di strumenti musicali diversi per generare output audio. Un utente potrebbe così riprodurre le note sulle tonalità di un pianoforte, mentre un altro potrebbe utilizzare il suono di una chitarra elettrica o una batteria. La capacità di astrazione dei dispositivi mobile moderni sembra essere molto ricca da questo punto di vista.

3.4 Interazione

In questa sezione vengono proposti dei diagrammi che illustrano le modalità di connessione e comunicazione tra i vari dispositivi appartenenti alla sessione. Per mantenere la semplicità, vengono omessi i router, che devono aver impostato il port-forwarding sulle porte 5004 e 5005 verso l'indirizzo locale del dispositivo interessato.

3.4.1 Sessione

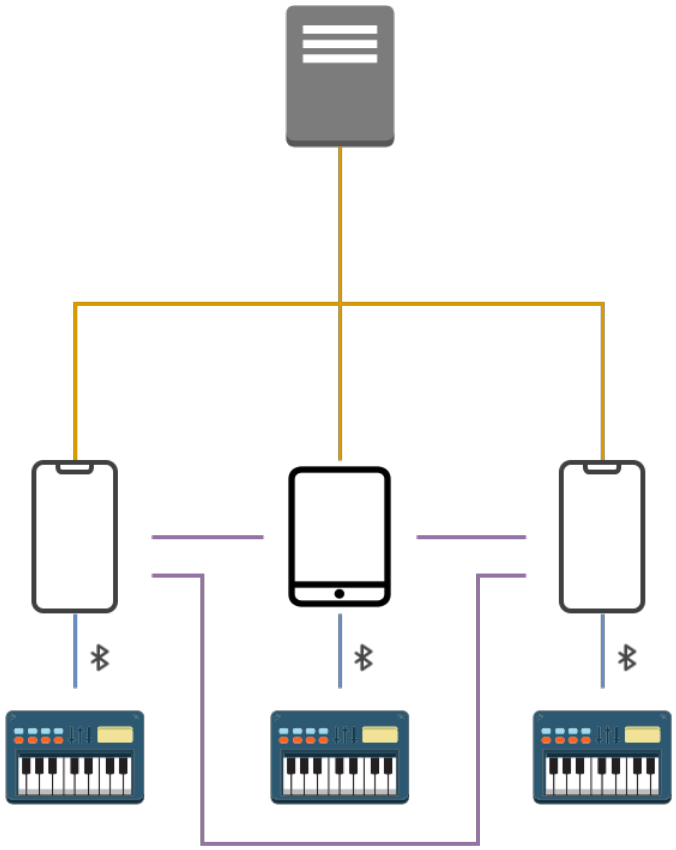


Figura 3.2: Sessione nello stato normale

Durante una sessione, ogni utente ha tre tipi di connessione attive sul proprio dispositivo mobile:

- **Bluetooth** - Collegamento al proprio dispositivo MIDI;
- **WebSocket** - Collegamento al server che tiene traccia delle informazioni riguardanti tutti gli utenti collegati alla sessione;
- **RTP-MIDI** - Collegamento diretto con gli altri dispositivi nella sessione, basandosi sui dati provenienti dal server.

Il server si occupa soltanto di gestire la connessione e disconnessione. Le note vengono trasmesse direttamente tra i client sulla sessione RTP-MIDI.

3.4.2 Connessione e disconnessione

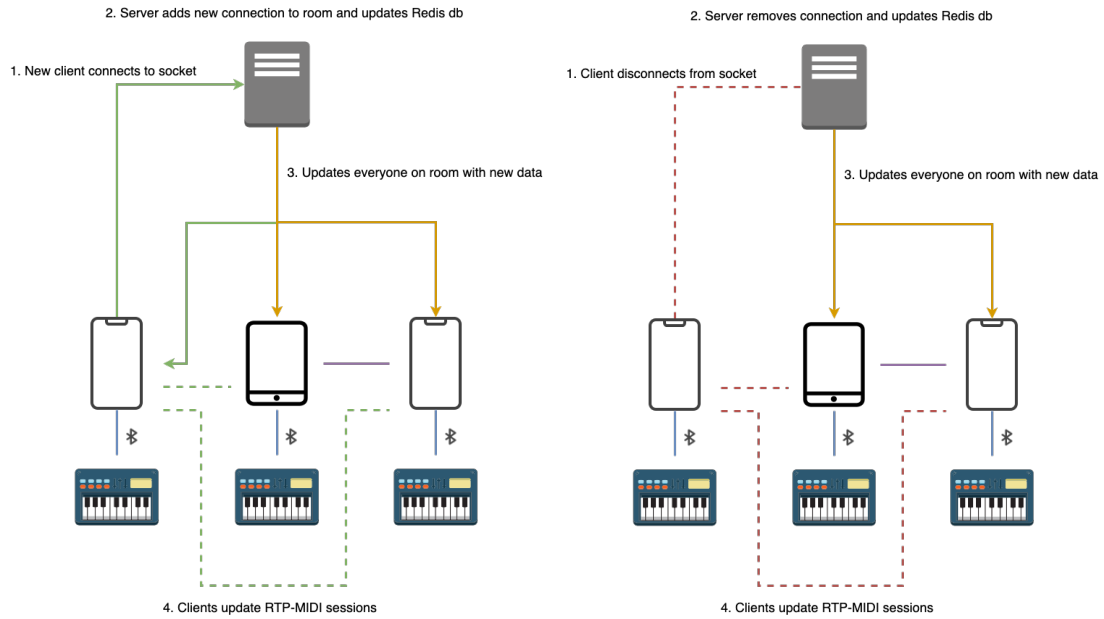


Figura 3.3: Connessione e disconnessione di un utente dalla Jam Session

Per effettuare la connessione alla Jam Session, il client effettua la richiesta di connessione alla WebSocket del server specificando tutti i dati illustrati nei paragrafi precedenti: token identificativo dell'utente, identificativo della band alla cui sessione ci si vuole collegare, indirizzo IP pubblico, porta e nome della sessione RTP-MIDI. Il server, dopo aver verificato la validità dell'utente e la sua appartenenza alla band, salva i suoi dati e lo aggiunge al canale di comunicazione, aggiornando immediatamente tutti gli altri client partecipanti alla stessa sessione tramite un messaggio sulla WebSocket. I client, dopo essere stati notificati dal server, aggiungono alla propria sessione RTP-MIDI il collegamento all'host del nuovo utente collegato.

Appena il server rileva la disconnessione di un client dalla WebSocket, lo rimuove dalla lista dei client connessi, i quali vengono aggiornati tramite un evento di notifica sulla WebSocket. I client rimuovono dalla loro sessione RTP-MIDI il collegamento all'host mancante.

Capitolo 4

Implementazione

In questo capitolo si spiegano le scelte implementative e le motivazioni che stanno dietro ad esse. Vengono illustrati gli ambienti di sviluppo e le tecnologie e librerie principali, insieme al codice che gestisce le interazioni cruciali.

L'applicazione ha due lati, quello server e quello client, i quali corrispondono ai rispettivi componenti architetturali descritti nel capitolo precedente. L'utente utilizza il client iOS per collegarsi al server Node.js, potendo così gestire la propria band e suonare insieme ad altri membri in maniera peer-to-peer, attraverso le sessioni RTP-MIDI. Nei paragrafi successivi si illustra come viene eseguita l'interazione tra le varie componenti e come queste sono state organizzate ed implementate.

4.1 Tecnologie e ambienti di sviluppo

Di seguito si descrivono le tecnologie e gli ambienti utilizzati nello sviluppo del progetto. Si discutono inoltre i fattori che ne hanno motivato la scelta.

4.1.1 Redis

Redis[9] è uno store in-memory, utilizzato come database, cache e message broker ed è distribuito sotto licenza BSD. Il modo in cui vengono salvati i record all'interno di un database Redis è abbastanza semplice: si utilizzano chiavi e valori. Il valore di una determinata chiave può essere creato, letto, aggiornato, cancellato tramite dei comandi atomici. Redis implementa inoltre delle strutture dati come le liste ed espone dei comandi atomici per la loro gestione. Operazioni come il push di un valore o il remove di varie sue occorrenze sono atomiche e immediate.

Il motivo principale per cui è stato scelto Redis è per la sua velocità. Quando un utente si aggiunge alla sessione oppure la lascia, il database viene aggiornato immediatamente e gli altri membri della sessione vengono subito notificati. Un database tradizionale sarebbe stato troppo lento e meno adatto in questa situazione, anche se l'applicazione ne fa comunque utilizzo in altri contesti.

4.1.2 Node.js

Node.js[5] è un ambiente di runtime per JavaScript, basato sul motore V8 di Google. Il contesto in cui viene utilizzato maggiormente al giorno d'oggi è lo sviluppo di applicazioni server-side e REST API.

I motivi per cui è stato scelto questo ambiente sono la velocità di esecuzione, la familiarità personale e la sua popolarità, la quale fa si che si trovi una notevole quantità di documentazione online. Inoltre vi è la presenza di un gran numero di framework e librerie open source installabili tramite Node Package Manager (NPM) e Yarn. Tra i più importanti nell'implementazione di questo progetto sono Strapi e Socket.io.

Strapi

Strapi[13] è un headless CMS distribuito sotto licenza MIT. I vantaggi che Strapi offre sono la facilità di definizione dei modelli e delle loro relazioni, il suo design database agnostic, la generazione automatica di chiamate CRUD che rispettano l'architettura REST e la possibilità di personalizzarle completamente. Essendo un CMS, presenta una potente interfaccia per amministratore nella quale si possono gestire facilmente permessi, definire modelli, creare istanze di modelli e utenti e tante altre funzionalità che tuttavia sono attualmente fuori dal nostro scope.

Nel progetto attuale lo si utilizza per verificare che un utente sia effettivamente membro della band alla quale si collega e per ricavare i dati personali da mostrare graficamente.

Socket.io

Socket.io[12] è una libreria che fa utilizzo di HTTP e WebSocket al fine di permettere una comunicazione bidirezionale, real-time, basata sugli eventi. Vi è un server che gestisce il collegamento dei client, la loro eventuale attribuzione ad un determinato canale (room) e rimane in ascolto di eventi provenienti da essi. Il server è a sua volta in grado di notificare tutti i client, oppure uno singolo oppure soltanto quelli collegati ad una determinata room.

Siccome si appoggia sul protocollo TCP, lo si è inizialmente utilizzato per effettuare le prove di misurazione del delay, illustrate nel paragrafo precedente. Nonostante i problemi di TCP nella trasmissione dei dati real-time, risulta comunque molto utile per notificare gli utenti di una sessione riguardo l'entrata o l'uscita di una nuova persona.

4.1.3 iOS

iOS e iPadOS sono i famosi sistemi operativi per dispositivi mobile Apple. L'ambiente è stato scelto in quanto implementa già nativamente librerie per la gestione delle connessioni RTP-MIDI e MIDI in generale. L'ambiente mobile si presta inoltre molto bene allo studio delle trasmissioni real-time tramite reti remote, con la possibilità di fare uso delle tecnologie 4G e 5G.

Swift

Swift[14] è un linguaggio compilato, multi-paradigma, nel quale accanto ad approcci object oriented possono convivere approcci funzionali e reactive. Presenta inoltre un sistema di tipi molto avanzato e un'implementazione interessante del pattern matching.

È il linguaggio più moderno sviluppato dalla Apple ed insieme ad Objective-C sono i principali se non in certi casi gli unici linguaggi attraverso i quali sviluppare applicazioni eseguibili sui suoi sistemi operativi. Tuttavia al giorno d'oggi esistono compilatori ed ambienti di esecuzione per Swift anche su sistemi Linux e Windows.

SwiftUI

SwiftUI è un nuovo approccio dichiarativo tramite il quale definire e gestire interfacce grafiche nelle applicazioni. Contrappone al tradizionale design pattern Model View Control (MVC) con le storyboard il design pattern Model View View-Model (MVVM): una vista può essere considerata come un observer che rimane in ascolto di aggiornamenti sui dati (observable o published) contenuti nel View-Model.

CoreMIDI e MIKMIDI

I sistemi operativi Apple per Mac e dispositivi mobile implementano nativamente la gestione del MIDI, tramite API raccolte nel framework CoreMIDI, il quale è suddiviso in sette servizi:

- **MIDI Services** - Comunicazione con l'hardware tramite l'utilizzo di Universal MIDI Packets;
- **MIDI System Setup** - Configurazione del sistema MIDI globale;
- **MIDI Messages** - Creazione e configurazione dei messaggi;
- **MIDI Thru Connection** - Creazione di connessioni play-through tramite sorgenti e destinazioni;
- **MIDI Networking** - Creazione e gestione di dispositivi connessi in una rete locale;
- **MIDI Drivers** - Creazione di plug-in per driver;
- **MIDI Capability Inquiry** - Supporto per il discovery bidirezionale e configurazione di dispositivi.

Il servizio utilizzato in modo diretto all'interno del progetto è MIDI Networking per la creazione e gestione di sessioni RTP-MIDI.

Gli altri servizi necessari sono stati invece utilizzati per mezzo della libreria MIKMIDI, che estende CoreMIDI ed è scritta utilizzando Objective-C. MIKMIDI semplifica la gestione dei dispositivi MIDI collegati, permettendo di applicare facilmente listener agli eventi recepiti da sorgenti MIDI.

AudioKit

AudioKit è una piattaforma per la sintesi, il processo e l'analisi di audio per iOS, MacOS e tvOS ed è distribuita sotto licenza MIT. I concetti chiave di questa libreria sono:

- **Nodi** - Componenti per il signal processing, collegabili tra di loro. Tutti i nodi hanno un output, ma solo quelli che devono elaborare segnali esterni hanno anche un input. Un esempio di nodo è il sintetizzatore;
- **Operazioni** - Sono dei componenti di elaborazione simili ai nodi, tuttavia la loro istanza può esistere soltanto all'interno di un singolo nodo;
- **Tap** - Utilizzano i nodi come proprie sorgenti di dati, tuttavia non ridirezionano immediatamente il segnale verso altri nodi. Questo permette di muovere con un certo grado di libertà i tap da nodo a nodo.

Attualmente esistono due versioni di AudioKit supportate:

- La versione 4 è quella attualmente più utilizzata;
- La versione 5 è attualmente in beta e propone un'implementazione più pulita e corretta, tuttavia vi è ancora (per poco) mancanza di qualche funzionalità.

Bander utilizza la versione 5, in quanto gli sviluppatori consigliano di utilizzare quest'ultima versione per le nuove implementazioni e non vi è ancora necessità delle funzionalità mancanti. All'interno del progetto, i componenti AudioKit vengono utilizzati principalmente per la sintetizzazione del suono, con l'obiettivo futuro di poter permettere agli utenti di specificare il proprio strumento musicale nella sessione.

4.1.4 JSON Web Token

JSON Web Token[2] (JWT) è uno standard che definisce un modo per trasmettere informazioni tra due entità in modo sicuro, sotto forma di oggetto JSON. Il JWT è una stringa firmata digitalmente tramite algoritmi a chiave simmetrica (e.g. HMAC) oppure asimmetrica (e.g. RSA) da parte del fornitore del servizio o di un terzo di cui il fornitore si fida (e.g. Auth0). La stringa è suddivisa in tre parti, separate da un punto. Le tre parti sono degli oggetti JSON codificati in Base64 e sono le seguenti:

- **Header** - Contiene il campo `typ` che descrive il tipo di token (e.g. JWT) e un campo `alg` che indica l'algoritmo utilizzato per la firma del JWT (e.g. RSA, HMAC, SHA256);
- **Payload** - Contiene le informazioni riguardanti l'entità che deve provare la sua validità. Nel caso di un utente registrato i campi potrebbero essere uno o più identificatori come l'email o lo username, in aggiunta vi potrebbe essere un campo che specifica il ruolo di tale entità. Le password e qualsiasi tipo di informazione confidenziale non vanno assolutamente aggiunte al payload, in quanto quest'ultimo non è di base criptato;
- **Signature** - Una firma che verifica la validità del JWT tramite l'algoritmo specificato.

4.2 Server

L'implementazione del server è stata effettuata in ambiente Node.js, attraverso il framework Strapi. Viene fatto utilizzo di PostgreSQL per la gestione dei dati persistenti e di Redis per i dati riguardanti gli utenti che suonano nella sessione, che da qui in poi chiameremo "Jam Session".

Di seguito si spiega l'implementazione e si illustrano le modalità di utilizzo e API dei seguenti servizi offerti dall'applicazione server:

- **Autenticazione** assieme alla gestione degli **utenti**;
- Gestione della **band**, dei **membri, inviti** ed **offerte**;
- Gestione della **Jam Session**, i controlli che effettua e il modo in cui tiene traccia degli utenti collegati.

4.2.1 Utenti e Autenticazione

Per permettere l'autenticazione degli utenti al servizio, viene fatto utilizzo del classico approccio nome utente e password. Il vantaggio che ci viene dato da Strapi è l'utilizzo di un plugin pre-implementato ai fini della gestione dell'autenticazione e dei permessi, chiamato "strapi-plugin-users-permissions". All'interno di Bander, il plugin gestisce le seguenti operazioni:

- Generazione del modello **User** con i seguenti attributi e relazioni principali:
  - `id` - Univoco per ogni utente, il suo tipo dipende dal database sottostante: nel caso di database SQL è un classico intero con incremento automatico, nel caso di MongoDB è un ObjectID di 12 byte;
  - `username` - Valore testuale univoco per ogni utente, può essere utilizzato per effettuare l'autenticazione;
  - `email` - Valore testuale univoco per ogni utente, utilizzato per l'autenticazione e le comunicazione esterne con l'utente;
  - `password` - Valore testuale risultante dal hashing della password dell'utente;
  - `person` - Relazione 1:1 con entità di tipo `Person` creata appositamente per compensare alle mancanze dell'entità `User`.
- Generazione di **endpoint** REST utili alla gestione degli utenti, tra i quali:
  - `POST /auth/local/register` - Per la registrazione dell'utente. Il plugin gestisce tutto il processo, inclusa la validazione e la sanitization dei dati. Al termine della registrazione, viene restituito un JSON contenente i dati non riservati dell'utente e un JWT firmato;
  - `POST /auth/local` - Per l'autenticazione dell'utente tramite email (o username) e password. Avendo ricevuto le credenziali corrette, il servizio firma un JWT e lo spedisce al client il quale lo utilizzerà per chiamare gli endpoint riservati;

- Possibilità di creare nuovi utenti, gestire i permessi e abilitare o disabilitare gli endpoint tramite l'interfaccia di amministrazione.

Manualmente sono state gestite le seguenti operazioni:

- Creazione del modello **Person** con i seguenti attributi e relazioni principali:
  - `userName` - Copia esatta dell'attributo `username` dell'entità `User` corrispondente;
  - `firstName` - Valore testuale corrispondente al nome dell'utente;
  - `lastName` - Valore testuale corrispondente al cognome dell'utente;
  - `isArtist` - Flag di tipo boolean per verificare che un utente sia un artista o no. Un artista è capace di effettuare determinate richieste all'API che un utente non artista non è in grado di fare;
  - `user` - Relazione 1:1 con l'entità `User` corrispondente;
  - `memberships` - Relazione con l'entità `Membership`, la quale mette a sua volta in relazione `Person` con `Band` al fine di identificare l'appartenenza di un utente ad una determinata band;
  - `offerResponses` - Relazione con l'entità `OfferResponse`, la quale mette in relazione `Person` con `Offer`. Permette di ricavare tutte le offerte di partecipazione ad una band alle quali l'utente si è candidato.
- Creazione dei seguenti endpoint e dei relativi controller:
  - `GET /people` - Restituisce una lista in formato JSON contenente tutte le persone registrate;
  - `POST /people` - Utilizzato subito dopo la registrazione di un utente. Estrapola il JWT contenuto nell'Authorization header per creare l'entità
    `Person` in base ai dati appartenenti all'entità `User` specificata nel payload del JWT;
  - `GET /people/:id` - Restituisce in formato JSON i dati dell'utente corrispondente al parametro `id`;
  - `PUT /people/me` - Identifica l'entità `Person` in base al payload del JWT e aggiorna i suoi attributi con quelli presenti nel body della richiesta;
  - `POST /people/me/isArtist` - Imposta al valore `true` il flag `isArtist` dell'utente autenticato e crea automaticamente una band corrispondente al singolo artista, alla quale non possono essere aggiunti altri membri.

In sintesi, l'utente si registra al servizio tramite la chiamata `POST /auth/local/register` e può effettuare l'accesso con identificativo e password tramite `POST /auth/local`. Dopo aver effettuato l'accesso, l'utente provvisto di JWT firmato correttamente e non scaduto può effettuare tutte le chiamate precedentemente elencate, aggiungendo alla richiesta HTTP il header Authorization con il valore `Bearer {jwt}`, dove `jwt` è il valore del proprio token.

4.2.2 Bands API

Questo modulo del server si occupa delle operazioni CRUD riguardanti le band e la loro interazione con i membri, gli eventi e i post. In questo paragrafo ci occuperemo di descrivere solo la parte relativa alle operazioni principali e ai membri, in quanto la gestione degli eventi e dei post non sono momentaneamente interessanti per quanto riguarda l'implementazione della Jam Session.

Funzionamento generale

Quando viene chiamato per la prima volta l'endpoint `POST /people/me/isArtist`, l'utente autenticato diventa artista, ottenendo così l'accesso agli endpoint per la gestione delle band, membri, offerte e inviti. In automatico, viene generata una band avente lo username dell'utente, alla quale non possono essere aggiunti ulteriori membri. Questa band è da considerarsi profilo dell'artista.

Un artista può creare una band attraverso l'endpoint `POST /bands`. I membri possono essere aggiunti nei seguenti modi:

- **Invito** - Un amministratore della band decide di spedire un invito ad un utente specificato, il quale potrà poi confermare o rifiutare l'invito. Nel primo caso, l'utente diventa così nuovo membro della band.
- **Offerta** - Un amministratore della band decide di pubblicare un'offerta alla quale potranno rispondere più artisti, tra i quali verrà infine scelto colui che diventerà il nuovo membro.

L'owner della band è l'unico che potrà modificare o cancellare i membri della band già arruolati. Qualsiasi membro della band potrà pubblicare post a nome della band stessa e unirsi ad una Jam Session.

Entità

Segue una descrizione delle entità e dei loro attributi e relazioni principali:

- `Band`
  - `name` - Valore testuale contenente il nome della band o dell'artista;
  - `isArtist` - Flag di tipo boolean che indica se l'istanza di `Band` attuale è relativa ad un unico artista, oppure è a tutti gli effetti una band che può avere più membri. Quando un utente effettua la chiamata `POST /people/me/isArtist` viene automaticamente creata una band con questo flag impostato a `true`, la band così creata sarà da considerarsi a tutti gli effetti come profilo dell'artista. Tutte le band che un utente potrà successivamente creare avranno il flag impostato a `false`;
  - `memberships` - Relazione con l'entità `Membership`, descritta di seguito;
  - `invitations` - Relazione con l'entità `Invitation`, descritta di seguito;
  - `offers` - Relazione con l'entità `Offer`, descritta di seguito;
  - `genres` - Relazione con l'entità `Genre`. Corrisponde ad una lista di generi che la band o l'artista suona;

- `Membership` - Mette in relazione `Band` con `Person`, denotando così l'appartenenza di un utente ad una band;
  - `band` - Relazione con l'entità `Band`;
  - `person` - Relazione con l'entità `Person`;
  - `role` - Valore testuale che indica il ruolo dell'utente all'interno della band;
  - `isAdmin` - Flag di tipo boolean che accerta che l'utente sia un amministratore della band. Gli amministratori possono aggiornare i dati della band, spedire inviti ed offerte e far partecipare la band ad un determinato evento;
  - `isOwner` - Flag di tipo boolean che accerta che l'utente sia il creatore della band. Oltre ad avere tutti i permessi di un amministratore, è l'unico che ha il totale controllo sulle membership della band.
- `Invitation` - Mette in relazione `Band` con `Person`. Corrisponde ad un invito di partecipazione alla band specificata. Se l'utente destinatario accetta, viene automaticamente creata la sua membership;
  - `band` - Relazione con l'entità `Band`;
  - `person` - Relazione con l'entità `Person`;
  - `message` - Un messaggio che il destinatario possa leggere (e.g. "Hey, vuoi unirti alla mia band?").
- `Offer` - Offerta di reclutamento alla band che non ha un utente specificato, ma alla quale possono candidarsi più utenti con l'obiettivo di diventare il nuovo membro della band;
  - `message` - Un messaggio che descriva la band ed invogli gli utenti ad unirsi;
  - `instrument` - Relazione con l'entità `Instrument`. Indica lo posizione che si avrà all'interno della band;
  - `band` - Relazione con l'entità `Band`. Denota la band che ha pubblicato l'offerta;
  - `offerResponses` - Relazione con l'entità `OfferResponse`, descritta di seguito.
- `OfferResponse` - Corrisponde alla risposta di un utente ad una specifica offerta;
  - `message` - Messaggio attraverso il quale il candidato si presenta;
  - `offer` - Relazione con l'entità `Offer`. Indica l'offerta alla quale è rivolta la risposta;
  - `person` - Relazione con l'entità `Person`. Denota l'utente che ha creato la risposta.

4.2.3 Jam Session

La Jam Session è il fulcro di questa ricerca e consiste nel dare ai membri della band la possibilità di suonare insieme in remoto. Il collegamento degli utenti alla sessione viene gestito tramite la libreria Socket.io, il traffico MIDI viene invece gestito direttamente dai client, attraverso sessioni RTP-MIDI.

Funzionamento generale

Procediamo ad illustrare l'implementazione della socket lato server. La gestione della Jam Session è composta da due funzioni principali che vengono eseguite durante il boot del server Strapi, vale a dire nel file `bootstrap.js`. Le due funzioni sono:

- `startRedis()`
- `startSocket()`

Esecuzione delle funzioni

```
startRedis();
process.nextTick(() => {
  startSocket();
});
```

La funzione `process.nextTick(callback[, ...args])` ritarda l'esecuzione della funzione `startSocket()` in quanto quest'ultima necessita di funzionalità che non sono disponibili nella fase di bootstrap del server Strapi.

Collegamento a Redis

La funzione `startRedis()` fa utilizzo dell'interfaccia Redis per Node.js ed esegue le seguenti operazioni:

1. Chiama la funzione `createClient(url: string)` per effettuare il collegamento al database Redis;
2. Imposta una callback che in caso di errore stampi il messaggio contenuto;
3. Dichiara delle versioni asincrone delle funzioni che verranno utilizzate per interfacciarsi con il database Redis. Queste funzioni restituiscono valori di tipo `Promise` che saranno "resolved" in caso di successo o "rejected" in caso di errore.
4. Aggiunge l'istanza della connessione al database come attributo dell'istanza `strapi`, così da poter essere utilizzata sempre all'interno del framework.

Gestione della socket

La funzione `startSocket()` agisce nel seguente modo:

1. Crea l'istanza del server;

2. Si mette in ascolto sul namespace `/jam`. I namespace sono dei canali di comunicazione che permettono di separare la logica dell'applicazione su una singola connessione condivisa. Il default namespace è `/` ed è l'unico che non va specificato tramite la funzione `of(namespace: string)`;

3. Dichiara una funzione di callback che sarà eseguita ad ogni nuova connessione. La funzione di callback dovrà verificare che l'utente che richiede di essere collegato al namespace `/jam` sia effettivamente membro della band alla quale vuole unirsi. La funzione esegue le seguenti operazioni:

   (a) Estrapola il jwt e l'id della band che il client deve aver fornito nella richiesta di connessione;

   (b) Verifica la validità del jwt fornito e decodifica l'id dell'utente dal payload;

   (c) Richiede al database il valore di `User` e di `Person` corrispondente all'id fornito, nel caso non ci sia, restituisce un messaggio d'errore 404;

   (d) Verifica che l'utente sia membro della band;

   (e) Se l'utente risulta essere membro, viene invocata la funzione di callback `next()` per eseguire la procedura di connessione, descritta di seguito;

4. Dichiara una funzione di callback che si occupa della gestione della richiesta di connessione nel caso di un utente valido. La funzione di callback esegue a sua volta le seguenti procedure:

   (a) Estrae il l'id della band, il nome della sessione RTP-MIDI e la sua porta dai parametri di richiesta forniti dal client;

   (b) Crea un oggetto contenente l'id della persona, il nome della sessione RTP-MIDI e la porta precedentemente ottenuti e l'indirizzo IP del client che ha effettuato la richiesta;

   (c) Applica la funzione `JSON.stringify()` all'oggetto appena creato e utilizza la funzione `lpush()` per aggiungere la stringa dell'oggetto alla lista identificata dalla chiave avente il valore dell'id della band;

   (d) Richiede la lista di tutti i partecipanti alla chiave denotata da `bandId` e la salva in un array;

   (e) Aggiunge il socket del client alla room identificata da `bandId`;

   (f) Emette un evento di tipo `update`, contenente la lista di tutti gli utenti collegati, verso tutte le socket appartenenti alla room identificata da `bandId`, del namespace `/jam`;

   (g) Dichiara una funzione di callback che gestisca il caso di errore;

   (h) Dichiara una funzione di callback che gestisca la disconnessione del client dalla socket nel seguente modo:

      i. Rimuove dalla lista del database il valore del client attuale;

      ii. Richiede nuovamente la lista con tutti i client collegati;

      iii. Emette l'evento `update`.

4.3 Client

Il client è stato scritto per gli ambienti iOS e iPadOS, utilizzando il linguaggio Swift. Vi è stato fatto un pesante utilizzo di SwiftUI, di conseguenza il modo in cui è organizzato il codice cerca di rispettare il pattern MVVM. Inoltre, per effettuare la condivisione dei dati tra tutte le viste dell'applicazione, viene utilizzato un Environment Object.

Il client si interfaccia costantemente con il server, tramite gli endpoint illustrati nella sezione precedente. Utilizza Socket.io e le funzionalità del framework CoreMIDI e della libreria MIKMIDI nella gestione della Jam Session.

Nelle sezioni successive verranno descritte le classi e strutture principali. Si partirà con una veloce discussione sul modo in cui l'applicazione utilizza le `struct` per astrarre le entità del server. Si passerà poi a descrivere lo `Store` condiviso e il modo in cui vengono gestite le richieste al server. Verranno infine illustrate le classi `BandStore`, `SocketHelper`, `BanderMIDI`, l'estensione `NoteCommand` e le principali viste.

4.3.1 Modelli

I modelli sono definiti come `struct` che possono essere istanziate partendo dal dato di tipo `JSON` ed implementano il protocollo `Entity`.

```
protocol Entity: Hashable, Identifiable, Codable {
  var id: Int { get }
  var createdAt: DateInRegion { get }
  var updatedAt: DateInRegion { get }

  init(_ data: JSON)
}
```

Il tipo `DateInRegion` è definito nella libreria SwiftDate che permette una gestione più avanzata delle date rispetto a quella di default. Il tipo `JSON` è definito nella libreria SwiftyJSON ed è un dizionario che rappresenta un oggetto in formato JSON.

La differenza principale tra `struct` e `class` è che il passaggio come parametro di funzioni delle prime avviene per valore, mentre le classi vengono passate per reference.

Per ogni modello, sono stati definiti due tipi di strutture diversi, in base al formato del JSON che il server fornisce come risposta ad una particolare richiesta:

- Oggetto contenente soltanto gli attributi dell'entità che rappresenta e privo dei dati relativi alle sue relazioni. Le strutture che rappresentano questi tipi di dato sono state definite col nome dell'entità (e.g. `Band`);
- Oggetto contenente sia gli attributi che la rappresentazione delle entità relative alle sue relazioni. Le strutture che rappresentano questi tipi di dato sono state definite col nome dell'entità, seguito da "Full" (e.g. `BandFull`).

Le strutture "semplici" possono essere istanziate a partire dalle strutture "full" tramite l'apposito costruttore.

4.3.2 Store

La classe `Store` è il riferimento principale di tutta l'applicazione. Essa contiene i dati relativi all'utente autenticato (se presente), al dispositivo utilizzato, alla localizzazione dell'utente (se attiva). Definisce inoltre le funzioni per interfacciarsi con le API del server.

`Store` implementa il protocollo `ObservableObject` in quanto definisce istanze di dati (decorate con `@Published`) che devono aggiornare le viste che renderizzano i loro valori. Affinché l'istanza singola di `Store` possa essere condivisa tra le varie viste, si effettua una dependency injection nella vista principale, all'interno della funzione `scene()` della classe `SceneDelegate`. Sarà così possibile fare riferimento all'istanza di `Store` all'interno di una vista:

```
@EnvironmentObject private var store: Store
```

4.3.3 Band Store

La classe `BandStore` contiene i dati principali ai quali fanno riferimento le viste relative ad una specifica band. Come `Store`, implementa il protocollo `ObservableObject` e dichiara degli attributi published. Contrariamente a `Store`, la sua istanza non è unica all'interno dell'applicazione, ma viene creata all'interno di una vista e passata come proprietà alle sue subview. All'interno di una vista, un attributo che fa riferimento ad un `ObservableObject` deve essere decorato con `@ObservedObject` se si vuole rimanere in ascolto dei suoi aggiornamenti.

```
@ObservedObject private var bs = BandStore()
```

4.3.4 Socket Helper

La classe `SocketHelper` si occupa di gestire la comunicazione con la WebSocket, attraverso l'utilizzo della libreria Socket.io per Swift. La classe contiene:

- Una funzione per effettuare la connessione alla WebSocket, avendo come parametri di connessione il jwt dell'utente autenticato, l'id della band, il nome e la porta della sessione RTP-MIDI;
- Una funzione per effettuare la disconnessione dalla WebSocket;
- Una funzione per verificare lo stato della connessione alla socket;
- Un enumeratore di tipi di eventi che possono essere spediti o ricevuti tramite la socket. Espone dei metodi `emit()` e `listen()` che servono rispettivamente a emettere un evento e assegnare un listener che rimanga in ascolto di eventi in arrivo dal server. Questi metodi effettuano pattern matching sul tipo stesso dell'enumeratore per distinguere i vari eventi, i quali possono avere tipi di ritorno diversi oppure richiedere parametri specifici nell'emissione di un messaggio;
- Un'istanza shared di se stessa. Questo approccio fa si che non sia necessario istanziare la classe esternamente per poterla utilizzare, basta semplicemente fare utilizzo dell'attributo `shared`.

4.3.5 Note Command

Il file `NoteCommand` definisce un'estensione della classe `MIKMIDINoteCommand`, definita nella libreria MIKMIDI. In Swift, la keyword `extension` viene utilizzata per arricchire una classe con metodi o attributi che potrebbero risultare utili alla nostra implementazione. Ovviamente queste modifiche non possono impattare il funzionamento interno della classe e della libreria alla quale appartiene.

L'estensione aggiunge i seguenti attributi computed alla classe:

- `noteOnOctave` - Restituisce il valore numerico di una nota senza prendere in considerazione l'ottava sulla quale si trova;
- `noteName` - Restituisce una coppia contenente il nome della nota secondo la nomenclatura europea e quella anglosassone;
- `noteNumber` - Valore numerico della nota in un tipo compatibile con AudioKit;
- `noteVelocity` - Valore numerico della velocità con cui è stata riprodotta la nota in un tipo compatibile con AudioKit.

4.3.6 Bander MIDI

La classe `BanderMIDI` è il gestore principale della Jam Session. Unisce tra di loro le funzionalità della sessione RTP-MIDI e della WebSocket e si interfaccia con gli strumenti musicali connessi al dispositivo. `BanderMIDI` implementa `ObservableObject` in quanto contiene attributi il cui aggiornamento deve scaturire l'aggiornamento della vista.

Per potere utilizzare una istanza della classe bisogna prima di tutto invocare il metodo `start()`, che verrà descritto di seguito. La classe espone i seguenti metodi:

- `start(appStore: Store, band: Band)` - Deve sempre essere invocato prima di utilizzare gli altri metodi. Inizializza tutto il sistema eseguendo le seguenti procedure:
  1. Assegna il riferimento dello store ricevuto come parametro all'attributo della classe e assegna a `sessionId` il valore dell'id della band;
  2. Assegna il mixer come output dell'engine, sul quale invoca successivamente il metodo `engine.start()`;
  3. Invoca il metodo `startMIDISession()`, descritto di seguito;
  4. Invoca il metodo `connectSocket()` sull'istanza condivisa della classe `SocketHelper` per effettuare la connessione remota alla WebSocket;
  5. Dopo aver eseguito la connessione invoca il metodo locale `listenForSocketUpdate()`.
- `startMIDISession()` - Inizializza la sessione RTP-MIDI nel seguente modo:
  1. Assegna all'attributo locale `midiSession` la sessione default del dispositivo mobile;

2. Abilita la sessione e permette la creazione di una connessione con qualsiasi dispositivo in rete;
3. Invoca il metodo `connectSessionDevice()`.

- `listenForSocketUpdate()` - Mette il client in ascolto di eventi di tipo `update` provenienti dal server. Imposta come listener una funzione di callback che connette la sessione RTP-MIDI all'indirizzo e porta di tutti i client ricevuti dall'evento;
- `refreshDevices()` - Sincronizza l'attributo `devices` con i dispositivi disponibili elencati da `MIKMIDIDeviceManager`;
- `connectLocalDevice(_ device: MIKMIDIDevice)` - Esegue il collegamento ad un dispositivo MIDI con almeno una sorgente disponibile. Istanzia un `Synth` come `localInstrument` e assegna un listener agli eventi della sorgente. Il listener è una funzione di callback che esegue il metodo opportuno (riproduci nota o smetti di riprodurre nota) su `localInstrument` in base al tipo di comando MIDI ricevuto dalla sorgente. Infine spedisce la nota all'endpoint di destinazione di `midiSession`, condividendo così il segnale con tutti gli altri client in ascolto sulla sessione RTP-MIDI;
- `connectSessionDevice()` - Funziona come `connectLocalDevice()`, ma effettua il collegamento con la sorgente di `midiSession` al fine di riprodurre i segnali MIDI ricevuti dalla sessione RTP-MIDI;
- `disconnectLocalDevice()` - Esegue le procedure di disconnessione del dispositivo MIDI locale;
- `disconnectSessiondevice()` - Esegue le procedure di disconnessione del dispositivo che astrae il collegamento alla sessione RTP-MIDI;
- `end()` - Esegue tutte le procedure di disconnessione e ferma il motore audio di AudioKit.

4.3.7 Viste

Il collegamento alla Jam Session viene eseguito quando un utente autenticato accede alla `JamView` tramite il link di navigazione presente nella `BandView` relativa ad una band di cui egli fa parte.

Band View

La vista della band è definita tramite la struct `BandView` di tipo `some View`. Un utente esterno può visualizzare i dati e i post della band. Un membro può pubblicare nuovi post a nome della band e navigare verso la `JamView`. Un amministratore può modificare i dati del profilo della band. Un owner può gestire anche le membership.

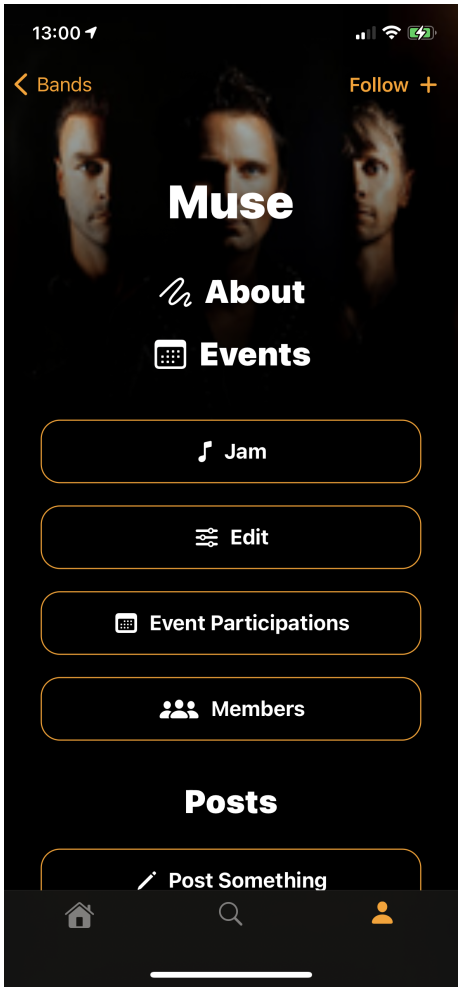


Figura 4.1: Vista della band dal punto di vista dell'owner

La classe `NavigationLink` renderizza un pulsante che imposta come destinazione una nuova istanza di `JamView` alla quale viene passata come parametro l'istanza attuale di `BandStore`. Si noti che la vista che gestisce la Jam Session viene già istanziata qui, ma non è desiderabile inizializzare adesso tutte le interfacce che gestiscono il MIDI. È per questo motivo che nella classe `BanderMIDI` le operazioni di inizializzazione non vengono fatte nel costruttore, ma nell'apposito metodo `start()`.

Jam View

All'apparizione di questa vista, viene invocato il metodo `start()` dell'istanza locale di `BanderMIDI`, l'utente viene quindi collegato automaticamente alla Jam Session. Gli vengono illustrati gli username degli altri partecipanti e può scegliere lo strumento che vuole suonare tra quelli collegati al dispositivo. Ogni nota che l'utente suona sul proprio dispositivo MIDI viene riprodotta dal dispositivo mobile e viene condivisa con gli altri utenti attraverso la sessione RTP-MIDI. Anche i comandi ricevuti dagli altri utenti sulla sessione vengono riprodotti dal dispositivo mobile.

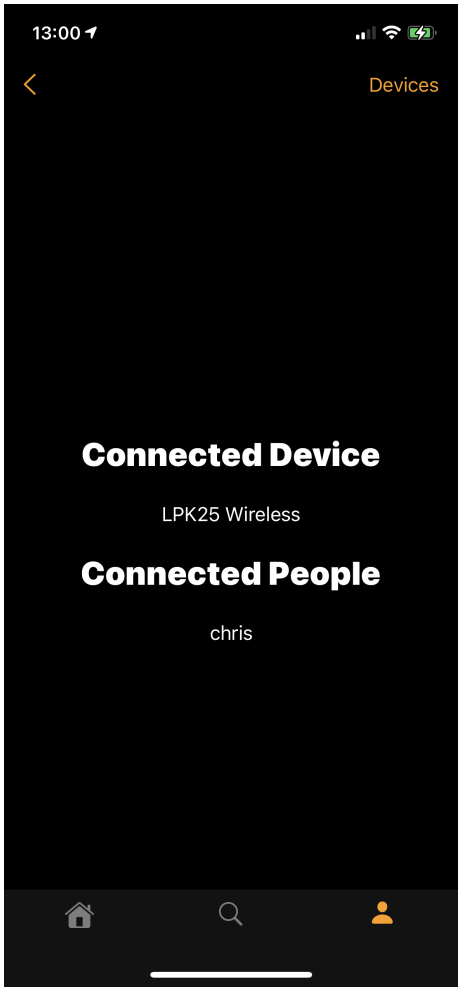


Figura 4.2: Vista della Jam Session con due utenti collegati

Capitolo 5

Validazione

In questo capitolo viene illustrato lo studio di fattibilità eseguito sull'implementazione descritta nel capitolo precedente. L'obiettivo è poter affermare con certezza che l'applicazione riesca a tutti gli effetti trasmettere i segnali MIDI tramite dispositivi collegati in remoto, senza perdita di pacchetti ed entro tempi accettabili per un'esibizione musicale. Il successo di tale verifica confermerebbe la possibilità che più utenti siano in grado di suonare insieme a distanza, con un grado di semplicità maggiore rispetto a quello precedente e che ha la possibilità di essere ampliato ulteriormente tramite il proseguimento della ricerca in questo ambito.

5.1 Esecuzione della prova

La prova effettuata vede l'utilizzo di due dispositivi iOS, ciascuno collegato alla propria rete Wi-Fi locale e con l'applicazione Bander in esecuzione. Sui gateway delle reti locali è stato impostato il port-forwarding sulle porte 5004 e 5005 verso l'indirizzo IP interno del dispositivo sul quale viene eseguita l'applicazione. La porta 5004 riceve i segnali di inizio e fine connessione, mentre la 5005 riceve i segnali MIDI veri e propri. Si prenda inoltre in considerazione il fatto che una delle reti locali utilizza il 4G per effettuare il collegamento a Internet, mentre l'altra utilizza un collegamento basato su fibra ottica.

Uno dei due dispositivi mobile è stato collegato ad un controller MIDI tramite Bluetooth, attraverso l'interfaccia che viene offerta di default dal sistema operativo. Ciascuno dei client ha effettuato l'accesso con il nome utente e password di due utenti appartenenti alla stessa band. Entrambi hanno navigato verso la vista della band che hanno in comune e hanno dato inizio alla Jam Session, effettuando così tutti i collegamenti necessari alla loro comunicazione. A quel punto, l'utente con lo strumento collegato lo ha scelto dalla lista dei dispositivi in input e lo ha iniziato a suonare. Ogni nota suonata è stata riprodotta localmente dall'applicazione e spedita in remoto al client dell'altro membro della band, il quale ha a sua volta riprodotto la nota.

In totale sono state spedite 1000 note, senza perdita di dati. Per ogni nota sono stati registrati i tempi di spedizione e arrivo e sono stati successivamente calcolati i ritardi. È stato poi eseguito uno studio sui tempi simile a quello già utilizzato nel confronto tra i vari protocolli di rete [2.4].

5.2 Interazioni

Di seguito vengono illustrate graficamente le interazioni principali che sono avvenute tra i dispositivi. La prima immagine descrive il collegamento di un client alla Jam Session e l'aggiornamento del client già collegato, con il successivo collegamento diretto dei due tramite RTP-MIDI. La seconda figura illustra il processo che viene scaturito tramite la ricezione di un evento MIDI proveniente dallo strumento musicale collegato a uno dei due client. Il messaggio viene elaborato e riprodotto localmente e contemporaneamente spedito all'altro client sulla sessione RTP-MIDI.

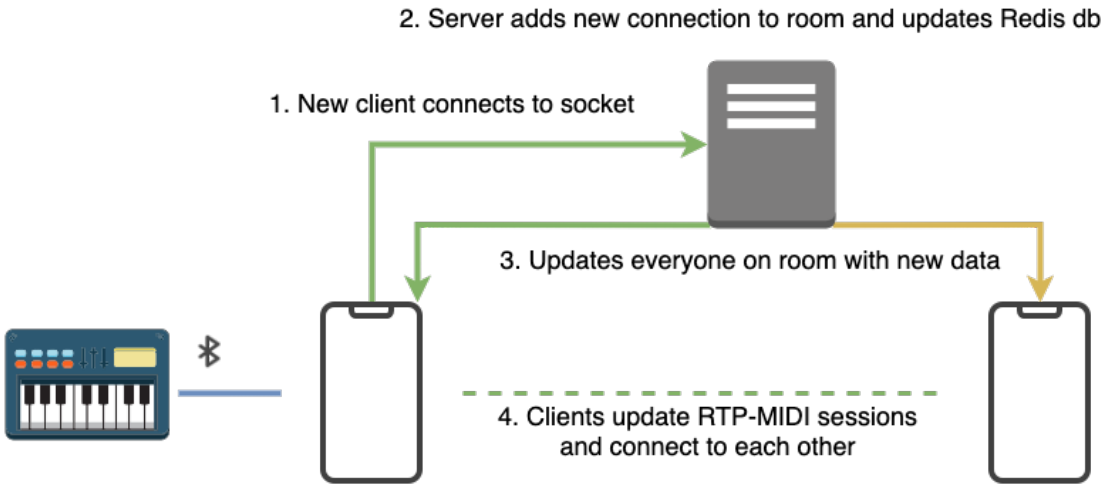


Figura 5.1: Connessione di un client alla sessione

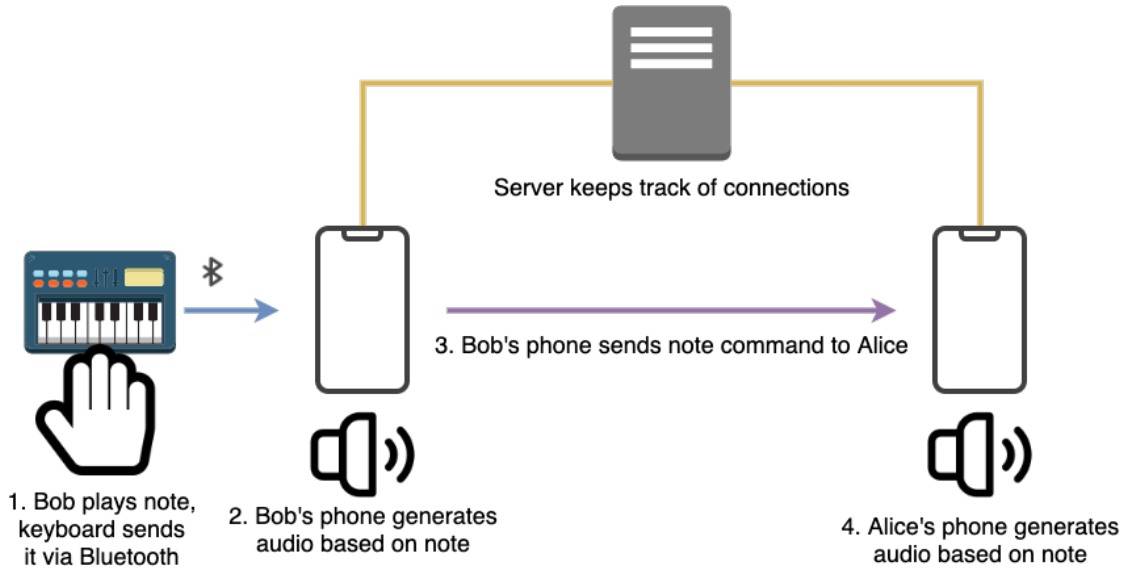


Figura 5.2: Propagazione dell'evento MIDI generato dallo strumento

5.3 Risultati

I grafici 5.3 e 5.4 illustrano i risultati (espressi in millisecondi) ottenuti mettendo a confronto i delay nella ricezione dei 1000 messaggi spediti. La figura 5.3 mostra come la concentrazione dei delay avvenga nell'intorno di un valore di mediana pari a 41.9ms. I punti rossi indicano gli outlier, i quali risultano leggermente più numerosi rispetto a quelli riscontrati nell'analisi del protocollo RTP-MIDI sulla rete locale. La figura 5.4 illustra il valore medio di arrivo di un messaggio, che corrisponde a (52.3 ± 0.9)ms.

Si nota subito un peggioramento rispetto ai dati ricavati nella fase di confronto dei protocolli su rete locale. L'aumento del ritardo di arrivo della nota e la minore concentrazione dei dati è molto probabilmente dovuto alla latenza del 4G, utilizzato dalla rete del mittente. Tutto sommato, con l'avvento del 5G o l'utilizzo di reti che si appoggiano sulla fibra ottica questo peggioramento dovrebbe essere alleviato. Nello stato attuale risulta comunque possibile suonare insieme, a distanza, con dei tempi accettabili.

La mancanza di dispositivi ad-hoc per la gestione del traffico MIDI è un punto di forza dell'implementazione attuale. Lo smartphone fa tutto il lavoro necessario alla spedizione, ricezione e riproduzione del messaggio e un eventuale utilizzatore del servizio può effettuare una Jam Session senza dover acquistare ulteriori dispositivi.

La scomodità maggiore è dovuta alla configurazione del port-forwarding, che è omissibile nelle connessioni RTP-MIDI che avvengono su rete locale, ma non in quelle via Internet. Nel seguente capitolo verranno discussi metodi che possono essere considerati in una eventuale implementazione futura per evitare tale sconforto da parte dell'utente finale.

Sicuramente il sistema non è perfetto in quanto limitato dagli standard di comunicazione attuali, ma rappresenta comunque un passo avanti nell'ambito della Networked Music Performance. Come spiegato precedentemente[2.2.4], il successo di tutti gli ambiti di ricerca collegati all'Intenet Of Musical Things dipende fortemente dalle standardizzazioni che potranno avere luogo solamente attraverso l'impegno collettivo degli enti di ricerca scientifica e industriali.

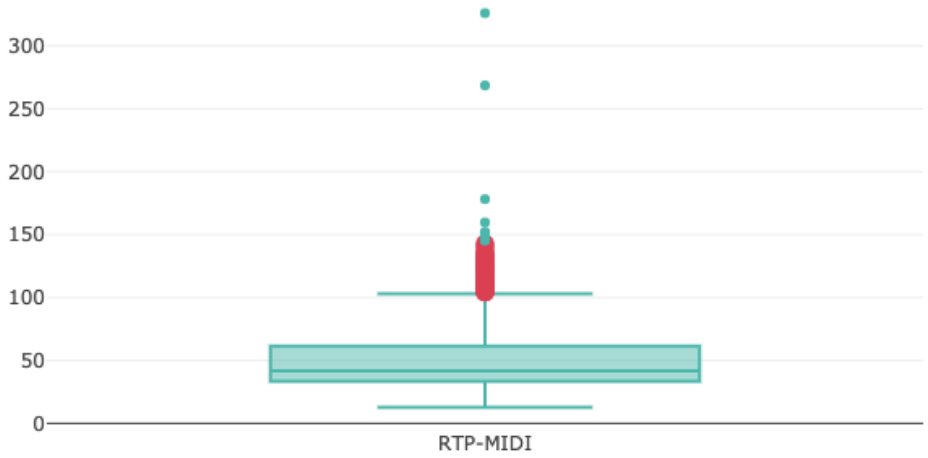


Figura 5.3: Concentrazione dei delay dei messaggi spediti

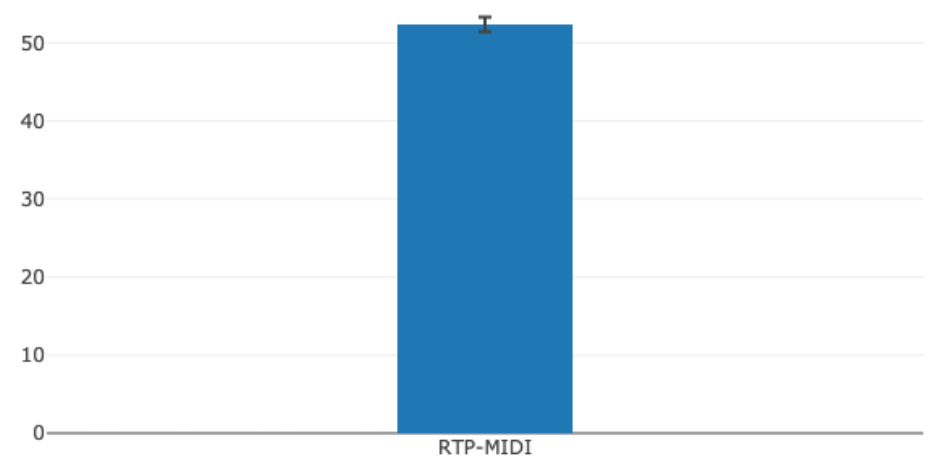


Figura 5.4: Media dei delay dei messaggi spediti

Capitolo 6

Conclusioni

In questa tesi sono stati illustrati metodi per la condivisione di traffico MIDI via internet. L'interazione vede coinvolti un server che tiene traccia dello stato di ciascuna sessione e una moltitudine di client che comunicano direttamente tra di loro tramite delle sessioni RTP-MIDI. Le prestazioni di tale sistema dipendono dalle caratteristiche dell'infrastruttura di comunicazione. Con l'avvento del 5G e di altri mezzi di comunicazione a bassa latenza, diventa sempre più possibile poter assicurare un ambiente veloce ed affidabile.

A livello comunicativo, abbiamo visto che il port-forwarding è l'unico fattore che necessita di conoscenze più o meno tecniche in ambito di reti durante l'utilizzo dell'applicazione. Può essere sicuramente un processo sconveniente per un eventuale utente finale e la sua eliminazione o automazione renderebbe ancora più usufruibile questo tipo di servizio. Una soluzione a questo inconveniente può essere implementata attraverso meccanismi di Hole Punching, i quali permettono la connessione diretta tramite dispositivi che si trovano all'interno di diverse reti locali con NAT attivo. Uno dei modi per effettuare tale collegamento vede l'utilizzo di un server esterno che tiene traccia di tutte le informazioni riguardanti gli indirizzi IP e porte degli host ai quali si vuole accedere. I client andrebbero poi ad utilizzare quelle informazioni per effettuare collegamenti diretti all'host interessato. L'implementazione di un tale meccanismo diventa praticamente un'estensione dell'applicazione proposta. Bisognerebbe analizzare come tale modifica possa essere gestita dalla sessione RTP-MIDI. È possibile che lo standard non ammetta tale comportamento e che quindi sia opportuno effettuare aggiornamenti su questo fronte.

Molto interessante da un punto di vista più ludico dovrebbe essere l'introduzione di una interfaccia grafica che emuli il comportamento di strumenti musicali fisici su schermo multitouch.

Ulteriori miglioramenti possono essere effettuati nella distinzione dei segnali provenienti dagli utenti esterni attraverso la sessione RTP-MIDI ed assegnare a ciascuno di essi un proprio sintetizzatore o addirittura uno strumento polifonico che riproduca altri tipi di suono, per esempio un pianoforte o una chitarra. Si possono inoltre sviluppare moduli per il controllo del segnale audio generato, creando una vera e propria workstation. A tal proposito, andrebbero sviluppati dei meccanismi che gestiscano al meglio la sincronizzazione temporale tra i vari utenti.

Bibliografia

[1] W. Eddy A. Zimmermann e L. Eggert. *Moving Outdated TCP Extensions and TCP-Related Documents to Historic or Informational Status*. RFC 7805. RFC Editor, apr. 2016. URL: https://tools.ietf.org/html/rfc7805.

[2] M. Jones. *JSON Web Token (JWT)*. RFC 7519. RFC Editor, mag. 2015. URL: https://tools.ietf.org/html/rfc7519.

[3] J. Lazzaro. *RTP Payload Format for MIDI*. RFC 6295. RFC Editor, giu. 2011. URL: https://tools.ietf.org/html/rfc6295.

[4] *MIDI 2.0 Specification Collection*. https://www.midi.org/specifications/midi-2-0-specifications/midi2-core/midi-2-0-specification-collection-2.

[5] *Node.js v15.3.0 Documentation*. https://nodejs.org/api.

[6] C. Perera, C. H. Liu e S. Jayawardena. «The Emerging Internet of Things Marketplace From an Industrial Perspective: A Survey». In: *IEEE Transactions on Emerging Topics in Computing* 3.4 (2015), pp. 585–598. DOI: 10.1109/TETC.2015.2390034.

[7] J. Postel. *User Datagram Protocol*. RFC 768. RFC Editor, ago. 1980. URL: https://tools.ietf.org/html/rfc768.

[8] Ammar Rayes e Samer Salam. «The Things in IoT: Sensors and Actuators». In: *Internet of Things From Hype to Reality: The Road to Digitization*. Cham: Springer International Publishing, 2017, pp. 57–77. ISBN: 978-3-319-44860-2. DOI: 10.1007/978-3-319-44860-2_3. URL: https://doi.org/10.1007/978-3-319-44860-2_3.

[9] *Redis Documentation*. https://redis.io/documentation.

[10] H. Schulzrinne. *RTP: A Transport Protocol for Real-Time Applications*. RFC 3550. RFC Editor, lug. 2003. URL: https://tools.ietf.org/html/rfc3550.

[11] Luca Sciullo. «Midi networking: Metodologie di interconnessione di strumenti musicali tramite reti fisiche e virtuali». URL: http://amslaurea.unibo.it/7448/.

[12] *Socket.io Documentation*. https://socket.io/docs/v3.

[13] *Strapi Documentation*. https://strapi.io/documentation.

[14] *Swift Documentation*. https://swift.org/documentation.

[15] Luca Turchet et al. «Internet of musical things: Vision and challenges». In: *IEEE Access* 6 (2018), pp. 61994–62017.

[16] *Web MIDI*. https://github.com/djipco/webmidi.