Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE
Corso di Laurea in Informatica

# MIDI networking: metodologie di interconnessione di strumenti musicali tramite reti fisiche e virtuali

Relatore:
Chiar.mo Prof.
Renzo Davoli

Presentata da:
Luca Sciullo

Sessione II
Anno Accademico 2013-2014

*There is no dark side in the moon, really.*
*Matter of fact it's all dark.*
PINK FLOYD

# Indice


**1 Introduzione** **1**
- 1.1 Presentazione . . . 1
- 1.2 La scelta di affrontare questo tema . . . 2

**2 Scenario** **5**
- 2.1 Mixer digitale . . . 5
- 2.2 Protocolli musicali . . . 6
  - 2.2.1 MIDI . . . 6
  - 2.2.2 File MIDI . . . 7
- 2.3 Protocolli di comunicazione tramite rete . . . 8
  - 2.3.1 UDP . . . 8
  - 2.3.2 VDE . . . 8
- 2.4 Toolkit e tecnologie . . . 8
  - 2.4.1 Gtk+ . . . 8
  - 2.4.2 Glade . . . 9
- 2.5 Dispositivi di elaborazione . . . 9
  - 2.5.1 Raspberry Pi . . . 9
  - 2.5.2 Udoo . . . 9
  - 2.5.3 Switch ethernet fisico . . . 10

**3 Implementazione** **11**
- 3.1 Obiettivi e requisiti . . . 11
  - 3.1.1 Efficienza . . . 11
  - 3.1.2 Interoperabilità . . . 11
  - 3.1.3 Convergenza verso l'utilizzo di strumenti hardware di rete . . . 12
  - 3.1.4 Software libero . . . 12
- 3.2 Contesto di applicazione . . . 12
  - 3.2.1 Infrastruttura fisica . . . 12
  - 3.2.2 Infrastruttura software . . . 13
  - 3.2.3 Infrastruttura di rete . . . 13

3.3 Sperimentazione . . . 13
3.3.1 Gestione traffico MIDI . . . 13
3.3.2 Inoltro del traffico MIDI mediante rete . . . 14
3.3.3 Rete fisica: UDP . . . 14
3.3.4 Rete virtuale: VDE . . . 14
3.3.5 Interfaccia grafica: GTK+ . . . 14

**4 Virtual Switch** **17**
4.1 Funzionamento generale . . . 17
4.2 Le feature . . . 18
4.3 Software dei nodi intermedi . . . 18
4.3.1 Divisione in moduli . . . 18
4.3.2 Strutture dati . . . 22
4.4 Software dello switch . . . 23
4.4.1 Back-end . . . 24
4.4.2 Interfaccia grafica (Front-end) . . . 29
4.5 Considerazioni sulle scelte . . . 33
4.5.1 UDP – TCP . . . 33
4.5.2 Qt - Gtk+ . . . 34
4.5.3 Glade – Gtk+ . . . 35
4.6 Analisi finale . . . 35
4.6.1 Codice sorgente . . . 36

**5 Conclusioni** **37**
5.1 Stato attuale del progetto e sviluppi futuri . . . 37
5.1.1 Stato attuale del progetto . . . 37
5.1.2 Integrazione con RTP MIDI . . . 37
5.2 Limiti del progetto . . . 38
5.3 Valutazioni . . . 38
5.4 Conclusioni . . . 40

**A How to** **43**
A.0.1 midi_send.c . . . 43
A.0.2 midi_recv.c . . . 46

# Capitolo 1

# Introduzione

## 1.1 Presentazione

La musica rappresenta da sempre uno dei più concreti piaceri per l'essere umano e per questo è amata e prodotta in tutto il mondo fin dall'antichità. Il progredire tecnologico e culturale ha chiaramente contribuito al diffondersi e al cambiamento di questa. Sono dunque nati dapprima nuovi strumenti ed in seguito movimenti e generi differenti, testimoni delle epoche in cui si sono sviluppati. Si può notare, infatti, come la diffusione degli stessi sia direttamente proporzionale al progredire tecnologico. Quello che ha caratterizzato i primi decenni del XX secolo, ad esempio, ha avuto un'influenza enorme anche in campo musicale. Le importanti scoperte ed innovazioni che in fisica hanno riguardato l'elettromagnetismo prima e l'elettronica poi da un lato hanno coinvolto l'aspetto della strumentazione (sia con l'implementazione di strumenti esistenti che con la progettazione di nuovi), dall'altro hanno consentito la possibilità di registrare e diffondere la musica. La radio ed i supporti di registrazione hanno permesso per la prima volta alla musica di raggiungere un pubblico vasto ed eterogeneo, sia dal punto di vista culturale che socio-economico; ciò ha favorito la diffusione e la definizione di nuovi generi musicali, spesso nati in contesti popolari legati a ben definite realtà geografiche e culturali; accanto alla musica *alta*, si affermano movimenti musicali che partono dal *basso*, come il blues ed il jazz. Si è soliti riconoscere quest'ultimo come il precursore di molti altri, come il rock'roll, il rock, etc...

L'ultimo grande passo è forse rappresentato dall'avvento dei calcolatori, che hanno permesso la sperimentazione di musica creata digitalmente, come la musica elettronica.

Il lavoro di cui ci si è interessati riguarda quest'ultimo settore della musica. Non si confonda *settore* con *genere*: quello di cui ci si è occupati principalmente sono gli strumenti messi a disposizione e richiesti spesso da chi produce musica

elettronica, ma non esclusivamente legati a questa.

In effetti le tecnologie che analizzeremo *parlano* la lingua della musica, e possono dunque dialogare con qualsiasi delle sue sfaccettature.

L'utilizzo di tali strumenti, non necessariamente per operazioni o composizioni musicali complesse, ha da subito messo in luce due principali problematiche: estrema dipendenza da software proprietario (per l'ottenimento di buoni risultati) e grandi costi nel reperire tutti gli strumenti di cui si necessita.

In questo studio si vuole dunque proporre un differente modello di infrastruttura, software e hardware, per l'utilizzo di tecnologie digitali musicali che sia il più economicamente accessibile possibile, e che non dipenda in nessun modo da software proprietari.

Si propone l'utilizzo delle reti, fisiche e virtuali, per eliminare l'uso di strumenti hardware specifici e l'utilizzo di uno switch virtuale per l'interconnessione degli strumenti per eliminare l'uso di software proprietario.

La struttura del documento è la seguente:

**Scenario** Vengono passati in rassegna i principali protocolli, tecnologie e strumenti hardware utilizzati.

**Implementazione** Vengono definiti gli obiettivi, i requisiti e il contesto di applicazione dello studio in esame. Vengono inoltre illustrate le principali sperimentazioni che sono servite per il progetto.

**Virtual Switch** Si descrive nel dettaglio il lavoro svolto, presentando l'applicazione software oggetto dello studio in esame. Vengono inoltre discusse le scelte effettuate in fase di studio di fattibilità del progetto.

**Conclusioni** Vengono presentati i possibili sviluppi futuri del progetto, i limiti e le valutazioni di questo.

## 1.2 La scelta di affrontare questo tema

Sono sempre stato d'accordo con il proverbio: "La necessità aguzza l'ingegno". E credo anche che chiunque abbia a che fare con il mondo della scienza non possa che esserlo. D'altro canto la storia ci insegna che praticamente qualsiasi cosa è frutto del bisogno, non solo le questioni più materiali: la religione nasce laddove l'uomo sente l'esigenza di protezione, la filosofia quando urge trovare delle risposte concrete a domande complesse, la fisica per spiegare la natura, etc...

Soprattutto durante le giornate di studio intenso, ho sentito spesso la necessità di abbandonare per un po' il mio lavoro e rilassarmi, magari poggiando le dita su dei tasti di un pianoforte o battendo le bacchette di una batteria digitale. È stato così che ho avuto il primo approccio con il mondo degli strumenti musicali digitali e, devo ammetterlo, non è stato per niente piacevole. In primo luogo perché ho usato del software scadente, come spesso si trova allegato allo strumento all'atto dell'acquisto; in secondo luogo perché, al contrario di quanto credessi e avessi voglia, prima di poter usufruire del software ho impiegato molto tempo ed energie, e così magari ho rinunciato. La scoperta poi che ottenere dei risultati soddisfacenti comportava l'acquisto di ulteriore strumentazione, per la maggior parte costosa, mi ha fatto capire che bisognava trovare una soluzione e cercare di ottenere buone prestazioni senza l'ausilio di software ed hardware specifici.

# Capitolo 2

# Scenario

Una delle configurazioni standard nell'ambito di un concerto musicale è la connessione di tutti gli strumenti ad un mixer, dal quale, in maniera remota, si controllano effetti, volumi, direzioni del segnale d'audio, etc... . Inoltre il mixer, come suggerisce il termine inglese, provvede a miscelare i segnali che gli giungono per poi crearne uno solo ed eventualmente inoltrarlo verso altri apparati per ulteriori elaborazioni.

Con l'introduzione di uno standard di formato audio digitale, si è avuta l'esigenza di creare un omologo del mixer analogico. Esattamente come quest'ultimo, infatti, era necessario riuscire ad instradare direttive, configurazioni o l'effettiva generazione di comandi di suoni agli strumenti digitali collegati. Al momento i dispositivi più diffusi per fare ciò sono strumenti che utilizzano hardware/software specifico per questo tipo di operazioni. Anche quando funzionano in maniera generica, i segnali vengono instradati mediante protocolli che richiedono precise apparecchiature per quella determinata funzione.

Scopo di questa tesi è dunque mostrare un prototipo di mixer digitale innovativo, che usa strumenti di rete standard per la diffusione di questi segnali.

## 2.1 Mixer digitale

Il mixer digitale è uno strumento capace di controllare il traffico dati dei dispositivi digitali ad esso connesso. A differenza di quello analogico, oltre all'hardware che lo compone, vi è una buona percentuale di software. Le funzionalità di cui dispone sono all'incirca le stesse dell'omologo analogico, tenendo però conto che sono progettate secondo precise indicazioni di specifici protocolli.

## 2.2 Protocolli musicali

### 2.2.1 MIDI

Il termine MIDI[5](Musical Instrument Digital Interface) nasce negli anni '80 con la definizione del protocollo omonimo, il quale è composta da:

- definizione del connettore fisico
- definizione dei messaggi MIDI
- definizione dello Standard di un file MIDI

#### Connettore fisico

Il connettore fisico, essendo stato progettato nella prima metà degli anni '80, non può garantire le attuali velocità di comunicazione digitale. Ciò nonostante, per lo scambio di dati con un singolo device offre ancora oggi delle notevoli prestazioni; inoltre rimane tuttora spesso necessario per i collegamenti tra strumenti digitali. L'uso di questo connettore risulta oggi essere sempre più in disuso, dato che ad esso si preferiscono le più comode, diffuse ed economiche porte USB.


Figura 2.1: Connettore MIDI (da `http://www.interfacebus.com/PC_MIDI_Pinout.html`)

Il connettore è formato da cinque poli, di cui solo tre vengono utilizzati. Esistono tre differenti tipi di connettori:

- connettore IN: si occupa della ricezione di messaggi MIDI.
- connettore OUT: si occupa dell'invio di messaggi MIDI.
- connettore THRU: si occupa di inoltrare i dati ricevuti dalla porta IN verso un altro device. Si può definire una soluzione naïve per l'interconnessione tra strumenti musicali digitali prima dell'avvento dei mixer digitali.

### Messaggi MIDI

La definizione dei messaggi MIDI è di per sè molto semplice, e proprio per questo molto efficace. Si basa sul traffico di pacchetti, composti da due tipi di "message byte".

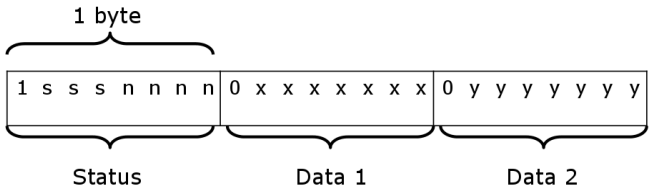


Figura 2.2: Pacchetto MIDI[13]

**Status byte** Inizia sempre per uno ed ha quindi 127 valori possibili. I primi tre bit disponibili (i bit s in figura 2.2) denotano il tipo di messaggio, gli ultimi quattro (i bit n in figura 2.2) sono utilizzati per indicare il numero del canale al quale il contenuto del pacchetto è destinato.

**Data byte** Inizia sempre per zero e dunque utilizza di fatto solo 7 bit, per un totale di 127 valori possibili; rappresenta il valore del dato passato.

Esistono due principali tipi di messaggi: quelli channel e quelli system.

**Channel messages** Sono indirizzati ad un particolare canale, a cui ci si riferisce mediante i bit n da 0 a 15. Appartengono a questa categoria i messaggi "note on", "note off", "polyphonic aftertouch", "control change", "program change", "channel aftertouch", "pitch weel".

**System messages** Sono indirizzati all'intero sistema. Appartengono a questa categoria i messaggi "System Exclusive" e i "System Common messages".

## 2.2.2 File MIDI

Lo Standard MIDI File (SMF) è un insieme di eventi che descrivono le operazioni che deve effettuare la scheda audio, o un altro sistema di riproduzione, per generare i corretti suoni. I file MIDI sono generalmente creati mediante l'uso di sequencer, software appositi per questo tipo di operazioni.

## 2.3 Protocolli di comunicazione tramite rete

### 2.3.1 UDP

Il protocollo User Datagram Protocol[14] (UDP) è un protocollo di livello trasporto connectionless, ovvero senza la necessità di creare prima una connessione tra mittente e destinatario. Non implementa il controllo di flusso, né la consegna affidabile o in giusta sequenza, ma assicura la correttezza del messaggio.

### 2.3.2 VDE

Più che un protocollo, Virtual Distributed Ethernet[12, 16] (VDE) è un ambiente di virtualizzazioni delle reti a livello data link che offre diversi tool. Ne sono stati utilizzati due:

#### VDE Switch

Si tratta della virtualizzazione di uno switch fisico ethernet a cui si possono connettere applicazioni, processi, macchine virtuali etc...

#### VXVDE

Si tratta di uno switch virtuale distribuito, a cui ci si collega mediante l'utilizzo di un indirizzo multicast comune a tutti i device che vi sono connessi.

## 2.4 Toolkit e tecnologie

### 2.4.1 Gtk+

Gimp Toolkit[10] (Gtk+) è un toolkit multi-piattaforma per la creazione di interfacce grafiche.

Nonostante sia stato scritto in linguaggio C, utilizza sostanzialmente un paradigma orientato agli oggetti e supporta molti altri linguaggi di programmazione, quali C++, C#, Python, Ruby, etc... .

Esistono strumenti visuali per facilitare la scrittura del codice in Gtk+. Questi software consentono di predisporre gli elementi grafici dell'interfaccia secondo scelta. Il programmatore può così utilizzare tecnologie What You See Is What You Get (wysiwyg) per disegnare le finestre che verranno visualizzate dall'utente.

### 2.4.2 Glade

Glade[1] è un tool messo a disposizione da GNOME[2] per la costruzione agevolata di interfacce grafiche.

Il programma permette di progettare la Graphical User Interface (GUI) mediante il semplice drag & drop dei window gadget (widget) di Gtk+, predefiniti da Glade stesso. Fatto ciò, di ognuno di quelli richiesti si possono settare facilmente tutte le impostazioni riguardanti l'oggetto. Inoltre, se richiesto l'uso di Cascading Styling Sheets (CSS) per la modifica dell'aspetto dell'interfaccia, permette la visualizzazione in anteprima dell'applicazione che si sta progettando. Infine, cosa non da poco, vi è modo di creare widget personali senza eccessivo sforzo, continuando ad avere comunque la possibilità di monitorare l'anteprima di ciò che si sta realizzando praticamente in tempo reale.

Glade genera un file xml che può essere caricato dinamicamente mediante l'utilizzo di un oggetto, il GtkBuilder, messo a disposizione dall'Application Programming Interface (API) di Gtk+. A differenza però del caso precedente, la creazione di un widget personale richiede una fase di compilazione, in quanto ciò che si è prodotto viene usato come fosse una libreria e quindi linkato all'atto della compilazione dei moduli principali.

## 2.5 Dispositivi di elaborazione

Vengono qui passati in rassegna i principali dispositivi di elaborazione che possono essere utilizzati all'interno della rete predisposta allo scambio di flussi MIDI.

### 2.5.1 Raspberry Pi

Raspberry Pi[7] è un calcolatore di dimensioni contenute, all'incirca una carta di credito, sul quale è possibile installare una distribuzione GNU/Linux[3]. Nonostante le piccole dimensioni, offre delle prestazioni interessanti, soprattutto per lo studio didattico dell'informatica.

### 2.5.2 Udoo

La board Udoo[11], nata da un progetto italiano, come nel caso di Raspberry Pi, ha come caratteristiche principali le moderate dimensioni. A differenza del calcolatore precedente, però, offre maggiori prestazioni e per questo risulta essere molto valida ai fini del progetto in esame. Permette l'installazione di distribuzioni GNU/Linux e di Android.

### 2.5.3 Switch ethernet fisico

Lo switch, o commutatore, è un dispositivo che lavora a livello data link e che si occupa dell'instradamento verso gli indirizzi indicati nei frame ethernet dei pacchetti che gli arrivano.

Gli indirizzi a cui spedire sono i Media Control Access address[14] (MAC address) delle schede di rete dei device a cui è connesso.

# Capitolo 3

# Implementazione

Il problema che si vuole risolvere è la creazione di uno switch virtuale, che interconnetta tutti gli strumenti digitali presenti e che permetta il controllo e l'eventuale modifica dei pacchetti MIDI che vi passano attraverso. Lo switch si troverà presso un calcolatore di comune potenza di calcolo; gli strumenti potranno essere collegati ad esso mediante connessione di rete fisica, virtuale o direttamente mediante connessione MIDI. Ulteriori calcolatori, ma con potenza di calcolo ridotta, saranno usati come nodi intermedi fra strumenti e switch nella rete. In questo caso la loro mansione sarà quella di leggere il traffico MIDI e di inoltrarlo nella rete per fare in modo che giunga allo switch.

## 3.1 Obiettivi e requisiti

L'obiettivo che si intende raggiungere è ottenere una rete - in parte fisica, in parte virtuale - che riesca a convogliare tutto il traffico allo switch MIDI per poi gestirlo ed eventualmente inoltrarlo nuovamente nella rete.

I requisiti che si vogliono rispettare sono:

### 3.1.1 Efficienza

La gestione del traffico deve essere efficiente: non vi deve essere la possibilità che un pacchetto compia un ciclo né tantomeno quella che un pacchetto ritardi o impedisca il fluire del resto dei messaggi.

### 3.1.2 Interoperabilità

Deve essere possibile:

- utilizzare il software creato su tutte le distribuzioni GNU/Linux semplicemente compilandolo ed installando un sintetizzatore MIDI;

- poter interconnettere un qualsiasi strumento digitale che abbia un'interfaccia compatibile.

### 3.1.3 Convergenza verso l'utilizzo di strumenti hardware di rete

La possibilità di utilizzare strumenti hardware di rete standard, permettendo il riuso di questi in altre circostanze ed andando ad abbattere considerevolmente i costi di costruzione dell'infrastruttura, deve essere alla base del progetto. Un ulteriore aspetto è l'enorme semplicità con cui si devono poter reperire gli strumenti necessari.

### 3.1.4 Software libero

Tutto ciò che viene usato, e creato, viene rilasciato con una licenza libera; questo requisito è fondamentale perché il progetto possa andare avanti e crescere anche al di fuori del contesto universitario.

## 3.2 Contesto di applicazione

Essendo gli strumenti fisici facilmente trasportabili, il requisito più importante per procedere alla fase di testing è lo spazio. In effetti, l'idea parte dal voler simulare un palcoscenico musicale, sul quale sono presenti più strumenti musicali.

### 3.2.1 Infrastruttura fisica

Partendo dal presupposto che ogni strumento musicale digitale ha come unica possibilità di connessione ad un calcolatore l'uso della propria interfaccia MIDI, affinchè ciascuno di essi riesca a raggiungere lo switch via rete esistono due possibilità: lo strumento si connette direttamente a quest'ultimo oppure si connette ad un calcolatore intermedio che a sua volta crea un collegamento con lo switch. I calcolatori nella rete possono anche diventare degli end-point laddove si decida di riprodurre mediante sintetizzatore MIDI il traffico che vi dirottiamo. Questa peculiarità fa sì che l'intero sistema risulti molto duttile, adattandosi alle esigenze degli utenti, che in questo caso potrebbero in effetti essere dei musicisti che richiedono di ascoltarsi su dei monitor da palco.

### 3.2.2 Infrastruttura software

L'esistenza di due tipi diversi di utilizzazione di calcolatore nella rete ha portato all'esigenza di creare software differenti a seconda dell'uso. Mentre per lo switch si utilizza software con interfaccia grafica e che richiede maggiore potenza di calcolo dovendo gestire più flussi dati, un calcolatore intermedio ha quasi sempre la sola esigenza di leggere/scrivere e/o inoltrare traffico MIDI. Questo fa sì che il software che vi si utilizza sopra sia più leggero e abbia meno pretese di computazione.

### 3.2.3 Infrastruttura di rete

Come si è detto, l'intero sistema è interconnesso mediante reti fisiche e virtuali. Non risulta importante per lo switch conoscere attraverso che tipo di connessione arrivi il flusso MIDI, ma lo è piuttosto per il calcolatore intermedio: quest'ultimo, infatti, ha l'esigenza - fin da quando parte il programma - di sapere come verrà interfacciato alla rete, in quanto si utilizzerà software differente a seconda del tipo.

## 3.3 Sperimentazione

L'obiettivo di questa sperimentazione è riuscire a capire se, portando il progetto a sviluppi maggiori, vi siano le basi per un concreto utilizzo dello switch virtuale come mixer reale, consumando poche risorse ed ottenendo prestazioni nell'ordine della soddisfacibilità. L'intero progetto si compone di piccoli pezzi, così come un puzzle, che sono stati uniti per dare vita a del software più potente e di maggiori prospettive.

### 3.3.1 Gestione traffico MIDI

Un primo passo è stato quello di verificare che la lettura di pacchetti MIDI prima, e la conseguente scrittura di questi poi, portasse ai risultati sperati. In effetti così è stato, grazie anche all'essersi basati su principi di progettazione di questo genere di software comune ad altri prodotti open source, quali Rosegarden[8]. Il passo successivo è stato provare a modificare i dati letti ed inoltrarli così come da modifica. Oltre all'implementazione del codice sperimentale, vi è da considerare una grande fetta di studio e ricerca nel fare propri i principi del protocollo MIDI. In merito, la documentazione è abbastanza scarna, ma - bisogna riconoscerlo - molto omogenea.

### 3.3.2 Inoltro del traffico MIDI mediante rete

A differenza della semplice gestione, inoltrare traffico lungo la rete presupponeva l'esistenza di due tipi differenti di software: un server ed un client. Un'ulteriore differenziazione di software era dovuta al diverso tipo di rete che si stava utilizzando: fisica o virtuale.

### 3.3.3 Rete fisica: UDP

L'utilizzo del ben affermato e conosciuto protocollo UDP non ha richiesto grossi sforzi di ricerca, quanto piuttosto il cimentarsi in un tipo di attività pressochè nuova e poco documentata. Gli iniziali esperimenti sono stati svolti con l'ausilio di una board Udoo che facesse da destinatario dei messaggi ed un normale calcolatore che funzionasse da nodo sorgente. Verificato che la rete fisica era un buono strumento sul quale far circolare il traffico, si è passati alla rete virtuale.

Un esempio rappresentativo di tale sperimentazione è fornito in figura 3.1.

### 3.3.4 Rete virtuale: VDE

L'utilizzo del software VDE ha comportato una buona fetta di studio per quanto riguarda la trasmissione dei suoi pacchetti. A differenza del protocollo UDP, che come è ben noto si appoggia sul livello network, ovvero su Internet Protocol[14] (IP), la API di VDE utilizza frame ETHERNET e si appoggia dunque sul livello data link. Una volta superato questo scoglio, ci si è concentrati sul verificare il corretto fluire di traffico MIDI, sia utilizzando dei VDE switch sia utilizzando VXVDE.

Un esempio rappresentativo è fornito in figura 3.2.

### 3.3.5 Interfaccia grafica: GTK+

L'ultimo passo prima di avviare definitivamente il progetto è stata la conferma che il toolkit GTK+ rispecchiasse realmente le necessità: grafica potente ma senza uso di grosse risorse. A tal proposito è stata necessaria la documentazione sulla creazione di widget personalizzati, ovvero oggetti dell'interfaccia non messi a disposizione direttamente da gtk+.

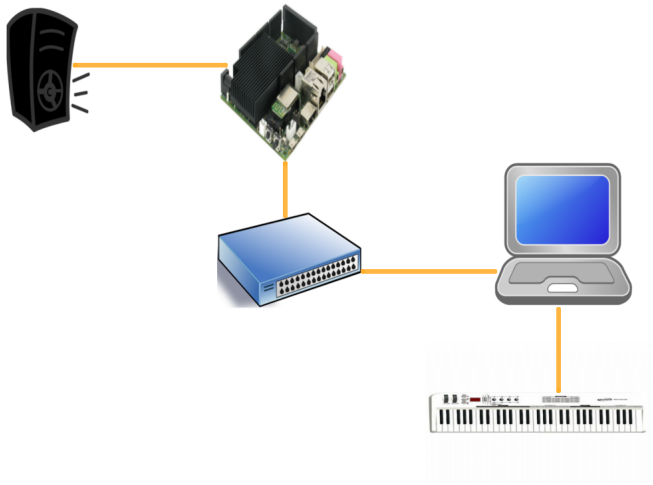

Figura 3.1: Infrastruttura per l'inoltro del traffico di pacchetti MIDI mediante UDP/IP. La tastiera musicale digitale viene collegata via MIDI ad una macchina, la quale riceve il traffico prodotto dallo strumento musicale. Il calcolatore e la board Udoo sono tra di loro connessi mediante uno switch ethernet fisico; il flusso di dati arriva fino all'Udoo dove viene riprodotto con l'ausilio di un sintetizzatore e un collegamento a delle casse.

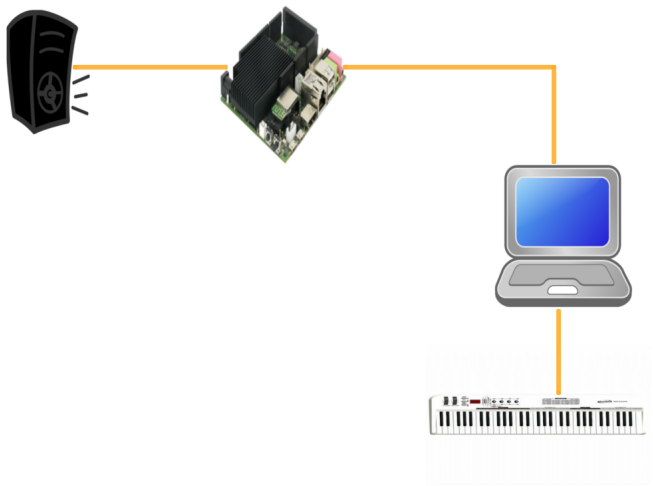

Figura 3.2: Infrastruttura per l'inoltro del traffico di pacchetti MIDI mediante VDE/VXVDE. La tastiera musicale digitale viene collegata via MIDI ad una macchina, la quale riceve il traffico prodotto dallo strumento musicale. Il calcolatore e la board Udoo sono tra di loro connesse mediante VDE_SWITCH o VXVDE; il flusso di dati arriva fino all'Udoo dove viene riprodotto con l'ausilio di un sintetizzatore e un collegamento a delle casse.

# Capitolo 4

# Virtual Switch

L'obiettivo finale è costruire un'applicazione con una ben definita interfaccia grafica che garantisca la facile gestione del traffico MIDI che attraversa la rete. Sarà necessario dunque, oltre all'implementazione del software che utilizzaranno i calcolatori intermedi, riunire e aggiornare gli esperimenti fatti in fase di studio di fattibilità del progetto e precedentemente descritti.

## 4.1 Funzionamento generale

Il progetto è suddiviso in due parti: la prima che implementa lo switch virtuale, la seconda un nodo intermedio connesso alla rete. Quest'ultimo ha il compito di riprodurre, leggere o inoltrare traffico MIDI.

Dati per scontati i collegamenti fisici e quelli software, il primo servizio messo a disposizione dal software è l'autoconfigurazione della rete. Mentre nel caso si utilizzi una connessione fisica è necessario conoscere l'indirizzo IP[14] dello switch per mandare il primo messaggio, nel caso di connessione virtuale i nodi intermedi alla rete mandano dei messaggi broadcast per "presentarsi". Lo switch, una volta ottenuti i messaggi di presentazione, acquisisce e memorizza gli indirizzi dei mittenti in caso di utilizzo futuro e registra un timestamp che indica il momento in cui è pervenuto il pacchetto. I nodi intermedi si occupano di confermare la loro esistenza all'interno della rete ogni tot prestabilito. Il software setta un timeout, scaduto il quale si avvia un controllo su tutti i nodi presenti in memoria per verificare che non sia passato oltre un eccessivo tempo dall'ultima conferma di presenza. Una volta che lo switch ha la lista dei nodi connessi, dall'interfaccia principale si possono selezionare i device della rete e da lì richiedere una specifica feature.

Un particolare riferimento va fatto allo switch virtuale che, essendo anche lui nodo interno alla rete, ha dunque la possibilità di essere controllato, localmente e non.

## 4.2 Le feature

Dato l'obiettivo che ci si è posti, è necessario poter scegliere gli end-point della connessione che si vuole controllare. A tal proposito è stato previsto che vi sia un nodo sorgente ed uno destinazione: il primo verrà selezionato dalla schermata principale, che quindi aprirà una finestra di controllo appositamente per quel device, mentre il secondo sarà scelto dalla pagina appena aperta. Chiaramente, in entrambi i casi, deve essere possibile poter scegliere la porta MIDI del nodo che si intende utilizzare; anche questa operazione viene effettuata dalla finestra del primo device aperto. Questo, inoltre, viene considerato il nodo di input, a differenza del secondo che rappresenta invece l'output.

Una volta ottenuto il controllo del traffico sulla connessione precedentemente instaurata, si può richiedere il cambio volume, strumento e canale.

## 4.3 Software dei nodi intermedi

Il software che sta alla base dei nodi intermedi deve permettere il soddisfacimento di una richiesta remota. L'idea di fondo di questa implementazione infatti è fortemente connessa al concetto di Remote Procedure Call[14] (RPC). In effetti, un thread si occupa di ricevere delle richieste dallo switch e di gestirle. A seconda del tipo di richiesta può venire lanciato un altro thread, che sarà la vera implementazione del controllo di flusso effettuato dallo switch. Una volta che il servizio è stato lanciato ed è in fase di funzionamento, l'attività principale diventa la lettura, scrittura e/o inoltro del traffico MIDI.

Quando l'utente richiede la modifica di suddetto traffico MIDI da interfaccia grafica, lo switch traduce la richiesta in pacchetto MIDI e la inoltra istantaneamente. In questo modo, per il device che si occupa della scrittura dei pacchetti, non vi è alcuna differenza su come trattare quelli che gli arrivano: dopo la lettura verranno gestiti per quello che sono.

Il thread che gestisce il flusso MIDI può essere chiuso solo dal processo di cui è parte; in questo caso o arriva una richiesta dallo switch con riferimento alla sua chiusura, oppure si aspetta il termine del programma che lo gestisce.

### 4.3.1 Divisione in moduli

La suddivisione del codice in moduli rispecchia quella in funzionalità. In particolare, si è cercato di scindere il più possibile la componente MIDI dal resto.

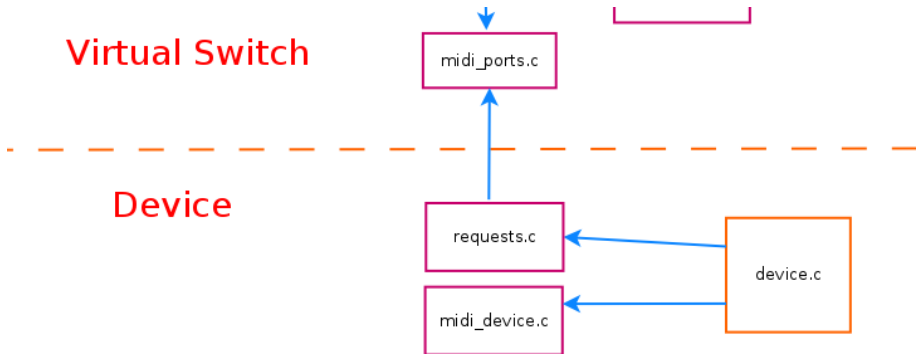


Figura 4.1: Suddivisione del codice del device in moduli.

### Ciclo principale

Il ciclo principale si trova nel file device.c. Qui sono implementate le procedure che si occupano di notificare la presenza del device nella rete allo switch e di confermarla dopo un certo numero di secondi. Oltre alla main, responsabile dell'avvio del programma, le altre principali procedure sono:

- `void keep_alive_vde(thread_arg *arg)`

  Procedura che, in caso di connessione tramite l'uso di VDE_SWITCH o VXVDE, rinnova la sua presenza all'interno della rete ogni tot tempo prestabilito.

- `void keep_alive_ip(thread_arg *arg)`

  Procedura che, in caso di connessione tramite uso del protocollo IP, rinnova la sua presenza all'interno della rete ogni tot tempo prestabilito.

- ```
  void give_response(char* string_remote_address,
  unsigned short int remote_port_number,
  mes request, int connection_type)
  ```

  Procedura che, chiamata dopo aver ricevuto una richiesta, si occupa di soddisfarla e nel caso rispondere al mittente.

### Le richieste

Il modulo per gestire l'arrivo di richieste e l'eventuale risposta è requests.c. Qui vi si trovano:
  ```

- `void wait_ip_request()`

  Procedura che, in caso di connessione tramite l'uso del protocollo IP, rimane in ascolto sulla connessione in attesa di richieste da parte dello switch.

- `void wait_vde_request()`

  Procedura che, in caso di connessione tramite l'uso di VDE_SWITCH o VXVDE, rimane in ascolto sulla connessione in attesa di richieste da parte dello switch.

- `void ip_send_midi_ports(char *string_remote_ip_address, unsigned short int remote_port_number)`

  Procedura che, in caso di connessione tramite l'uso del protocollo IP, si occupa di inoltrare allo switch le porte MIDI aperte sul device.

- `void vde_send_midi_ports(char *string_remote_ip_address, unsigned short int remote_port_number)`

  Procedura che, in caso di connessione tramite l'uso di VDE_SWITCH o VXVDE, si occupa di inoltrare allo switch le porte MIDI aperte sul device.

### Gestione MIDI

Probabilmente la parte più importante dell'applicazione, il modulo midi_device.c contiene l'implementazione delle procedure che si occupano di gestire, leggere, scrivere ed eventualmente riprodurre i pacchetti MIDI. Nonostante vengano chiamate anche queste in risposta ad una richiesta dallo switch, si è preferito aggiungerle in un modulo a parte, in modo da incapsulare le procedure che utilizzano MIDI e di conseguenza separarle dalle procedure rimanenti.

- `int play_midi()`

  Procedura che scrive sullo stream MIDI i byte contenuti in un buffer.

- `void* midi_send_vde(void * thread_arg)`

  Procedura che, in caso di connessione tramite l'uso di VDE_SWITCH o VXVDE, legge dallo stream MIDI aperto e invia i byte incapsulati in frame ethernet lungo la connessione virtuale.

- `void* midi_send_ip(void * thread_arg)`

  Procedura che, in caso di connessione tramite l'uso del protocollo IP, legge dallo stream MIDI aperto e invia i byte incapsulati in frame UDP lungo la connessione fisica.

- `void *midi_recv_vde(void *thread_arg)`

  Procedura che, in caso di connessione tramite l'uso di VDE_SWITCH o vxvde, riceve dallo switch dei frame ethernet con all'interno dei byte MIDI e, con l'lausilio di play_midi(), li riproduce.

- `void *midi_recv_ip(void *thread_arg)`

  Procedura che, in caso di connessione tramite l'uso del protocollo IP, riceve dallo switch dei frame UDP con all'interno dei byte MIDI e, con l'ausilio di play_midi(), li riproduce.

### Gestione porte MIDI

Il codice che si occupa di recuperare le porte MIDI aperte su un device si trova su midi_ports.c. Questo modulo ha richiesto una scelta di progettazione particolare: è infatti condiviso con il software dello switch. Il motivo risiede nel fatto che sia il software del device che quello switch stesso hanno la necessità di recuperare le porte su di essi aperte.

Le principali procedure che vi si trovano sono:

- `midi_card* get_midi_cards(midi_card *root)`

  Procedura che si occupa di ottenere le informazioni sulle schede audio rilevate sul dispositivo. Se vengono riconosciute come schede audio MIDI, si vanno a ricercare le porte di quella scheda aperte.

- `midi_dev* get_midi_dev(midi_card *port, int n_card, snd_ctl_t *ctl)`

  Procedura che si occupa di ottenere le informazioni sulle porte aperte di una data scheda audio MIDI.

- `midi_subdev* get_midi_subdevice(midi_dev *dev, snd_ctl_t *ctl, int n_card, int n_dev)`

  Procedura che si occupa di contare quanti canali possiede una determinata porta MIDI e - nel caso di uno solo - riportare quale.

### 4.3.2 Strutture dati

L'utilizzo di porte well-known, dichiarate nel file header net.h, in comune tra codice dello switch e dei device, ha permesso l'utilizzo di strutture dati essenziali e ridotte.

La più importante è di certo la struct device_midi_thread_arg, ovvero quella che viene passata al thread MIDI che è responsabile della gestione del flusso MIDI.

La struttura è definita come segue:

```
struct device_midi_thread_arg {

  char string_remote_address[100];
  char port_in[100];
  char port_out[100];
  unsigned short int port;

}
```

La specifica dei campi è la seguente:

**char string_remote_address[100]** Rappresenta l'indirizzo del destinatario /mittente dei pacchetti rispettivamente mandati o ricevuti nella connessione. Questo può essere un MAC address o un indirizzo IP. Le procedure che utilizzano questa struct, infatti, usano una stringa come indirizzo.

**char port_in[100]** Rappresenta il nome della porta dalla quale si deve leggere lo stream MIDI.

**char port_out[100]** Rappresenta il nome della porta sulla quale si deve scrivere lo stream MIDI.

**unsigned short int port** In caso di connessione tramite l'uso del protocollo IP, è la porta UDP che, in aggiunta all'indirizzo, fornisce tutte le indicazioni del destinatario.

Un'altra importante struttura è quella che è servita per la memorizzazione delle porte MIDI, o meglio delle schede audio con le loro porte. La struct è definita come segue:

```
typedef struct midi_card {    /*struct for card*/
```

```
    int card;
    char name[50];
    char hw[20];
    int n_dev;
    int is_midi;
    struct midi_card *next;
    struct midi_dev *dev_root;  /*NULL if there are 16 subdevices*/
} midi_card;
```

La specifica dei campi è la seguente:

**int card** Rappresenta il numero della scheda audio.

**char name[50]** Rappresenta il nome della scheda audio.

**char hw[20]** Rappresenta il nome nella forma "hw:x,y,z", dove x è il numero di scheda, y il numero di porta e z il numero di canale se quest'ultimo è uno solo, altrimenti il nome è nella forma "hw:x,y".

**int n_dev** Rappresenta il numero di porte aperte sulla porta.

**int is_midi** Specifica se la scheda audio è MIDI o meno.

**struct midi_dev *dev_root** Punta alla lista che descrive le porte MIDI aperte sulla scheda

**struct midi_card *next** Punta alla struttura della lista successiva.

## 4.4 Software dello switch

Come ormai noto, una parte del software dello switch riguarda l'interfaccia grafica. Si è cercato di renderla il più facile e comprensibile possibile, tenendo comunque conto che l'utente a cui è rivolta è colui che possiede un minimo di dimestichezza con il MIDI, le sue interfacce e gli strumenti.

Tutto il resto del programma è eseguito in background. Come accennato in 4.1, la costruzione della rete che vede lo switch, la gestione della lista dei device che mantiene in memoria, il controllo del traffico MIDI, etc..., sono tutte operazioni che non fanno parte dell'interfaccia. Per questo, vengono presentate separatamente le due parti che compongono il programma.

### 4.4.1 Back-end

La parte più consistente del programma è di sicuro questa: all'avvio lo switch fa partire un thread per tipo di connessione, che resterà in vita fino alla chiusura del programma. Ognuno di questi ha il compito di ricevere i messaggi di presentazione dai device nella rete e, se non presenti, aggiungerli nella lista che mantiene in memoria.

All'atto dell'apertura del controllo di un dispositivo, a questo viene inoltrata la richiesta MIDI_PORTS, ovvero gli si domanda quali porte MIDI abbia aperte.

Viene dunque richiesto un ulteriore thread, che si occupa di aspettare i messaggi contenenti i nomi delle porte MIDI provenienti dal device in esame.

A questo punto, dall'interfaccia grafica, si può finalmente controllare il traffico MIDI, attraverso una tab (sotto-pagina specifica) dedicata per ogni nodo di cui si è richiesto il controllo.

Quando viene aperto il canale di trasmissione tra i device selezionati, si apre un nuovo thread, con la specifica funzione di incanalare correttamente il flusso MIDI source (sorgente) e sink (destinatario), eventualmente con le modifiche apportate dall'utente tramite interfaccia; a seconda di quali siano input e output, il thread farà scelte diverse, richiamando opportunamente procedure differenti. Sono previste queste varianti:

- SWITCH_LOCAL_IN_OUT: il device input e quello output sono lo switch. Il traffico viene gestito dunque localmente.
- DEVICE_LOCAL_IN_OUT: il device input e quello output sono gli stessi, in particolare uno specificato nodo della rete.
- IN_OUT: il device input e quello output sono differenti, in particolare due nodi della rete.
- SWITCH_IN: il device input è lo switch, quello output è un nodo della rete.
- SWITCH_OUT: il device input è un nodo della rete, quello output è lo switch.

#### Divisione in moduli

Le diverse funzionalità richieste dallo switch in background sono state la naturale traccia per la divisione dei moduli.

**Gestione lista nodi** I thread che si occupano di gestire la lista di device collegati allo switch sono implementati nel modulo midi_nodes.c. Le principali procedure sono:

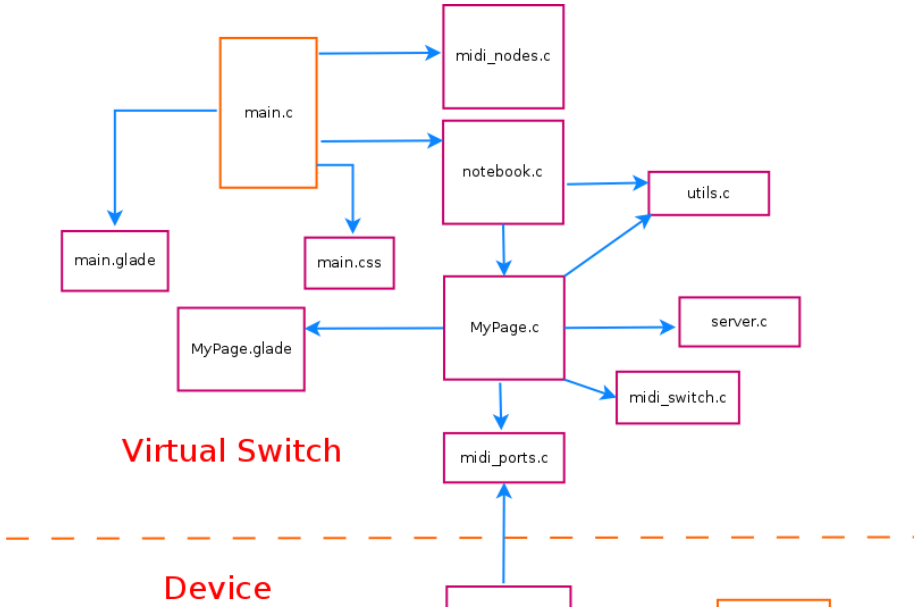


Figura 4.2: Suddivisione del codice dello switch in moduli.

- `void create_midi_node(char *name, char *string_remote_address, int connection_type)`

  Procedura che si occupa di aggiungere alla lista un nuovo nodo sulla base del suo nome, del suo indirizzo e del tipo di connessione che instaura con lo switch.

- `void * check_node_to_remove()`

  Procedura che si occupa di scandire periodicamente la lista alla ricerca di un nodo da rimuovere in caso non abbia riconfermato la sua presenza nei tempi prestabiliti.

- `void wait_for_eth_nodes()`

  Procedura che, in caso di connessione switch tramite l'uso di VDE_SWITCH o VXVDE, si occupa di aspettare i messaggi di presentazione e le riconferme di presenza nella rete da parte dei nodi.

- `void wait_for_ip_nodes()`

  Procedura che, in caso di connessione tramite l'uso del protocollo IP, si occupa di aspettare i messaggi di presentazione e le riconferme di presenza nella rete da parte dei nodi.

**Gestione porte MIDI** Il codice che si occupa di recuperare le porte MIDI aperte sullo switch si trova su midi_ports.c (si faccia riferimento a 4.3.1 nella sezione "Gestione porte MIDI").

**Gestione richieste** Le procedure che inviano le richieste verso un device sono implementate nel modulo server.c. Qui vi si trovano:

- `void send_ip_request(char *string_remote_ip_address, int req, int page_index, char *msg_text)`

  Procedura che, in caso di connessione tramite l'uso del protocollo IP, si occupa di mandare una richiesta ad un determinato device, sulla base del suo indirizzo e di una porta well-known.

- `void send_vde_request(char *string_remote_ip_address, int req, int page_index, char *msg_text)`

  Procedura che, in caso di connessione tramite l'uso di VDE_SWITCH o VXVDE, si occupa di mandare una richiesta ad un determinato device, sulla base del suo indirizzo.

**Gestione MIDI** Uno dei moduli più importanti è di sicuro midi_switch.c, in cui vi sono implementate le procedure per la gestione del traffico MIDI che passa per lo switch. In particolare, sono definite:

- `void switch_local_in_out(midi_local_thread_arg *midi_arg)`

  Procedura che, in caso di traffico MIDI interamente su switch, implementa le funzioni di gestione ed elaborazione dei pacchetti che lo attraversano.

- `void in_out(midi_ext_thread_arg *midi_arg)`

  Procedura che, in caso di traffico MIDI interamente su device esterni differenti, implementa le funzioni di inoltro ed eventuale elaborazione dei pacchetti che arrivano allo switch.

- `int send_midi_ip(int socketfd, char *address, char *buffer, int page_index)`

  Procedura che, in caso di connessione tramite l'uso del protocollo IP, si occupa di inoltrare i pacchetti MIDI verso il nodo di output.

- `int recv_midi_ip(int socketfd, char *buffer, int page_index)`

  Procedura che, in caso di connessione tramite l'uso del protocollo IP, si occupa di ricevere i pacchetti MIDI dal nodo di input.

- `int send_midi_vde(int socketfd, char *address, char *buffer)`

  Procedura che, in caso di connessione tramite l'uso di VDE_SWITCH o VXVDE , si occupa di inoltrare i pacchetti MIDI verso il nodo di output.

- `int recv_midi_vde(int socketfd, char *buffer)`

  Procedura che, in caso di connessione tramite l'uso di VDE_SWITCH o VXVDE, si occupa di ricevere i pacchetti MIDI dal nodo di input.

### Strutture dati

Oltre alle strutture progettate appositamente per il passaggio di parametri ai thread, le più importanti risultano essere quella per le richieste di servizi ai device, quella per il controllo del traffico MIDI e quella che rappresenta un nodo della rete nella memoria dello switch.

**mes** La definizione del messaggio per le richieste è veramente semplice, ma proprio grazie a questa sua caratteristica è stata utilizzata non solo come domanda, ma anche come risposta di un servizio richiesto.

```
struct mes {
  int type;
  char text[100];
}
```

La specifica dei campi è la seguente:

**int type** Rappresenta il tipo di richiesta/risposta che il messaggio contiene.

**char text[100]** Rappresenta il contenuto della richiesta/risposta che il messaggio contiene.

**midi_control_packet** La struttura rappresenta la modifica al traffico MIDI che l'utente richiede da interfaccia. Questa, a seconda se il servizio giri in locale o meno, verrà inoltrata al thread che si occupa della lettura del flusso mediante socketpair o procedure di altri moduli che forniscono altri servizi di rete.

```
struct midi_control_packet {
  int type;
  int value;
}
```

La specifica dei campi è la seguente:

**int type** Rappresenta il tipo di modifica al traffico MIDI che si è richiesto di effettuare da interfaccia.

**int value** Rappresenta il valore della modifica al traffico MIDI che si è richiesto di effettuare da interfaccia.

**midi_node** La definizione della struttura di un nodo all'interno della rete è identica per nodi connessi mediante protocolli differenti, se non per il valore di un flag che ne indica appunto il tipo. Questo ha permesso di tenere in memoria una sola lista, e di effettuare quindi tutte le operazioni che la riguardano in maniera più efficiente.

```
struct midi_node {
  char name[50];
  char address[100];
  int connection_type;
  int is_in;
  struct timeval tv;
  struct midi_node *next;
  MyPage *page;
}
```

La specifica dei campi è la seguente:

**char name[50]** Rappresenta il nome del device connesso allo switch.

**char address[50]** Rappresenta l'indirizzo del device connesso allo switch. Sia per connessioni mediante l'uso del protocollo IP, sia per quelle mediante l'uso di VDE_SWITCH o VXVDE, è rappresentato da una stringa. Le procedure che utilizzano questa struttura si occuperanno in seguito di fare le opportune operazioni di conversione.

**int connection_type** Rappresenta il tipo di connessione tra nodo e switch.

**int is_in** Specifica se il device è da considerarsi ancora operativo all'interno della rete o se non ha ripresentato il suo messaggio di conferma entro il timeout prefissato.

**struct timeval tv** Specifica l'istante in cui il device si è presentato per la prima volta o quello dell'ultima riconferma.

**struct midi_node *next** Punta al nodo successivo memorizzato nella lista.

**MyPage *page** Punta alla pagina allocata all'interno dell'interfaccia per quel nodo. Se invece non ne è stato richiesto il controllo, il puntatore è NULL.

### 4.4.2 Interfaccia grafica (Front-end)

L'interfaccia si presenta con la scelta dei device che si vogliono controllare. Una volta selezionato uno di questi, si apre una pagina dedicata al controllo del traffico MIDI che passa, in entrata e/o in uscita, sul dispositivo. L'utilizzo del widget GtkNotebook di Gtk+ si è rivelata una scelta azzeccata, soprattutto per la comodità nel passare da una pagina all'altra senza utilizzare più finestre diverse del programma. Questo è stato comunque il frutto della creazione del widget personale MyPage, che ha permesso così l'utilizzo di un template specifico per ogni nuova tab.

All'interno di questo widget vi è una netta separazione tra gli oggetti che riassumono le informazioni principali del traffico MIDI sotto gestione (indirizzi device, canale in uso, volume per il canale, etc... ) e i widget che permettono l'uso delle feature precedentemente elencate in 4.2.

Infine un piccolo terminale si occupa di riportare gli eventi rilevanti che avvengono in background.

In figura 4.3 è rappresentata la pagina di gestione del traffico MIDI così come è allo stato attuale.

#### Divisione in moduli

Come brevemente illustrato in 2.4.2, Glade crea un file xml che può essere analizzato al volo; da qui viene creata l'interfaccia. Discorso differente va fatto invece per i widget personali, che hanno bisogno di essere prima compilati e poi linkati all'eseguibile principale. Nonostante ciò, si è cercato, per quanto possibile, di mantenere una netta separazione tra moduli in cui sono presenti le procedure per l'interfaccia e moduli in cui vi sono quelle per le funzionalità in background. Si faccia riferimento alla figura 4.2 per una visione di insieme dei moduli.

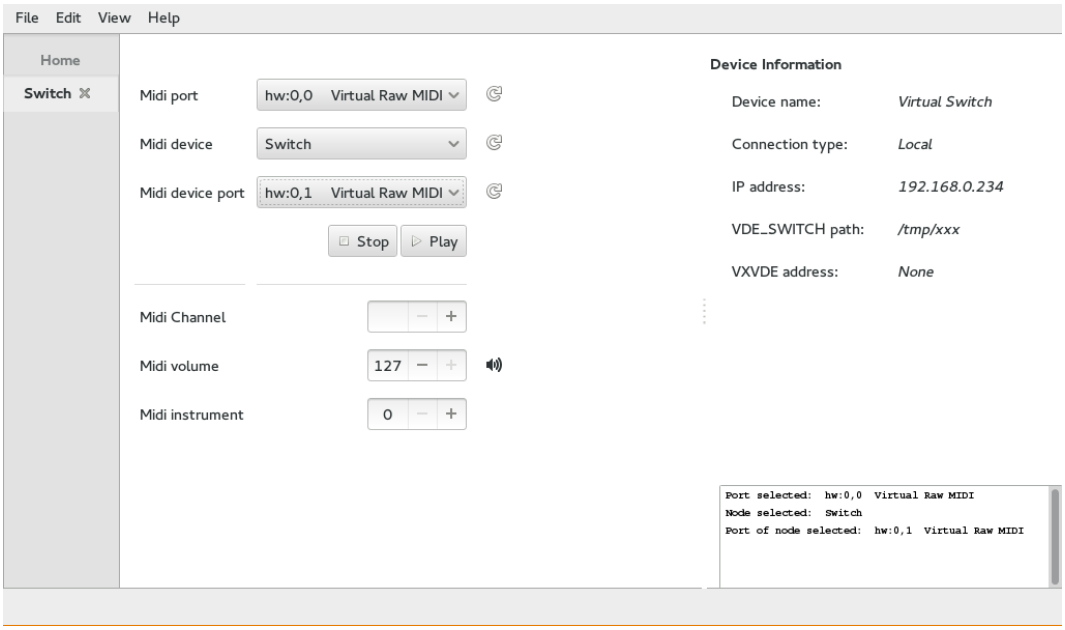


Figura 4.3: Screenshot della pagina da cui si controlla il traffico MIDI tra due device.

**Moduli Glade** Il file che descrive l'interfaccia principale si chiama main.glade. Qui vi sono le gerarchie che compongono la struttura dell'interfaccia. Tra queste, la più importante è quella che riguarda il widget GtkNotebook, ovvero il contenitore delle pagine di controllo dei device della rete.

Un ulteriore file in xml, MyPage.glade, descrive lo scheletro del widget personale che si è creato. A differenza del caso precedente, per il corretto funzionamento di questo nuovo oggetto, si è dovuti intervenire con una buona percentuale di programmazione in C, mediante l'utilizzo di metodi messi a disposizione dall'API di Gtk+.

**Moduli C** Tutte le funzioni di aggiornamento, visualizzazione e modifica dei contenuti dell'interfaccia hanno richiesto dei moduli a parte. In aggiunta a questi, si ricorda la parte di programmazione necessaria alla creazione del widget personale che si è utilizzato.

Il modulo dunque che contiene la parte più ponderante è MyPage.c. In primo luogo qui si trovano le procedure che permettono la cattura dei requisiti presenti nel modulo MyPage.glade, la definizione del nuovo tipo MyPage e l'allocazione del nuovo oggetto quando richiesto. Secondariamente sono implementati i servizi sopracitati.

Allo stesso modo, il modulo notebook.c contiene le procedure per le funzionalità del GtkNotebook, come l'aggiunta di una pagina e quindi un widget MyPage.

In questa sezione va ricordato anche il modulo del ciclo principale, main.c; questo avvia l'interfaccia e lancia i thread principali, sia del back-end che del front-end.

Viene riportata una delle procedure più importanti per la gestione del traffico MIDI, nonostante si preoccupi esclusivamente di raccogliere le richieste dell'utente dall'interfaccia ed inoltrarle al processo che le smaltisce.

```
void
midi_control(GtkWidget *page, gpointer *user_data) {
  MyPage *tmp_page = MY_PAGE(page);
  midi_control_packet control_packet;

  GtkWidget *from = GTK_WIDGET(user_data);
  int volume = 63;
  int instrument = 0;
  int channel = 0;

  if(from == tmp_page->volume_button) {
    volume = gtk_scale_button_get_value(GTK_SCALE_BUTTON(from));
    control_packet.type = VOL_CHANG;
    control_packet.value = volume;
    printf("Volume changed : %d\n", volume);
  } else if(from == tmp_page->volume_spin) {
    instrument = gtk_spin_button_get_value_as_int (GTK_SPIN_BUTTON(from));
    control_packet.type = VOL_CHANG;
    control_packet.value = volume;
    printf("Volume changed : %d\n", volume);
  } else if(from == tmp_page->instrument_spin) {
    instrument = gtk_spin_button_get_value_as_int (GTK_SPIN_BUTTON(from));
    printf("Instrument changed: %d\n", instrument);
    control_packet.type = INSTR_CHANG;
    control_packet.value = instrument;
  } else if(from == tmp_page->channel_spin) {
    channel = gtk_spin_button_get_value_as_int (GTK_SPIN_BUTTON(from));
    control_packet.type = CHAN_CHANG;
    control_packet.value = channel;
    printf("Channel changed: %d\n", channel);
  }  else if(from == tmp_page->stop_midi_thread_button) {
    control_packet.type = STOP;
    control_packet.value = 0;
  }

  send(tmp_page->other.midi_fd[writesocket], &control_packet,
  sizeof(midi_control_packet), MSG_NOSIGNAL); /*socketpair send*/
}
```

A seconda di quale widget si è utilizzato per modificare il flusso MIDI, la procedura cattura il valore richiesto dall'utente e lo inoltra al processo che si occupa di trasformarlo in messaggio MIDI, così da poter esser letto.

**Strutture dati** Quasi tutte le strutture dati relative all'interfaccia grafica sono state progettate in funzione del widget personale che si è utilizzato. A tal proposito, merita di essere riportata quella che forse è la più importante, dato che rappresenta la pagina che si alloca nel GtkNotebook ogni volta che si richiede il controllo di un device.

**MyPage** La struttura è composta quasi per intero da puntatori agli oggetti di cui è composta, ma contiene anche una struttura dati per la gestione di tutti quei campi utili alla pagina che non sono però necessariamente del front-end.

```
struct MyPage
{
  GtkPaned Paned;


  GtkWidget *button_ports;
  GtkWidget *combo_ports;
  GtkTextBuffer *page_textbuffer;
  GtkWidget *page_terminal;
  GtkWidget *page_combo_nodes;
  GtkWidget *page_node_combo_ports;
  GtkWidget *volume_button;
  GtkWidget *volume_spin;
  GtkWidget *stop_midi_thread_button;
  GtkWidget *instrument_spin;
  GtkWidget *channel_spin;
  page_other other;

}
```

## 4.5 Considerazioni sulle scelte

L'adozione delle tecnologie e dei protocolli suddetti è stata fatta sulla base di alcune considerazioni in merito a ciò che fosse più opportuno, veloce ed efficiente da utilizzare per il progetto in esame.

Si riportano i principali confronti fatti sulla base delle caratteristiche che i protocolli, o le tecnologie, presentano.

### 4.5.1 UDP – TCP

Essendo nella necessità di utilizzare un protocollo di livello trasporto, la scelta poteva chiaramente ricadere su Trasmission Control Protocol[14] (TCP) o UDP (per quest'ultimo si veda 2.3.1). Verrà fornita una breve descrizione di TCP per completezza, e per permettere al lettore di fare propri i punti chiave della scelta; la quale è ricaduta su quest'ultimo in quanto, per caratteristiche, si adatta meglio al servizio che doveva fornire.

#### TCP

TCP, essendo un protocollo orientato alla connessione, prevede che vi sia uno scambio di messaggi iniziali tra gli host con l'obiettivo di instaurare un collegamento logico tra questi. Un'altra peculiarità del protocollo in esame è l'essere stato progettato per garantire affidabilità, controlli di flusso e di congestione. Un'ultima caratteristica da prendere in considerazione è la modalità di trasmissione dei dati: una volta aperta la connessione, vengono spediti flussi di byte.

#### Confronto

L'affidabilità garantita da TCP si presta ad essere richiesta da applicazioni che necessitano di garantire sempre il corretto fluire dei dati. Il fatto poi di trasmettere flussi, detti segmenti, permette all'applicazione di non effettuare controlli ed ordinamenti sui pacchetti che riceve. Infine il controllo di flusso e di congestione sono sicuramente degli ottimi strumenti per coloro i quali abbiano a che fare con grosse quantità di dati.

Tuttavia bisogna considerare i servizi che l'applicazione, di cui si sono precedentemente descritti i principi, intende realizzare. In primo luogo le quantità di dati che circolano nell'intera rete non sono tali da richiedere controlli particolari, tenendo anche conto del fatto che vi possono essere connessioni virtuali. Secondariamente, bisogna considerare che l'affidabilità di TCP, da indubbio vantaggio, nel caso in esame potrebbe passare ad essere uno svantaggio. Nell'insistere col rimandare un pacchetto non ricevuto dal destinatario, per esempio, si potrebbe incorrere nel rischio di farlo giungere quando ormai non ve ne è più bisogno, anzi quando questo stona con il contesto.

Infine, oltre alla non necessità di tali aspetti, vi è da considerare l'aggiunta di overhead che TCP comporta.

Dunque, per le ragioni appena messe in luce, si apprezza maggiormente il servizio best-effort fornito dal protocollo UDP, così definito per le sue caratteristiche.

### 4.5.2 Qt - Gtk+

Essendo nella necessità di utilizzare un toolkit/framework per la creazione di interfacce grafiche, il confronto è stato fatto sui due principali contendenti della scena: Gtk+ e Qt[6], il primo alla base di GNOME (come visto in 2.4.1), il secondo alla base di K Desktop Environment[4] (KDE). Verrà fornita una breve descrizione di Qt per completezza, e per permettere al lettore di fare propri i punti chiave della scelta.

#### Qt

Qt è un framework multi-piattaforma impiegato prevalentemente per lo sviluppo di interfacce grafiche. Solitamente utilizzato con il linguaggio C++ standard, offre però anche la possibilità di binding con altri linguaggi. Inoltre mette a disposizione feature per il back-end, come l'accesso a database sulla base di SQL, parsing XML, supporto per le reti, etc... Dispone di un suo Integrated Development Environment (IDE) per la creazioni di applicazioni complete negli ambiti previsti dai progettisti di Qt.

#### Confronto

La documentazione presente in rete non permette di ottenere un confronto sulle prestazioni evidente e oggettivo. Spesso infatti, confrontando caratteristiche così simili e altrettanto valide, la scelta ricade sul gusto personale o su sottigliezze. Detto ciò, sembra non vi siano grosse differenze dal punto di vista dell'utilizzo di risorse.

Dovendo scegliere dunque, una delle motivazioni principali che hanno portato all'utilizzo di Gtk+ in questo progetto è stata la maggiore vicinanza al linguaggio C di questo toolkit, tenendo chiaramente conto del fatto che il resto del progetto è implementato in C. La leggerezza e la concretezza del software di aiuto Glade hanno infine eliminato qualsiasi dubbio rimasto.

### 4.5.3 Glade – Gtk+

Come detto, per costruire l'interfaccia si è usato l'ausilio di un software aggiuntivo, nonostante Gtk+ metta a disposizione un'API completa, che permette di ottenere gli stessi risultati. Chiaramente quest'ultimo e Glade non sono comparabili, essendo due tecnologie del tutto differenti. Vengono dunque messi in risalto i principali punti per cui si è scelto l'utilizzo del software di cui si è precedentemente parlato (si veda 2.4.2 per ulteriori dettagli).

**Confronto**

Le difficoltà riscontrate nei primi utilizzi di Glade, dovute in parte alla poca documentazione fornita, non sono state comunque tali da portare al non impiego del software in esame. In effetti, la possibilità di definire graficamente l'interfaccia, e caricarla dinamicamente, sono aspetti davvero notevoli se rapportati all'alternativa: usare solo l'API di Gtk+ significa costruire alla cieca la GUI per poi verificarla di volta in volta mediante compilazione di tutto il codice prodotto.

Non si può tralasciare la grande velocità con la quale si genera l'interfaccia e si settano tutte le impostazioni degli oggetti; cosa che, senza l'ausilio di Glade, comporterebbe l'utilizzo di un metodo per ogni impostazione del widget. Infine, qualità da non sottovalutare affatto in progetti di medie/grande dimensioni è la grande facilità con la quale si ottiene una netta separazione tra i moduli di codice contenenti l'ambiente grafico e quelli del back-end. Questo aiuta la mantenibilità del codice; modifiche dell'interfaccia comportano solo la modifica grafica e non un grande impatto sulla riscrittura del codice.

Sulla base di tutte queste considerazioni, è risultato più conveniente l'utilizzo del software di cui si è appena discusso.

## 4.6 Analisi finale

Così come è stato presentato, il progetto sembra soddisfare in pieno i requisiti e gli obiettivi che ci si era posti in 3.1, in particolare:

**Efficienza** L'utilizzo di connessioni diverse permette un maggiore dislocamento delle risorse utilizzate; nonostante infatti il traffico raggiunga sempre lo switch, l'utilizzo di VXVDE e VDE_SWITCH fa sì che siano presenti nella rete più switch virtuali che si occupano di gestire i pacchetti. Inoltre l'infrastruttura, per come è stata definita, non consente la presenza di cicli.

**Interoperabilità** Il software creato non presenta particolari requisiti per cui non possa essere eseguito su determinate distribuzioni GNU/Linux. Inoltre, grazie all'utilizzo di device che permettono il collegamento degli strumenti musicali digitali alla rete, ogni tipo di questi con interfaccia compatibile può essere connesso.

### 4.6.1 Codice sorgente

Il codice sorgente del progetto di cui finora si è discusso è rilasciato sotto licenza GNU GPL[3] (General Public License) versione 2 ed è reperibile all'indirizzo:
*http://github.com/lukesmolo/MIDI-Virtual-Switch*

# Capitolo 5

# Conclusioni

## 5.1 Stato attuale del progetto e sviluppi futuri

Nato da una semplice esigenza, il progetto ha dato subito l'impressione che espandendosi avrebbe potuto soddisfarne delle altre. Si può quindi affermare che, rispetto a come era stato concepito inizialmente, il lavoro di cui ci si è occupati si è notevolmente allargato. Inoltre, esistono molte soluzioni di miglioramento e sviluppo che richiedono studi a parte.

### 5.1.1 Stato attuale del progetto

Allo stato attuale del progetto non sono ancora state implementate alcune delle funzionalità di gestione del traffico MIDI maggiormente usate, come il "polyphonic aftertouch", il "sustain", etc...

Un altro punto su cui non ci si è ancora potuti concentrare è l'apertura di più connessioni verso un singolo device, in modo, ad esempio, che questo possa ricevere più flussi MIDI. Allo stesso tempo, se pur nell'ordine delle cose da fare, non è ancora stata gestita la possibilità di separare i canali MIDI e di inoltrarli su device differenti.

Non vi è stato tempo di concretizzare le suddette idee, ma ciò non toglie che siano sicuramente degli sviluppi futuri per il progetto.

### 5.1.2 Integrazione con RTP MIDI

Real Time Protocol MIDI[9] (RTP MIDI), o AppleMIDI, è un protocollo sviluppato dalla Apple per la trasmissione in rete dei pacchetti MIDI. Si basa sul protocollo di livello applicazione RTP, a sua volta basato su UDP; mette a disposizione un meccanismo di recupero dei pacchetti persi nella rete, eliminando dunque la necessità di ritrasmissione di un pacchetto e di conseguenza evitando possibili ritardi.

Purtroppo esistono ancora poco supporto e documentazione per l'interfacciamento con i moduli del kernel Linux, ma di certo non appena possibile si cercherà di rendere compatibile il progetto di cui si è discusso finora con questo protocollo.

## 5.2 Limiti del progetto

Come tutti i lavori, anche questo progetto ha dei punti deboli. In particolare, forse quello maggiore riguarda l'inoltro di ogni singolo pacchetto MIDI: nonostante questo sia composto al più da 3 byte, se ne usano ben 42 per impacchettarlo in un frame UDP (8 per l'header UDP, 20 per quello IP, e 14 per quello ethernet), ovvero ottenendo circa il 1400% di overhead. Tuttavia questo dato non deve far spaventare il lettore, in quanto una tipica trasmissione dati con interfaccia MIDI ha una velocità dell'ordine di grandezza di alcune decine di Kb/s, mentre una connessione di rete è in grado di ottenere prestazioni anche centinaia di migliaia di volte migliori. Di conseguenza l'overhead può essere un problema per il consumo di risorse, ma non di certo per le prestazioni dello strumento musicale.

Una possibile soluzione comunque sarebbe rappresentata dal settare un timeout entro il quale racccogliere più pacchetti MIDI possibili e, scaduto questo, spedirli in un frame unico come fossero un flusso.

Tuttavia questa soluzione rappresenta un trade-off rispetto a ciò che si vuole ottenere: se da un lato infatti utilizzando meno byte diminuisce l'uso di risorse, dall'altro si introduce un ritardo fittizio per la raccolta dei pacchetti.

Analisi di prestazione e soprattutto di accettabilità di trasmissione e riproduzione del suono verranno fatti in futuro, proprio per capire quale dei due aspetti sia meglio privilegiare.

## 5.3 Valutazioni

Per verificare che l'incapsulamento dei pacchetti MIDI ed il relativo inoltro non fossero troppo costosi in termini di tempo, sono state effettuate delle valutazioni volte a capire che tipo di ritardo venga aggiunto nel flusso di dati.

Per fare ciò, si è calcolato quanto tempo impiegasse un pacchetto ad essere letto da uno strumento musicale, spedito, ricevuto ed infine scritto su una porta predefinita.

L'obiettivo era dunque dimostrare che, nonostante si introducano necessariamente dei ritardi, i tempi di computazione siano pressochè indifferenti da quelli ottenuti non utilizzando nessun tipo di connessione di rete.

In effetti, come si evince dal confronto tra le tabelle 5.1, 5.2, 5.3 e 5.4, non vengono percepiti ritardi significativi, anzi spesso i valori sono quasi identici.

Tabella 5.1: Flusso di 10000 pacchetti MIDI da Virtual Switch a board Udoo

| **Connessione** | **Tempo massimo (ms)** | **Tempo minimo (ms)** | **Tempo medio (ms)** |
|---|---|---|---|
| UDP | 186.00 | 0 | 44.71 |
| UDP[1] | 186.00 | 0 | 44.71 |
| VDE_SWITCH | 496.00 | 0 | 49.99 |
| VDE_SWITCH[1] | 406.00 | 0 | 44.42 |

[1] è stato aggiunto uno switch fisico ethernet nella connessione tra i due dispositivi

Tabella 5.2: Flusso di 10000 pacchetti MIDI da board Udoo a Virtual Switch

| **Connessione** | **Tempo massimo (ms)** | **Tempo minimo (ms)** | **Tempo medio (ms)** |
|---|---|---|---|
| UDP | 186.00 | 0 | 44.75 |
| UDP[1] | 186.00 | 0 | 44.73 |
| VDE_SWITCH | 192.00 | 0 | 49.57 |
| VDE_SWITCH[1] | 194.00 | 0 | 47.96 |

[1] è stato aggiunto uno switch fisico ethernet nella connessione tra i due dispositivi

Essendo stati considerati solo possibili ritardi nell'ordine dei millisecondi, non si possono però escludere quelli nell'ordine dei nanosecondi; ciò è ovviamente superfluo ai fini della valutazione, soprattutto tenendo conto del fatto che l'udito umano percepisce appena il decimo di secondo.

Un'altra considerazione che si può fare dai dati sottomano è che sembra esserci un valore massimo che normalmente non si supera. Questo significa che tutte le configurazioni che si stanno esaminando potrebbero avere gli stessi tempi di computazione anche nel caso pessimo; è un'ulteriore prova, insieme ai tempi medi, che tutti gli strumenti utilizzati nel progetto si sono rivelati all'altezza.

Va messo anche in evidenza che i tempi delle computazioni che riguardano connessioni mediante VDE_SWITCH sembrano essere leggermente maggiori delle connessioni UDP. Ulteriori analisi future saranno effettuate per chiarire meglio questo punto.

Come ultimo aspetto si deve considerare come la board Udoo abbia ottenuto praticamente gli stessi risultati dello switch virtuale, il quale però è installato su un calcolatore molto più potente. Questo giustifica la nostra scelta iniziale: un dispositivo con buone prestazioni, ma con risorse e dimensioni ridotte.

Tabella 5.3: Flusso di 10000 pacchetti MIDI interamente su Virtual Switch

| **Connessione** | **Tempo massimo (ms)** | **Tempo minimo (ms)** | **Tempo medio (ms)** |
|---|---|---|---|
| MIDI[1] | 186.00 | 0 | 44.74 |
| UDP[2] | 186.00 | 0 | 44.75 |
| VDE_SWITCH | 186.00 | 0 | 44.74 |

[1] è stata usata l'interfaccia MIDI dello strumento musicale ed un adattatore USB 2.0
[2] è stata usata l'interfaccia di rete di loopback

Tabella 5.4: Flusso di 10000 pacchetti MIDI interamente su board Udoo

| **Connessione** | **Tempo massimo (ms)** | **Tempo minimo (ms)** | **Tempo medio (ms)** |
|---|---|---|---|
| MIDI[1] | 186.00 | 0 | 44.76 |
| UDP[2] | 186.00 | 0 | 44.73 |
| VDE_SWITCH | 186.00 | 0 | 44.76 |

[1] è stata usata l'interfaccia MIDI dello strumento musicale ed un adattatore USB 2.0
[2] è stata usata l'interfaccia di rete di loopback

## 5.4 Conclusioni

Il lavoro di cui si è discusso fin qui ha messo in luce la possibilità di poter realmente gestire il traffico MIDI mediante protocolli di rete.

L'implementazione dello switch MIDI è stato finora il fine del progetto; in un'ottica però differente, questo può essere considerato invece lo strumento per raggiungere un obiettivo diverso e di maggiore portata: nell'ottica di Internet of Things[15] infatti, il lavoro svolto rappresenta un tentativo per cercare di rendere MIDI un sotto-protocollo di rete, fisica e virtuale. L'utilizzo poi dei tipici strumenti hardware di rete è stata la conferma definitiva che questa filosofia può essere applicata in qualsiasi contesto.

I risultati fin qui discussi sono il frutto di diverse tappe: come prima cosa si è analizzato il contesto nel quale un lavoro di questo genere poteva essere affrontato, mediante la selezione delle tecnologie che risultavano più convenienti. Definiti obiettivi e requisiti, ci si è concentrati sulla sperimentazione di ciò che si era scelto, affrontando uno per volta i vari protocolli e strumenti. Infine si sono riuniti i diversi pezzi del puzzle per dar vita a ciò che è il risultato finale.

L'utilità di un progetto di questo calibro sta anche nel fatto di aver permesso di cimentarsi per la prima volta con protocolli di natura differente e aver dovuto affrontare scelte di progettazione non sempre banali.

In conclusione dunque, nonostante non si siano potute implementare tutte le funzionialità in modo da rendere il progetto completo sotto ogni punto di vista, si sono sicuramente gettate le basi per degli sviluppi futuri.

# Appendice A

# How to

In questa sezione si presenta brevemente uno degli esperimenti effettuati in fase di valutazione di fattibilità del progetto. In particolare si mostra come sia possibile incapsulare in frame ethernet dei pacchetti MIDI letti da uno strumento musicale ed il successivo inoltro di questi mediante VDE_SWITCH (si faccia riferimento alla figura 3.2 per la strutturazione della rete).

Per comodità e leggibilità del codice, sono stati utilizzati due moduli: il primo si occupa di leggere da interfaccia MIDI il flusso di byte ed il secondo di riceverli e scriverli su una determinata porta.

### A.0.1 midi_send.c

La prima cosa da fare è dichiarare le variabili che servono, ovvero il path dello switch virtuale che si utilizza, la porta MIDI dalla quale leggere il traffico e le informazioni per spedire un pacchetto ethernet.

```
/*numero di protocollo fittizio per i frame ethernet*/
unsigned short proto = 0x1234;

/*destinatario dei pacchetti ethernet*/
char src[6];

/*mittente dei pacchetti ethernet*/
char dest[6];

/*path dello switch virtuale*/
char _switch[] = "/tmp/xxx";

/*struttura del frame ethernet*/
union ethframe frame;char sender = 'a';

char destination = 'b';
```

```
int data_len = sizeof(char)*3;

/*connessione verso lo switch*/
VDECONN *conn;

int frame_len;
/*socket per la conessione allo switch virtuale*/
int socketfd;

/*parametri necessari all'apertura della connessione*/
struct vde_open_args open_args = {.port=0,.group=NULL,.mode=0700};

/*porta dalla quale si legge il traffico MIDI*/
char portname[] = "hw:2,0,0";

/*modalita' con la quale interagire con l'interfaccia MIDI*/
int mode = SND_RAWMIDI_SYNC;

/*maniglia per connessione MIDI*/
snd_rawmidi_t* midiin = NULL;

/*struttura per la poll della connessione MIDI*/
struct pollfd *pfd_in;

/*struttura per la poll della connessione vde*/
struct pollfd *pfd_out;

int npfd_in = 1;
int npfd_out = 1;
```

A questo punto le operazioni da fare sono: aprire una connessione verso lo switch virtuale ed una verso l'interfaccia MIDI dalla quale si legge il flusso.

```
/*apertura della connessione verso lo switch virtuale*/
conn = vde_open(_switch, "test", &open_args);

if (conn == NULL) {
  printf("Error opening vde connection: %s\n",strerror(errno));
  exit(1);
}
/* "bind" del socket con la connessione*/
socketfd = vde_datafd(conn);

if ((status = snd_rawmidi_open(&midiin, NULL, portname, mode)) < 0) {
  printf("Problem opening MIDI input: %s\n", snd_strerror(status));
  exit(1);
}

pfd_in = (struct pollfd *)alloca(npfd_in * sizeof(struct pollfd));
```

```
/*procedura che lega la poll all'interfaccia MIDI*/
snd_rawmidi_poll_descriptors(midiin, pfd_in, 1);

pfd_in[0].events = POLLIN;
pfd_in[0].revents = 0;


pfd_out = (struct pollfd *)alloca(npfd_out * sizeof(struct pollfd));

pfd_out[0].events = POLLOUT;
pfd_out[0].revents = 0;
pfd_out[0].fd = socketfd;
```

Una volta aperte le connessioni l'ultima impostazione da settare sono i campi del frame ethernet.

```
/*per il MAC address del destinatario e del mittente si usano sei caratteri
  char; puo' esssere utilizzato qualsiasi carattere, basta che il primo byte
  sia pari in modo che l'indirizzo non risulti essere MULTICAST*/
memset(dest, destination, sizeof(dest));
memset(src, sender, sizeof(dest));

/*caratteri pari, ovvero 64 nella conversione esadecimale*/
dest[0] = 'd';
src[0] = 'd';

frame_len = data_len + ETH_HLEN;

/*assegnamento del protocollo fittizio che si e' dichiarato precedentemente*/
frame.field.header.h_proto = htons(proto);

memcpy(frame.field.header.h_dest, dest, ETH_ALEN);
memcpy(frame.field.header.h_source, src, ETH_ALEN);
```

Adesso è tutto pronto per avviare un loop nel quale si legge - se disponibile un pacchetto in lettura - e, se il socket è disponibile in scrittura, si invia di conseguenza.

```
while(1) {
  /*controllo che ci sia un pacchetto da leggere*/
  if (poll(pfd_in, npfd_in, -1) > 0) {
    if (pfd_in[0].revents & POLLIN) {
      if((status = snd_rawmidi_read(midiin, frame.field.data, 3)) < 0) {
        printf("Problem reading MIDI input: %s", snd_strerror(status));
      }

    }
  }
  /*controllo che il socket sia disponibile in scrittura*/
  if (poll(pfd_out, npfd_out, 0) > 0) {
```

```
      if (pfd_out[0].revents & POLLOUT) {
        vde_send(conn, frame.buffer, frame_len, 0);
      }
    }
  }
```

### A.0.2 midi_recv.c

Le variabili necessarie sono più o meno quelle del modulo di cui si è discusso in precedenza, con l'eccezione del nome della porta sulla quale si vuole scrivere il traffico MIDI che si riceve. Anche le operazioni di apertura delle connessioni e il settaggio delle poll sono equivalenti, mentre cambia il loop nel quale si leggono e scrivono i pacchetti.

```
while(1) {
  /*controllo che ci sia qualche pacchetto da leggere*/
  if (poll(pfd_in, npfd_in, -1) > 0) {
    if (pfd_in[0].revents & POLLIN) {
      count = vde_recv(conn, buffer_recv, ETH_FRAME_LEN, 0);
      frame = (struct field*) buffer_recv;
    }
  }

  /*controllo che si possa scrivere sulla porta MIDI*/
  if (poll(pfd_out, npfd_out, 0) > 0) {
    if (pfd_out[0].revents & POLLOUT) {
      if((status = snd_rawmidi_write(midiout, frame->data, 3)) < 0) {
        printf("Problem writing to MIDI output: %s",
          snd_strerror(status));
      }
    }
  }

}
```

# Bibliografia


[1] Glade - A User Interface Designer. http://glade.gnome.org. Accessed: 07-2014.

[2] GNOME. http://www.gnome.org/. Accessed: 07-2014.

[3] GNU Operating System. http://www.gnu.org/. Accessed: 04-2014.

[4] KDE. http://www.kde.org/. Accessed: 07-2014.

[5] MIDI manufacturers association (MMA). The complete MIDI 1.0 detailed specification. http://www.midi.org. Accessed: 07-2014.

[6] Qt Project. http://qt-project.org/. Accessed: 07-2014.

[7] Raspberry Pi. http://www.raspberrypi.org/. Accessed: 04-2014.

[8] Rosegarden. http://www.rosegardenmusic.com/. Accessed: 04-2014.

[9] RTP payload Format for MIDI. http://www.rfc-editor.org/info/rfc6295. Accessed: 08-2014.

[10] The GTK+ Project. http://www.gtk.org/. Accessed: 07-2014.

[11] Udoo. http://www.udoo.org/. Accessed: 04-2014.

[12] Virtual Square: all the virtuality you always wanted but you were afraid to ask. http://wiki.v2.cs.unibo.it/wiki/index.php/Main_Page. Accessed: 02-2014.

[13] Juan Pablo Bello. MIDI code. Technical report, New York University, http://www.nyu.edu/classes/bello/FMT.html.

[14] Larry Peterson, Bruce Davie. *Reti di Calcolatori*. Apogeo, 2012.

[15] Friedemann Mattern, Christian Floerkemeier. From the Internet of Computers to the Internet of Things. Technical report, Distributed Systems Group, Institute for Pervasive Computing, ETH Zurich, 2010.

[16] Renzo Davoli, Michael Goldweber. Virtual Square: Users, Programmers & Developers Guide. http://wiki.v2.cs.unibo.it/wiki/index.php/Main_Page, 2008.