Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE
Corso di Laurea in Informatica

Integrazione di macchine virtuali User-Mode Linux relativistiche nell'ambiente di emulazione Marionnet

Relatore:
Chiar.mo Prof.
RENZO DAVOLI

Presentata da:
GIANLUCA GUIDI

Sessione II
Anno Accademico 2013-2014

Ai miei genitori, che mi hanno consentito di proseguire gli studi.

Indice

Introduzione

Nel corso degli anni sono stati sviluppati programmi in grado di emulare vari dispositivi, come macchine o computer. Questi dispositivi virtuali sono prima di tutto pensati per essere il più possibile simili ai corrispondenti dispositivi reali. L'obiettivo è essenzialmente quello di raggiungere un buon livello di fedeltà dell'emulazione, in modo tale che non sia possibile, o sia comunque difficile, riuscire a distinguere tra un dispositivo emulato ed uno reale. Una caratteristica importante del *software* in generale è quella di essere molto più flessibile e adattabile rispetto all'*hardware*. Grazie a questa proprietà è possibile introdurre nelle macchine emulate alcune proprietà che sarebbero altrimenti difficili da ottenere fisicamente. Ad esempio, si possono introdurre artificialmente dei difetti per testare il comportamento dei programmi in situazioni avverse. Nella realtà, è difficile introdurre difetti nei dispositivi fisici e sarebbe in ogni caso una pratica costosa, in quanto spesso le modifiche non sono reversibili. Ciò che avviene più raramente anche nei programmi di emulazione invece è l'introduzione di risorse *migliori* di quelle disponibili, come ad esempio la velocità di calcolo o la velocità di trasferimento dei dati. Avere a propria disposizione una rete molto veloce può essere utile per testare i programmi applicativi in condizioni che non sarebbero altrimenti riproducibili nel presente, ma che potrebbero esserlo in futuro, in seguito all'abbassamento dei costi e/o al miglioramento della qualità delle tecnologie esistenti. Per questo motivo sono state create macchine virtuali relativistiche, cioè macchine il cui tempo scorre ad una velocità che non corrisponde necessariamente a quella del tempo reale. Dal punto di vista della macchina virtuale relativistica il cui tempo scorre più lentamente tutto appare essere più veloce, perché il tempo di completamento reale delle operazioni rimane invariato, ma alle macchine virtuali sembra essere minore. Per informazioni sul contesto tecnologico in cui il progetto si inserisce, si veda il capitolo 2.

Le macchine relativistiche sono state realizzate con una piccola modifica a User-Mode Linux [10, 11, 32]: ogni volta che deve essere letta l'ora reale,

essa viene ricalcolata per ottenere l'ora virtuale. La definizione del tempo virtuale ed il conseguente calcolo sono basati su due parametri: la frequenza virtuale ed il punto di convergenza. Il primo parametro indica la velocità alla quale scorre il tempo virtuale, mentre il secondo è un punto temporale in cui tempo reale e virtuale assumono lo stesso valore. Sono state realizzate due versioni della modifica a User-Mode Linux, una che opera a livello di intercettazione delle *system call* dei processi virtualizzati ed una che opera al livello di interazione tra User-Mode Linux e il sistema operativo che lo ospita.
Per facilitare la creazione e l'utilizzo di una rete di macchine virtuali relativistiche, è stato scelto di sfruttare l'interfaccia grafica di Marionnet [24–26], un emulatore di reti basato su User-Mode Linux e VDE [8, 9]. L'interfaccia e il livello di controllo sottostante sono stati adattati per consentire all'utente di inserire i parametri necessari alla definizione del tempo virtuale. I parametri necessari alla "relativizzazione" possono essere specificati per un singolo dispositivo o globalmente, per includere tutti i dispositivi aggiunti alla rete.
Per una presentazione della funzionalità offerta, si veda il capitolo 1. Per maggiori informazioni sul funzionamento di User-Mode Linux e Marionnet e sull'implementazione della modifica, si veda il capitolo 3.

Capitolo 1

Scopo e utilizzo

Macchine virtuali

In questo progetto sono state realizzate macchine virtuali relativistiche, facilmente collegabili in una rete virtuale. Normalmente si è abituati a pensare alle macchine (computer) come a dispositivi strettamente fisici che includono almeno un processore e necessitano di corrente elettrica per funzionare. Tuttavia la componente fisica è solamente un mezzo per poter eseguire qualche attività di interesse per la persona che utilizza la macchina. Se la macchina è programmabile (d'ora in poi ci si riferirà sempre implicitamente a macchine programmabili) le attività che devono essere svolte sono definite dal *software*. Tutte o alcune delle caratteristiche di una macchina sono riproducibili via *software*. È possibile quindi ottenere un programma che sia in grado di eseguire molte delle attività che possono essere svolte da un macchina fisica. Una macchina realizzata via *software* viene chiamata macchina virtuale. L'aggettivo virtuale indica che l'oggetto in questione è una imitazione di un oggetto reale. Sebbene un oggetto virtuale, come una macchina, abbia alcune limitazioni rispetto ad un oggetto reale (ad esempio, un caffè fatto con una macchinetta del caffè virtuale non può essere bevuto), per esso sono assenti alcuni vincoli fisici altrimenti inevitabili nella realtà.

Macchine relativistiche

Cos'è una macchina relativistica? È una macchina il cui tempo scorre in modo anomalo, diversamente dal tempo reale. Il concetto che nella realtà il tempo non scorra a velocità costante è stato introdotto per la prima volta nella teoria della relatività di Einstein. Un'importante conseguenza di tale teoria è infatti che varie misurazioni, tra cui quelle del tempo, sono relative alla velocità dell'osservatore. All'interno di una macchina virtuale relativistica il tempo scorre in maniera uniforme, ma non necessariamente alla stessa velocità del tempo nella macchina reale. In altre parole, la frequenza

della macchina virtuale è costante, ma non necessariamente unitaria. Questa proprietà fornisce molte possibilità di utilizzo.

Il primo obiettivo del progetto è quello di consentire la creazione di reti virtuali in cui alcuni collegamenti abbiano un'ampiezza di banda superiore al normale. In particolare è possibile produrre ampiezze di banda virtuali superiori a quelle fisiche ottenibili con le risorse limitate di una macchina reale. Infatti, anche se il tempo virtuale scorre ad una velocità diversa, la velocità effettiva delle operazioni rimane sempre la stessa; se il tempo virtuale scorre lentamente tutte le attività appariranno, dal punto di vista dei processi virtualizzati, essere completate in poco tempo. È così possibile creare l'illusione di reti veloci, perché i tempi di trasferimento sembrano essere accorciati. Anche la latenza viene ridotta come conseguenza di questo effetto, ma essa può essere aumentata artificialmente, ad esempio se la rete è creata sfruttando VDE. Quest'ultimo fornisce i mezzi per introdurre vari difetti nella rete, inclusi appunto dei ritardi elevati che sono in grado di contrastare l'effetto della frequenza virtuale sulle macchine coinvolte. La simulazione di reti veloci è utile perché, anche se non c'è un riscontro immediato nella realtà, molto probabilmente ci sarà in futuro: le tecnologie impiegate per la trasmissione dei dati sono in evoluzione, soprattutto per quanto riguarda la capacità della trasmissione. L'unico limite insuperabile è dato dalla velocità della luce, ma questo si applica solamente al ritardo nella trasmissione dei dati, non alla capacità del canale.

Ci sono anche altri possibili utilizzi della funzionalità aggiunta. Ad esempio, ponendo valori diversi per la frequenza di due o più macchine virtuali, è possibile verificare il funzionamento di protocolli e algoritmi di sincronizzazione, come NTP (*Network Time Protocol*) [30]. Normalmente il processo di emulazione di una macchina tende a ridurre la precisione delle misurazioni del tempo, quindi test di questo tipo possono produrre risultati grossolani. Tuttavia, per valori di frequenza sufficientemente bassi, la precisione della macchina virtuale risulta essere migliore (ma mai superiore rispetto alla macchina fisica), perché le misurazioni sembrano essere effettuate con una frequenza maggiore.
Un altro contesto in cui può essere utile una macchina relativistica è quello in cui si vuole osservare in poco tempo l'effetto di azioni ripetute automaticamente ad intervalli regolari. In questo caso è sufficiente aumentare la frequenza virtuale per ridurre l'ampiezza degli intervalli.

La funzionalità aggiunta verrà di qui in avanti chiamata "relativizzazione", con riferimento alla capacità delle macchine virtuali di percepire il tempo diversamente rispetto alla macchina reale. Per poter utilizzare efficacemente la relativizzazione è necessario comprendere il modo in cui viene definito il tempo virtuale. Tale definizione è composta da due elementi, chiamati frequenza e punto di convergenza (a cui ci si riferisce sinteticamente con il termine "convergenza").
La frequenza determina la velocità a cui scorre il tempo virtuale. L'unità di misura per questo valore è l'Hertz (Hz). In altre parole questo valore viene espresso come il numero di secondi virtuali che passano per ogni secondo reale.
La convergenza determina lo spostamento del tempo virtuale rispetto a quello reale: il "presente virtuale" può infatti trovarsi più o meno lontano nel passato o nel futuro. Questo elemento è espresso con un istante temporale. Il modo in cui viene espresso un istante temporale corrisponde a quello dello standard POSIX, cioè contando il numero di secondi passati dalla *Epoch* (00:00:00 del 1 gennaio 1970) [16].

1.1 Singola macchina virtuale

Il supporto per la relativizzazione è stato aggiunto a User-Mode Linux, un particolare tipo di macchina virtuale che non emula i componenti fisici. Per maggiori informazioni sul principio di funzionamento di User-Mode Linux, consultare la sezione 3.1. Per poter utilizzare la funzione di relativizzazione è necessario applicare la *patch* alla corretta versione di Linux. La *patch* non è molto intrusiva, quindi è compatibile con o facilmente adattabile a molte versioni del *kernel*. Prima della compilazione del *kernel* a cui è stata applicata la *patch*, al momento della configurazione, è possibile disattivare la funzione di relativizzazione. Il simbolo si chiama `CONFIG_RELATIVISTIC_TIME`, ed è presentato con il nome "*Relative virtual time*" nella sezione specifica di User-Mode Linux (UML) nei menù di configurazione (figura 1.1).

Dopo aver applicato la *patch* e compilato i sorgenti, verrà creato un file eseguibile chiamato `linux`: la sua esecuzione determina l'avvio del *kernel* virtuale. Per il corretto funzionamento della macchina virtuale, è necessario immettere almeno due parametri nel comando di avvio, chiamati `mem` e `ubd0`. Una stringa di avvio deve avere quindi la seguente forma:

```
$ linux mem=⟨memoria⟩ ubd0=⟨file system⟩
```

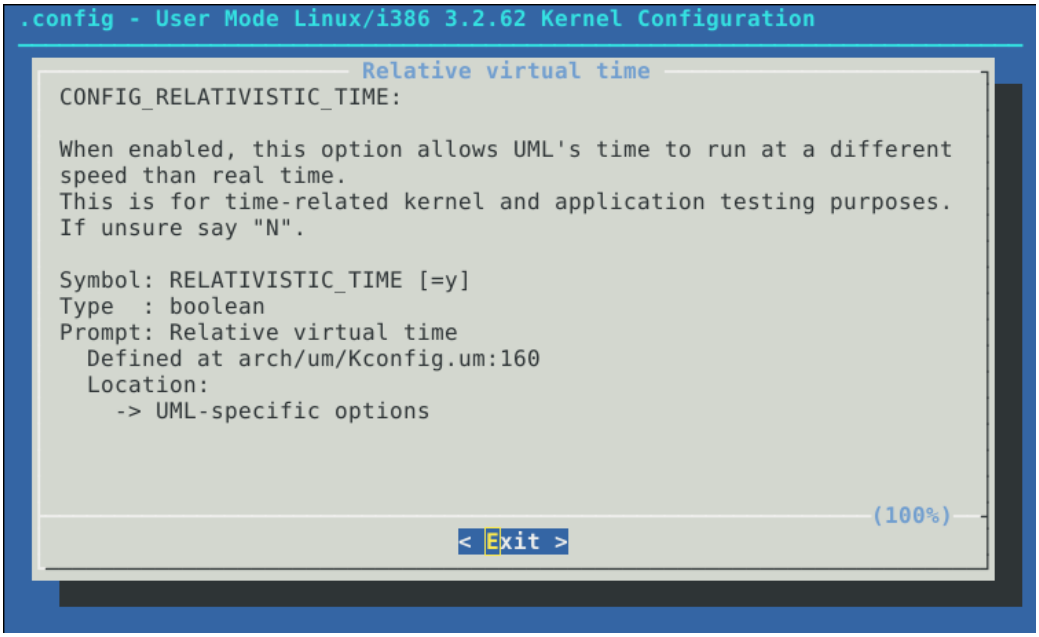

Figura 1.1: Simbolo nel menù di configurazione del *kernel*.

Il parametro `mem` serve per indicare la quantità di memoria principale che la macchina virtuale avrà a disposizione, in byte. È possibile usare abbreviazioni per esprimere valori grandi, ad esempio 64M corrispondono a 64 MiB. Il parametro `ubd0` serve per specificare il percorso del file system virtuale a cui la macchina virtuale farà riferimento. Il file system virtuale non è altro che un file formattato con un file system.
I parametri specifici per la relativizzazione sono `timefreq` e `timeconv`: essi corrispondono rispettivamente ai valori di frequenza e convergenza descritti in precedenza. La frequenza è espressa con un numero reale maggiore di zero. La convergenza è un numero intero che corrisponde al numero di secondi passati dalla *Epoch*. Ad esempio, per definire un tempo virtuale che scorre il doppio più veloce di quello reale e assume lo stesso valore di esso esattamente alle 02:00 del 1 gennaio 1970, si sceglieranno i seguenti parametri:

```
timefreq=2.0 timeconv=7200
```

Il valore di default della frequenza è 1, mentre quello della convergenza è 0.

Quando la macchina è stata avviata, è possibile recuperare le informazioni relative ai parametri di relativizzazione attraverso la lettura di due file vir-

tuali. Il file `/proc/uml_reltime/frequency` indica il valore della frequenza e il file `/proc/uml_reltime/convergence` indica il valore della convergenza. È possibile che il valore della frequenza sia leggermente diverso da quello inserito all'avvio; si tratta di un errore dovuto alla rappresentazione interna del dato, che non deve preoccupare. Il dato interno è infatti a virgola mobile e può assumere un insieme finito di valori, perciò non tutti i valori reali sono rappresentabili.
Il modo più semplice per accedere ai parametri è attraverso il comando `cat`:

```
 $ cat /proc/uml_reltime/frequency
2.0
 $ cat /proc/uml_reltime/convergence
7200
```

Una volta avviato, il tempo del sistema scorre alla velocità indicata dai parametri, ma i processi non ne saranno consapevoli: essi vedranno solamente il tempo necessario al completamento delle operazioni accorciarsi (in caso di frequenza virtuale minori di uno) o allungarsi (in caso di frequenza virtuale maggiore di uno).

1.2 Interfaccia grafica per reti virtuali

Fino a quando bisogna gestire solamente una o due macchine virtuali, la riga di comando può essere considerata un'interfaccia adeguata. Tuttavia per reti leggermente più complesse essa risulta essere poco pratica: è desiderabile poter interagire con un'interfaccia grafica. Quest'ultima corrisponde a quella di Marionnet, un programma di emulazione di reti. In effetti questo programma offre più di una semplice interfaccia, e fornisce gli strumenti per creare reti virtuali anche complesse.

Il menù superiore della finestra principale di Marionnet segue uno schema familiare ad altri programmi con interfaccia grafica: "*Project*" corrisponde alla tipica voce "*File*", in cui è possibile creare, salvare e caricare i documenti che in questo caso si identificano nei progetti. Un progetto corrisponde ad una specifica configurazione della rete, che include i vari dispositivi e lo stato dei dischi virtuali. Nella finestra è presente sulla sinistra un pannello utilizzato per la gestione dei dispositivi virtuali e sulla destra il grafo che mostra la topologia della rete. In basso sono presenti dei bottoni per sostituire la vista del grafo con altri tipi di vista. Ne esistono ad esempio una dedicata all'introduzione di difetti nella rete ed una per l'inserimento degli indirizzi IP delle schede di rete virtuali. La figura 2.3 mostra la finestra principale

del programma.
Marionnet consente di gestire vari tipi di dispositivi, che verranno brevemente illustrati.

Macchina Un normale computer con sistema operativo Linux. Questo elemento è realizzato con User-Mode Linux, quindi presenta tutti i vantaggi e le limitazioni di quest'ultimo. I programmi applicativi vengono eseguiti su questo tipo di dispositivo: essi possono usufruire della funzionalità di relativizzazione qualora sia presente nell'istanza di User-Mode Linux utilizzata.

Router Una macchina in grado di collegare tra loro più sottoreti, attraverso lo smistamento di pacchetti. I router di Marionnet sono realizzati allo stesso modo delle normali macchine, ma i file system dedicati sono più leggeri e includono *quagga* [18], una suite di routing che fornisce un'interfaccia simile a quella dei router reali Cisco. Essendo a tutti gli effetti delle macchine virtuali complete, i router possono essere relativizzati allo stesso modo dei dispositivi di tipo *Macchina*.

Hub Un oggetto che consente di creare "incroci" nei collegamenti con i cavi. Permette di aumentare il numero dei dispositivi collegati assieme; questo effetto è ottenuto tramite la ripetizione su tutte le porte di ogni segnale in arrivo . Si tratta di un dispositivo molto umile, che nel mondo reale è spesso sostituito da *switch*. Può risultare utile per la cattura di pacchetti con sorgenti diverse.

Switch Questo dispositivo consente di creare reti locali complesse. I *frame* in arrivo non vengono inviati ciecamente a tutte le porte, ma solo a quelle che possono portare i dati alla destinazione finale. Questi dispositivi sono implementati con VDE e supportano funzionalità avanzate: per maggiori informazioni su come sfruttarle si consiglia il testo *Virtual Square: Users, Programmers & Developers Guide* [9].

Rete sconosciuta di livello 2 Questo dispositivo può essere collegato a due cavi e sostanzialmente non introduce nessuna proprietà aggiuntiva nella rete. Inizialmente è stata pensata per aggiungere un elemento di casualità e imprevedibilità [25], ma grazie alla possibilità di inserire difetti nei cavi e nelle schede di rete questo elemento è completamente ridondante.

Cavo dritto e cavo incrociato I cavi sono utilizzati per connettere tra loro gli altri componenti della rete. A seconda del tipo della coppia di dispositivi da collegare è necessario utilizzare un diverso tipo di cavo. Anche i cavi sono realizzati con VDE.

Gateway e Bridge Due dispositivi virtuali realizzati con slirpVDE: essi consentono ai dispositivi virtuali di collegarsi alla rete reale, permettendo ad esempio l'accesso ad internet.

I dispositivi che possono essere relativizzati sono *Macchina* e *Router*. Quando viene aggiunta una nuova macchina, viene mostrata una finestra simile a quella di figura 1.2. La finestra per la modifica dei router, molto simile a quella per la macchina, è mostrata in figura 1.3. Si notino in particolare i due campi nella sezione *Relativization*: *Frequency* e *Convergence*. Essi corrispondono ai parametri `timefreq` e `timeconv` illustrati nella sezione 1.1. È possibile modificare il valore di questi campi attraverso i bottoni raffiguranti frecce, con un passo di un centesimo di Hertz, oppure digitando le cifre direttamente con la tastiera. È anche possibile cambiare i valori di default di questi due campi, attraverso una finestra di dialogo dedicata. L'uso di tale finestra verrà descritto successivamente.
Affinché questi parametri vengano applicati con successo alle macchine virtuali, è necessario che queste supportino la funzione di relativizzazione. Per questo motivo deve essere scelta una versione opportuna di User-Mode Linux, alla quale sia stata applicata la *patch*, nel campo denominato *Kernel*. Marionnet crea la lista dei possibili *kernel* leggendo il contenuto di un direttorio, il cui percorso di default è `/usr/share/marionnet/kernels`. I file locati in questo direttorio devono includere il prefisso "`linux-`" per essere riconosciuti.
La finestra per la modifica di dispositivi esistenti è del tutto analoga a quella per l'aggiunta dei nuovi dispositivi, con l'eccezione che i vari campi non sono inizializzati con i valori di default, ma con quelli della macchina che si vuole modificare.

Come anticipato è anche possibile modificare le impostazioni di relativizzazione a livello globale, cambiando il valore di default dei nuovi dispositivi ed opzionalmente aggiornando quelli esistenti. Per accedere alla finestra delle impostazioni globali, bisogna selezionare la voce *Options* dal menù principale in alto e successivamente selezionare *Relativization* dal menù a tendina che compare. Nella figura 1.4 si può osservare la finestra che compare. In tale finestra sono presenti i due parametri di relativizzazione ed un *checkbox*

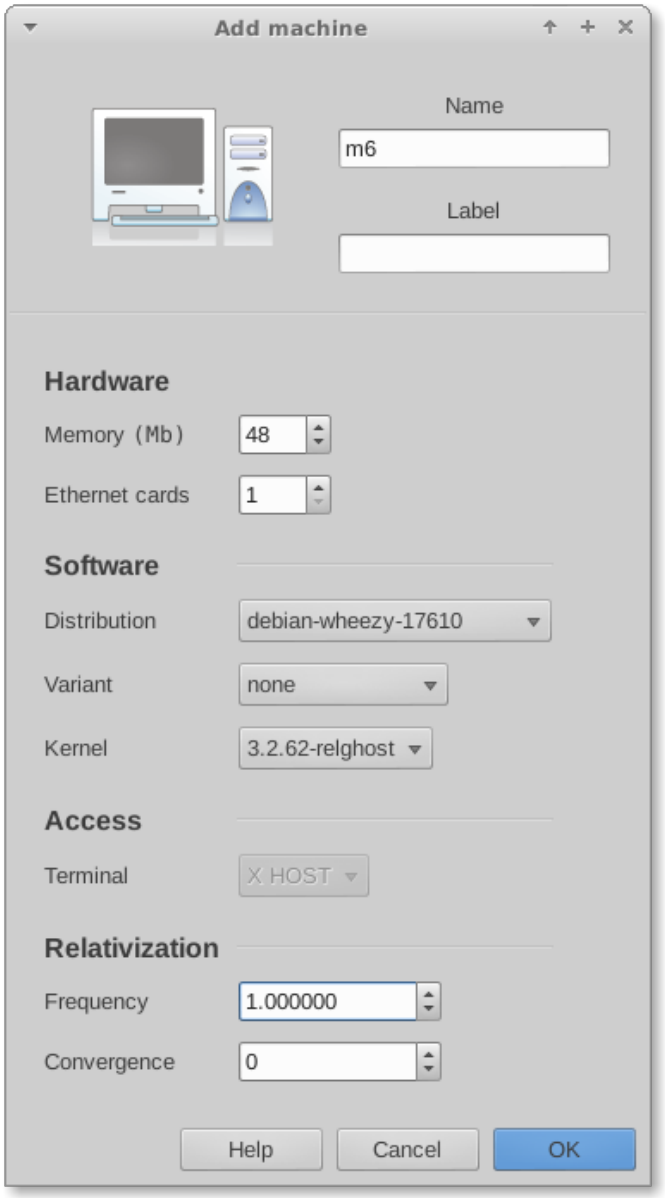

Figura 1.2: Aggiunta di una nuova macchina.

Figura 1.3: Modifica di un router.

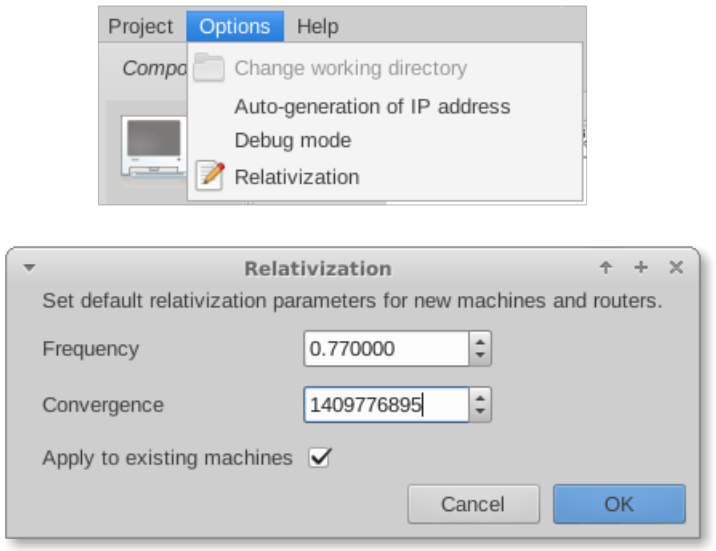

Figura 1.4: In alto, la voce *Options* del menù principale. In basso, la finestra di dialogo per le impostazioni globali di relativizzazione.

etichettato "*Apply to existing machines*". Il bottone di default è spuntato, ma la spunta si può rimuovere cliccandolo; se spuntato, tutte le macchine ed i router spenti esistenti all'interno del progetto corrente vengono modificati in modo da riflettere il cambiamento. A prescindere dallo stato del *checkbox*, il risultato della finestra, se viene premuto il bottone di conferma indicato con "*OK*", è quello del cambio dei valori di default per i nuovi dispositivi.

Dopo aver aggiunto i dispositivi al progetto, è possibile avviarli singolarmente dal menù laterale dalla voce relativa al tipo del dispositivo. Alternativamente è possibile avviare tutti i dispositivi spenti premendo il bottone "*Start all*". Per modificare le caratteristiche dei dispositivi esistenti è necessario spegnerli. Il procedimento è analogo a quello per l'accensione: si possono spegnere individualmente dal menù laterale o globalmente con il tasto "*Shutdown all*". Per ogni macchina virtuale avviata compare una finestra che presenta un terminale per controllare la relativa macchina.

Capitolo 2

Contesto tecnologico

Sono disponibili vari emulatori di macchine, e sono già presenti tutti i mezzi per creare reti virtuali e gestirle graficamente. La funzionalità di relativizzazione è stata ideata dal Prof. Renzo Davoli ed inserita all'interno del progetto *ViewOS* [13]. ViewOS fa parte di *Virtual Square* [7], una suite di virtualizzazione che comprende strumenti di varia natura. Uno degli obiettivi di Virtual Square è quello di aumentare l'interoperabilità del *software* di virtualizzazione esistente. Questo risultato è ottenuto introducendo uno strato intermedio di compatibilità: VDE (Virtual Distributed Ethernet) ne è un buon esempio: esso permette la creazione di reti virtuali emulando una rete Ethernet reale. Tutti i dispositivi che supportano lo standard IEEE 802.3 [17] sono in grado di connettersi ad una rete di questo tipo. ViewOS è invece un particolare sistema di virtualizzazione che permette di cambiare l'aspetto di alcune informazioni che il sistema operativo mostra al processo.

2.1 ViewOS

Nei tradizionali emulatori di macchine la virtualizzazione ha un effetto globale sui processi, nel senso che ogni processo virtuale fa riferimento all'ambiente della macchina virtuale che lo ospita. Perciò ogni processo ha una visione comune delle risorse e dello stato del sistema, condivisa da tutti gli altri processi della stessa macchina. ViewOS si distacca da questo tipo di approccio, creando una virtualizzazione "per processo": ogni processo può avere una percezione diversa del sistema. In questo modo i processi possono anche essere isolati tra loro e "vivere" nel proprio ambiente individuale; è così possibile creare ambienti protetti a livello di processo. La granularità di ViewOS è tuttavia ancora più fine. È infatti possibile virtualizzare soltanto alcune componenti del sistema, piuttosto che tutte. Ad esempio, è possibile

cambiare la struttura del file system (anche solo di un sottoalbero all'interno del file system) o la configurazione della rete per un solo processo, senza modificare altre informazioni a livello globale. Il concetto introdotto è quello di *virtualizzazione parziale*: solo gli aspetti di interesse del sistema vengono modificati, mantenendo inalterate le altre caratteristiche.

Esistono due implementazioni di ViewOS: *UMView* e *KMView*. UMView si appoggia alla *system call ptrace* per catturare le chiamate al sistema dei processi: queste chiamate sono sostituite con semplici *getpid* e i processi vedono gli effetti generati dal processo di controllo di UMView. Tutto il procedimento avviene in modalità utente, perciò non è richiesto nessun supporto particolare da parte del *kernel*. Per maggiori informazioni sul meccanismo di emulazione che sfrutta *ptrace*, usato anche da User-Mode Linux, si consulti la sezione 3.1. KMView sfrutta un modulo per il *kernel* per tracciare i processi attraverso *utrace*: il meccanismo è lo stesso, ma il tracciamento avviene in modalità *kernel*, con un'efficienza conseguentemente maggiore. Dato che, a parte lo spazio in cui operano, le due implementazioni si comportano allo stesso modo, ci si riferisce spesso ad esse con il termine **MView* per includerle entrambe. *MView fornisce il meccanismo di base di intercettazione delle *system call* ma, a meno che non venga specificato dall'utente, non effettua nessuna virtualizzazione delle stesse. *MView dispone infatti di moduli, nella forma di librerie dinamiche, che virtualizzano alcune *system call* in base alla funzionalità che essi offrono. L'utente può decidere quali moduli caricare: ogni modulo caricato determina esclusivamente la sostituzione delle *system call* necessarie per ottenere l'effetto desiderato. È in questo contesto che è nata la funzione di relativizzazione: il modulo *UMmisc* offre la possibilità di fare in modo che un processo veda il tempo scorrere diversamente rispetto a tutti gli altri processi.
Nonostante sia possibile "relativizzare" i singoli processi con ViewOS, risulta impossibile creare una rete di intere macchine virtuali con questo metodo. Per realizzare una rete virtuale relativistica, è opportuno poter creare macchine virtuali separate che supportino tale funzionalità. Per questo motivo è stato analizzato lo stato dell'arte del *software* di emulazione di macchine, per trovare una base su cui appoggiarsi per aggiungere la funzionalità desiderata.

2.2 Macchine virtuali

Esistono numerosi emulatori di macchine, ognuno con i propri pregi e difetti. In generale, lo stato in cui si trovano questi emulatori è abbastanza avanzato: molti emulatori sono in grado di riprodurre il comportamento delle

macchine in modo abbastanza simile da permettere l'installazione di svariati sistemi operativi, non solo Linux, e di testarli come verrebbero testati su una macchina reale. Si può fare la distinzione tra due tipi di emulatori: emulatori di macchine veri e propri, chiamati anche emulatori di CPU, ed emulatori basati sull'intercettazione di *system call*, che introducono un livello di compatibilità tra il sistema operativo reale e quello virtuale. Gli emulatori di CPU riproducono il comportamento di una specifica architettura, compreso l'insieme dei registri e delle istruzioni. Alcuni esempi di emulatori *open source* sono QEMU [2,3], KVM [21], VirtualBox [28], mentre un noto programma di emulazione proprietario è VMware [29]. Il metodo classico per l'emulazione prevede un interprete che converta dinamicamente (cioè a tempo di esecuzione della macchina virtuale) le istruzioni della macchina emulata in istruzioni che possano essere comprese dalla macchina fisica. Questa attività aggiuntiva di traduzione risulta spesso piuttosto costosa in termini di tempo e per l'emulazione è anche necessaria una quantità di memoria almeno uguale a quella della macchina emulata. Sebbene il costo sia in molti casi sostenibile nel caso dell'emulazione di una singola macchina, l'emulazione di un ampio numero di macchine può facilmente diventare eccessivo.

Dato che l'obiettivo è quello di consentire la creazione di reti di macchine virtuali relativistiche, che potenzialmente comprendono molte macchine, è stata considerata una soluzione più leggera in termini di memoria e di costo computazionale. Un'alternativa all'emulazione della CPU è infatti l'emulazione del *sistema operativo*, perché l'interazione dei processi utente con la macchina è mediata da quest'ultimo. Questo tipo di emulazione è possibile attraverso l'intercettazione delle *system call* effettuate dai processi: questo tipo di meccanismo è riscontrabile in User-Mode Linux e nel già nominato ViewOS e viene descritto in maggiore dettaglio nella sezione 3.1. User-Mode Linux è stato scelto come base per la creazione di macchine virtuali relativistiche, principalmente per la compatibilità con Marionnet, lo strumento scelto per la gestione grafica delle reti virtuali, per la relativa efficienza e per la sua natura simile a quella di ViewOS.

2.3 Emulatori di reti

Al presente sono stati realizzati vari strumenti che consentono di creare reti di macchine virtuali su di un unico calcolatore fisico. Gli scopi per cui sono stati prodotti questi programmi non sono gli stessi: alcuni di essi sono stati creati principalmente a scopo di ricerca, altri a fini didattici, altri ancora per test o debugging. Di questi programmi solo alcuni forniscono un'interfaccia

grafica. Gli esempi considerati per l'aggiunta del supporto della relativizzazione sono stati principalmente Cloonix [5] e Marionnet. È stato preso in considerazione anche GNS3 [15], ma quest'ultimo si è dimostrato particolarmente inadatto: esso presenta infatti un'interfaccia piuttosto complessa fornendo allo stesso tempo funzionalità simili a quelle degli altri emulatori di reti. Questo programma, sebbene *open source*, è comunemente usato per l'apprendimento della gestione dei sistemi Cisco, mentre l'integrazione della funzionalità di relativizzazione è orientata ad ambienti completamente liberi.

Cloonix è un ambiente di emulazione scritto in Bash e C che utilizza KVM per la creazione di macchine virtuali. I collegamenti tra le macchine sono realizzati dallo stesso Cloonix con l'uso di memoria condivisa e *futex*: i *frame* vengono posti in locazioni di memoria note a tutti i processi coinvolti nella rete; nella fase di invio e di ricezione viene utilizzata *futex*, una *system call* di Linux per la creazione di meccanismi di sincronizzazione, per bloccare e sbloccare i processi appropriati. Cloonix presenta un'interfaccia grafica che permette di aggiungere e rimuovere elementi dalla rete dinamicamente e di visualizzarne la topologia attraverso un grafo. La figura 2.1 mostra la finestra principale dell'interfaccia. L'aggiunta degli elementi avviene trascinando il relativo nome dal pannello laterale al grafo, mentre la modifica dei dispositivi esistenti è effettuata cliccando con il tasto destro del mouse sulla rappresentazione nel grafo dell'elemento da modificare. La figura 2.2 mostra l'interfaccia per la modifica delle caratteristiche delle macchine virtuali. Cloonix include funzionalità interessanti come il supporto per il collegamento di reti virtuali contenute in macchine diverse, ma l'interfaccia è stata ritenuta poco intuitiva, soprattutto per gli utenti meno esperti.

Marionnet è un programma di emulazione scritto in OCaml [23] che si appoggia a User-Mode Linux per rappresentare le macchine virtuali. Marionnet consente di creare i corrispondenti virtuali di varie tipologie di dispositivi, come hub, router e cavi Ethernet. Gli switch virtuali sono configurabili in dettaglio e supportano varie funzioni, di cui un esempio è la partizione delle rete in sottoreti virtuali (VLAN). I dispositivi che operano a livello di collegamento dei dati sono realizzati con VDE. La finestra principale di Marionnet presenta un classico menù da cui è possibile gestire alcuni aspetti globali del programma, un pannello laterale da cui è possibile gestire i singoli dispositivi, e una vista del grafo della rete.

La figura 2.3 mostra la finestra principale. L'interfaccia per l'aggiunta e la modifica dei dispositivi è minimale, ma permette di inserire i parametri fondamentali per la definizione di macchine virtuali. Le interfacce, adattate

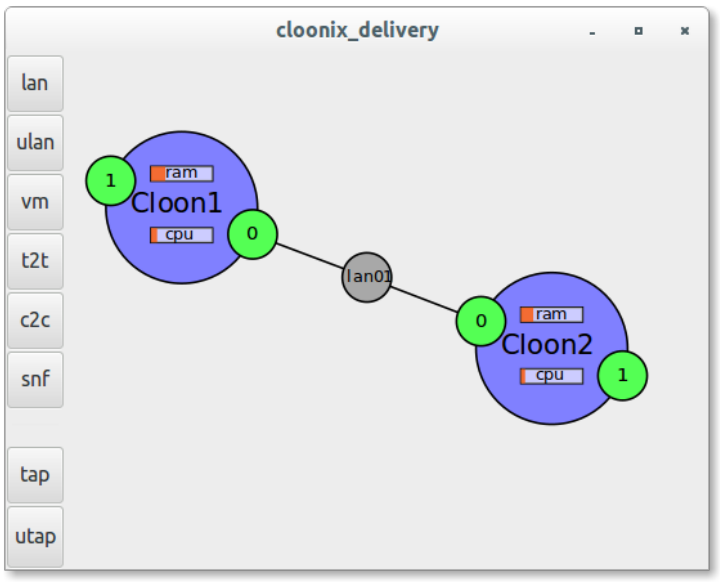

Figura 2.1: Finestra principale di Cloonix.

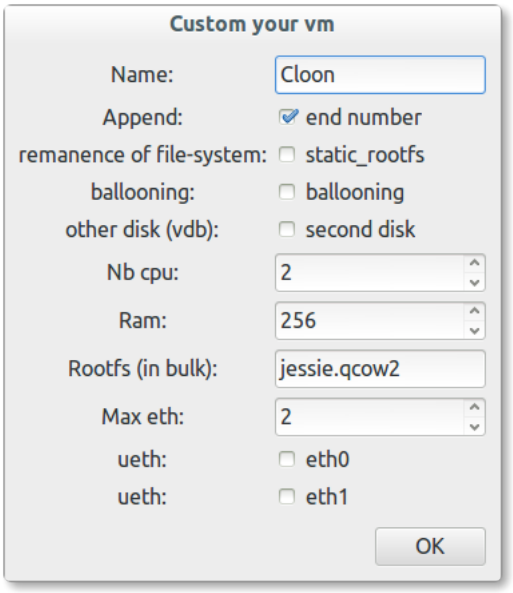

Figura 2.2: Modifica delle caratteristiche di una macchina virtuale in Cloonix.

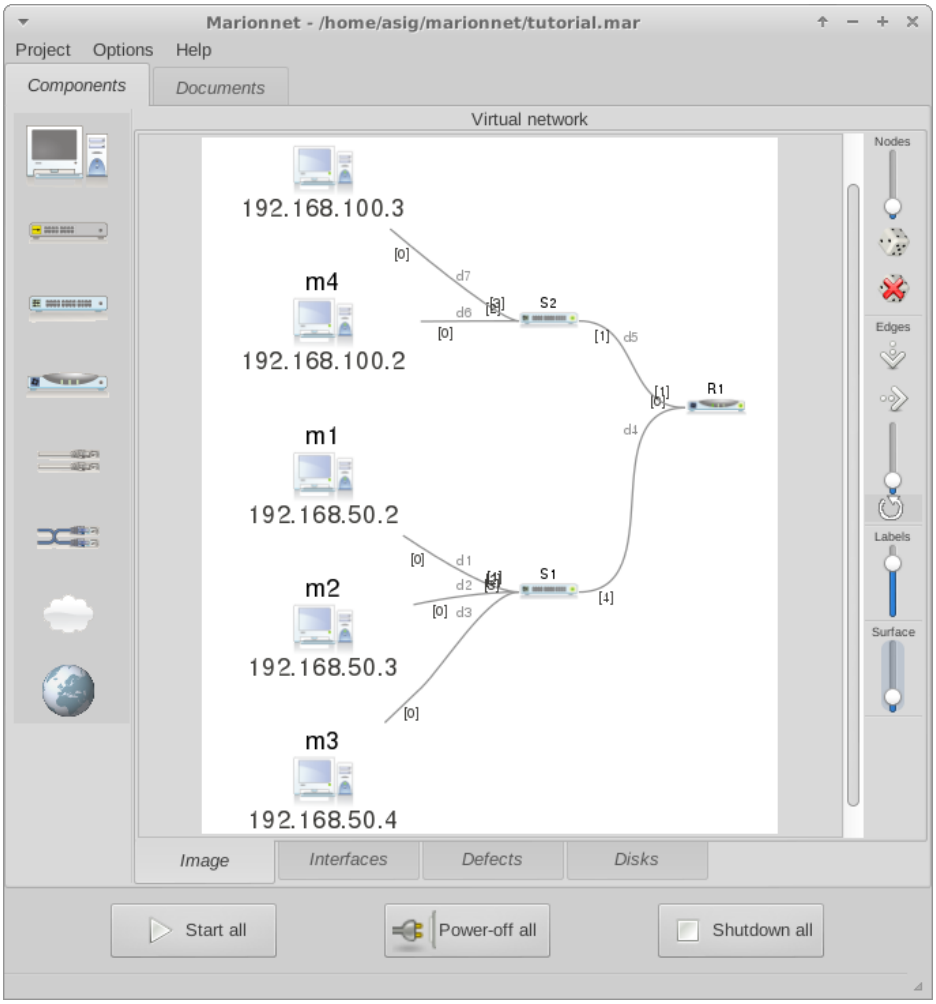

Figura 2.3: Finestra principale di Marionnet.

per supportare la relativizzazione, sono visibili nelle figure 1.2 e 1.3. In Marionnet è anche possibile, attraverso l'interfaccia grafica, introdurre difetti nei collegamenti e impostare gli indirizzi IP delle schede di rete delle macchine virtuali. Marionnet è stato scelto per l'adattamento alla relativizzazione per la sua interfaccia semplice dall'aspetto "nativo", per le sue funzionalità relativamente avanzate rispetto ad altri emulatori dovute all'utilizzo di VDE e per il fatto che il programma è attivamente mantenuto dagli autori.

Capitolo 3

Progetto

Lo scopo del progetto è quello di creare macchine relativistiche collegabili in rete, in modo da sfruttare gli effetti della relativizzazione per modificare alcune proprietà relative alla trasmissione dei dati, come la velocità di trasferimento. Come già spiegato anche nella sezione 1, con macchina relativistica si intende una macchina il cui tempo scorre ad una velocità diversa dal normale. L'idea è quella di permettere all'utente di scegliere la frequenza della macchina e sfruttare le conseguenze del diverso scorrimento del tempo per creare l'illusione di risorse migliori o peggiori rispetto a quelle effettivamente disponibili.
Il progetto è stato diviso in due fasi principali: la prima fase ha previsto la creazione di macchine virtuali relativistiche, mentre la seconda si è concentrata sulla realizzazione di uno strumento di semplice utilizzo per la creazione e gestione di reti di macchine relativistiche.
Dato che esistono varie soluzioni *open source* per la virtualizzazione di macchine, è stato ritenuto opportuno appoggiarsi ad una di esse, aggiungendo semplicemente la funzionalità desiderata. Il *software* di virtualizzazione scelto per la singola macchina è *User-Mode Linux*. Esso si distingue dai comuni emulatori di CPU, in quanto non emula caratteristiche fisiche come registri ed insieme di istruzioni specifici di un'architettura: i processi virtualizzati vengono eseguiti nativamente dal sistema operativo Linux installato sulla macchina reale. Questo particolare tipo di emulazione, basato sull'intercettazione delle *system call* effettuate dai processi, è spiegato in maggior dettaglio nella sezione 3.1. Il principale vantaggio dell'utilizzo di User-Mode Linux rispetto agli emulatori di CPU è l'efficienza: emulare le istruzioni di un'architettura è infatti un processo costoso, che non è richiesto nell'emulazione basata sull'intercettazione di *system call*. User-Mode Linux funziona sfruttando lo stesso meccanismo utilizzato da *UMview*, l'implementazione in modalità utente di *View-os*, che già implementa la funzionalità di relati-

vizzazione. Per maggiori informazioni su *View-os* si consulti la sezione 2.1. Si noti anche che molti emulatori di CPU richiedono uno speciale supporto da parte del sistema operativo e/o richiedono particolari privilegi, per aumentare l'efficienza; la virtualizzazione basata su *system call* come quella sfruttata da User-Mode Linux invece non richiede nessun supporto speciale. User-Mode Linux è inoltre già supportato da programmi di emulazione di reti come *Marionnet*. Quest'ultimo è stato scelto come base per la seconda fase del progetto: a sua volta User-Mode Linux è stato preferito rispetto alle alternative grazie al supporto già presente su Marionnet.
Marionnet è un programma con interfaccia grafica pensato innanzitutto per l'insegnamento; esso presenta un'interfaccia intuitiva e funzionalità avanzate rispetto ad altri gestori di reti virtuali, come ad esempio la possibilità di introdurre difetti nei cavi virtuali. Come altri programmi che consentono di creare e gestire reti virtuali, Marionnet si appoggia a vari componenti per l'emulazione dei dispositivi; in particolare i programmi utilizzati sono User-Mode Linux e VDE. Per maggiori informazioni su Marionnet e sulla modifica ad esso apportata si veda la sezione 3.2.

3.1 User-Mode Linux

L'emulatore scelto per realizzare le macchine virtuali relativistiche è User-Mode Linux. Questo programma (che è in realtà un adattamento e non un programma a sé stante), scritto in C da Jeff Dike, consente di emulare un sistema operativo Linux, detto "*guest*", sul sistema operativo Linux che lo ospita, detto "*host*".

User-Mode Linux differisce dall'approccio degli emulatori di CPU, che cercano di riprodurre il comportamento di una specifica architettura: esso non emula l'intera macchina, ma permette di eseguire i processi utente come se si trovassero su una macchina diversa da quella reale. Dato che l'obiettivo finale dell'emulazione sono i processi utente, non è necessario emulare tutte le caratteristiche fisiche della macchina, ma è sufficiente emularne il sistema operativo. Il codice di User-Mode Linux è infatti quello di Linux [32]: nell'albero dei sorgenti esiste in particolare una specifica architettura riservata a User-Mode Linux. Se il *kernel* viene compilato per questa architettura, esso viene adattato in modo da poter essere eseguito come un normale processo utente. Tale processo, chiamato anche *kernel thread*, rappresenta il *kernel* virtuale della macchina emulata. I processi utente virtuali sono eseguiti come *thread* reali sull'*host*. Questi *thread* sono nativi, nel senso che non viene effettuato nessun tipo di traduzione delle istruzioni eseguite.

Il classico mezzo di dialogo tra processi utente e *kernel* è costituito dalle *system call*. La percezione del mondo esterno di un processo dipende quindi totalmente dagli effetti delle *system call* effettuate. Questa situazione è sfruttata da User-Mode Linux nel seguente modo: il *kernel thread* cattura le *system call* dei processi emulati e le gestisce esso stesso, sostituendosi al sistema operativo *host*. L'intercettazione avviene indirettamente tramite una particolare *system call* chiamata *ptrace*, originariamente pensata per fornire una comodo mezzo di *debugging*. Questa *system call* consente al processo chiamante di monitorare le chiamate al sistema dei processi figli. Essa fornisce la possibilità di interrompere il normale flusso dell'esecuzione prima e/o dopo della gestione da parte del sistema operativo di una *system call* effettuata dal processo monitorato. User-Mode Linux sfrutta *ptrace* in questo modo:

1. Quando uno dei processi figli (del *kernel thread*, quindi i processi utente da emulare) effettua una *system call*, prima che essa venga gestita dal sistema *host*, il controllo passa al *kernel thread*

2. Il *kernel thread* sostituisce il numero della chiamata al sistema con quello relativo alla chiamata *getpid*. Questa chiamata non ha nessun effetto sul sistema *host*, poiché viene semplicemente restituito il valore del numero identificativo del processo.

3. Il sistema *host* gestisce la *system call getpid*.

4. Prima che il risultato venga restituito al processo da emulare, il *kernel thread* riprende il controllo scartando il risultato di *getpid* ed effettuando la propria gestione della *system call* iniziale, secondo quanto previsto dal *kernel* da emulare.

5. Il controllo viene restituito al processo emulato, il quale può osservare il risultato della gestione da parte del *kernel* virtuale, ma non quello da parte del sistema *host*.

Il meccanismo appena descritto è schematizzato in figura 3.1.

Grazie a questa tecnica i processi in *user mode* non hanno accesso al *kernel* reale, ma fanno riferimento a quello virtuale. I processi in *kernel mode* vengono rilasciati dal tracciamento di *ptrace* e interagiscono direttamente con il *kernel* reale. In questo senso il *kernel* reale viene trattato da User-Mode Linux come se fosse l'*hardware*: solo i processi in *kernel mode* possono accedervi direttamente senza la mediazione del *kernel*.

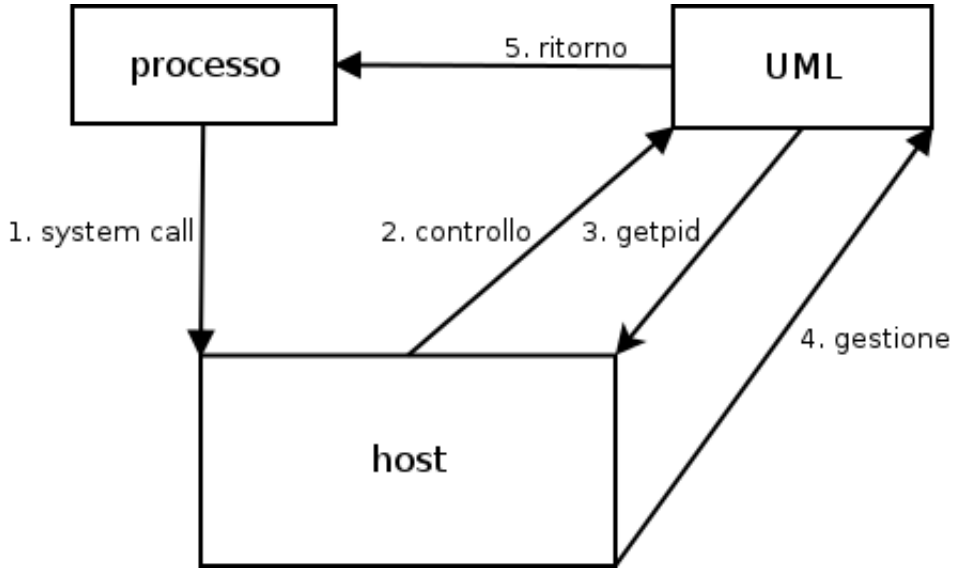

Figura 3.1: Schema di emulazione con ptrace.

Il principio del funzionamento di User-Mode Linux spiega perché esso emuli necessariamente un sistema Linux su un altro sistema Linux. Non è possibile usare un altro sistema *guest* perché il codice di User-Mode Linux è strettamente connesso a quello di Linux (ne è anzi parte); non è possibile usare un altro sistema *host* perché i processi virtualizzati sono processi reali del sistema. In altre parole User-Mode Linux è un *port*, un adattamento, da Linux a Linux. Questo è il principale limite di User-Mode Linux: non è possibile emulare altri sistemi operativi. D'altra parte però questa caratteristica conferisce all'emulazione una maggiore efficienza rispetto agli emulatori di *hardware*: è assente infatti l'*overhead* necessario alla traduzione di ogni istruzione. Nelle prossime sezioni verrà mostrato come il codice di User-Mode Linux è stato modificato per aggiungere la funzionalità di relativizzazione.

3.1.1 Caratteristiche comuni

Il tempo virtuale può scorrere in diversi modi: può essere più lento, più veloce, può trovarsi spostato nel passato o avanti nel futuro rispetto a quello reale. Il modo più naturale e sensato per dare una misura al comportamento del tempo virtuale è metterlo in confronto con quello reale. Da questo confronto sono stati individuati due valori: *frequenza* e *convergenza*. La frequenza indica la velocità del tempo virtuale: quanti secondi virtuali passano in un secondo reale. La convergenza è un'istante di tempo che serve ad esprimere lo spostamento del tempo virtuale rispetto a quello reale: è il punto

in cui i due tempi si incontrano ed assumono lo stesso valore. Per la rappresentazione di un istante di tempo è stato usato il sistema dei *timestamp POSIX*: ogni istante è rappresentato dal numero di secondi passati dal 1 gennaio 1970.
Questa definizione si riflette nella struttura `reltime` definita in `arch/um/kernel/reltime.c`:

```
struct uml_reltime_t {
   long long convergence;
   long double frequency;
};

struct uml_reltime_t reltime = {
   .convergence= 0,
   .frequency= 1
};
```

Il codice precedente mostra anche i valori di default scelti per i parametri: 1 per la frequenza e 0 per la convergenza. Il valore della frequenza posto ad uno determina un tempo identico a quello reale, perciò il valore della convergenza non è rilevante, perché i due tempi coincidono su tutti i punti.
L'istante attuale virtuale t_v viene calcolato in modo da rispettare la seguente equazione:

$$t_v = (t_r - c) * f + c$$

in cui t_r è l'istante di tempo reale, mentre f e c sono rispettivamente i valori di frequenza e convergenza.
L'idea è quella di riuscire ad ottenere lo spostamento dell'ora virtuale rispetto al punto di convergenza. Infatti $t_r - c$ rappresenta la distanza tra l'ora attuale reale ed il punto di convergenza: moltiplicando questo valore per quello della frequenza si ottiene la distanza tra l'ora virtuale attuale ed il punto di convergenza. Ottenuta la distanza dal punto di convergenza, è sufficiente sommarla al valore di quest'ultimo per ottenere l'ora virtuale. Questo processo è illustrato graficamente in figura 3.2.

I parametri inviati da linea di comando vengono letti da funzioni definite nel file `arch/um/kernel/um_arch.c` insieme ad altre funzioni di estrazione dei parametri. Le funzioni aggiunte sono state chiamate `uml_conv_setup` e `uml_freq_setup`. La frequenza è di tipo `long double`, ma Linux normalmente non gestisce valori di questo tipo, perciò la funzione `sscanf` già presente nel *kernel* non consente di convertire una stringa in un valore di tipo `long double`. Per questo motivo è anche definita anche una sempli-

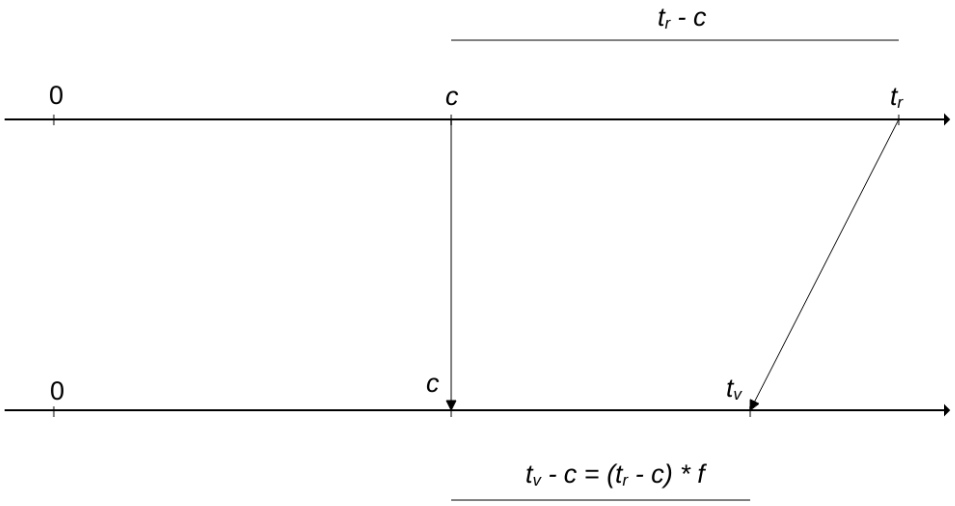

Figura 3.2: Il tempo può essere rappresentato con una retta orientata. In figura appaiono due istanze della retta, per meglio raffigurare gli effetti del calcolo. Il punto 0 corrisponde alla *Epoch*. Il punto di convergenza è chiamato c. La distanza tra l'istante di tempo virtuale t_v ed il punto di convergenza è ottenuta dalla moltiplicazione tra frequenza f e distanza tra l'istante di tempo reale t_r ed il punto di convergenza. A questo risultato, $(t_v - c)$, è sufficiente sommare c per ottenere l'instante di tempo virtuale t_v.

ce funzione chiamata `atold`: questa definizione si trova nel file `arch/um/kernel/reltime.c`.

Precisione

Anche senza alcuna modifica, User-Mode Linux è piuttosto impreciso nella gestione del tempo. Questa imprecisione è dovuta al fatto che viene mantenuto un orologio virtuale realizzato tramite un *timer* che scatta ad intervalli regolari. L'ampiezza di questi intervalli è configurabile in fase di compilazione attraverso il simbolo `CONFIG_HZ`, che indica la frequenza degli aggiornamenti. Di default il valore associato a questo simbolo è 100, che significa che l'intervallo tra un aggiornamento e l'altro dell'orologio virtuale è di un centesimo di secondo. La precisione dell'orologio nella versione non modificata di User-Mode Linux perciò non può superare questo valore. La modifica apportata però introduce un ulteriore fattore di imprecisione, perché la frequenza virtuale è espressa con un `long double`, quindi un numero *floating point* con precisione limitata: dato che la distanza dell'ora reale dal punto di convergenza viene moltiplicata per la frequenza, l'ora virtuale calcolata risente dell'errore dovuto alla rappresentazione tramite *floating point*. La precisione limitata dei numeri a virgola mobile è dovuta al fatto che l'insieme di essi è finito, ma sono solitamente usati (come anche nel caso della modifica a User-Mode Linux) per rappresentare valori reali. L'insieme dei numeri reali è infinito, perciò infiniti valori verranno rappresentati con lo stesso numero. Inoltre la distribuzione dei numeri *floating point* rispetto a quella dei numeri reali non è uniforme, perciò la precisione varia in base al valore del numero da rappresentare. Questo concetto è mostrato nella figura 3.3.

Nonostante l'imprecisione dei numeri *floating point*, la soluzione è stata ritenuta accettabile per vari motivi. In primo luogo, i valori di tipo `long double` sono piuttosto precisi in quanto, almeno per l'architettura x86, vengono utilizzati 80 bit per la rappresentazione [31]. La distribuzione non uniforme, che si concentra nei numeri vicini allo zero, implica che la precisione nel calcolo dell'ora virtuale è maggiore se questa è più vicina all'ora reale, perché la frequenza deve essere moltiplicata per la differenza tra ora reale e convergenza. La scelta di un punto di convergenza vicino al presente riduce quindi gli effetti dell'errore introdotto dalla rappresentazione *floating point*, così come valori non eccessivamente alti della frequenza. Un dato importante è che il problema dell'imprecisione dei *floating point* normalmente non si pone, in quanto per intervalli di aggiornamento dell'orologio di un centesimo di secondo l'errore non è significativo se la moltiplicazione tra frequenza e distanza dal punto di convergenza non causa uno spostamento dell'ordine di 10^7 nanosecondi. Nel caso di frequenza virtuale sufficientemente bassa la

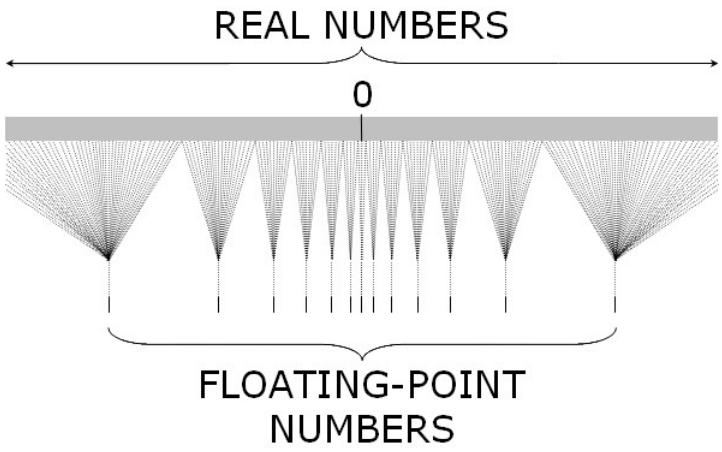

Figura 3.3: Distribuzione dei numeri *floating point* rispetto ai numeri reali [20].

precisione dell'ora virtuale potrebbe aumentare, perché la frequenza di aggiornamento dell'orologio virtuale dal punto di vista dei processi virtualizzati appare essere inversamente proporzionale al parametro di frequenza virtuale (il valore utilizzato nel calcolo). Tuttavia le conseguenze di questa proprietà non sono ancora state studiate approfonditamente.

Sono state esplorate due alternative implementative. Nonostante condividano alcuni comportamenti e definizioni, come quelli già descritti, esse presentano alcune differenze fondamentali. Queste alternative sono state entrambe realizzate, poiché la seconda non era nota durante la fase iniziale della progettazione.

3.1.2 Modifica a livello di gestione di *system call*

Per fare in modo che i processi emulati abbiano una visione distorta del tempo, bisogna in qualche modo alterare la loro percezione del mondo esterno. Come già detto questa percezione avviene esclusivamente tramite *system call*, perciò è possibile alterarla attraverso una modifica nella fase di gestione delle chiamate al sistema. Con riferimento alla figura 3.1, questa fase corrisponde al momento successivo a *gestione* ed immediatamente prima di *ritorno*. Questo approccio è lo stesso adottato da *UMview*, il quale fornisce un modulo chiamato *UMmisc* che offre esattamente la funzionalità di relativizzazione che si vuole portare su User-Mode Linux.

Nel file `arch/um/kernel/reltime.c` si trova il cuore della prima versione della *patch* che è stata scritta per introdurre la relativizzazione in User-Mode Linux. In questo file è infatti presente il codice che ridefinisce la gestione delle *system call*.
Per ottenere la relativizzazione del sistema è sufficiente modificare solo i gestori di *system call* che servono per inserire nuovi valori di tempo nel sistema oppure per leggere valori di tempo da quest'ultimo. Sarebbe stato possibile modificare direttamente i gestori di interesse, ma una tale soluzione sarebbe difficile da mantenere con nuove versioni di Linux. Si è scelto invece di creare delle funzioni "*wrapper*" per i gestori originali: queste funzioni modificano i valori che riguardano in qualche modo il tempo del sistema e contengono al loro interno una chiamata alla funzione originale. In questo modo il resto del *kernel* è a conoscenza del reale valore del tempo, ma questo valore non coincide con quello presentato ai processi da virtualizzare.
In Linux esiste una tabella dei gestori delle chiamate al sistema, chiamata `sys_call_table`. Questa tabella è di fatto un *array* che mantiene i puntatori a tutte le funzioni di gestione. Ad ogni indice della tabella corrisponde il numero di una *system call*. Per modificare il comportamento dei gestori si è fatto in modo che, a tempo di esecuzione, venga cambiato il valore di alcuni puntatori di `sys_call_table`: i puntatori alle funzioni originali sono sostituiti dai puntatori alle corrispondenti funzioni "*wrapper*". La ridefinizione del singolo gestore è osservabile nella macro `REDEFINE_SYSCALL`. Il momento preciso in cui avviene la ridefinizione è osservabile nel file `arch/um/kernel/um\_arch.c`, quando viene chiamata la funzione `redefine_syscalls`.

3.1.3 Modifica a livello di interfacciamento con l'*host*

La seconda alternativa implementativa che è stata analizzata prevede la relativizzazione ad un livello più basso. Invece che cambiare la percezione del tempo dei processi emulati, è infatti possibile fare in modo che User-Mode Linux modifichi la propria percezione: per transizione anche i processi emulati che fanno riferimento al *kernel thread* saranno portati ad avere una visione distorta del tempo. In figura 3.4 è possibile osservare una rappresentazione schematica dei due livelli a cui le due alternative di implementazione operano: il livello di questa implementazione è quello etichettato *Alternativa 2*, mentre quello che opera sulla gestione delle *system call* è denominato *Alternativa 1*.
User-Mode Linux implementa un orologio virtuale il cui tempo di aggiornamento è realizzato sfruttando un *interval timer* fornito dal sistema *host*. Il valore del tempo corrente viene letto regolarmente effettuando una *gettimeofday*. Per ottenere l'effetto desiderato è quindi sufficiente modificare

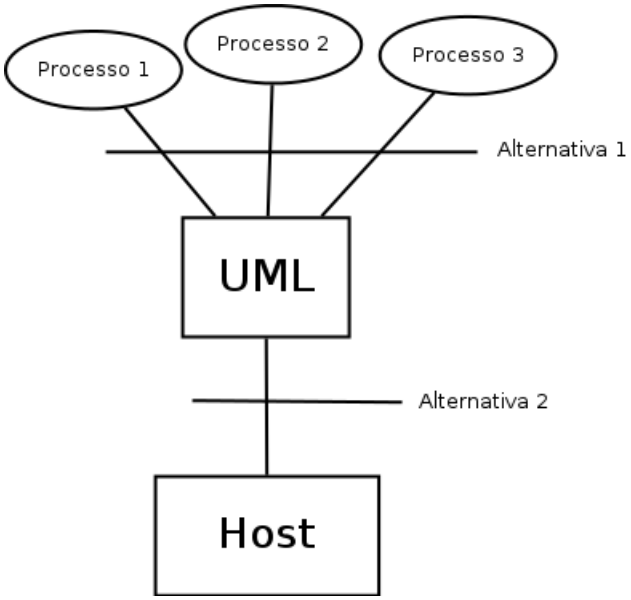

Figura 3.4: Diversi livelli a cui operano le due alternative.

questo valore. La lettura e modifica del valore avviene all'interno del file `arch/um/os-Linux/time.c` nella funzione `os_nsecs`. Quest'ultima restituisce l'ora attuale espressa come il numero di nanosecondi passati dalla *Epoch*. L'unico adattamento necessario, oltre alla conversione del risultato di *gettimeofday*, consiste nel moltiplicare per 10^9 il valore della convergenza, perché altrimenti sarebbe espresso in secondi.

3.1.4 Confronto tra le due alternative di implementazione

Sebbene entrambe le alternative presentate siano corrette è stata preferita la seconda, cioè quella che prevede la modifica a livello di interfacciamento con l'*host*. Un obiettivo nella relativizzazione della macchina virtuale relativistica è quello di creare un sistema coerente: tutti i processi della stessa macchina devono veder scorrere il tempo allo stesso modo, qualunque sia la chiamata al sistema effettuata. Nella prima alternativa, presentata nella sezione 3.1.2, è quindi necessario modificare il comportamento di ogni *system call* che in qualche modo interagisce con il tempo. Esistono molte *system call* con questa proprietà (si pensi ad esempio ai vari *timer* come in *select* e *poll*), alcune di esse meno utilizzate di altre. Inoltre in futuro potrebbero essere aggiunte nuove chiamate che includono un'interazione con il tempo mantenuto dal sistema. Per questi motivi il risultante codice cresce in dimensione

e diventa più difficile da mantenere con l'evolversi di Linux. L'alternativa presentata nella sezione 3.1.3 concentra invece in un solo punto la modifica dei valori del tempo: non solo il codice è più breve, ma è anche molto più facile da mantenere e compatibile con l'introduzione di nuove *system call*. La mantenibilità è stato perciò il motivo principale che ha portato alla scelta dell'alternativa che prevede la modifica a livello di interfacciamento con l'*host*.

3.1.5 Esposizione dei parametri nel file system virtuale *proc*

È stata fornita all'utente la possibilità di controllare i valori dei parametri di relativizzazione a tempo di esecuzione. Questa funzionalità è utile se l'utente che opera con una macchina virtuale già avviata non ricorda o non conosce i parametri in questione. A questo scopo si è scelto di creare due file virtuali all'interno del file system *proc*. Questa soluzione è stata adottata perché relativamente semplice da usare dal punto di vista dell'utente, oltre al fatto che il *kernel* espone già molti altri valori e parametri in questo modo. L'inserimento nel file system avviene durante l'avvio di User-Mode Linux; il punto nella sequenza di avvio è osservabile nel file `arch/um/kernel/reltime.c` in cui è utilizzata la macro `__initcall`. La gestione del file virtuali in *proc* avviene attraverso un'apposita interfaccia di programmazione interna al *kernel*. La documentazione relativa a questa interfaccia consiste soprattutto nel codice che la definisce, nei sorgenti di Linux. Alcuni elementi sono anche descritti nel libro *Linux Device Drivers* [6]. Contestualmente alla creazione dei file virtuali avviene la definizione dei metodi di accesso: i file sono stati definiti come di sola lettura; l'output consiste nella conversione in stringa del valore corrispondente (frequenza o convergenza).

3.2 Marionnet

L'emulatore di reti scelto per il progetto è Marionnet. Questo programma, scritto in OCaml da Jean-Vincent Loddo e Luca Saiu, consente di creare reti virtuali anche complesse e di modificarle dinamicamente, ovvero anche quando i dispositivi virtuali sono già stati avviati. Il campo applicativo principale di Marionnet è quello dell'insegnamento. Il programma è stato infatti pensato per essere utilizzato da studenti di informatica nei corsi di reti. Nonostante il suo scopo principalmente didattico, Marionnet è una suite di emulazione piuttosto completa ed è perfettamente utilizzabile anche nel contesto di test di configurazioni di rete, di programmi e di protocolli. Il

programma fornisce un'interfaccia grafica per facilitare la gestione dei vari componenti. Attraverso l'interfaccia è possibile creare e salvare un progetto. Un progetto rappresenta una specifica configurazione di rete. Anche lo stato dei file system virtuali viene salvato permanentemente insieme alla topologia della rete, attraverso dei file "*copy-on-write*", cioè che memorizzano incrementalmente le differenze tra il nuovo stato del file system e quello predefinito. L'utente può aggiungere vari dispositivi, alcuni dei quali operano al livello 2 ed altri anche a livelli superiori (con riferimento alla pila di protocolli del modello OSI [19]). Ad esempio *switch* e *hub* gestiscono *frame* di dati ed operano solo al livello 2. Macchine e *router* sono in grado di reindirizzare *pacchetti* di dati e collegare tra loro varie reti, ma anche di gestire protocolli che si trovano a livelli di astrazione più alti, da quello di trasporto (con protocolli come TCP e UDP) a quello applicativo (con protocolli come HTTP). In teoria una generica macchina con molte interfacce di rete può essere utilizzata per lo smistamento dei pacchetti, ma in pratica nella realtà si utilizzano macchine specializzate chiamate router. Marionnet effettua la distinzione a livello concettuale tra macchina e router, per meglio riflettere la realtà, anche se in pratica essi hanno le stesse proprietà.

I due elementi di maggior interesse sono la *Macchina* ed il *Router* (con riferimento ai nomi utilizzati da Marionnet, "*Machine*" e "*Router*"), poiché si collocano ad un livello più alto rispetto agli altri dispositivi. Le macchine ed i router virtuali, essendo realizzati con User-Mode Linux, ospitano necessariamente un sistema operativo Linux. La modifica per aggiungere la funzionalità di relativizzazione deve essere applicata a questi due tipi di dispositivo. Tale funzionalità deve essere supportata dai *kernel* virtuali. La ragione di ciò è che Marionnet fondamentalmente sfrutta vari programmi separati e li mette insieme per fornire uno strumento centrale per la creazione di reti virtuali. User-Mode Linux è uno di questi componenti: i file binari eseguibili di User-Mode Linux devono essere posti in uno specifico percorso (di default `/usr/share/marionnet/kernels` anche se la locazione è configurabile). Il contenuto del direttorio viene letto dal programma per generare una lista dei possibili *kernel* virtuali.

L'utente può selezionare graficamente il *kernel* virtuale e i parametri da passare a quest'ultimo. Tale scelta può avvenire in due momenti: quando si aggiunge un nuovo dispositivo (macchina o router) oppure quando si decide di modificare le caratteristiche di un dispositivo esistente spento. Quando l'utente effettua una di queste due azioni, viene generata e mostrata una finestra che permette, attraverso vari *widget*, di scegliere le caratteristiche

desiderate. Uno degli obiettivi è stato quindi la modifica di queste finestre, per fare in modo che esse conferiscano all'utente anche la possibilità di cambiare in maniera comoda i parametri per la relativizzazione.

La funzionalità di relativizzazione può trovare impiego per test di protocolli di sincronizzazione degli orologi (ad esempio il protocollo NTP, Network Time Protocol) e simili. In questi casi la possibilità di modificare i parametri individualmente per ogni macchina è desiderabile, in quanto il loro valore dovrà essere diverso per ottenere una rete di macchine inizialmente non sincronizzate, che operano a frequenze diverse. Tuttavia è stato previsto un utilizzo più frequente della relativizzazione per ampliare o restringere l'ampiezza di banda della rete. In questo caso d'uso è desiderabile avere una frequenza coerente tra le varie macchine, perciò è stato ritenuto utile fornire un modo per cambiare facilmente il valore dei parametri a livello globale. Per questo motivo è stato definito l'obiettivo di creare un'interfaccia per semplificare la gestione dei parametri di relativizzazione a livello dell'intero progetto (progetto inteso come rete virtuale in Marionnet, incluso lo stato dei file system virtuali).

3.2.1 Linguaggio e librerie

In seguito verrà data una spiegazione della modifica apportata al programma per l'aggiunta degli elementi grafici desiderati. Per poter comprendere il codice è necessaria una conoscenza del linguaggio *OCaml* e delle librerie *lablgtk*. In questo documento verrano descritti solamente alcuni aspetti fondamentali; per informazioni più approfondite si consiglia di consultare la documentazione online per OCaml [23], per *lablgtk* [14] e per *ocaml-gettext* [22].

OCaml

Abbreviazione per *Objective Caml*, è un linguaggio funzionale che supporta anche il paradigma *object-oriented* e imperativo. Questa flessibilità del linguaggio è ampiamente sfruttata dagli autori di Marionnet, in quanto molti elementi del programma sono codificati con degli oggetti, come ad esempio i vari dispositivi della rete. Molto utilizzato in Marionnet è anche il meccanismo dei *label* per il passaggio dei parametri alle funzioni: gli argomenti vengono "etichettati", in modo che siano identificabili attraverso il proprio nome piuttosto che attraverso la propria posizione. Un meccanismo simile è presente anche nel linguaggio Python [33]. Gli argomenti etichettati possono anche essere dichiarati come opzionali; in questo caso è possibile definire un

valore di default che viene usato nel caso in cui l'argomento non venga passato. Gli autori fanno un ampio uso dei *functor*, costrutti che permettono di definire moduli parametrici, e del passaggio di "funzioni parziali", cioè funzioni delle quali sono già stati definiti alcuni argomenti. Si ricorda inoltre che nel linguaggio utilizzato ogni file con estensione ".ml" costituisce un modulo, ma i moduli possono anche essere annidati, è cioè possibile definire un modulo internamente ad un altro. In seguito i termini "modulo" e "file" (quando il file ha estensione ".ml") verrano usati liberamente, in quanto non c'è una distinzione concettuale tra i due. Inoltre i moduli annidati verranno indicati così come lo sono nel codice oppure indicando il nome del modulo interno ed il nome del file che definisce il modulo esterno. Ad esempio le espressioni "`Gui_bricks.Dialog_run`" e "il modulo `Dialog_run` contenuto nel file `gui_bricks.ml`" denotano lo stesso modulo.
OCaml è un linguaggio potente e complesso; per maggiori informazioni si consiglia di fare riferimento al già citato manuale.

lablgtk

Questo insieme di librerie è un adattamento per OCaml del *framework GTK+* [4] nelle sue versioni 1.2 e 2.0. GTK+ fornisce gli strumenti per la creazione di interfacce grafiche native su sistemi a finestre. In particolare in lablgtk è possibile creare oggetti che rappresentano le finestre dell'applicazione, per poi collegare a questi altri oggetti che rappresentano i vari elementi contenuti nella finestra. Gli elementi sono posizionabili in speciali contenitori che dispongono il proprio contenuto secondo determinati parametri: ad esempio un *vbox*, o *vertical box*, dispone gli elementi contenuti in esso uno sopra all'altro secondo un ordinamento a colonna, mentre un *hbox*, o *horizontal box*, dispone gli elementi contenuti uno a fianco a l'altro secondo un ordinamento a riga. La creazione di un nuovo elemento prevede sempre un parametro etichettato con `pack`, che definisce il contenitore e il modo in cui l'elemento si comporta in relazione ad esso.
L'input dell'utente viene gestito attraverso un meccanismo basato su *signal* e *callback*. I *signal* sono segnali che vengono generati in seguito a *click* e movimenti del *mouse* oppure pressione dei tasti della testiera da parte dell'utente. I *callback* sono azioni, esprimibili tramite funzioni, che vengono eseguite in risposta ad un segnale. Per rispondere all'input dell'utente è quindi sufficiente collegare funzioni *callback* ai segnali desiderati. Nel caso di Marionnet ad esempio è già stata definita dagli autori una funzione che permette di creare una finestra di dialogo e collegare un'azione alla pressione del tasto "*OK*" (tasto che indica l'accettazione dei dati immessi da parte dell'utente): si tratta di `Dialog_run.ok_or_cancel`, contenuta nel modulo

`gui_bricks.ml`.

ocaml-gettext
In Marionnet è già presente il supporto per l'internazionalizzazione, perciò il codice aggiunto è stato adattato in modo da sfruttare il supporto già esistente. Il *framework* usato a questo scopo è chiamato *ocaml-gettext*: esso include due implementazioni di cui una è un adattamento ad OCaml della libreria C *gettext* [12] e l'altra è un'implementazione completamente riscritta in OCaml secondo il paradigma funzionale. Per quanto riguarda il codice aggiunto, l'unica funzione utilizzata di questa libreria è `s_` : essa accetta in input una stringa, per la quale viene creato il supporto per semplificarne la traduzione attraverso appositi file. Tutte le stringhe aggiunte nel codice sono perciò in inglese e passate in input a `s_` .

3.2.2 Componenti

Marionnet fornisce solamente un'interfaccia che permette all'utente di gestire comodamente una rete virtuale. La rete vera e propria è composta fondamentalmente sfruttando due elementi: User-Mode Linux e VDE. In particolare, con VDE vengono realizzati i dispositivi di livello 2 e la connessione tra i vari elementi della rete, mentre le macchine virtuali (macchine e router) sono realizzate con User-Mode Linux. Dato che la relativizzazione vera e propria avviene a livello di User-Mode Linux, il codice di Marionnet è stato adattato per poter passare i corretti parametri all'avvio delle macchine virtuali. Il codice per la gestione interna a Marionnet dei vari componenti è principalmente situato nel file `simulation_level.ml`. In questo file viene generata la stringa che contiene tutti i parametri della macchina User-Mode Linux da avviare: la classe `uml_process` rappresenta infatti un processo di User-Mode Linux, cioè una macchina o un router virtuale. La figura 3.5 mostra la divisione a strati di Marionnet, da User-Mode Linux all'interfaccia grafica.

3.2.3 Aggiunta e modifica dei dispositivi virtuali

La finestra che permette di cambiare i parametri delle macchine virtuali viene generata dinamicamente quando l'utente seleziona $Machine \rightarrow Add$ oppure $Machine \rightarrow Modify \rightarrow \langle nome\ macchina \rangle$ dal menù laterale. Il sottoalbero *Machine* del menù laterale viene prodotto da un *functor* chiamato `Make_menus` all'interno del file `machine.ml`. All'interno del *functor* in questione sono definiti vari moduli, uno per ogni voce del sottomenù. I moduli

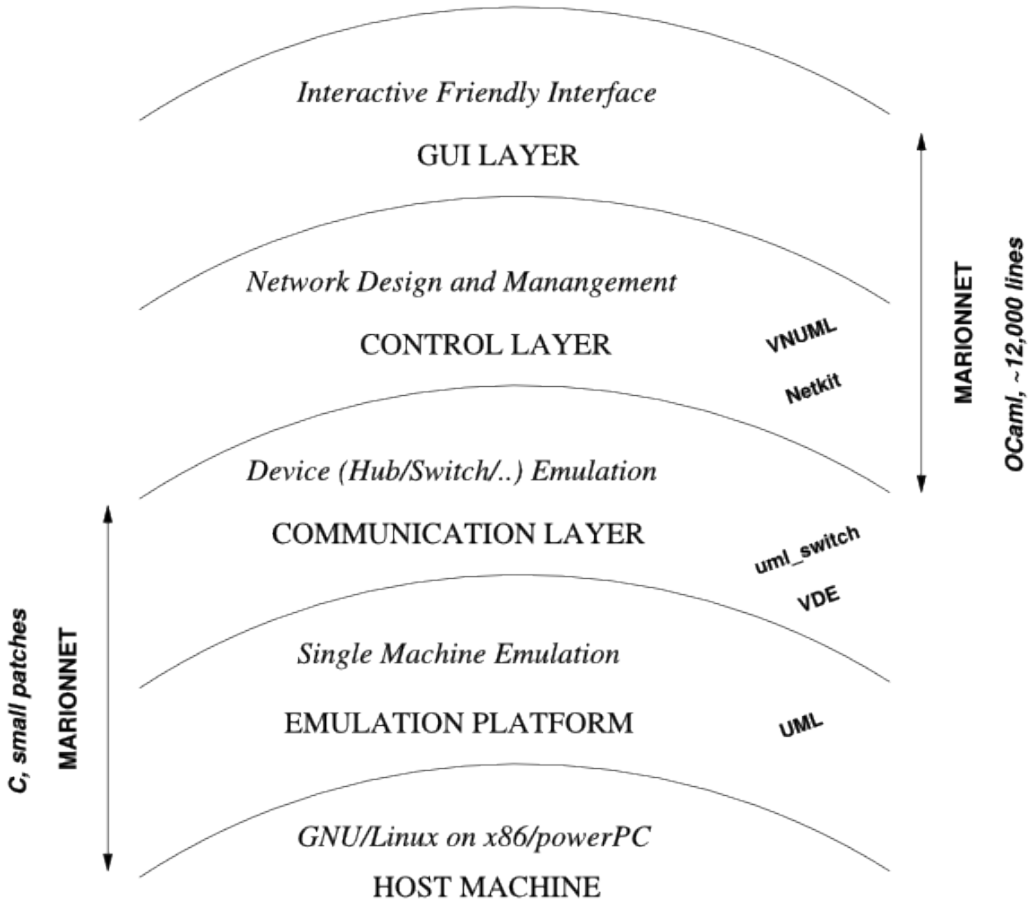

Figura 3.5: Divisione in strati di Marionnet [25]. La rete vera e propria è costruita con User-Mode Linux e VDE.

di interesse per l'inserimento dei parametri delle macchine virtuali sono `Add` e `Properties`.

I moduli `Add` e `Properties` si comportano in maniera molto simile. Le uniche differenze sono che il primo deve presentare all'utente dei valori di default e permettere di modificare tutti i parametri, mentre il secondo deve presentare i valori correnti della macchina da modificare e non consentire di cambiare il file system (perché esso rappresenta in un certo senso l'identità della macchina). Per questo motivo esiste un'unica struttura per la finestra da generare. La finestra viene generata con la funzione `Dialog_add_or_update.make`, la quale prende in input parametri opzionali. Se i valori dei parametri sono specificati, come avviene nel modulo `Properties`, all'utente saranno inizialmente proposti quei valori, mentre se non sono specificati assumeranno dei valori di default, i quali sono tutti racchiusi nel modulo `Const`. Gli autori di Marionnet fanno largo uso del meccanismo dei *label* di OCaml e sfruttano spesso la possibilità di definire funzioni con parametri opzionali, come in questo caso.
La creazione della struttura di base della finestra, che include un'immagine e i campi per scegliere *Name* e *Label* da associare alla macchina, avviene con la chiamata a
`Gui_bricks.Dialog_add_or_update.make_window_image_name_and_label`.
Il resto della struttura dell'interfaccia viene creato con
`Gui_bricks.make_form_with_labels`.
Il risultato di tale chiamata è un oggetto a cui vengono aggiunte le varie sezioni del *form* con i relativi *widget* che consentono all'utente di inserire le informazioni. In questo contesto è stata inserita una nuova sezione per la relativizzazione, denominata Relativization, che include gli elementi frequenza e convergenza (*Frequency* e *Convergence*).

Per consentire all'utente un inserimento comodo dei dati sono stati scelti degli *spin button*. Gli *spin button* sono dei *widget* forniti dalle librerie *lablgtk* e permettono all'utente di inserire dati numerici. L'utente può digitare il valore con la tastiera oppure premere dei bottoni raffiguranti delle frecce per incrementare o diminuire il valore. Per adattare questi *widget* alle esigenze specifiche del problema, sono state definite funzioni di creazione apposite nel modulo `gui_bricks.ml`: `spin_freq` e `spin_seconds`.
Queste funzioni accettano parametri opzionali per determinare caratteristiche quali i limiti massimo e minimo dei valori che lo *spin button* può assumere. In realtà le funzioni non vengono mai chiamate con questi parametri, perciò essi hanno sempre il valore di default visibile nella definizione della

funzione. È stato adottato questo meccanismo soprattutto per uniformare lo stile del codice a quello preesistente. Le due nuove funzioni sono infatti state definite seguendo lo schema usato nella funzione *spin_byte*, anch'essa parte del modulo `gui_bricks.ml`, usata dagli autori del programma per creare gli *spin button* che prendono in input il valore di un byte.
Questo tipo di *widget* fornito dalle librerie lavora solamente con i tipi float, tuttavia per esprimere il numero di secondi dalla Unix Epoch per il campo *Convergence* è opportuno utilizzare degli interi. Questo problema è stato aggirato convertendo il valore restituito in intero con la funzione `int_of_float` fornita dal modulo *Pervasives* (un modulo che fornisce i tipi e le funzioni di base del linguaggio OCaml). Per nascondere all'utente il reale tipo del dato, creando l'illusione di un tipo intero, alla funzione di creazione del *widget* viene passato l'argomento `digits:0`, che serve ad impostare il numero di cifre da mostrare dopo la virgola.

La creazione della finestra vera e propria avviene all'interno del file `machine.ml` con la chiamata in `Dialog_add_or_update.make` di `Gui_bricks.Dialog_run.ok_or_cancel`. Questa funzione, definita in `gui_bricks.ml`, crea la finestra passata come argomento e restituisce l'insieme dei valori scelti dall'utente. I valori vengono raccolti dalla funzione `get_widget_data`, che è uno dei parametri accettati dalla funzione di creazione. Nel momento in cui `get_widget_data` ottiene i valori di frequenza e convergenza, espressi rispettivamente come `float` e `int`, essi vengono convertiti in stringhe. Il motivo della conversione è che ultimamente devono essere passati in input alla macchina User-Mode Linux, che appunto accetta l'input solamente sotto forma di stringa.

3.2.4 Impostazioni globali

L'interfaccia della finestra principale di Marionnet presenta già alcune impostazioni che hanno effetto sull'intero progetto. Queste opzioni sono accessibili dalla barra del menù nella parte alta della finestra, sotto la voce *Options*. Il bottone che permette di visualizzare e modificare globalmente i parametri di relativizzazione è stato quindi inserito sotto tale voce.
Il codice relativo al bottone che lancia la finestra è stato inserito nel modulo `gui_menubar_MARIONNET.ml`, il quale comprende anche la definizione degli elementi del menù principale. Quando il bottone viene premuto, viene costruito un opportuno modulo chiamato `Relativization`, che viene poi usato per creare e mostrare la finestra di dialogo (attraverso la funzione chiamata `make_window`). La funzione può produrre fondamentalmente due tipi di risultato: un risultato nullo (`None`) se l'utente non ha confermato la

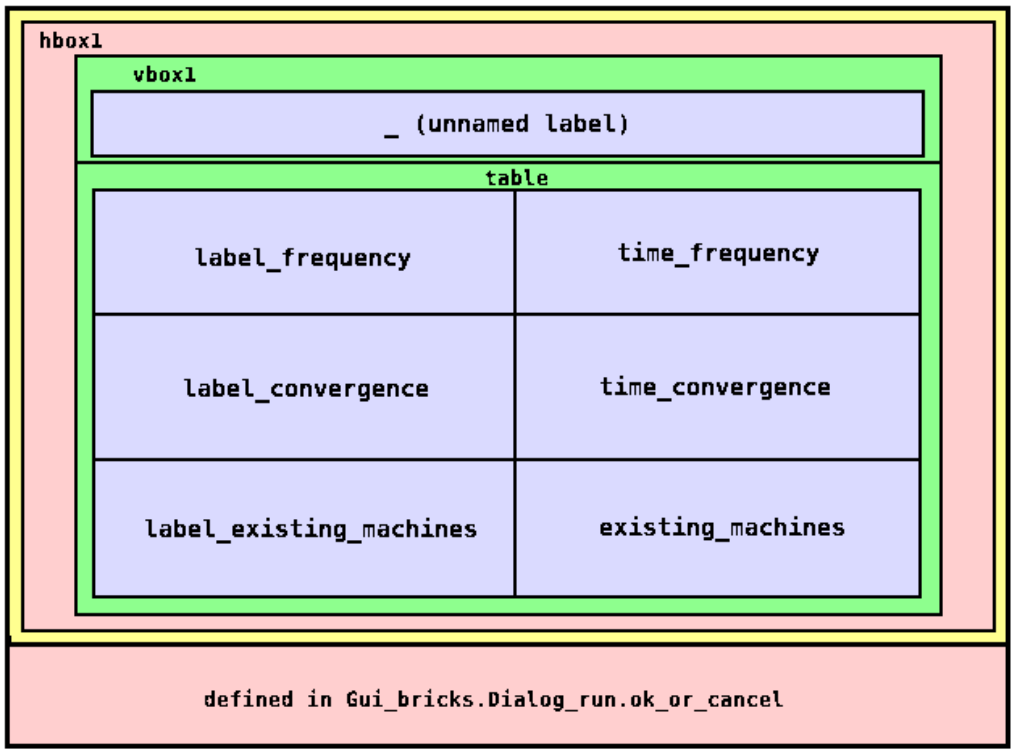

Figura 3.6: Struttura della finestra di dialogo. I nomi corrispondono agli identificatori degli oggetti nel codice.

propria scelta; un risultato non-nullo (`Some result`) nel caso in cui l'utente abbia confermato la propria scelta. Nel primo caso non deve essere effettuata alcuna azione in risposta alla chiusura della finestra. Nel secondo caso i parametri ottenuti attraverso il dialogo con l'utente devono essere applicati, perciò in questo caso viene chiamata la funzione `apply_changes`.

La definizione del costruttore del modulo e delle due funzioni è stata collocata in un modulo a parte, chiamato `relativization.ml`. Il *functor* è chiamato `Make`, e prende in input un modulo la cui interfaccia contenga un oggetto di tipo `State.globalState`. L'input è necessario perché l'oggetto che viene passato rappresenta lo stato globale della rete, quindi comprende anche la rappresentazione di tutti i dispositivi esistenti che potrebbero dover essere modificati.
La funzione `make_window` fa uso della libreria *lablgtk* per creare la struttura della finestra. La figura 3.6 mostra la struttura della finestra ed evidenzia l'incapsulamento degli oggetti che corrispondono agli elementi grafici; i no-

mi mostrati in figura sono quelli utilizzati come identificatori nel codice. Si confronti la figura 3.6 con la figura 1.4 per vedere gli effetti della definizione della struttura. Il risultato della funzione coincide con quello di `Gui_bricks.Dialog_run.ok_or_cancel`. Il tipo del risultato, chiamato `reltime_t`, è definito internamente al modulo `relativization.ml` e comprende i tre valori ottenibili dall'interazione con l'utente. Oltre ai valori dei due parametri di relativizzazione, è presente anche un valore booleano che indica se l'utente ha scelto di applicare la modifica ai dispositivi già esistenti oppure no. Graficamente questa scelta è realizzata con un *checkbox* (bottone su cui si può porre una spunta).

Se l'utente conferma la propria scelta, i parametri di relativizzazione inseriti devono essere sostituiti ai precedenti valori di default per le nuove macchine virtuali. Per rendere possibile questo cambiamento, sono stati modificati i moduli `Const` interni a `machine.ml` e `router.ml`. Questi due moduli, simili tra loro, sono costituiti da due semplici insiemi di valori (statici, cioè non modificabili a tempo di esecuzione) che vengono usati per stabilire quali siano i valori di default per le nuove macchine ed i nuovi router che vengono aggiunti alla rete. I due valori aggiunti per la relativizzazione (frequenza e convergenza) sono stati definiti come riferimenti a zone di memoria, le quali sono modificabili. Sono quindi state definite funzioni *getter* e *setter* (ovvero rispettivamente per ottenere il valore e impostare il valore). Questo approccio è comune nel paradigma *object-oriented*, ma la creazione di un oggetto per queste semplici operazioni è stato ritenuto eccessivo.
I valori relativi a frequenza e convergenza contenuti in `Const` vengono impostati con i valori ottenuti dal dialogo con l'utente.

Se l'utente, prima di confermare la propria scelta, decide di spuntare il *checkbox* (o meglio, non togliere la spunta già presente), i parametri devono essere applicati anche ai dispositivi già esistenti, a patto che essi siano spenti. La funzione che implementa questo è stata chiamata `update_existing_devices` ed è parte del modulo costruito da `Relativization.Make`. Questa funzione recupera tutti i dispositivi spenti del tipo desiderato (macchina o router) dall'oggetto che mantiene lo stato globale del progetto, ottenuto dal modulo preso in input dal costruttore e chiamato `st`. I dispositivi sono identificati dal proprio nome, perciò la lista dei dispositivi è una lista di stringhe. Le due liste ottenute, una per le macchine ed una per i router, vengono visitate dalla funzione ricorsiva in coda (ottimizzata quindi dal compilatore in modo da occupare spazio costante) `loop`. Quest'ultima è una funzione di ordine superiore: essa prende in input la lista dei dispositivi e la funzione da appli-

care ad ognuno di essi. È stata sfruttata questa funzionalità del linguaggio principalmente per evitare la ripetizione di codice.
I router richiedono un trattamento diverso dalle macchine, ma le differenze sono minimali. Per riutilizzare il codice esistente, invece che aggiornare solamente i parametri di frequenza e convergenza, vengono aggiornate anche altre caratteristiche del dispositivo: queste ultime vengono reimpostate al loro valore attuale, senza introdurre cambiamenti indesiderati. La funzione riutilizzata per le macchine è `update_machine_with`, un metodo degli oggetti appartenenti alla classe `Machine.User_level_machine.machine`. Per poter utilizzare tale funzione funzione viene usata `Obj.magic`, una funzione di OCaml che permette sostanzialmente di effettuare una conversione da un tipo di oggetto ad un altro.

Capitolo 4

Conclusioni

Sono state realizzate macchine virtuali relativistiche, cioè macchine alle quali è possibile assegnare una percezione arbitraria dello scorrere del tempo, che possono essere collegare tra loro per formare "reti relativistiche". La relativizzazione può essere impiegata per il test di applicazioni che effettuano operazioni periodicamente, oppure per creare l'illusione della disponibilità di risorse minori o maggiori rispetto a quelle realmente disponibili. Grazie alla funzionalità introdotta è possibile ad esempio emulare reti con ampiezza di banda superiore a quella reale.
Il processo di sviluppo ha previsto due fasi: la creazione di macchine virtuali relativistiche e la creazione di un'interfaccia intuitiva che permetta di creare reti di tali macchine.

Per lo sviluppo della singola macchina virtuale, ci si è appoggiati a due tipologie di *software* di virtualizzazione già esistenti. La funzionalità era già presente nel modulo UMmisc dell'implementazione del progetto ViewOS, applicata però ad un solo processo. La relativizzazione è stata portata ad un'intera macchina. Considerato il contesto tecnologico attuale, è stato ritenuto opportuno creare una *patch* per un programma già esistente. Il *software* di emulazione che è stato modificato a questo scopo è User-Mode Linux, un emulatore molto efficiente che non tenta di riprodurre il comportamento di una specifica architettura, ma che consente di eseguire un *kernel* Linux come processo in modalità utente su un sistema Linux. Sono state prodotte due implementazioni per User-Mode Linux. Le due implementazioni agiscono a due diversi livelli: una opera a livello di intercettazione e gestione delle *system call* effettuate dai processi da emulare; l'altra a livello di interfacciamento, tramite *system call*, di User-Mode Linux con il sistema *host*. La seconda alternativa è stata considerata migliore grazie a proprietà quali la

coerenza nella virtualizzazione del tempo e la maggiore mantenibilità del codice prodotto.

Per semplificare lo sfruttamento della funzionalità di relativizzazione nel contesto di reti virtuali, è stato ritenuto opportuno fornire un'interfaccia grafica per la gestione dei parametri di relativizzazione e per la gestione della rete. A questo scopo è stato scelto Marionnet, un programma scritto in OCaml che fornisce un'interfaccia grafica per la gestione di reti virtuali. Questo programma è stato pensato per l'insegnamento, ma è stato ritenuto sufficientemente completo e versatile anche per scopi diversi. Il programma si basa su componenti quali User-Mode Linux e VDE per creare gli elementi della rete e connetterli tra loro. È stata prodotta una *patch* che aggiunge elementi grafici che permettono di modificare i parametri di relativizzazione a livello di singolo dispositivo virtuale o a livello globale del programma. Nel secondo caso l'utente può decidere di cambiare solamente i valori di default per i nuovi dispositivi oppure di modificare anche i valori dei dispositivi esistenti. I parametri vengono passati alle macchine User-Mode Linux sottostanti, che devono supportare la funzionalità di relativizzazione. I dati vengono inseriti attraverso degli *spin button*: l'utente può scorrere i valori cliccando sugli appositi tasti oppure inserire il valore desiderato digitandolo con la tastiera. I parametri vengono mostrati in forma esplicita: la convergenza è espressa come il numero di secondi passati dall'Epoca, secondo lo standard POSIX. Grazie a Marionnet e alle modifiche introdotte, la creazione di una rete virtuale relativistica anche complessa diventa un processo veloce e relativamente semplice.

Capitolo 5

Sviluppi futuri

Il sistema prodotto è funzionante e usabile, ma alcuni aspetti di esso non sono stati studiati approfonditamente, quindi c'è spazio per qualche miglioramento. Ad esempio la modifica a User-Mode Linux per la creazione di macchine relativistiche potrebbe essere aggiornata in modo da ottenere una maggiore precisione nel mantenimento del tempo. Si potrebbe provare ad esprimere il valore della frequenza evitando di utilizzare il tipo `long double`, ma prima è necessario un controllo accurato per verificare l'effettiva utilità di una tale modifica.
L'esposizione dei parametri nel file system virtuale *proc*, allo stato attuale, crea un dato facilmente leggibile da un essere umano ma più complicato da gestire da parte di un calcolatore. I valori letti dai file sono stringhe che rappresentano dei numeri, quindi per poter effettuare dei calcoli su tali numeri le stringhe devono prima essere riconvertite; questo processo ha un effetto particolarmente negativo sul dato "frequenza", perché è rappresentato internamente con un `long double` quindi la conversione potrebbe portare ad una perdita di precisione. Si potrebbe quindi pensare di aggiungere una versione "binaria" dei parametri all'interno di *proc*: file che quando letti restituiscono i parametri "puri" che corrispondono alla rappresentazione interna al *kernel*. Questa funzionalità potrebbe risultare utile se si vuole risalire al tempo reale calcolandolo in base al tempo virtuale e al valore dei parametri di relativizzazione.

L'interfaccia grafica potrebbe essere semplificata per quanto riguarda la gestione del parametro di convergenza, perché allo stato attuale esso viene mostrato in maniera esplicita, cioè come il numero di secondi passati dall'*Epoca* secondo lo standard POSIX. Questo formato è facilmente gestibile da un calcolatore ma più difficile da interpretare da parte di un essere umano, che si deve affidare a strumenti esterni come il comando standard

"*date*". Una rappresentazione alternativa, potenzialmente più comoda per l'utente, potrebbe prevedere un calendario in cui scegliere anno e mese e degli *spin button* per ora, minuti e secondi. Tuttavia questa scelta potrebbe portare a confusione se il fuso orario della macchina virtuale non corrisponde a quello della macchina reale. Una funzionalità forse ancora più utile sarebbe la possibilità di ottenere l'ora attuale attraverso la pressione di un pulsante. Un altro miglioramento per l'interfaccia potrebbe essere una ristrutturazione della finestra per l'aggiunta o la modifica dei dispositivi di tipo "Macchina": questa finestra di dialogo possedeva un'altezza minima molto grande già prima della modifica introdotta, che da questo punto di vista ha peggiorato la situazione. L'altezza è accettabile sugli schermi delle dimensioni più comuni per computer, ma per schermi particolarmente piccoli la finestra potrebbe non essere visualizzata interamente. Una possibile soluzione potrebbe prevedere l'aggiunta di barre di scorrimento che permettano di scorrere il contenuto della finestra; un'altra consiste nel disporre orizzontalmente alcuni elementi. Infine, l'utente potrebbe non essere interessato alla funzionalità offerta, quindi si potrebbe aggiungere un'opzione, ad esempio nel menù "*Options*", per nascondere le sezioni riguardanti la relativizzazione.

Altri possibili sviluppi potrebbero consistere nel portare la funzionalità di relativizzazione su macchine virtuali diverse da User-Mode Linux, per supportare ad esempio sistemi operativi diversi da Linux. A Marionnet potrebbe essere aggiunto il supporto per *software* di virtualizzazione come Xen [1, 27] (una prospettiva auspicata anche dagli stessi autori [25]) o KVM, per fare in modo che le macchine virtuali gestite in questo ambiente di emulazione possano ospitare anche altri sistemi operativi.

Bibliografia

[1] P. Barham *et al.*: *Xen and the art of virtualization.* ACM SIGOPS Operating Systems Review, 37(5):164–177, 2003.

[2] F. Bellard: *QEMU home page.* `http://www.qemu.org`, accesso a 2014.

[3] F. Bellard: *QEMU, a Fast and Portable Dynamic Translator.* Nel *USENIX Annual Technical Conference, FREENIX Track*, pp. 41–46, 2005.

[4] M. Clasen *et al.*: *The GTK+ Project home page.* `http://www.gtk.org`, accesso a 2014.

[5] clownix@clownix.net: *Cloonix home page.* `http://clownix.net/`, accesso a 2014.

[6] J. Corbet, A. Rubini e G. Kroah-Hartman: *Linux device drivers.* O'Reilly Media, Inc., 2005.

[7] R. Davoli: *Virtual Square Wiki.* `http://www.virtualsquare.org`, accesso a 2014.

[8] R. Davoli: *Vde: Virtual distributed ethernet.* Nel *Testbeds and Research Infrastructures for the Development of Networks and Communities, 2005. Tridentcom 2005. First International Conference on*, pp. 213–220. IEEE, 2005.

[9] R. Davoli e M. Goldweber: *Virtual Square: Users, Programmers & Developers Guide.* Lulu Book, 2011.

[10] J. Dike: *User-Mode Linux home page.* `http://user-mode-linux.sourceforge.net`, accesso a 2014.

[11] J. Dike: *User-Mode Linux*, vol. 2. Prentice Hall Englewood Cliffs, 2006.

[12] U. Drepper, P. Miller e B. Haible: *Gettext home page*. `https://www.gnu.org/software/gettext`, accesso a 2014.

[13] L. Gardenghi, M. Goldweber e R. Davoli: *View-os: A new unifying approach against the global view assumption*. Nel *Computational Science–ICCS 2008*, pp. 287–296. Springer, 2008.

[14] J. Garrigue, H. Fauque, J. Furuse e K. Kagawa: *LablGTK: an objective caml interface to GTK+*. `http://lablgtk.forge.ocamlcore.org/`, accesso a 2014.

[15] J. Grossmann: *GNS3 home page*. `http://www.gns3.net`, accesso a 2014.

[16] IEEE: *Standard for Information Technology - Portable Operating System Interface (POSIX(R))*, 2008.

[17] IEEE: *Standard for Ethernet*, 2012.

[18] K. Ishiguro *et al.*: *Quagga home page*. `http://www.nongnu.org/quagga`, accesso a 2014.

[19] ISO/IEC: *Information technology – Open Systems Interconnection – Basic Reference Model: The Basic Model*, 1994.

[20] L. R. Izquierdo e J. G. Polhill: *Is your model susceptible to floating-point errors?* Journal of Artificial Societies and Social Simulation, 9(4), 2006.

[21] A. Kivity, Y. Kamay, D. Laor, U. Lublin e A. Liguori: *kvm: the Linux virtual machine monitor*. Nel *Proceedings of the Linux Symposium*, vol. 1, pp. 225–230, 2007.

[22] S. Le Gall: *OCaml gettext home page*. `https://forge.ocamlcore.org/projects/ocaml-gettext`, accesso a 2014.

[23] X. Leroy, D. Doligez, A. Frisch, J. Garrigue, D. Rémy, J. Vouillon *et al.*: *The OCaml system release 4.01: Documentation and user's manual*. 2013. `http://caml.inria.fr/pub/docs/manual-ocaml-4.01/index.html`.

[24] J. V. Loddo e L. Saiu: *Marionnet home page*. `http://www.marionnet.org`, accesso a 2014.

[25] J. V. Loddo e L. Saiu: *Status report: marionnet or how to implement a virtual network laboratory in six months and be happy*. Nel *Proceedings of the 2007 workshop on Workshop on ML*, pp. 59–70. ACM, 2007.

[26] J. V. Loddo e L. Saiu: *Marionnet: a virtual network laboratory and simulation tool*. Nel *First International Conference on Simulation Tools and Techniques for Communications, Networks and Systems*, 2008.

[27] Linux Foundation: *Xen home page*. `http://www.xenproject.org`, accesso a 2014.

[28] Oracle Inc.: *VirtualBox home page*. `http://www.virtualbox.org`, accesso a 2014.

[29] VMware Inc.: *VMware home page*. `http://www.vmware.com`, accesso a 2014.

[30] D. L. Mills: *Internet time synchronization: the network time protocol*. IEEE Transactions on Communications, 39(10):1482–1493, 1991.

[31] G. Project: *Using the GNU Compiler Collection, i386 and x86-64 Options*. `http://gcc.gnu.org/onlinedocs/gcc/i386-and-x86-64-Options.html`, accesso a 2014.

[32] L. Torvalds: *Linux home page*. `http://www.linux.com`, accesso a 2014.

[33] G. Van Rossum e F. L. Drake: *Python language reference manual*. Network Theory, 2011.