UNIVERSITÀ DEGLI STUDI DI PISA
FACOLTÀ DI SCIENZE MATEMATICHE, FISICHE E NATURALI

CORSO DI LAUREA IN INFORMATICA

MONITORAGGIO E AMMINISTRAZIONE DI RETI VIRTUALI

Tesi di laurea specialistica

Relatore: Prof. F. Baiardi
Candidato: Carlo Bertoldi

ANNO ACCADEMICO 2007/2008

Sommario

Negli ultimi anni l'attenzione, dei ricercatori e dell'industria, nei confronti dei sistemi di virtualizzazione è incrementata notevolmente. Un sistema di virtualizzazione ha lo scopo di fornire diversi ambienti di esecuzione software, comunemente chiamati *macchine virtuali*, astraendo dalla risorse hardware e software sottostanti.
Uno dei vantaggi principali offerti dalla virtualizzazione, è costituito dalla possibilità di razionalizzare l'utilizzo delle risorse di calcolo a disposizione: innanzitutto è possibile destinare un solo computer allo svolgimento del lavoro che prima veniva svolto da più nodi, garantendo così un notevole risparmio, abbattendo i costi hardware, di manutenzione e il consumo di energia elettrica. Inoltre, è possibile sfruttare la tecnologia di virtualizzazione per progettare un sistema più sicuro. È infatti possibile, eseguire i processi critici, ad esempio un server web, all'interno di macchine virtuali separate, garantendo un elevato grado di isolamento fra applicazioni differenti.
Questa tesi presenta uno strumento software per il monitoraggio e l'amministrazione remota di macchine virtuali. Tale strumento analizza le *hypercall* invocate dal sistema operativo virtualizzato, nei confronti del *virtual machine monitor*, per fornire una vista dettagliata dello stato di esecuzione della macchina virtuale.
La tesi presenta, inoltre, il lavoro svolto per cercare di definire il concetto di *sense of self* di una macchina virtuale *Xen*. Tale ricerca ha lo scopo di individuare un metodo efficace per distinguere l'attività lecita svolta da una macchina virtuale, dall'attività anomala, dovuta, ad esempio, ad un tentativo di intrusione.
Infine, dopo aver descritto l'architettura complessiva dello strumento, la tesi presenta una prima valutazione delle prestazioni del prototipo realizzato.

Ringraziamenti

Grazie, grazie.

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Virtualizzazione e Xen

Questo capitolo introduce la tecnologia della virtualizzazione, descrivendo le varie tipologie esistenti e, in particolare, quella hardware. Vengono presentati i principali casi di questa virtualizzazione ed i principali vantaggi offerti. Il capitolo presenta poi una panoramica dei principali protocolli utilizzabili per la creazione di una rete virtuale, descrivendo brevemente il funzionamento di ognuno. Successivamente, viene descritto Xen, il Virtual Machine Monitor sviluppato dall'Università di Cambridge. Infine, sono presentati due strumenti di monitoraggio di un sistema distribuito. I due strumenti sono focalizzati, rispettivamente, sui servizi offerti (Nagios) e sul traffico di rete (Ntop).

1.1 La virtualizzazione

La virtualizzazione è una tecnologia di cui si è iniziato a parlare sin dai primi anni 60: già allora era noto il concetto di macchina virtuale (VM), che indicava un'astrazione software atta a fornire la stessa interfaccia messa a disposizione dalla macchina reale [8].

La prima VM a supportare interamente la virtualizzazione è stata rilasciata da IBM come parte integrante del mainframe S/360. Questo sistema poteva essere partizionato in maniera ricorsiva in macchine virtuali [41]. I moderni sistemi operativi contengono una versione semplificata del concetto di virtualizzazione: ogni processo in esecuzione

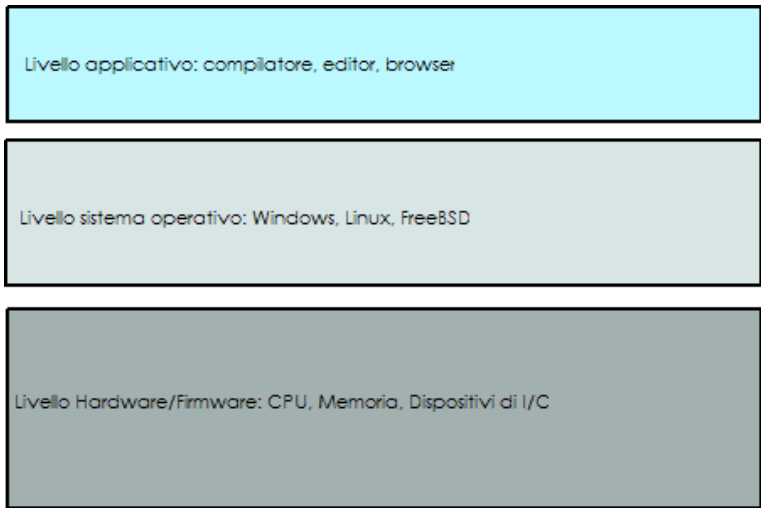

Figura 1.1: I livelli in cui può essere suddiviso un elaboratore

può comportarsi come se fosse l'unico processo in esecuzione su un certo computer. Questo è possibile perché le risorse che utilizza, in primis la CPU e la memoria, sono virtualizzate. Il concetto di virtualizzazione è comunque molto ampio, basti pensare alla tecnologia sviluppata da Sun della Java Virtual Machine.

1.1.1 Livelli di virtualizzazione

Osservando i livelli in cui può essere idealmente suddiviso un sistema di elaborazione (vedi figura 1.1), possiamo distinguere tre tipologie principali di virtualizzazione: a livello hardware, a livello di sistema operativo (SO) e a livello di software applicativo.

- **Virtualizzazione a livello hardware:** il livello di virtualizzazione è implementato direttamente sul livello hardware/firmware, e deve presentare al livello successivo, quello del SO, un insieme di risorse virtuali, ottenute attraverso l'emulazione delle risorse fisiche della macchina (CPU, memoria, dispositivi di I/O...)

- **Virtualizzazione a livello di SO:** il software di virtualizzazione è intermedio fra il sistema operativo e le applicazioni. Queste macchine virtuali permettono di eseguire applicazioni scritte per il medesimo SO.

- **Virtualizzazione a livello applicativo:** la macchina virtuale è un'applicazione per il SO sottostante, ed esegue software compilato ad hoc.

La tipologia di virtualizzazione che più ci interessa è quella a livello hardware, in quanto essa permette di condividere la stessa macchina fisica fra più istanze di sistemi operativi diversi. Il software che si occupa della creazione e gestione delle macchine virtuali è il Virtual Machine Monitor (VMM), o Hypervisor, che deve essere in grado di garantire tre proprietà fondamentali:

- **Compatibilità:** : il software scritto per essere eseguito da una macchina reale deve poter essere eseguito anche da una macchina virtuale.
- **Isolamento:** una macchina virtuale in esecuzione non deve poter interferire con le altre VM in esecuzione sul medesimo nodo fisico.
- **Incapsulamento:** tutto il software deve essere eseguito all'interno delle macchine virtuali, in modo tale che solo il VMM abbia accesso all'hardware. Questo permette di garantire anche la proprietà dell'isolamento.

I VMM possono essere classificati in due categorie principali (vedi fig. 1.2):

- *Unhosted VMM:* il software di virtualizzazione è quello tra l'hardware e il sistema operativo guest. Xen è un VMM di questo tipo.
- *Hosted VMM:* il software di virtualizzazione è un processo eseguito all'interno di un sistema operativo. Il VMM si interpone fra il SO e il software che esso esegue. Un esempio di VMM di questo tipo è User Mode Linux [40]

1.1.2 Livelli di privilegio

I processori x86 implementano un meccanismo di protezione basato su una gerarchia di livelli, ciascuno dei quali viene definito anello (ring).

Ad ogni anello corrisponde un livello di privilegio, partendo dal livello 0, che gode dei privilegi massimi, fino al 3, a cui sono concessi i privilegi minimi. Solitamente il

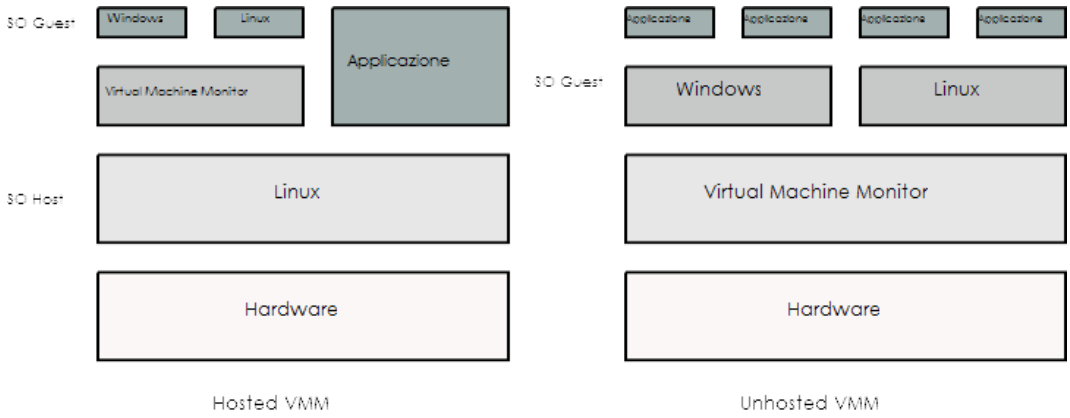

Figura 1.2: I tipi di VMM

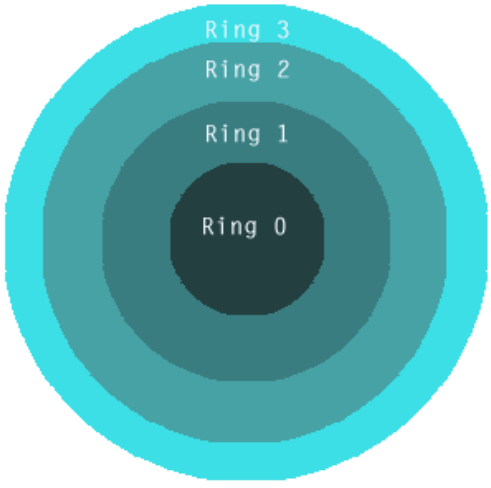

Figura 1.3: I livelli di privilegio

SO viene eseguito nel livello 0, in modo da avere il controllo totale della CPU, mentre le applicazioni sono ristrette al livello 3. Chiaramente, questo non è possibile in un sistema virtualizzato, in quanto il VMM deve poter garantire l'isolamento fra le diverse macchine virtuali. Un sistema virtualizzato esegue generalmente il VMM al livello 0, il SO guest al livello 1 e le applicazioni ancora al livello 3. Questo, però, non è sempre vero, perché AMD, con l'introduzione delle CPU x86-64, ha eliminato i livelli di privilegio 1 e 2. In questo caso, la scelta obbligata è stata quella di eseguire il SO guest al livello 3 insieme alle applicazioni. Il problema è stato risolto con l'introduzione delle estensioni di supporto alla virtualizzazione hardware, che permettono di eseguire il SO guest sempre a livello 0, grazie all'introduzione di un nuovo livello di privilegio chiamato -1.

1.1.3 Tipologie di virtualizzazione

Possiamo individuare due tecniche di virtualizzazione:

1. *Full virtualization:* simula completamente l'hardware sottostante, garantendo l'esecuzione del sistema operativo guest all'interno della VM senza alcun tipo di modifica. Penalizza le prestazioni perché il VMM deve intercettare le operazioni privilegiate che il SO ospite vuole eseguire per poter eseguire le equivalenti istruzioni virtuali. Le tecniche di full virtualization maggiormente utilizzate sono la *dynamic recompilation* [17] e la *binary translation* [34].

2. *Paravirtualization:* non simula interamente l'architettura considerata e quindi rende necessaria la modifica di alcune parti del sistema operativo che si vuole virtualizzare. Le performance ottenibili sono molto simili a quelle che si otterrebbero eseguendo il SO direttamente sull'hardware.

1.1.4 Supporto hardware alla virtualizzazione

Chiaramente il sistema che più ci interessa emulare è l'architettura x86, essendo questa la piattaforma hardware più diffusa. Per cercare di migliorare le prestazioni dei sistemi virtualizzati, i principali produttori di CPU x86, Intel e AMD, hanno introdotto due

set di estensioni: Intel Virtualization Technology (IVT o VT) [39] e AMD-V. [1]. Queste istruzioni permettono di ottenere prestazioni simili a quelle dei sistemi non virtualizzati, eliminando la necessità di modificare il SO che si intende virtualizzare. Le Virtual Machine che sfruttano tali estensioni vengono definite Hardware Virtual Machine (HVM).

1.1.5 L'architettura Intel VT

Questa architettura introduce due nuove modalità di esecuzione che un processore può utilizzare:

- VMX root operation
- VMX non-root operation

Quando il processore esegue istruzioni nella prima modalità si comporta come una CPU tradizionale, ed è quindi la modalità di funzionamento utilizzata dal VMM. La seconda modalità fornisce, invece, un ambiente di esecuzione controllato da un VMM, ed ha lo scopo di facilitare la gestione delle macchine virtuali. La transizione da una modalità all'altra ha reso necessaria l'introduzione di una struttura dati chiamata Virtual Machine Control Structure (VMCS), nella quale viene salvato lo stato del processore.

1.1.6 Vantaggi della virtualizzazione

La virtualizzazione offre vari benefici [39]. Innanzitutto, è possibile effettuare il **consolidamento** del parco macchine; una singola macchina fisica può svolgere il lavoro che prima era allocato a più computer. In questo modo, le risorse di calcolo sono sfruttate con maggiore efficienza, consentendo inoltre un risparmio sui costi di manutenzione e di esecuzione (e.g. minor consumo di energia elettrica). La **migrazione** di una macchina virtuale da un server fisico ad un altro è un'operazione estremamente semplice, ed è così possibile effettuare un efficace bilanciamento del carico fra diversi computer fisici, garantire l'alta disponibilità di servizi critici e semplificare la gestione del processo di backup del parco macchine. Infine, una macchina virtuale fornisce un

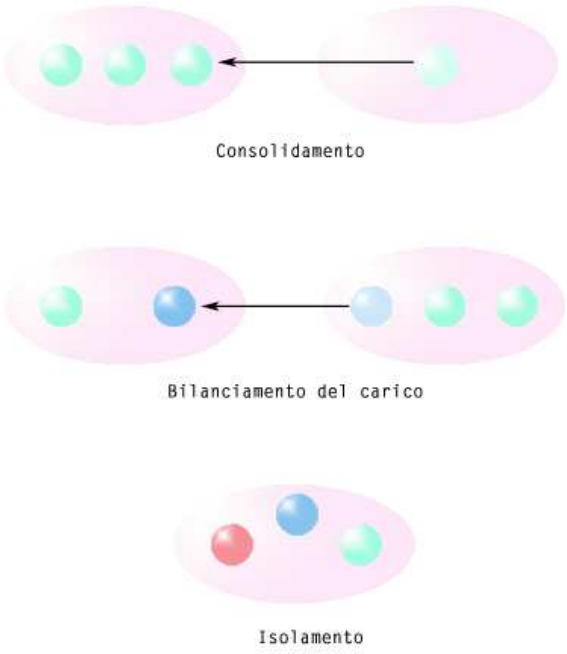

Figura 1.4: I vantaggi della virtualizzazione

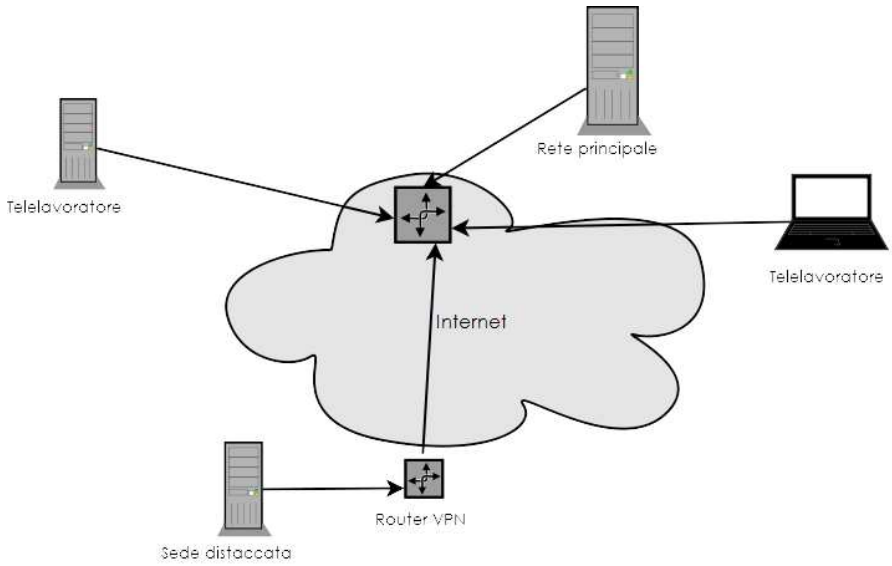

Figura 1.5: La struttura di una VPN

grado di **isolamento** sicuramente superiore rispetto a quello di un processo all'interno di un sistema operativo. In caso di compromissione di un processo, è possibile quindi garantire alle altre macchine virtuali un maggiore livello di sicurezza.

1.2 Reti virtuali

Attualmente, grazie ad una maggiore disponibilità di connessioni Internet ad alta velocità, il concetto di rete di comunicazione sta perdendo l'accezione di località spostandosi verso un'idea di rete più ampia, nella quale i vari nodi non comunicano necessariamente tramite un collegamento fisico diretto, ma possono essere ad esempio essere interconnessi tramite connessioni implementate da molti nodi della rete stessa. Questa sezione descrive alcuni fra i protocolli e metodi che permettono di realizzare quella che viene comunemente chiamata **Virtual Private Network** (VPN).

Una VPN è un tunnel punto a punto che permette di simulare una rete privata fra i due estremi del tunnel utilizzando una comunicazione crittata che ha Internet, o un'altra rete pubblica, come canale di trasporto [3].

1.2.1 PPP

In principio il metodo per stabilire una connessione diretta fra due nodi fisicamente collegati in rete era unicamente quello di utilizzare il Point to Point Protocol (PPP) [30], un protocollo che opera al livello 2 della pila OSI (data link layer). PPP è formato da tre componenti principali:

1. Un metodo per incapsulare datagrammi multiprotocollo, con l'ausilio di intestazioni aggiuntive.

2. Un *Link Control Protocol* (LCP), per stabilire, configurare e verificare la connessione al livello data link.

3. Un insieme di *Network Control Protocol* (NCP) per configurare differenti protocolli al livello Network, tipicamente IP.

Inoltre, il PPP supporta l'autenticazione del collegamento utilizzando due protocolli differenti: il *Password Authentication Protocol* (PAP) e il *Challenge Handshake Authentication Protocol* (CHAP). Tipicamente, il PPP viene attualmente utilizzato dai fornitori di connessioni ADSL incapsulato in un altro livello di trasporto, come ad esempio Ethernet (PPPoE) o ATM (PPPoA). La necessità di implementare le VPN ha portato alla progettazione di nuovi protocolli, che possono essere visti come estensioni del PPP.

1.2.2 PPTP

Il *Point to Point Tunneling Protocol* (PPTP) è stato sviluppato da un consorzio cui appartenevano, fra gli altri, Microsoft e 3Com [26], come un'estensione del PPP, che permette un trasferimento di dati fra un client e un server collegati attraverso una rete basata sulla suite di protocolli TCP/IP. Nella terminologia PPTP, il client è detto *PPTP Access Concentrator* (PAC), e deve dare la possibilità di stabilire connessioni TCP verso il server, detto *PPTP Network Server* (PNS). Una coppia PAC-PNS definisce un tunnel PPTP.

1.2.3 Funzionamento del protocollo

Il corretto funzionamento di un tunnel PPTP richiede due connessioni:

1. una connessione di controllo, basata su TCP, fra il PAC e il PNS.
2. un tunnel IP fra il PAC e il PNS che trasporta i pacchetti PPP incapsulati in pacchetti GRE (*Generic Routing Encapsulation*).

La connessione di controllo è una sessione TCP standard, che trasmette informazioni di controllo di chiamata e gestione delle informazioni PPTP, ovvero è responsabile della crezione, della gestione e della terminazione delle sessioni trasportate attraverso il tunnel. Questa connessione è logicamente associata, benché separata, dalle sessioni incapsulate nel tunnel PPTP. Per ogni coppia PAC-PNS è richiesta la creazione di un tunnel, nel quale possono transitare differenti sessioni PPP, distinte grazie ad un particolare campo contenuto nelle intestazioni dei pacchetti GRE. Le intestazioni permettono anche di implementare funzionalità di controllo di congestione e del flusso. Il PPTP soffre però di notevoli problemi di sicurezza: i messaggi scambiati sul canale di controllo non sono né autenticati né vengono sottoposti a verifiche di integrità, semplificando il compito di un attaccante che volesse intromettersi nella comunicazione. Inoltre, per evitare che i dati contenuti nei pacchetti PPP possano essere intercettati e modificati, è necessario adottare strumenti di crittografia [31].

1.2.4 L2F

L2F, per esteso *Layer Two Forwarding*, è un protocollo di tunneling sviluppato da Cisco Systems [18] per creare una VPN sfruttando Internet come canale di comunicazione. Per molti versi è un protocollo simile al PPTP, vediamo dunque le principali differenze:

- L2F utilizza datagrammi UDP per la comunicazione.
- Supporto per l'autenticazione tramite PAP e CHAP.
- Non esiste una connessione di controllo separata rispetto al tunnel vero e proprio.

1.2.5 L2TP

L2TP, per esteso *Layer Two Transport Protocol*, è un protocollo di tunneling standard [19] che permette di creare Virtual Private Network. Questo protocollo è stato progettato cercando di fondere gli aspetti migliori del PPTP e del L2F. Anche se si comporta come un protocollo di livello 2 della pila OSI, in realtà è un protocollo di livello 5, detto di sessione. La creazione di un tunnel L2TP coinvolge due nodi:

- *L2TP Access Concentrator* (LAC): un computer che vuole instaurare una VPN con una rete remota si deve collegare a questo nodo, che si occupa della comunicazione con l'altro lato del tunnel, l'LNS.
- *L2TP Network Server* (LNS): è il nodo che funge da terminazione logica della sessione PPP che viaggia all'interno del tunnel instaurato con un LAC.

1.2.6 Struttura del protocollo

L2TP utilizza due tipi di messaggi: il primo tipo trasmette informazioni di controllo, mentre il secondo dati da scambiare fra gli estremi del tunnel. I messaggi di controllo sono utilizzati per l'inizializzazione, la gestione e la terminazione di un tunnel. Una connessione di controllo viene avviata con un procedimento simile al 3-way handshake del TCP, e si fa carico di garantire la consegna dei messaggi, proprietà che invece non è garantita per la connessione dati. I pacchetti dati eventualmente persi non vengono ritrasmessi. Se dovesse essere necessaria una connessione dati affidabile, la scelta del protocollo incapsulato nel tunnel verrà effettuata rispettando tale requisito.

Per garantire un'adeguata possibilità di estensione del protocollo, senza per questo intaccare l'interoperabilità, è stato elaborato un formato standard per la codifica della tipologia dei messaggi e del contenuto: *Attribute Value Pair* (AVP). I primi 6 bit formano una maschera che descrive gli attributi generali della codifica. Due bit sono definiti, mentre i rimanenti quattro sono riservati per usi futuri. Il bit **M** (Mandatory - obbligatorio) controlla il comportamento di un'implementazione alla ricezione di un AVP sconosciuta. Se il bit è attivo la sessione, o il tunnel, cui è associata la coppia deve essere terminata immediatamente. In caso contrario, un'eventuale coppia scono-

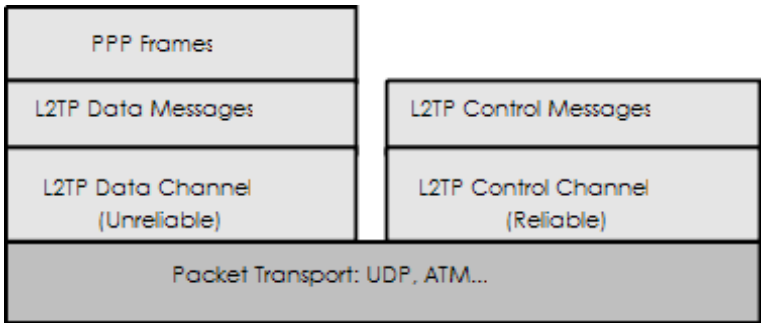

Figura 1.6: I messaggi L2TP

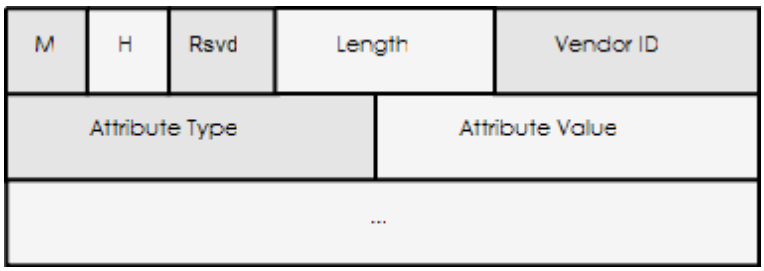

Figura 1.7: La struttura di un AVP

sciuta verrà semplicemente ignorata. Il bit **H** (Hidden - nascosto) codifica il fatto che si vogliono celare i dati contenuti nell'AVP. Questo bit permette ad esempio di evitare l'invio di dati sensibili, quale può essere una password, come testo in chiaro. Il campo **Lenght** codifica il numero di byte contenuti nella relativa AVP. Il campo è di 10 bit, permettendo così di inserire un massimo di 1023 byte in una singola AVP, mentre la lunghezza minima è di 6 byte. Il campo **Vendor ID** contiene un identificativo univoco, lungo 2 byte, per evitare incompatibilità fra estensioni differenti. Nel caso di AVP standard l'ID deve contenere il valore zero. I 16 bit contenuti nel campo **Attribute Type** identificano univocamente il tipo di attributo nello spazio relativo al Vendor ID. Lo spazio rimanente, specificato dal valore contenuto in Length a cui vanno sottratti 6 byte di intestazioni, contiene il valore vero e proprio dell'attributo descritto dalla coppia Vendor ID e Attribute Type.

1.2.7 Funzionamento del protocollo

Per trasmettere una sessione PPP mediante un tunnel sono necessari due passi:

1. creare un tunnel con relativa connessione di controllo

2. iniziare una sessione all'invio, o al ricevimento, di una chiamata

Una volta creata la sessione, è possibile iniziare la trasmissione di frame PPP all'interno del tunnel. L2TP prevede la possibilità di sessioni multiple all'interno del medesimo tunnel.

1.2.8 Tipologie di tunnel

Esistono diversi tipi di tunnel L2TP, a seconda dell'estensione del tunnel relativamente alla sessione PPP [20]:

- Voluntary tunnel: il tunnel viene creato dall'utente, tipicamente con l'ausilio di un client L2TP. L'utente invia il traffico all'ISP, che si occuperà di inoltrarlo verso l'LNS. Il provider non si deve preoccupare di offrire supporto L2TP. In questo modello il tunnel si estende attraverso tutta la sessione PPP, dal client L2TP all'LNS.

- Compulsory tunnel model - incoming call: il tunnel viene creato a insaputa dell'utente, che si limita ad inviare il traffico PPP al provider; tale traffico viene incapsulato dall'LAC dell'ISP, che lo inoltra all'LNS remoto. In questa modalità il tunnel si appoggia solamente al segmento della sessione PPP fra il provider e l'LNS.

- Compulsory tunnel model remote dial: è analogo al modello precedente, ma in questo caso è l'LNS remoto che si occupa di creare il tunnel, e il LAC del provider chiede di instaurare una connessione PPP verso un determinato host.

- L2TP Multi hop connection: questa modalità coinvolge un L2TP Multi-hop gateway, un nodo che agisce contemporaneamente da LAC e da LNS. Il tunnel è

in realtà la composizione di due tunnel: il primo viene instaurato fra un LAC e il multi-hop gateway, il secondo viene instaurato fra il multi-hop gateway e l'LNS.

1.2.9 Problematiche di sicurezza

Durante la creazione del tunnel, è possibile autenticare gli endpoint utilizzando un protocollo simile al CHAP, che prevede la condivisione di un segreto fra il LAC e l'LNS, tipicamente una password d'accesso. Questo meccanismo chiaramente non può proteggere la comunicazione da un attaccante che sia in grado di intercettare il traffico del tunnel per iniettare traffico malevolo. Nel caso sia necessario garantire la riservatezza della comunicazione è possibile incapsulare i messaggi L2TP in IPsec [21].

1.2.10 SOCKS

SOCKS [35] è un meccanismo di proxy trasparente che permette ad un nodo di una rete di connettersi verso nodi appartenenti ad altre reti, garantendo un buon livello di protezione, in quanto il nodo che inizia la connessione non viene mai esposto sulla rete pubblica. SOCKS opera al livello sessione, il 5^o della pila OSI. Esistono 3 versioni principali del protocollo SOCKS: 4, 4a e 5. La 4a è una semplice estensione della 4 che permette al client di utilizzare direttamente l'hostname, qualora non fosse in grado di risolverlo nel corrispondente indirizzo IP. La versione 5 [36] invece è in grado di gestire datagrammi UDP, in aggiunta a connessioni TCP come le precedenti versioni, inoltre prevede la possibilità di gestire diversi metodi di autenticazione.

1.2.11 SSH

Secure Shell (SSH) [37] è un protocollo che permette l'utilizzo sicuro di diversi servizi di rete, in primis il login remoto, attraverso una rete insicura. SSH utilizza la crittografia a chiave pubblica per autenticare l'utente presso il computer remoto e vice versa. Fra le possibili applicazioni di SSH una in particolare è molto interessante, perché permette di creare una sorta di VPN a livello di singola applicazione: il protocollo SSH

consente infatti di effettuare l'inoltro del traffico diretto ad una porta locale verso un host remoto, trasmettendo il traffico di rete all'interno di un canale crittato.

1.2.12 VLAN

Una LAN virtuale (VLAN) è composta da un insieme di nodi che comunicano come se appartenessero al medesimo dominio broadcast, senza tener conto dell'effettiva collocazione fisica, ovvero, una VLAN permette di raggruppare computer come accade in una normale LAN, anche se non connessi fisicamente allo stesso switch. Le VLAN permettono di dividere una rete fisica in più segmenti separati con facilità. Normalmente per partizionare una rete in due, o più, sottoreti dovremmo riconfigurare gli indirizzi IP dei vari nodi, operando quindi a livello 3. Invece, utilizzando una VLAN, possiamo operare a livello 2, in maniera non solo totalmente trasparente ai nodi connessi alla rete, ma anche riducendo la complessità di gestione. Infatti, è sufficiente modificare le configurazioni degli switch che compongono la rete e questi apparati tipicamente sono presenti in numero decisamente minore rispetto ai nodi della LAN stessa. È possibile decidere l'appartenenza ad una VLAN in due modi: statico e dinamico. Nella modalità statica si partizionano a priori le porte di uno switch decidendo quali appartengono ad una VLAN. In questo modo, qualunque nodo che si colleghi ad una porta è automaticamente connesso ad una specifica VLAN. Nella modalità dinamica, invece, si può decidere dinamicamente a quale VLAN appartenga la porta di uno switch, utilizzando alcune informazioni, ad esempio l'indirizzo MAC del nodo connesso a tale porta.

1.2.13 802.1Q

Il protocollo utilizzato per configurare una VLAN è l'802.1Q [22] che definisce un'intestazione di 4 byte, detta tag, contenente, fra gli altri campi, un VLAN identifier (VID) di 12 bit, che identifica univocamente la VLAN a cui appartiene il frame così marcato.

1.2.14 MPLS

Multi Protocol Label Switching (MPLS) [25] è un meccanismo di instradamento, utilizzato sulle reti a commutazione di pacchetto, che opera al livello 2 della pila OSI. Il multi protocollo dell'acronimo si riferisce al fatto che MPLS è utilizzabile con qualunque protocollo dello strato di rete (tipicamente IP). La motivazione principale che ha sostenuto la progettazione di MPLS è stata quella di creare un protocollo di switching ad alte prestazioni, perché non era ancora possibile implementare l'instradamento dei pacchetti IP interamente al livello hardware. Con la diffusione degli ASIC [2] l'MPLS ha visto venir meno lo scopo principale per cui era stato creato, ma non ha perso importanza, ed ora viene utilizzato prevalentemente per la gestione del traffico di rete e per la creazione di VPN sopra reti IPv4 esistenti.

1.2.15 Funzionamento del protocollo

Prima di descrivere il funzionamento di MPLS illustriamo brevemente il meccanismo standard di instradamento dei pacchetti in una rete IP. Quando un router riceve un pacchetto IP ne analizza l'intestazione, sceglie il prossimo hop in base ad un algoritmo di instradamento, e inoltra il pacchetto. L'header di un pacchetto contiene però numerose altre informazioni superflue per la scelta del successivo hop, possiamo quindi suddividere il problema della scelta del nodo come la composizione di due funzioni. La prima funzione partiziona i pacchetti in arrivo in alcune classi, dette *Forwarding Equivalence Classes* (FEC), in base ad alcune caratteristiche (indirizzo di destinazione, classe di QoS) dei pacchetti stessi. La seconda funzione, invece, deve mappare una FEC nell'hop verso cui deve essere inoltrato il pacchetto. È facile notare come i pacchetti appartenenti alla stessa classe seguiranno la stessa rotta, o insieme di rotte nel caso siano utilizzate strategie di routing multipath. Il normale meccanismo di instradamento IP non è molto efficiente, in quanto prevede che ogni router applichi anche la prima funzione. Al contrario con MPLS l'assegnamento di un pacchetto ad una FEC avviene solamente una volta, al momento dell'ingresso del pacchetto nella rete. La FEC di appartenenza viene codificata con un valore, il label, che verrà utilizzato come indice per

accedere ad una tabella contenente le possibili destinazioni del pacchetto. I vantaggi di questo meccanismo rispetto al tradizionale instradamento IP sono molteplici:

1. È possibile basare l'instradamento su più variabili, anche se queste non sono ricavabili direttamente dalle intestazioni del pacchetto, come ad esempio la porta dello switch/router sulla quale è arrivato.

2. L'instradamento del pacchetto può variare sulla base del router di ingresso.

3. Per instradare lungo un percorso prefissato, non è necessario utilizzare informazioni aggiuntive nell'intestazione del pacchetto, in quanto il label MPLS è sufficiente per codificare il percorso prescelto.

Un label MPLS comprende 4 campi per un totale di 32 bit, vediamoli nel dettaglio:

1. Il valore del label, 20 bit.

2. La priorità QoS, 3 bit .

3. 1 bit per indicare che il label si trova in fondo alla pila.

4. Un campo TTL, 8 bit.

Nulla vieta che un pacchetto possa essere marcato più volte, creando così una pila di label sopra le intestazioni.

In una rete MPLS i punti di ingresso e uscita prendono il nome di *Label Edge Routers* (LER), e si occupano rispettivamente di fare il *push* (aggiunta) e il *pop* (rimozione) dei label sul pacchetto in transito. I router che si occupano solamente di instradare i pacchetti sulla base dei label, sono detti *Label Switch Routers* (LSR). Quando un pacchetto transita per un LSR, questi analizza solo il label sulla cima della pila, e in base al valore effettuerà un'operazione di push, pop o *swap* (scambio, ovvero un pop seguito da un push).

1.2.16 IPsec

IPsec [13] è stato progettato per fornire servizi di sicurezza a livello IP, permettendo ad un sistema di decidere quali protocolli e algoritmi utilizzare, e di gestire le necessarie chiavi crittografiche. Per servizi di sicurezza si intende il controllo degli accessi, la mutua autenticazione dei nodi coinvolti in una comunicazione, la confidenzialità dei dati trasmessi e la protezione da attacchi di tipo reply. IPsec permette anche di gestire la compressione IP, poiché utilizzando la crittografia non è possibile utilizzare tecniche di compressione nei livelli inferiori della pila OSI. Possono essere individuate tre tipologie di comunicazione che IPsec è in grado di proteggere:

- fra due host (e.g. due computer)
- fra due security gateway (e.g. firewall, router)
- fra un security gateway e un host

IPsec è un protocollo altamente configurabile, ad esempio è possibile decidere di utilizzarlo per proteggere tutto il traffico scambiato fra due gateway utilizzando un singolo tunnel, oppure si può utilizzare un tunnel per ciascuna connessione TCP instaurata.

1.2.17 Funzionamento del protocollo

IPsec utilizza due protocolli per garantire la sicurezza del traffico:

- *IP Authentication Header* (AH): garantisce l'integrità e l'autenticazione dei datagrammi IP, inoltre fornisce protezione contro gli attacchi di tipo replay. AH deve anche proteggere le intestazioni IP esclusi i campi che possono essere modificati durante il transito del pacchetto, come ad esempio il campo TTL.
- *Encapsulating Security Payload* (ESP): ha funzionalità simili al protocollo AH, e in più è in grado di garantire la confidenzialità dei messaggi scambiati, inclusa l'identità della sorgente e del destinatario della comunicazione. A differenza di AH non si preoccupa di proteggere le intestazioni IP.

Tali protocolli possono essere utilizzati singolarmente o in maniera combinata, a seconda dei servizi che si vogliono fornire. Entrambi supportano due modalità di utilizzo:

- *Transport mode*: sono protetti solamente per i protocolli di livello superiore (TCP, UDP...) Viene utilizzato per la comunicazione fra due host.
- *Tunnel mode*: il pacchetto IP originale viene incapsulato in un nuovo pacchetto IP dopo essere stato elaborato da AH o ESP. È la modalità utilizzata ogni qual volta una delle parti coinvolta nella comunicazione è un security gateway. Questa modalità può essere utilizzata per la creazione di VPN.

1.2.18 Security Association

Una *Security Association* (SA) è un insieme di algoritmi e parametri, quale ad esempio le chiavi crittografiche, utilizzato per autenticare e crittare un flusso di comunicazione in una determinata direzione. Di conseguenza, nel caso di una normale comunicazione bidirezionale fra due host, per rendere sicuri i flussi di comunicazione occorre utilizzare due SA.

1.2.19 IKE

Internet Key Exchange (IKE) [10] è il protocollo utilizzato per instaurare una Security Association offrendo la mutua autenticazione degli host che vogliono stabilire la comunicazione. IKE è stato progettato prendendo spunto da tre protocolli preesistenti:

1. Internet Security Association and Key Management Protocol (ISAKMP) [14]: fornisce una metodologia per supportare diversi tipi di scambio di chiavi.
2. Oakley [28]: descrive diverse tipologie di scambio di chiavi, specificando le peculiarità di ciascuna (garanzia Perfect Forward Secrecy, autenticazione, protezione dell'identità...)
3. SKEME [16]: descrive una tecnica di scambio di chiavi che fornisce anonimità e ripudiabilità

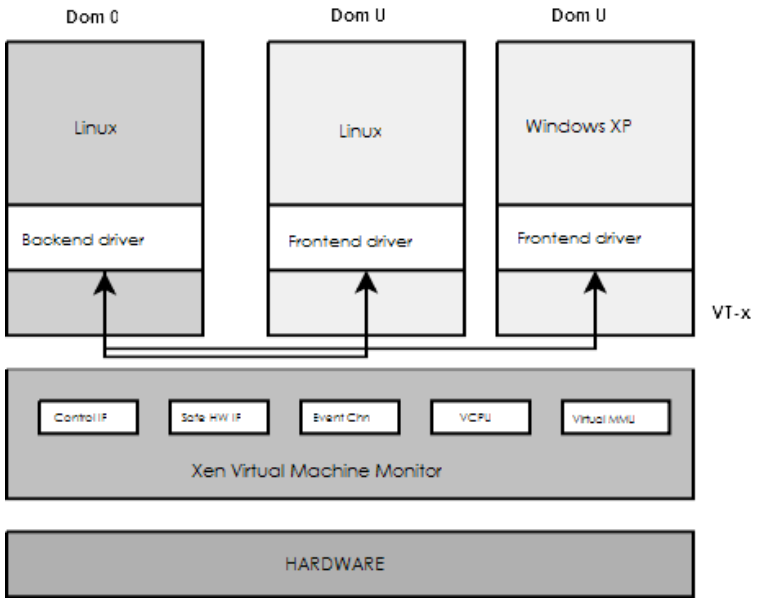

Figura 1.8: L'architettura di Xen

Il processo IKE per la creazione di una Security Association si articola in due fasi:

1. creazione di un canale sicuro autenticato fra i due nodi che vogliono comunicare. In questa fase viene utilizzato l'algoritmo Diffie Hellman per lo scambio di chiavi.

2. negoziazione dell'algoritmo, e delle relative chiavi, da utilizzare per proteggere la sessione IPsec.

1.3 Xen

Xen [44] è un Virtual Machine Monitor open source sviluppato dall'Università di Cambridge in grado di eseguire sia sistemi operativi *guest* modificati, tramite tecniche di paravirtualizzazione, sia sistemi operativi non modificati, grazie all'introduzione del supporto alle *Hardware Virtual Machine* (HVM). Il compito principale di Xen è quello di virtualizzare le risorse hardware per poterle condividere fra i sistemi operativi ospiti in esecuzione. L'ultima versione stabile di Xen è la 3.3.0.

La filosofia seguita dagli sviluppatori Xen è quella di implementare meno funzioni possibile all'interno dell'hypervisor, con l'obiettivo di sfruttare il più possibile la base di codice messa a disposizione del sistema operativo virtualizzato. Così facendo, si evita di riscrivere una notevole quantità di codice già esistente, soprattutto per quanto riguarda i driver delle varie periferiche, ed è possibile gestire una minor quantità di codice, riducendo la probabilità che errori di programmazione non siano rilevati per lungo tempo. La correttezza del codice dell'hypervisor è di cruciale importanza, in quanto una vulnerabilità a questo livello porterebbe alla compromissione di tutte le macchine virtuali in esecuzione sullo stesso nodo fisico.

1.3.1 I domini Xen

Osservando la figura 1.8 si può notare come Xen fornisca a ciascuna VM, o dominio, un ambiente di esecuzione isolato. Esistono due tipologie di dominio:

1. dominio 0 (dom0)
2. dominio U (domU), dove la U sta per unprivileged

Esiste un unico dominio 0, ed è il primo ambiente virtuale ad essere caricato all'avvio del VMM, e gode di privilegi di esecuzione elevati. Il dominio 0 in primo luogo fornisce l'interfaccia per interagire con l'hypervisor, ovvero permette di creare e gestire i domini utente. In secondo luogo, tramite i driver forniti dal sistema operativo, questo dominio si occupa della gestione delle periferiche che verranno condivise da tutte le macchine virtuali. I domU invece godono di meno privilegi, tipicamente infatti non hanno accesso diretto all'hardware sottostante, però, al contrario del dom0, possono essere migrati *a caldo* da un computer fisico ad un altro, riprendendo l'esecuzione da dove era stata interrotta.

1.3.2 L'interfaccia della macchina virtuale

Xen si occupa di virtualizzare e condividere fra i diversi domini tre componenti principali: la CPU, la memoria e i dispositivi di I/O.

1.3.3 CPU Virtuale

Una processore virtuale, benché diverso da uno fisico, è relativamente semplice da implementare: durante il normale funzionamento un processo ottiene l'utilizzo esclusivo della CPU per un breve lasso temporale, alla fine del quale avviene una commutazione di contesto, ovvero lo stato del processore viene salvato per permettere ad un altro processo di essere eseguito. Durante la commutazione di contesto è necessario che il processore sia in esecuzione in modalità supervisore; chiaramente questo non si può verificare in un sistema virtualizzato, altrimenti non sarebbe possibile garantire un effettivo isolamento dei domini virtuali. Dividiamo quindi le istruzioni che una CPU può eseguire in tre categorie:

Istruzioni privilegiate: sono le istruzioni eseguite in modalità supervisore, causano una trap se eseguite al di fuori di questa modalità. Una trap è un'eccezione sincrona che generalmente comporta il passaggio del processore alla modalità supervisore, definita anche come *kernel-mode*.

Istruzioni sensibili al controllo: sono le istruzioni che tentano di cambiare la configurazione delle risorse di sistema, ad esempio aggiornando la memoria fisica.

Istruzioni sensibili al comportamento: sono le istruzioni il cui risultato dipende dalla configurazione delle risorse presenti nel sistema.

Affinché sia possibile virtualizzare un sistema, Popek e Goldberg [29], hanno mostrato che le istruzioni del secondo tipo debbano diventare privilegiate. Per soddisfare questo requisito è necessario un meccanismo che permetta all'hypervisor di intercettare tutte le istruzioni che modificano lo stato della macchina in modo che potrebbe avere ripercussioni sugli altri domini in esecuzione.

1.3.4 Gestione della memoria

La gestione della memoria è un aspetto cruciale dell'implementazione di un VMM, avendo infatti un notevole impatto sulle prestazioni delle macchine virtuali. Ad esempio, ad ogni volta che una VM richiede l'intervento dell'hypervisor: normalmente si dovrebbe verificare un cambio di contesto, con la conseguente invalidazione del TLB, dovuto al caricamento in memoria delle pagine utilizzate da Xen. Quest'operazione comporta un notevole spreco di cicli di CPU, così è stata presa in prestito una tecnica già utilizzata da alcuni sistemi operativi: Xen riserva per se stesso i primi 64 MB dello spazio di indirizzamento di ogni processo, in modo da non causare l'invalidazione del TLB ad ogni cambio di contesto. Questa tecnica fa uso della segmentazione, il cui supporto però è venuto meno nelle CPU x86-64, nel qual caso è necessario fare affidamento su uno schema di protezione basato sui livelli delle pagine di memoria.
Sempre per perseguire l'obiettivo di ottenere buone prestazioni, Xen non implementa tecniche di swapping, bensì mette a disposizione di ogni VM il *balloon driver*. Interagendo con questo driver, un dominio virtuale può rilasciare parte della memoria non utilizzata, o fare richiesta per ottenerne una quantità maggiore, entro un limite massimo configurabile. Un altro problema relativo alla gestione della memoria è dovuto al fatto che un SO per l'architettura x86 normalmente assume che le pagine fisiche di memoria siano contigue, ma questa è una garanzia che Xen non si preoccupa di fornire ai domini virtuali, per il notevole impatto sulle prestazioni. Per risolvere il problema l'hypervisor gestisce due spazi di indirizzamento differenti: *machine-memory* e *pseudo-physical* memory.

1.3.5 Machine-memory e pseudo-physical memory

Per machine-memory si intende la memoria fisica installata sulla macchina (la memoria principale), la pseudo-physical memory è invece un livello di indirezione introdotto per dare l'illusione ai domini virtuali di avere a disposizione uno spazio di indirizzamento contiguo. La traduzione da un modello di indirizzamento all'altro è effettuata tramite l'ausilio di due tabelle:

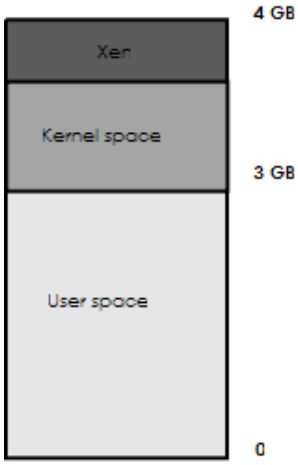

Figura 1.9: Il layout della memoria nei sistemi x86-32

- *Machine to Physical:* utilizzata dall'hypervisor per mappare la memoria fisica su quella pseudofisica, di dimensioni proporzionali alla memoria fisica
- *Physical to Machine:* contenuta in ciascun dominio virtuale e di dimensioni proporzionali alla memoria allocata dal dominio, per effettuare la traduzione inversa.

Quando si parla dunque di numero di pagina, potremmo indicare sia un *machine frame number* (MFN), ovvero il numero di pagina nello spazio di indirizzamento fisico, sia di *guest page frame number* (GPFN), utilizzato per accedere agli indirizzi lineari all'interno di un dominio virtuale.

1.3.6 Le periferiche di I/O

Per astrarre le periferiche virtuali, Xen solitamente usa tre componenti:

- Il driver vero e proprio
- Lo *split driver*
- Un ring buffer condiviso

Il driver vero è fornito tipicamente dal sistema operativo del dominio 0, e generalmente non richiede modifiche di sorta. Lo split driver è il meccanismo utilizzato da Xen per condividere i driver fra i domini virtuali, è formato da due componenti: *frontend* e *backend*. Il frontend si trova nei domU, ai quali fornisce un'interfaccia equivalente a quella del comune driver. Una volta ricevuta una richiesta dal sistema operativo, il frontend invia una richiesta al backend, situato nel dom0. Innanzitutto esso controlla che le richieste ricevute siano legittime e non vìolino il principio di isolamento fra i domini. Successivamente, il backend comunica con il driver reale per soddisfare la richiesta di I/O e, una volta che gli eventuali dati in output sono pronti, li carica nel ring buffer condiviso. Un ring buffer è implementato mediante pagine di memoria condivisa accedute utilizzando la tecnica del produttore-consumatore.

1.3.7 Event channel

Gli *event channel* sono il meccanismo che Xen mette a disposizione per la trasmissione asincrona di notifiche fra domini, e possono essere paragonati ai segnali di un tradizionale sistema operativo Unix. Ciascun dominio si preoccupa di associare una funzione *callback* che Xen invocherà per ogni interruzione rilevante. Per ogni dominio viene gestita una bitmap, un bit per ciascun tipo di evento che deve essere segnalato al sistema operativo. Inoltre, ad ogni evento è associata una flag che permette di specificare se esso debba essere mascherato o meno, analogamente a quanto accade con le interruzioni. Gli event channel sono, ad esempio, parte integrante del meccanismo degli split driver: ogni qual volta vi sono nuovi dati a disposizione di un dominio nel ring buffer, viene generato un evento.

1.3.8 XenStore

XenStore è un meccanismo di memorizzazione condivisa, simile ad un normale filesystem gerarchico, nel quale sono presenti directory e file, che prendono il nome di *chiavi*. Ogni chiave ha un valore associato, generalmente rappresentato da una breve stringa. Generalmente il contenuto delle chiavi riguarda la configurazione delle macchine virtuali. Lo store è gestito dal dominio 0 tramite un demone in user-space, così

come nei vari domU è possibile accedervi sempre grazie programmi ad-hoc. Le VM accedono allo store tramite pagine di memoria condivisa, mentre la notifica della presenza di dati aggiornati avviene tramite il meccanismo degli event channel. L'indirizzo della pagina di memoria condivisa si trova in una particolare struttura dati di Xen, la *shared info page*.

1.3.9 Grant table

Le grant table sono il meccanismo che Xen utilizza per permettere il trasferimento e la condivisione di pagine di memoria fra macchine virtuali. Le pagine condivise sono identificate da un intero, chiamato *grant reference*. Nel caso di condivisione di una pagina, essa viene inserita nello spazio di indirizzamento del dominio che richiede la condivisione, oltre che in quello che la offre. Al contrario, nel caso di trasferimento, la pagina viene rimossa dallo spazio di indirizzamento del dominio d'origine. L'operazione di mapping viene utilizzata per la creazione di aree di memoria condivisa fra le VM, mentre l'operazione di trasferimento viene comunemente utilizzata dal balloon driver, per permettere al dominio 0 di trasferire pagine di memoria ai domini guest. L'operazione di mapping viene detta di tipo pull: una volta che un dominio offre la condivisione di una o più pagine, un altro potrà effettivamente mapparle nel proprio spazio di indirizzamento. Il trasferimento è invece un'operazione di tipo push: il dominio che invoca l'operazione di trasferimento invia la pagina ad un altro dominio. Nel caso del trasferimento è comunque necessario che il dominio ricevente segnali la sua disponibilità a ricevere una pagina attraverso la creazione di una riga nella grant table.
L'operazione di trasferimento è indicata se si devono spostare grandi quantità di dati, dato che ha un costo quasi costante, in quanto richiede solamente qualche aggiornamento alla tabella delle pagine. Tale costo è pero relativamente alto, per questo, nel caso di piccoli trasferimenti, la copia delle pagine potrebbe essere la scelta più indicata. Uno degli usi più comuni delle pagine di memoria condivisa è certamente l'implementazione dei ring buffer, struttura dati alla base delle periferiche virtuali di Xen. La struttura della grant table è molto semplice, ogni riga contiene solamente tre elementi:

1. domid: l'identificativo del dominio a cui vengono concessi i privilegi relativi alla pagina da condividere.

2. frame: l'indirizzo della pagina da condividere o trasferire.

3. flags: un intero di 16 bit diviso in due parti, la prima indica la tipologia della condivisione (mapping o trasferimento), la seconda indica invece alcune proprietà aggiuntive in base al tipo di condivisione. Ad esempio nel caso del mapping, è possibile specificare se la condivisione sarà in sola lettura, o sia in lettura che in scrittura.

1.4 Strumenti di monitoraggio

Una volta costruita un'adeguata infrastruttura di rete è necessario effettuare continui controlli per scoprire al più presto eventuali problemi, per intervenire e ristabilire il corretto funzionamento del sistema. Si presenta pertanto la necessità di creare un'efficiente infrastruttura dedita al monitoraggio, operazione che non si riduce alla mera installazione di un certo numero di strumenti, bensì richiede un attento studio e una pianificazione dettagliata. Ad esempio, è facile sottovalutare gli effetti che l'introduzione di un sistema di monitoraggio avrà sull'infrastruttura esistente; presentiamo di seguito un breve elenco delle principali problematiche da affrontare durante la progettazione di un sistema di monitoraggio:

- l'ampiezza di banda utilizzata dagli strumenti di monitoraggio
- la richiesta in termini di risorse hardware
- i sistemi critici per il funzionamento dell'infrastruttura di monitoraggio
- il sistema di monitoraggio introduce nuove problematiche di sicurezza, ad esempio trasmettere sulla rete informazioni sensibili in chiaro
- se è possibile eventuali guasti dovessero interessare proprio i sistemi adibiti al monitoraggio

1.4.1 Esecuzione dei controlli

Un ulteriore aspetto da pianificare attentamente è decidere dove avranno luogo i controlli:

- ogni singolo nodo monitorato esegue autonomamente i controlli, per poi comunicare i risultati ad un server centrale, oppure
- un server esegue i controlli remoti su tutti i nodi monitorati

La seconda soluzione permette di non aumentare eccessivamente il carico della rete monitorata, molto importante nel caso di server che richiedano elevate prestazioni, ma non sempre è una strada percorribile, in quanto il numero di nodi da monitorare potrebbe essere molto grande. È possibile, dunque, scegliere la prima strada, oppure si può utilizzare un sistema di monitoraggio distribuito formato da più server che si spartiscono il carico di lavoro. Questa soluzione è particolarmente adatta quando la rete da monitorare occupa un'area geografica molto vasta. Particolare attenzione dovrà essere rivolta alla scelta di dove collocare i sistemi di monitoraggio all'interno della rete; a tal proposito è necessario:

- non alterare le misure di sicurezza esistenti
- minimizzare l'impatto sulla rete
- ridurre al minimo le dipendenze fra i sistemi di monitoraggio e i sistemi di importanza critica

Di fondamentale importanza è la scelta dei controlli da eseguire, in quanto un numero eccessivo non farà altro che produrre una mole di dati ingestibile, dalla quale risulterà impossibile estrarre informazioni rilevanti. Infine è bene tener presente che, come tutti i sistemi informatici, anche l'infrastruttura di monitoraggio potrà incorrere in guasti e malfunzionamenti, e sarà dunque necessario valutare la possibilità di introdurre un sistema di monitoraggio secondario.
Di seguito analizzeremo due degli strumenti di monitoraggio più noti: Nagios e ntop.

1.4.2 Nagios

Nagios [27] è un software in grado di monitorare reti e computer. Questa definizione è in apparenza fuorviante perché in realtà Nagios non è in grado di monitorare. Infatti Nagios è un framework che permette di schedulare l'esecuzione di programmi, chiamati plugin, che si occupano del monitoraggio vero e proprio. Il punto di forza di questa scelta progettuale risiede nel fatto che i plugin possono essere scritti in un qualunque linguaggio di programmazione, garantendo un'elevata adattabilità ai sistemi più disparati. Nagios si occupa inoltre di tenere traccia delle diverse esecuzioni dei vari plugin, permettendo così di elaborare uno storico del funzionamento dei sistemi monitorati.

1.4.3 Host e servizi

L'ambiente da monitorare viene visto come un insieme di host e servizi. Per host si intende un'entità fisica in grado di comunicare utilizzando il TCP, ad esempio un comune server, un sensore ambientale o una macchina del caffè... I servizi sono invece intesi come le entità logiche fornite da un host, ad esempio un server web. Nagios permette di configurare un singolo controllo per ciascun host e molteplici controlli di servizio per ciascun host. Per ovvi motivi, nel caso un host non dovesse rispondere, tutti i controlli relativi ai servizi di tale host non verranno eseguiti.

1.4.4 Architettura di Nagios

Nagios è altamente configurabile, e permette di decidere i controlli da eseguire su ciascun host che si intende monitorare. Questo è reso possibile dal *Nagios Remote Plugin Executor* (NRPE). L'NRPE è formato da due parti (vedi figura 1.10):

- *check_nrpe*: un plugin eseguito localmente da Nagios che si occupa di richiedere l'esecuzione dei plugin remoti
- un *demone* in esecuzione su ciascuna macchina monitorata, che si occupa di eseguire i plugin configurati e di restituire i risultati a check_nrpe

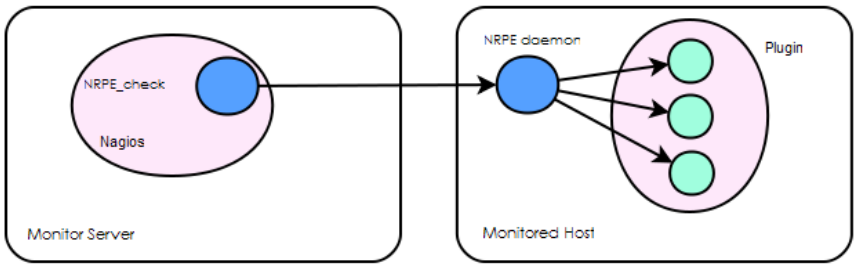

Figura 1.10: L'architettura di Nagios

La comunicazione fra le due parti può essere eventualmente autenticata e crittografata, nel caso debba attraversare una rete non fidata.

1.4.5 Ntop

Ntop è un software open source multi-piattaforma nato come strumento di misurazione e monitoraggio del traffico di rete. L'autore di ntop sentiva la necessità di avere un tool simile a top [6] che fosse in grado di misurare il traffico di rete e riportare informazioni sui pacchetti catturati. Mano a mano che la base di utenti si è ampliata, ntop si è evoluto sino a diventare uno strumento flessibile ed estendibile a seconda delle necessità.

1.4.6 Architettura di ntop

Ntop si può suddividere in quattro componenti principali (vedi figura 1.11):

- *packet sniffer*: basato sulle librerie pcap, cattura il traffico di rete dalle interfacce di rete specificate
- *packet analyser*: elabora i pacchetti catturati cercando di correlarli in base al flusso di appartenenza (ad esempio una connessione TCP), calcola statistiche relative ai singoli host e cerca di costruire una topologia di rete mantenendo una lista dei router e dei server DNS rilevati
- *report engine*: contiene un server http/https di dimensioni contenute, che permette agli utenti di visualizzare i rapporti sul traffico analizzato tramite un browser

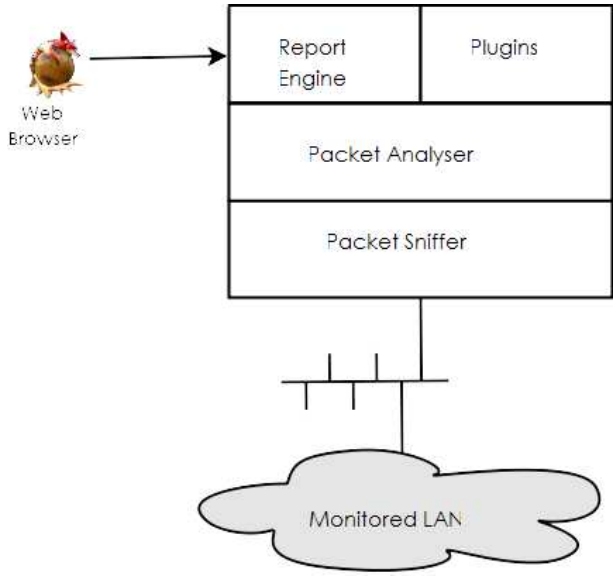

Figura 1.11: L'architettura di ntop

- *plugin*: utilizzati per estendere le funzionalità di ntop senza dover mettere mano al codice principale. Alcuni plugin permettono ad esempio di effettuare un'analisi dettagliata di protocolli quali ARP e ICMP

1.4.7 Misurazione del traffico

Per ogni host, ntop registra le seguenti informazioni:

- il traffico totale classificato in base al protocollo di rete
- statistiche sui pacchetti multicast IP
- uno storico delle sessioni TCP complete di sorgente, destinazione, durata, dati ritrasmessi...
- i servizi messi a disposizione dai vari host (server web, DNS...)
- percentuale del traffico generato da un determinato nodo in relazione al traffico totale della rete

- numero totale dei pacchetti trasmessi ordinati in base alla dimensione e alla tipologia (unicast, multicast, broadcast)
- statistiche sull'utilizzo dei diversi protocolli, organizzate anche in base alla tipologia della destinazione (host locale o remoto)
- analisi dei flussi di rete in base alle regole definite dall'utente

Avendo a disposizione tutte queste informazioni è possibile comprendere al meglio l'utilizzo che viene fatto della rete monitorata. Ad esempio, conoscendo la fascia oraria in cui si verifica il picco di traffico, è possibile pianificare al meglio le operazioni di manutenzione. In questo modo invece di un costoso aumento dell'ampiezza di banda a disposizione si potrà più semplicemente alleggerire il carico di rete spostando determinate operazioni nella fascia oraria meno sfruttata. Ntop rende inoltre possibile la rilevazione di servizi mal configurati.

1.4.8 Ntop come strumento di sicurezza

Grazie alle sue capacità di monitoraggio e all'espandibilità tramite l'architettura a plugin, ntop può essere utilizzato come un basilare *Network Intrusion Detection System* (NIDS), configurabile attraverso un semplice linguaggio di definizione regole. Vediamo quindi alcune delle principali attività che è possibile rilevare:

- portscan: sia il classico, e rumoroso, portscan basato sull'invio di un pacchetto a ciascuna porta, sia gli stealth scan, che si limitano ad esempio all'invio di un singolo pacchetto con la flag SYN impostata
- spoofing grazie al plugin arpWatch
- schede di rete configurate in modalità promiscua per catturare tutto il traffico in transito sul relativo segmento di rete
- utilizzo di scanner per vulnerabilità, come Nessus ad esempio, cosa che solitamente costituisce il preludio ad un tentativo di attacco

- cavalli di troia sui computer della rete
- attacchi di tipo DOS analizzando le statistiche relative al traffico di rete
- pacchetti sospetti: spesso per cercare di sfruttare eventuali bug presenti nello stack TCP/IP degli apparati di rete, un attaccante invia particolari pacchetti creati ad-hoc. Un esempio possono essere i pacchetti TCP con le flag SYN/FIN ma che non appartengono ad alcuna connessione esistente

1.5 Monitoraggio di sistemi Xen

Abbiamo visto come uno dei principali vantaggi dell'utilizzare macchine virtuali sia la possibilità di consolidare più macchine fisiche su un unico server. Una volta costruita un'infrastruttura virtuale per ottenere un elevato livello di prestazioni è bene monitorare l'esecuzione delle VM per costruire un profilo sufficientemente dettagliato dell'utilizzo delle risorse a disposizione. I vantaggi di un sistema di monitoraggio di questo tipo sono molteplici:

- È possibile implementare politiche di diverso tipo per l'allocazione delle risorse fra le VM.
- Permette di capire se sia possibile o meno aggiungere un'ulteriore macchina virtuale su un determinato server fisico.
- Fornisce informazioni utili per decidere quando sia il caso di effettuare la migrazione di una macchina virtuale.
- Permette di garantire una buona qualità del servizio (QoS) a tutte le macchine virtuali in esecuzione.

1.5.1 XenMon

Nel caso di Xen un sistema di monitoraggio di questo tipo è costituito da XenMon [9]. XenMon è formato da tre componenti principali (vedi figura 1.12):

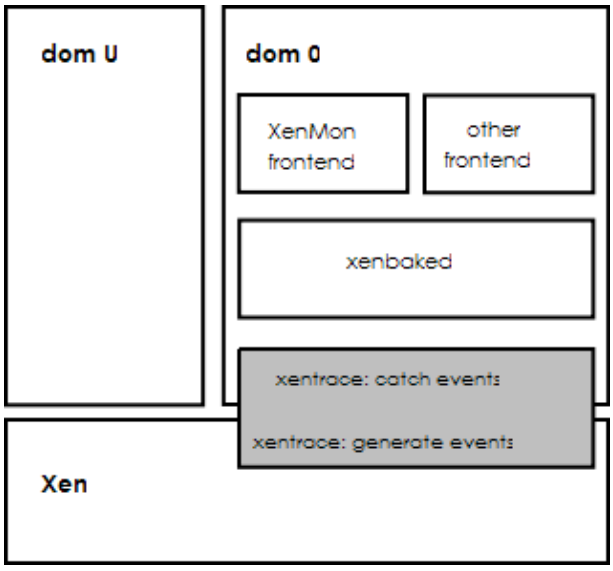

Figura 1.12: L'architettura di XenMon

- *Xentrace*: è un modulo che si occupa di lanciare eventi. È possibile lanciare eventi in punti arbitrari all'interno dell'hypervisor, inoltre xentrace permette di associare alcuni parametri agli eventi sollevati. Ad esempio, per l'evento "dominio schedulato" come parametri possiamo avere l'ID del dominio schedulato e la marca temporale dell'evento. Questo modulo è composto da due parti: una, che si fa carico di generare gli eventi, che si trova nel VMM, e un'altra, che si fa carico di raccogliere gli eventi generati, situata nel dominio 0.

- *Xenbaked*: dato che il flusso grezzo di eventi generato da xentrace non è di per sé molto significativo, è necessario uno strumento eseguito in spazio utente che si occupi di rielaborare tali informazioni per fornire un output di facile comprensione. Ad esempio partendo dagli eventi di schedulazione di un dominio è possibile calcolare il tempo in cui una VM è rimasta bloccata.

- *Xenmon*: è il front-end che si occupa della visualizzazione e del salvataggio dei dati raccolti. Il suo utilizzo comporta un peggioramento delle prestazioni di circa l'1-2%, overhead più che accettabile.

1.5.2 Metriche

XenMon si occupa di raccogliere dati in tre intervalli di tempo: il periodo di esecuzione, l'ultimo secondo trascorso e gli ultimi 10 secondi. Per capire come un determinato dominio utilizzi il tempo vi sono a disposizione tre metriche:

- *CPU usage*: la percentuale di tempo, su un determinato intervallo, che un dominio trascorre utilizzando il processore (riga "Gotten" nella figura).
- *Blocked time*: la percentuale di tempo che un dominio è bloccato in attesa di una qualche operazione di I/O.
- *Waiting time*: la percentuale di tempo che un dominio è nella coda di pronto in attesa di essere schedulato per l'utilizzo del processore.

Per analizzare approfonditamente il comportamento dello scheduler delle macchine virtuali, è possibile combinare il tempo di utilizzo e attesa della CPU con una terza metrica: *l'execution count*. Tale metrica permette di sapere quanto spesso un determinato dominio sia stato schedulato per l'esecuzione durante un certo intervallo temporale. Infine XenMon mette a disposizione una metrica per la valutazione del carico di I/O: *l'I/O count metric*. Questo contatore tiene traccia del numero di richieste di scambio di pagine fra un dominio utente e il dominio 0, e, pur non fornendo un conteggio accurato del numero di byte scambiati fra tali domini, è comunque utilizzabile per ottenere una buona stima dell'ordine di grandezza del numero di operazioni di I/O richieste dal dominio virtuale.

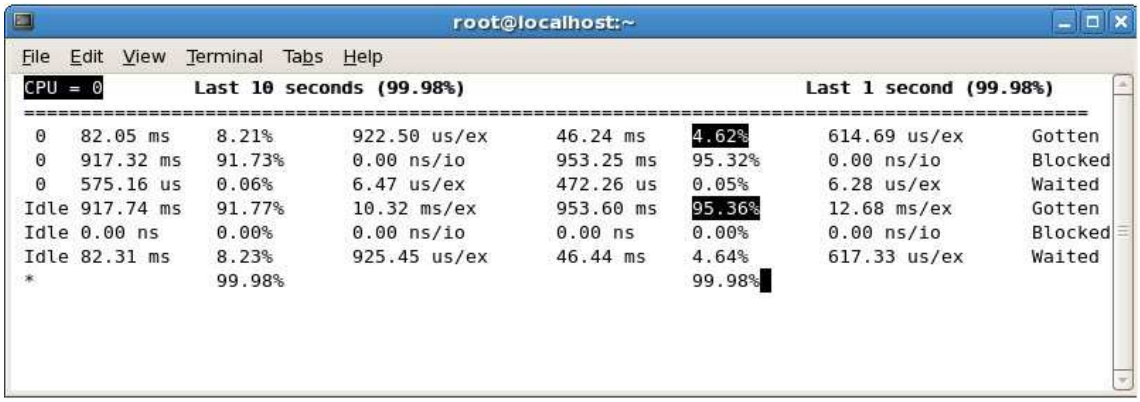

Figura 1.13: L'output fornito da XenMon

Capitolo 2

Hypercall

Questo capitolo, dopo aver introdotto il funzionamento delle syscall, introduce il meccanismo delle hypercall utilizzato da Xen e descrive brevemente le funzionalità offerte da ciascuna hypercall. Successivamente, viene introdotto il concetto di sense of self nell'ambito dei sistemi di rilevamento intrusioni. Infine, il capitolo presenta il lavoro svolto per cercare di costruire un sense of self per le macchine virtuali Xen.

2.1 Hypercall

Abbiamo visto che con l'utilizzo di Xen il kernel del sistema operativo viene spostato dall'anello di privilegio 0 all'anello 1, dove non dispone di sufficienti privilegi per il normale funzionamento. Per risolvere questo problema, gli sviluppatori di Xen si sono ispirati al meccanismo delle chiamate di sistema (*system call* o *syscall*). Le hypercall costituiscono, dunque, l'interfaccia tramite cui una macchina virtuale può interagire con Xen.

2.1.1 Syscall

In un comune sistema operativo non emulato, il codice utente viene eseguito a livello di privilegio 3. Oltre a limitare lo spazio d'indirizzamento del processo, ciò non concede l'interazione diretta con l'hardware sottostante, privilegio garantito esclusivamente al

kernel. Le chiamate di sistema sono l'interfaccia tramite cui un processo utente può richiedere al kernel un'operazione per cui il processo non disponga di sufficienti diritti per eseguirla nel proprio flusso di esecuzione.
Vediamo cosa accade quando avviene una chiamata di sistema:

1. un programma utente invoca una syscall, solitamente tramite una funzione di libreria che incapsula la chiamata vera e propria
2. gli argomenti della syscall vengono caricati in alcuni registri del processore o sullo stack
3. viene generata un'eccezione
4. il processore passa in modalità di esecuzione privilegiata e il flusso di esecuzione si sposta all'interno del kernel, precisamente all'interno del codice per la gestione delle eccezioni
5. il kernel esegue il codice relativo alla chiamata di sistema effettuata
6. il processore ritorna alla modalità di esecuzione non privilegiata
7. il flusso di esecuzione ritorna al processo che aveva invocato la chiamata

Sui sistemi x86 le operazioni del passo 2 consistono tipicamente nel caricare il numero della chiamata di sistema che si vuole invocare nel registro eax, e successivamente invocare l'eccezione, o l'interruzione software, 0x80. Attualmente, le CPU forniscono anche delle istruzioni apposite per implementare le cosiddette *fast system call* che, nel caso di processori Intel, prendono il nome di SYSCALL e SYSRET [11], mentre nel caso di AMD si chiamano SYSENTER e SYSEXIT [1]. Tali istruzioni permettono il trasferimento rapido del controllo al kernel per l'esecuzione di una chiamata di sistema, senza il costo addizionale dovuto al sollevamento di un'eccezione.

2.1.2 Invocazione di una hypercall

Il meccanismo utilizzato da un sistema operativo emulato per invocare un'hypercall, è molto simile a quello utilizzato normalmente per invocare una syscall: il kernel guest,

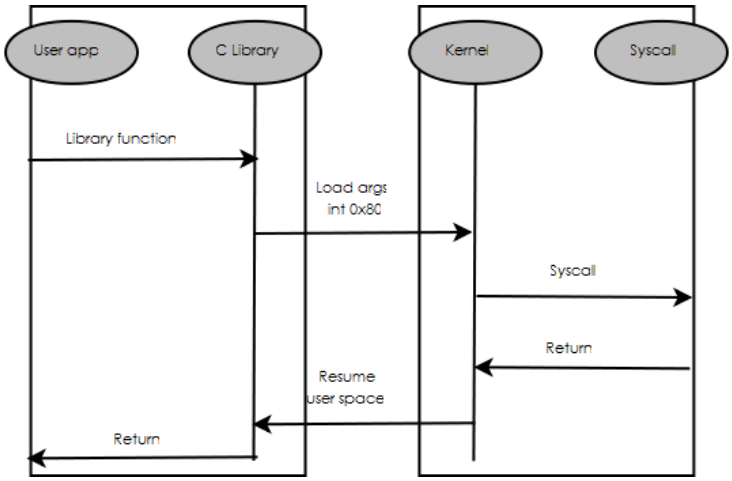

Figura 2.1: Il flusso di esecuzione di una syscall

dopo aver caricato gli argomenti dell'hypercall nei registri del processore, solleva l'eccezione 0x82, anziché la 0x80. A partire dalla versione 3 di Xen, questo meccanismo è sconsigliato e sostituito da un altro ben più efficiente: durante la fase di caricamento del SO guest, all'interno del suo spazio di indirizzamento viene mappata una particolare pagina di memoria, detta appunto *hypercall page*. Quando il kernel deve invocare una hypercall, non deve fare altro che accedere ad un preciso indirizzo all'interno di tale pagina, utilizzando l'istruzione assembler CALL. Dato che nell'architettura x86 la dimensione di una pagina di memoria è di 4 KB, Xen può mettere a disposizione delle VM un massimo di 128 hypercall. Attualmente ne sono implementate 37 [45]. È importante notare come non vi sia una corrispondenza biunivoca fra le chiamate di sistema e le hypercall, questo sia perché evidentemente le syscall sono un numero molto maggiore, attualmente più di 300 in un kernel Linux 2.6 [23] sia perché l'hypervisor implementa un insieme minimo di funzionalità perché i SO guest possano funzionare. Nel caso la macchina virtuale utilizzi le estensioni hardware per il supporto alla virtualizzazione, il kernel ha accesso allo stesso insieme di hypercall, ma l'implementazione di una chiamata è leggermente diversa. Abbiamo visto come in una macchina virtuale paravirtualizzata specifica la locazione dove la pagina delle hypercall debba essere caricata nelle intestazioni ELF. Questo non è chiaramente possibile nel caso di un guest HVM, e occorre quindi utilizzare un approccio diverso. Innanzitutto, il guest

deve capire a tempo di esecuzione se è in esecuzione sotto Xen, successivamente potrà ricavare l'indirizzo della pagina contenente le hypercall. Per quanto riguarda l'implementazione di una chiamata, invece, non è possibile utilizzare le eccezioni, perché queste vengono trasmesse direttamente al kernel guest, e non a Xen come nel caso di VM paravirtualizzate. Pertanto, le hypercall vengono trasmesse tramite l'istruzione assembler VMEXIT.

2.1.3 Le hypercall di Xen

Vediamo ora una descrizione delle hypercall messe a disposizione dei sistemi operativi guest:

- *set_callbacks*: le interruzioni sono virtualizzate dal VMM che le mappa su alcuni event channel. Tramite questi canali vengono consegnati messaggi al dominio di destinazione in maniera asincrona, utilizzando le callback fornite come parametro di questa hypercall. Durante la fase di boot, una delle prime operazioni effettuate dal SO guest è proprio quella che registra le funzioni di callback.

- *vcpu_op*: utilizzata per inizializzare una CPU virtuale, attivarla, disattivarla e controllarne lo stato.

- *iret*: normalmente un processo può essere messo in stato di attesa perché il processore ha ricevuto un'interruzione, che richiede l'esecuzione di una routine di gestione. Terminata tale procedura, il controllo della CPU viene restituito al processo interrotto utilizzando l'istruzione assembler IRET. Come abbiamo visto, in Xen le interruzioni sono implementate via software tramite gli eventi. Per questa ragione, è necessario implementare un'hypercall equivalente all'istruzione assembler.

- *vm_assist*: permette al SO guest di specificare quali *assistenze* richiedere al VMM. Sono disponibili tre tipologie di assistenza: 1) supporto all'emulazione per le istruzioni che si aspettano di operare su segmenti di memoria di 4 GB 2) specifica di callback nel caso sia utilizzata la precedente assistenza 3) permette di attivare/disattivare la tabella delle pagine scrivibile.

- *memory_op*: permette di eseguire operazioni relative alla memoria a disposizione di una macchina virtuale: 1) richiedere un aumento della memoria a disposizione 2) rilascio di parte della memoria a disposizione 3) richiesta dell'indirizzo più alto della memoria a disposizione 4) richiesta dello quantità di memoria attualmente a disposizione 5) richiesta della massima quantità di memoria che il dominio può utilizzare.

- *mmu_update*: un dominio guest, che non è in grado di modificare autonomamente la propria tabella delle pagine, utilizza questa hypercall per richiedere l'intervento del dominio 0 per aggiornare la tabella delle pagine o la tabella di mappatura machine to physical. Questa hypercall permette di eseguire più modifiche in una sola volta.

- *mmuext_op*: permette di eseguire alcune operazioni relative alla MMU che non siano un semplice aggiornamento della tabella. Ad esempio, è possibile caricare un certo valore nel registro di controllo cr3, richiedere un flush della cache, richiedere un flush del TLB, effettuare il *pinning* e l'*unpinning* di righe della tabella delle pagine. Alcune di queste operazioni possono essere eseguite solamente da un dominio con privilegi adeguati.

- *vm_assist*: utilizzata per attivare/disattivare le varie modalità di gestione della memoria.

- *set_segment_base*: utilizzata per impostare il valore di alcuni *Machine Specific Register* (MSR) nel caso di processori di architettura AMD64 che non offrono il supporto per la segmentazione.

- *update_va_mapping*: è simile alla hypercall mmu_update, ma permette di modificare un singolo elemento della tabella delle pagine. È utile, ad esempio, quando un programma eseguito in spazio utente effettua una malloc. È importante notare che sia nella mmu_update che nella update_va_mapping, Xen verifica che le modifiche richieste di un dominio siano sicure, ovvero che non vadano a modificare lo spazio di indirizzamento di un altro dominio.

- *update_va_mapping_otherdomain*: è funzionalmente identica alla precedente, ma permette ad un dominio dotato di sufficienti privilegi di manipolare la tabella delle pagine di un altro dominio.

- *set_gdt*: permette ad un dominio di manipolare la sua *Global Descriptor Table* (GDT). Anche in questo caso, Xen verifica che gli indirizzi da modificare siano validi, e non appartengano ad un altro dominio.

- *grant_table_op*: utilizzata per trasferire o condividere singole pagine di memoria fra domini in maniera sicura, attraverso il meccanismo delle tabelle dei privilegi (grant table).

- *set_trap_table*: utilizzata da un dominio guest per installare una tabella per la gestione delle interruzioni virtuali (*Interrupt Descriptor Table* IDT). Ciò permette alla VM di gestire autonomamente le interruzioni e le chiamate di sistema, senza coinvolgere l'hypervisor.

- *suspend*: utilizzata per invocare la sospensione di un dominio, il cui stato viene serializzato su una periferica di memorizzazione di massa, in modo da poterne ripristinare successivamente l'esecuzione. È interessante notare come sia possibile riprendere l'esecuzione di una macchina virtuale anche su una macchina fisica differente.

- *sched_op*: normalmente, Xen schedula le VM, ma questa hypercall permette ai domini di interagire con lo scheduler in 4 modi: 1) **yield**: rilascia volontariamente la CPU 2) **block**: rilascia volontariamente la CPU segnalando allo scheduler che il dominio non dovrà essere rimesso in esecuzione fino alla consegna di un evento 3) **shutdown**: pone fine all'esecuzione di una macchina virtuale, segnalandone il motivo (reboot, sospensione, crash) 4) **sched_poll**: simile alla block, ma permette di specificare un insieme di eventi che il dominio vuole attendere. Questo comando permette di specificare un timeout opzionale e, inoltre, richiede la disabilitazione degli eventi per la VM invocante.

- *sched_op_compat*: simile alla precedente ma senza il supporto per il comando sched_poll. È mantenuta per motivi di compatibilità, come si evince dal nome.

- *set_timer_op*: imposta un timeout, allo scadere del quale viene lanciato un evento di tipo timer.

- *multicall*: in certi momenti dell'esecuzione un dominio guest deve utilizzare funzioni ad alto livello che richiedono sempre la stessa sequenza di hypercall. Ad esempio, questo avviene ogni qualvolta il SO virtuale deve effettuare il cambio di contesto per eseguire un processo differente. Questa hypercall accetta un array contenente le chiamate da eseguire, che possono così essere portate a termine con una singola transizione della CPU dal dominio guest all'hypervisor (dal ring 1 o 3 al ring 0).

- *physdev_op*: solitamente un sistema operativo cerca di accedere al bus PCI per inferire la configurazione delle periferiche installate. Per ovvi motivi di sicurezza, non è possibile permettere ad un dominio guest di effettuare tale operazione. Per superare questo vincolo, Xen mette quindi a disposizione delle VM questa chiamata.

- *fpu_taskswitch*: cerca di migliorare le prestazioni dell'operazione di context switch permettendo la modifica lazy dei registri in virgola mobile della CPU. Invocando questa chiamata Xen imposta a 1 il bit *Task Switched* (TS) nel registro di controllo cr0. In questo modo il prossimo tentativo di utilizzare operazioni in virgola mobile causerà un'ecczione che potrà essere intercettata dal sistema operativo guest, il quale salverà/ricaricherà lo stato dei registri FP e rimetterà a 0 il valore del bit TS. Questa chiamata è utilizzata come ottimizzazione perché è comunque sempre possibile salvare e ricaricare il contenuto dei registri FP ad ogni cambio di contesto.

- *stack_switch*: questa chiamata deve essere utilizzata ad ogni cambio di contesto per richiedere all'hypervisor l'aggiornamento dello stack pointer del kernel.

- *event_channel_op*: permette di gestire il meccanismo di notifica asincrona implementato dagli event channel. Questa chiamata prevede un parametro per specificare il tipo di operazione richiesta: allocazione di un canale su cui attendere connessioni, associazione di una porta allocata ad un IRQ virtuale, associazione di una porta allocata ad un IRQ fisica, creazione di un event channel interdominio, chiusura di un event channel, invio di un evento su un canale, richiesta dello stato di un canale.

- *acm_op*: può essere utilizzata esclusivamente dal dominio 0 per la gestione dell'*Access Control Module* (ACM).

- *nmi_op*: si occupa della registrazione/deregistrazione del gestore di un'interruzione non mascherabile.

- *xenoprof_op*: utilizzata per gestire il profiling di Xen.

- *callback_op*: permette di registrare diversi tipi di callback, ad esempio una da richiamare nel caso venga sollevata un'eccezione non mascherabile.

- *domctl*: permette, ad un dominio che goda di privilegi sufficienti, tipicamente il dominio 0, di eseguire operazioni di controllo relativamente ad altri domini. I comandi permettono, ad esempio, di creare e distruggere domini, mettere in pausa o riprendere l'esecuzione di una VM, modificare la priorità di schedulazione di un dominio, ottenere informazioni relative alla memoria di un dominio o alle CPU virtuali assegnate ad un dominio.

- *platform_op*: permette al kernel in esecuzione nel dominio 0 di eseguire alcune operazioni messe a disposizione dalla piattaforma hardware in uso. Ad esempio è possibile impostare l'orologio di sistema, mettere il computer in modalità sleep, cambiare la frequenza del processore.

- *xen_version*: restituisce la versione di Xen attualmente in esecuzione.

- *console_io*: permette di interagire con la console di Xen, sia in scrittura che in lettura. Utilizzata a scopo di debugging.

- *set_debugreg*: utilizzata per il debugging di Xen, permette di inserire un certo valore nel registro di debugging specificato.
- *get_debugreg*: complementare alla precedente, permette di ottenere il valore contenuto nel registro di debugging specificato.
- *hvm_op*: utilizzata per fornire supporto ad alcune operazioni nel caso di una VM che utilizzi le estensioni della CPU per la virtualizzazione.
- *kexec_op*: in un normale kernel Linux la chiamata di sistema sys_kexec permette di avviare un altro kernel senza riavviare fisicamente la macchina, ovvero saltando la fase di avvio che coinvolge BIOS e boot loader. Questa hypercall è un'estensione di tale chiamata di sistema, e permette di passare da un kernel *vanilla* ad uno Xen, e viceversa, ma anche da un kernel Xen ad un altro kernel Xen.
- *update_descriptor*: utilizzata da un sistema operativo guest per modificare un singolo descrittore di segmento nella LDT o nella GDT. Xen verifica che le modifiche richieste siano valide. Questa hypercall è conveniente da utilizzare nel caso in cui il numero di descrittori da modificare sia piccolo, nel caso contrario sarà preferibile ricaricare completamente le tabelle.

2.2 Il sense of self dei processi Unix

In [7] viene presentato un metodo innovativo per rilevare intrusioni che cerca di trasferire in ambito informatico quelle che sono le peculiarità del sistema immunitario degli esseri viventi. La caratteristica principale del sistema immunitario è la capacità di distinguere fra le strutture endogene o esogene che non sono pericolose per l'organismo, e che quindi possono o devono essere preservate (*self*), e le strutture endogene o esogene nocive e che devono quindi essere eliminate (*non-self*). Vi sono però anche altri aspetti dei sistemi immunitari che contribuiscono alla progettazione di una robusta infrastruttura di sicurezza: 1) la rilevazione degli agenti estranei è distribuita 2) la rilevazione è probabilistica e in tempo reale 3) il sistema immunitario è virtualmente

in grado di rilevare tutte le possibili entità estranee, anche quelle mai incontrate prima. Nel progetto di un sistema di questo tipo è inoltre necessario calibrare con precisione la sensibilità del sistema; un sistema troppo sensibile rileverà problemi anche laddove non ve ne siano (falsi positivi), finendo così per essere ignorato. Al contrario, un sistema troppo poco sensibile, ovvero prono ai falsi negativi, sarebbe ancora più dannoso, in quanto classificherebbe come lecite attività in realtà nocive per l'integrità del sistema da proteggere.
Gli autori dell'articolo sono giunti così all'elaborazione di un sense of self basato su insiemi di brevi sequenze composte dalle chiamate di sistema effettuate dai processi privilegiati in esecuzione. Per ogni processo monitorato si costruisce un database di sequenze di quello che viene considerato come il normale comportamento. Una volta terminata la fase di apprendimento le sequenze rilevate vengono confrontate con quelle contenute nel database, e qualora non siano presenti vengono considerate come anomale, facendo scattare un qualche tipo di contromisura. È importante sottolineare come delle sequenze monitorate non si tenga conto né dei parametri passati alle chiamate di sistema, né di informazioni temporali, rendendo i sistemi di questo tipo passibili di attacchi *mimicry* [42].

Vediamo adesso il funzionamento dell'algoritmo più dettagliatamente. Innanzitutto è necessario decidere la lunghezza delle sequenze da inserire nel database; gli autori hanno sperimentalmente verificato che sequenze di lunghezza k = 6 sono un buon compromesso fra efficacia del sistema e un database dalle dimensioni accettabili. Una volta deciso il parametro k, inizia la fase di apprendimento durante la quale si costruirà il database contenente il comportamento normale del processo monitorato. A questo proposito è possibile seguire due strategie diverse:

- monitorare il programma durante il funzionamento in un ambiente controllato
- monitorare il programma durante il funzionamento in ambito di produzione

Nel primo caso si ha la certezza che il comportamento del programma sia un sottoinsieme di quello che può essere definito come comportamento normale; potrebbe essere però complesso riuscire a generare un insieme di tracce che copra la maggior

parte dei comportamenti accettabili, ciò porta a generare un elevato numero di falsi positivi quando il processo monitorato è eseguito al di fuori dell'ambiente di test. Al contrario, nel secondo caso si ha un'elevata probabilità di costruire un database delle tracce che sia una buona approssimazione del comportamento normale. Lo svantaggio è che nel database potrebbero essere inserite anche sequenze corrispondenti ad un comportamento anomalo, niente ci assicura infatti che durante la fase di apprendimento il processo monitorato non venga sottoposto ad attacchi, o che si trovi in situazioni non pericolose, ma comunque non classificabili come normali, e.g. lo spazio su disco è terminato.
Una volta terminata la fase di apprendimento, inizia quella di monitoraggio vero e proprio, durante la quale si confrontano le sequenze generate dal processo monitorato, con quelle presenti nel database. Le sequenze che non vengono trovate nella base di dati prendono il nome di *mismatches*, riscontri negativi. Maggiore il numero di mismatches, maggiore la probabilità di aver riscontrato un'anomalia nel comportamento del processo monitorato. Il confronto fra due sequenze i e j è basato sulla *distanza di Hamming* d(i,j), definita come il numero di elementi differenti fra le due sequenze i e j. Per ogni sequenza generata si calcola la minima distanza rispetto a tutte le sequenze contenute nel database:

$$d_{min}(i) = \min \{d(i,j) \text{ per tutte le sequenze normali } j\}$$

Per limitare il numero di falsi positivi è possibile impostare una soglia C tale che:

$$d_{min}(i) \geq C$$

con $1 \leq C \leq k$. Grazie a questa soglia verranno segnalate come anomale solamente le sequenze sufficientemente differenti dalle sequenze normali.

2.2.1 Alla ricerca di un sense of self per Xen

Ispirandosi all'idea appena descritta, si è pensato di provare ad effettuare lo stesso tipo di analisi alle sequenze di hypercall che il kernel delle macchine virtuali effettuano nei confronti di Xen; la speranza era quella di utilizzare tali sequenze per costruire un sense of self a livello di domU, in modo tale da rilevare eventuali anomalie nel

comportamento della macchina stessa. Inizialmente, si è deciso di creare all'interno del kernel del dominio utente, una struttura dati per tenere il conteggio di tutte le hypercall eseguite. Durante la raccolta dati si è cercato di sottoporre la macchina virtuale ad un carico di lavoro tipico per un server web: con l'ausilio di wget [43] sono state simulate richieste multiple nei confronti del server (vedi il grafico di figura 2.2). Successivamente, per effettuare una raccolta dati più raffinata, si è pensato di estendere il conteggio alle singole hypercall contenute in ciascuna multicall (vedi il grafico di figura 2.3).

2.2.2 Analisi dei dati raccolti

Analizzando i dati raccolti è subito evidente come quasi il 90% delle hypercall rilevate siano relative alla manipolazione della memoria della macchina virtuale, ed in particolare la chiamata update_va_mapping è quella più utilizzata in assoluto. Per cercare quindi di raffinare la granularità delle informazioni in nostro possesso, abbiamo pensato di controllare l'argomento dell'hypercall update_va_mapping, per verificare se l'area di memoria modificata appartenga allo spazio di indirizzamento del kernel, piuttosto che allo spazio utente. Per effettuare questa analisi abbiamo ridotto al minimo il carico di lavoro della macchina virtuale, in modo tale da garantire che le hypercall rilevate siano direttamente riconducibili al comando eseguito, minimizzando la possibilità di includere "rumore di fondo" nelle informazioni raccolte.

Esaminando le informazioni raccolte (vedi tabelle 2.1 e 2.2) si può notare come non sia possibile dedurre informazioni interessanti sulle attività di un processo in esecuzione all'interno della macchina virtuale. Consideriamo che, innanzitutto, i dati sono stati raccolti eseguendo un comando per volta, mentre ben diversa sarebbe la situazione su una macchina di produzione, che può eseguire numerosi demoni che generano decine, se non centinaia, di richieste contemporanee. Il problema di questo approccio è probabilmente insito nell'impossibilità di correlare le hypercall raccolte ai processi che hanno portato alla loro chiamata. D'altra parte, un'ipotetica soluzione al problema richiederebbe modiche sostanziali sia al kernel in esecuzione nel dominio guest che al kernel del dominio 0, con una probabile eccessiva penalizzazione dal punto di vista

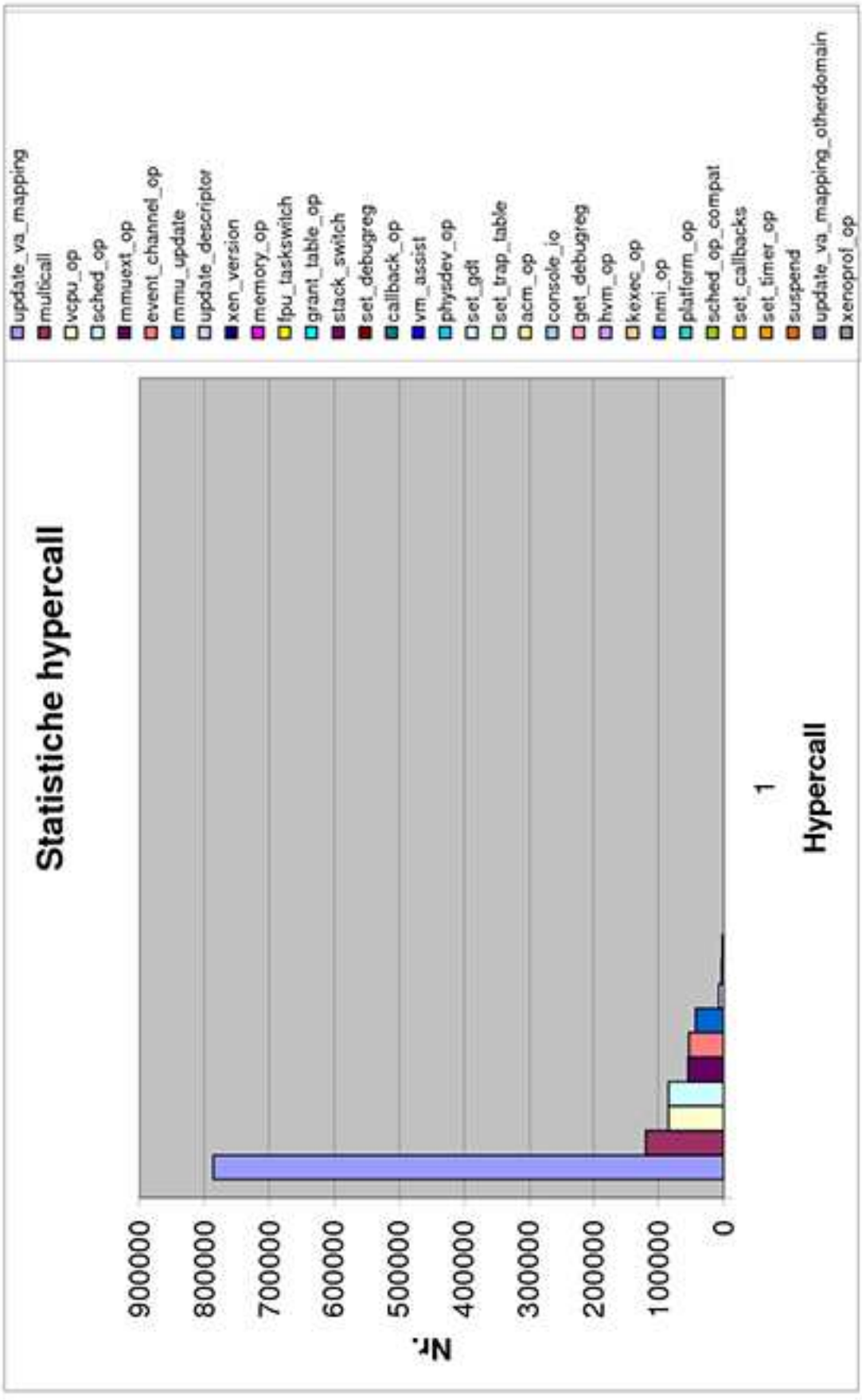

Figura 2.2: Grafico statistiche hypercall

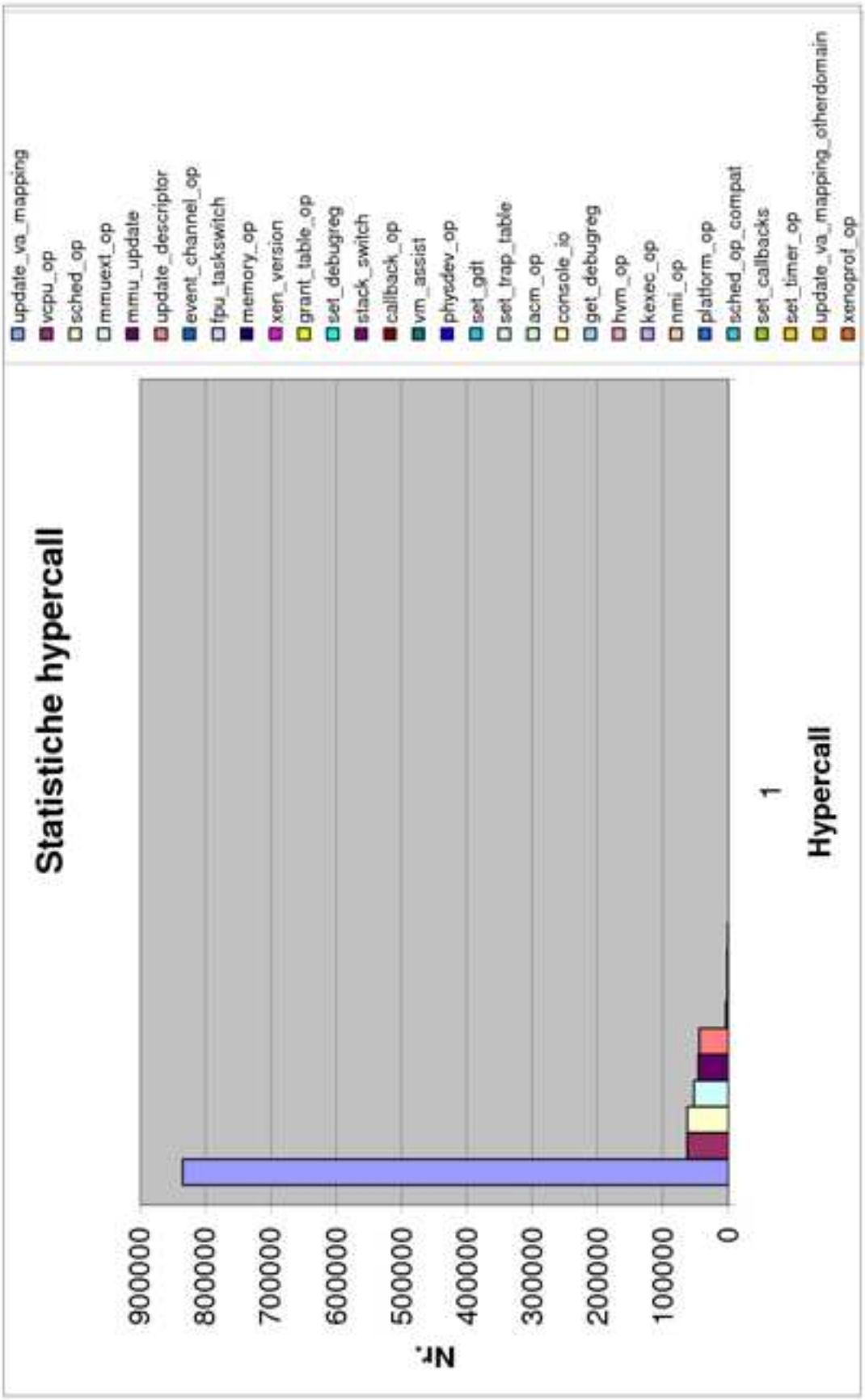

Figura 2.3: Grafico statistiche hypercall con multicall espansa

Hypercall	Nr.
update_va_mapping	786645
multicall	119897
vcpu_op	84837
sched_op	84771
mmuext_op	54379
event_channel_op	53825
mmu_update	43079
update_descriptor	6942
xen_version	3145
memory_op	1669
fpu_taskswitch	28
grant_table_op	6
stack_switch	6
set_debugreg	6
callback_op	3
vm_assist	3
physdev_op	1
set_gdt	1
set_trap_table	1
acm_op	0
console_io	0
get_debugreg	0
hvm_op	0
kexec_op	0
nmi_op	0
platform_op	0
sched_op_compat	0
set_callbacks	0
set_timer_op	0
suspend	0
update_va_mapping_otherdomain	0
xenoprof_op	0

Tabella 2.1: Tabella relativa al grafico di figura 2.2

delle prestazioni.

Hypercall	Nr.
update_va_mapping	835065
vcpu_op	62975
sched_op	62909
mmuext_op	52775
mmu_update	46173
update_descriptor	44749
event_channel_op	3806
fpu_taskswitch	1862
memory_op	1669
xen_version	622
grant_table_op	6
stack_switch	6
set_debugreg	6
callback_op	3
vm_assist	3
physdev_op	1
set_gdt	1
set_trap_table	1
acm_op	0
console_io	0
get_debugreg	0
hvm_op	0
kexec_op	0
nmi_op	0
platform_op	0
sched_op_compat	0
set_callbacks	0
set_timer_op	0
suspend	0
update_va_mapping_otherdomain	0
xenoprof_op	0

Tabella 2.2: Tabella relativa al grafico di figura 2.3

2.2.3 L'eccezione

Fra tutte le hypercall ve ne è una però, il cui rilevamento rappresenta un chiaro segnale d'allarme: la set_trap_table. Questa chiamata viene normalmente eseguita una sola

volta durante la fase di boot del sistema operativo per installare l'IDT ed è utilizzata dal processore per trovare il gestore delle eventuali interruzioni ed eccezioni ricevute. La sovrascrittura dell'indirizzo della tabella delle interruzioni è una tecnica utilizzata da una tipologia di rootkit, che cercano così di inserirsi direttamente all'interno del kernel per cercare di occultarsi [15]. Tale tecnica è ormai ben nota, e la maggior parte degli strumenti adibiti al rilevamento di rootkit salvano l'indirizzo della IDT per controllarlo successivamente e verificarne l'integrità.

Capitolo 3

Monitoraggio di macchine virtuali Xen

Questo capitolo descrive l'architettura del prototipo realizzato per monitorare lo stato delle macchine virtuali Xen. Viene inoltre descritto xmn, uno strumento per l'amministrazione remota del Virtual Machine Monitor.

3.1 VMtop

VMtop è uno strumento di monitoraggio remoto di macchine virtuali Xen. Nel seguito descriveremo l'architettura e l'implementazione di questo strumento.

3.1.1 Architettura

VMtop è formato da due componenti principali, secondo il tradizionale paradigma client server (vedi figura 3.1):

- *VMtopdomU* è il demone che si occupa di raccogliere le statistiche dell'host su cui è in esecuzione, per poterle inviare al client che ne faccia richiesta.
- *VMtop* è il client che si occupa di interrogare i demoni in esecuzione su ciascuna macchina virtuale. In aggiunta, il client è dotato di un'interfaccia grafica per la visualizzazione delle informazioni raccolte.

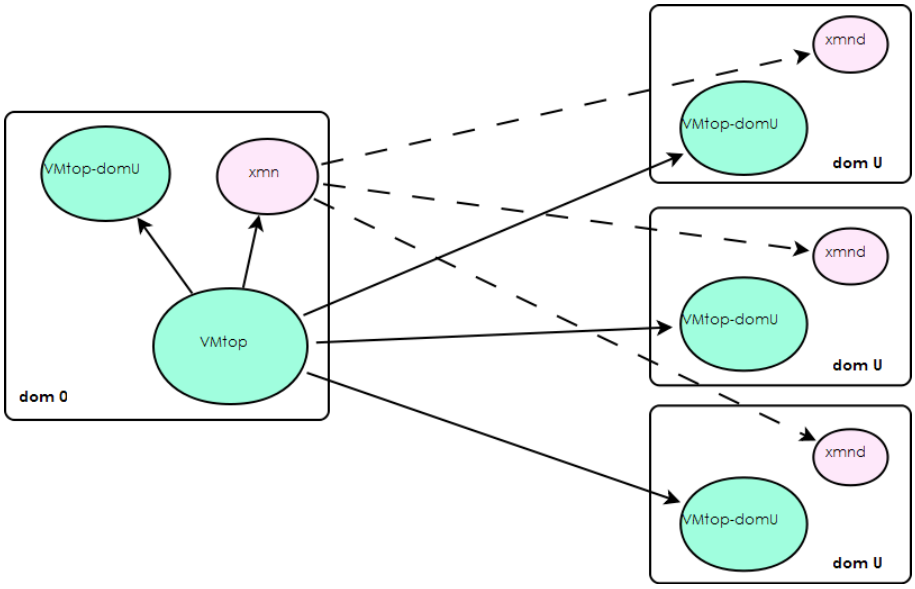

Figura 3.1: L'architettura di VMtop

Inoltre, all'interno di VMtop è stato integrato xmn, uno strumento di amministrazione remota di macchine virtuali che sarà descritto nel capitolo 3.4.

Le due applicazioni comunicano attraverso una connessione TCP. Il client, subito dopo l'avvio, crea una connessione con ciascuna macchina virtuale da monitorare. Normalmente, tale connessione rimarrà attiva per tutto il periodo di esecuzione del client. Il client invia quindi la stringa "GET" per richiedere l'invio delle ultime statistiche raccolte. La risposta del server consiste in una stringa contenente tutte le statistiche separate da uno spazio, e. g.:

0.00 0.00 0.00 131256 34252 97004 0 3200 13820 0 0 250.06 1 71687
68933 1669 0 1664 0 25552 25801 445945 246 0 155 23943

Una volta ricevuta la risposta, il client inserisce le metriche in essa contenute in alcune strutture dati, li visualizza e li salva in un file su disco. Più avanti vedremo l'utilizzo di tale file. Per fornire al server le statistiche relative alle hypercall, è stato necessario implementare un modulo per il kernel Linux che esporta le informazioni sulle hypercall creando un file nello pseudo file system *proc*.

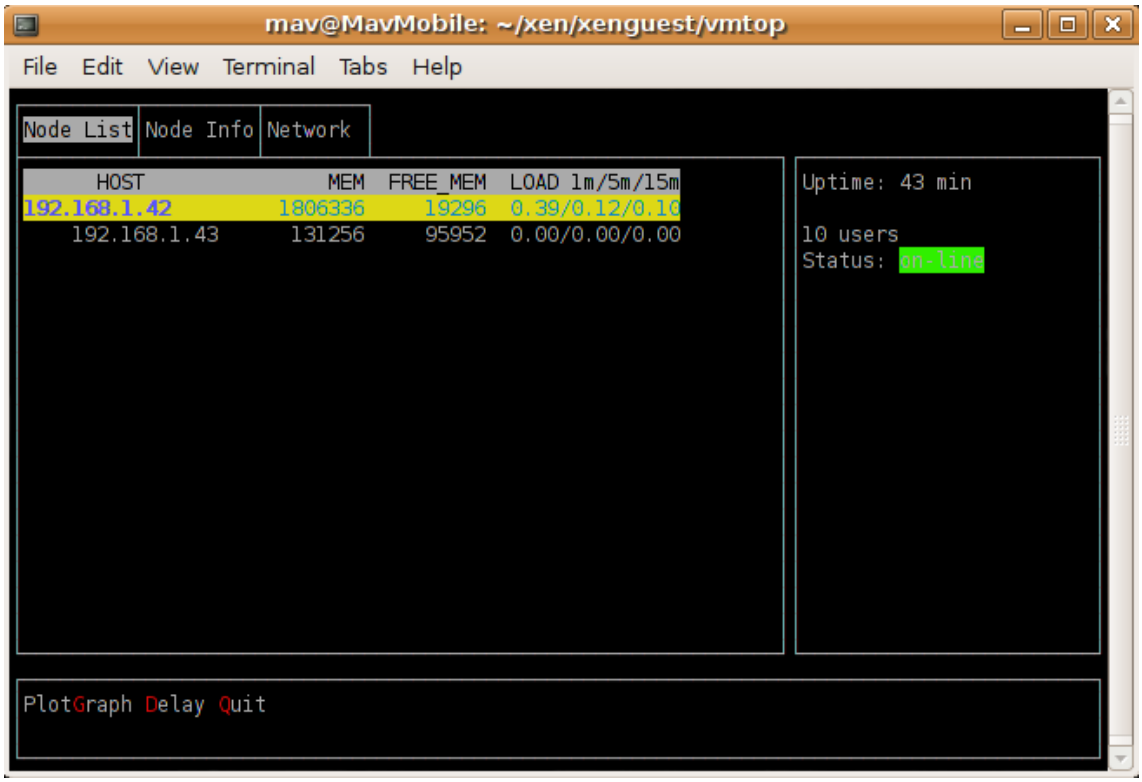

Figura 3.2: La schermata principale di VMtop

Nella figura 3.2 è possibile osservare la schermata principale di VMtop. Essa presenta nel riquadro principale la lista degli host monitorati con alcune informazioni di riepilogo sul loro stato. Le macchine virtuali si distinguono in quanto sono rappresentate con una maggiore indentazione rispetto al nodo fisico che le esegue. Nella figura 3.3 vediamo la schermata contenente le informazioni più dettagliate relative al nodo fisico che esegue il VMM. È possibile notare, in particolare, le statistiche relative al traffico di rete. In questo caso è evidente come il traffico di rete rilevato sia stato scambiato solamente fra il VMM e le macchine virtuali attive; infatti, la quantità di traffico rilevata sull'interfaccia di rete fisica è pari a zero.

Nella figura 3.4 è illustrata, invece, la schermata relativa alle informazioni dettagliate di una macchina virtuale. In particolare, si notano:

- Le informazioni relative ad alcune hypercall invocate. Nel dettaglio, tali hypercall sono relative ad operazioni di memoria, (vedi §2.1.3) perché riteniamo siano

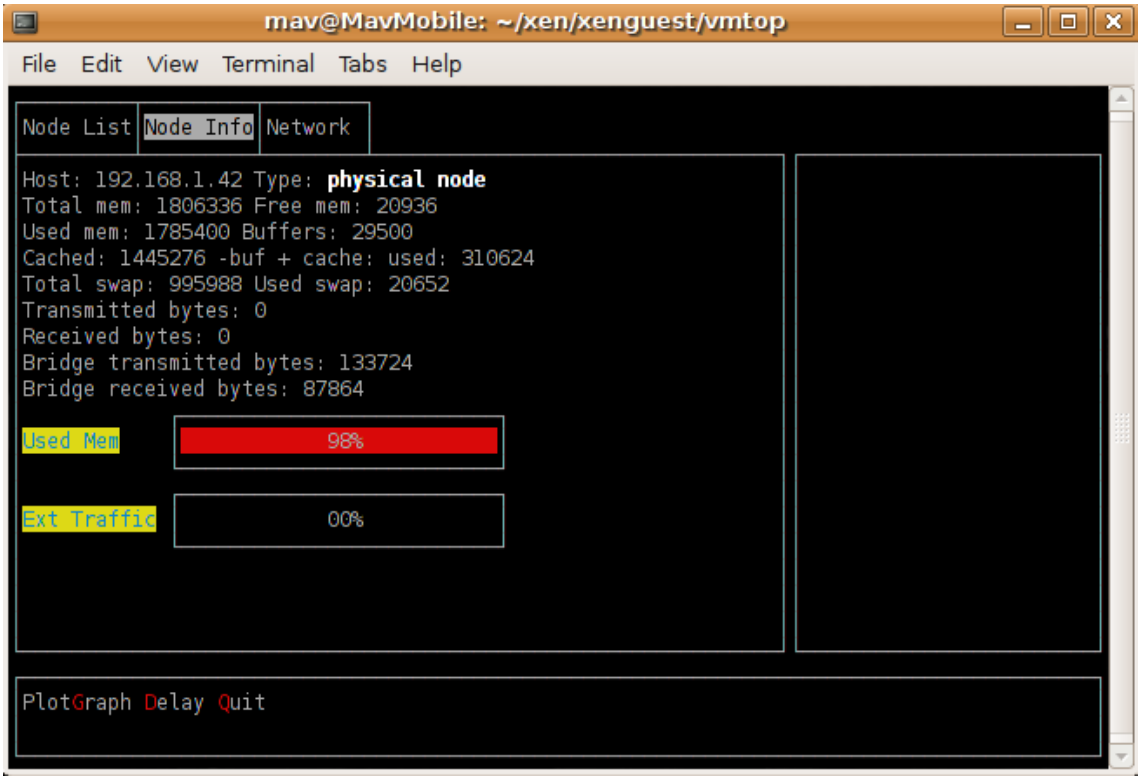

Figura 3.3: VMtop: informazioni dettagliate di un nodo fisico

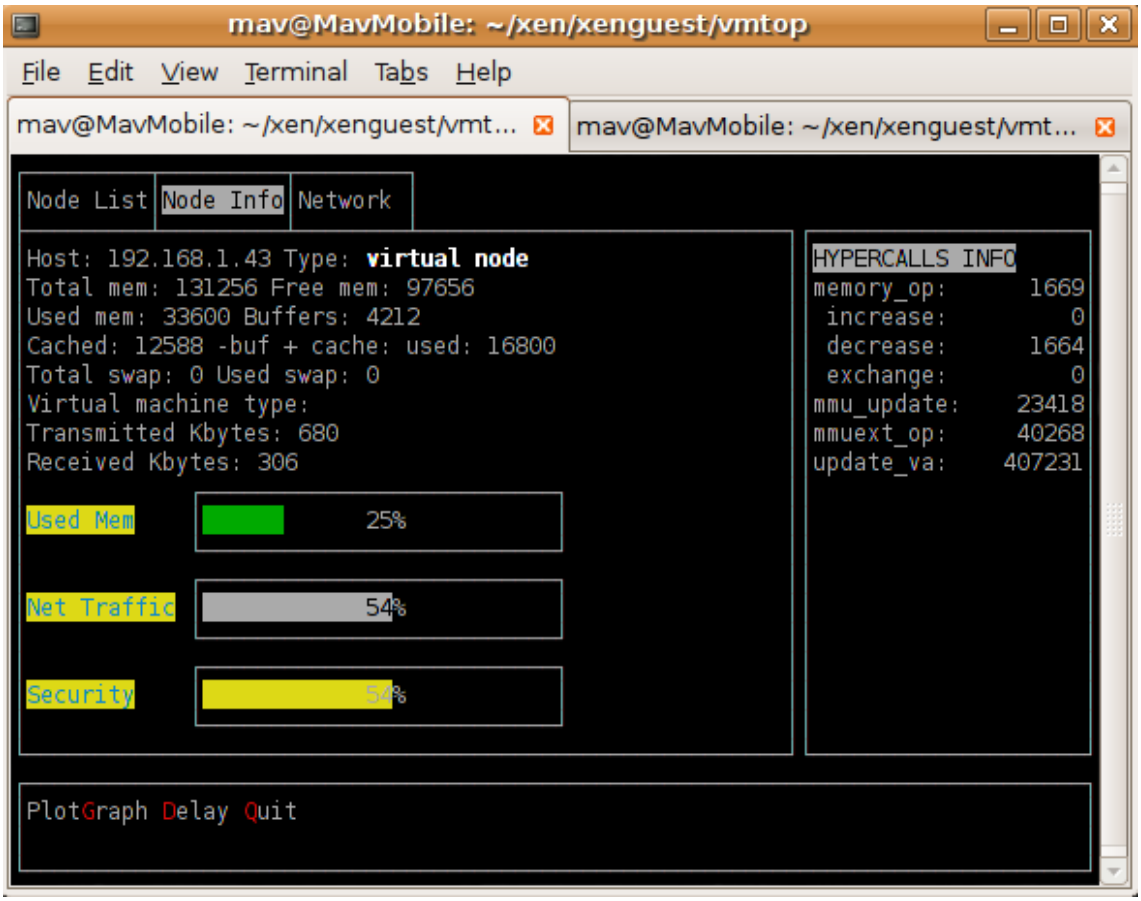

Figura 3.4: VMtop: informazioni dettagliate di un nodo fisico

molto utili per comprendere più a fondo l'effettivo utilizzo di memoria da parte della macchina virtuale.

- La barra di sicurezza, che offre un riepilogo su quello che viene considerato il livello di sicurezza della macchina virtuale in questione. Il coefficiente di sicurezza viene calcolato utilizzando varie informazioni, quali ad esempio: presenza di un firewall sulla macchina, configurazione aggiornamenti automatici, utilizzo di tecniche di introspezione per verificare l'integrità del kernel, frequenza dei controlli di introspezione.

3.2 RRDtool

RRDtool (Round Robin Database tool) [33] è uno strumento open source in grado di salvare dati numerici provenienti dalle fonti più disparate, in maniera sistematica ed efficiente. Ad esempio, alcuni dei dati che vengono frequentemente trattati da questo programma possono essere il carico e la temperatura della CPU, la memoria utilizzata, la quantità di traffico di rete... Come intuibile dal nome [32], RRDtool per memorizzare i dati, utilizza archivi in modalità round robin. La caratteristica principale di questo tipo di archivio è quella di occupare una quantità fissa di spazio di memorizzazione. Infatti, nel caso in cui il database sia pieno, i nuovi dati vengono inseriti in maniera circolare, sovrascrivendo quelli più vecchi. La gestione di serie di dati che si ripetono periodicamente nel tempo risulta così notevolmente semplificata. Inoltre, dato che la raccolta dei dati può non avvenire sempre all'istante previsto, RRDtool è in grado di effettuare l'interpolazione dei dati, garantendone così l'uniformità. Una volta raccolta una quantità di informazioni sufficienti, RRDtool è in grado di rappresentarle in forma grafica (vedi figura 3.5). Si è deciso quindi di utilizzare RRDtool per salvare alcuni dei dati raccolti da VMtop [1], producendo così grafici utili per ottenere una rapida visione d'insieme dell'utilizzo delle risorse a disposizione dele macchine virtuali monitorate. Al momento i database esistenti sono tre:

1. utilizzo della memoria
2. carico della CPU
3. traffico di rete generato

In particolare, l'analisi del traffico di rete ci permette di capire se l'allocazione delle macchine virtuali sui nodi fisici garantisce un adeguato bilanciamento del carico, o se invece è necessaria una qualche modifica. Sia TVi il traffico di rete generato dall'i-esima macchina virtuale, TR il traffico transitato sull'interfaccia utilizzata dall'hypervisor per connettere le VM alla rete esterna, sia essa una rete fisica o una VPN. Se TE è definito come:

[1] Il contenuto del file scritto da VMtop, viene utilizzato per popolare i vari database RRD.

$$TE = \frac{\sum_{i=1}^{n} TV_i * 100}{TR} \tag{3.1}$$

allora TE rappresenta la percentuale del traffico generato dalle macchine virtuali, diretto all'esterno del nodo fisico. Di conseguenza, se TE assume valori elevati, può essere conveniente ripensare l'allocazione dei domini virtuali su differenti nodi fisici, in modo tale da ottenere così una riduzione del traffico presente sulla rete fisica.

3.3 Implementazione

VMtop è stato scritto utilizzando il linguaggio di programmazione C. In particolare, l'interfaccia utente di VMtop è stata realizzata utilizzando le librerie Ncurses [4]. Questa libreria è stata scelta perché permette di implementare applicazioni con un'interfaccia che anche se testuale, è comunque paragonabile ad una vera e propria interfaccia grafica (GUI) per quanto riguarda l'immediatezza nell'interazione con il programma. Inoltre, un'interfaccia testuale è particolarmente adatta per l'utilizzo di programmi su un server remoto, in quanto il volume di traffico generato è notevolmente inferiore rispetto a quello che è richiesto per la gestione di una GUI completa. Come Virtual Machine Monitor è stato utilizzato Xen versione 3.1.0, per il dominio zero è stata utilizzata la distribuzione Linux Ubuntu 7.10 [38], mentre per i domini utente è stata utilizzata la distribuzione Linux Debian [5].

3.4 Xmn

Xmn (Xen Monitor Network) è utilizzato per la gestione remota di un numero arbitrario di VMM interconnessi tramite una VPN. All'interno della rete si stabilisce una semplice gerarchia: un nodo agisce come **VMM master**, mentre i rimanenti sono semplici **VMM slave**. Il nodo master controlla le macchine virtuali allocate sui nodi slave. Xmn è un'estensione del comando *xm* (Xen Manager) [46]. Utilizzando xmn è possibile effettuare numerose operazioni :

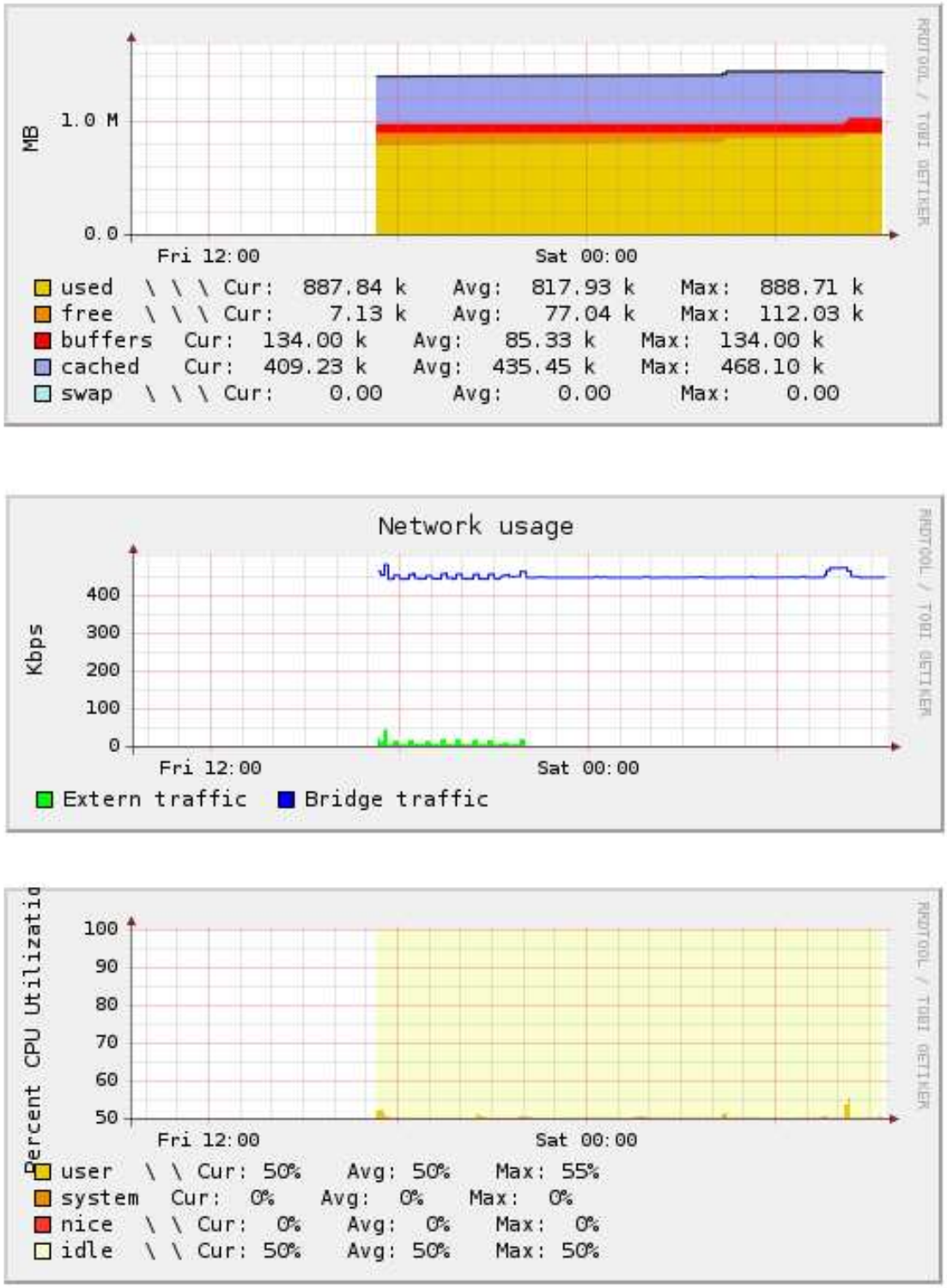

Figura 3.5: I grafici prodotti con RRDtool

1. Inviare ai VMM slave i file necessari per la creazione di una macchina virtuale, quali ad esempio la configurazione, l'immagine del kernel o anche l'immagine di una partizione, necessaria nel caso la VM venga migrata.

2. Creare una macchina virtuale su un nodo della rete.

3. Connettersi alla console di una macchina virtuale.

4. Distruggere una macchina virtuale, operazione equivalente allo spegnimento forzato di un computer, ad esempio per un'improvvisa mancanza di corrente.

5. Spegnere o riavviare una o più macchine virtuali.

6. Ogni VM è identificata sia da un *domain name* che da un *domID*. Xmn, dato un domain name, permette di ottenere il corrispondente domID. Ovviamente è possibile effettuare anche la risoluzione inversa.

7. Stampare alcune informazioni relative ad una o più macchine virtuali in esecuzione. Ad esempio, è possibile conoscere il numero di CPU virtuali allocate ad un dominio, oppure lo stato in cui si trova la macchina virtuale.

8. Configurare la quantità massima di memoria che una VM può utilizzare.

9. Migrare una macchina virtuale dal nodo su cui si trova ad un altro. Chiaramente, il nodo di destinazione deve essere in grado di allocare sufficienti risorse per ricevere la VM.

10. Sospendere e riprendere l'esecuzione di una VM. Una macchina virtuale sospesa, continua ad occupare le risorse a sua disposizione, ma non viene mai schedulata dall'hypervisor.

11. Spegnere una macchina virtuale dopo aver salvato il suo stato in un file, liberando così le risorse ad essa allocate. In questo modo è possibile riprendere l'esecuzione della VM in un secondo momento, ripartendo esattamente dal punto in cui era stata interrotta. Questa operazione è l'equivalente dell'ibernazione di un sistema reale.

12. Modificare il numero di CPU virtuali a disposizione di un determinato dominio.

13. Aggiungere o rimuovere un dispositivo a blocchi virtuale ad una VM. Ad esempio è possibile utilizzare un'immagine ISO come se fosse un hard disk.

14. Aggiungere o rimuovere una scheda di rete virtuale ad una VM.

15. Configurare una politica di controllo degli accessi (ACP) relativa ad una macchina virtuale.

3.4.1 Architettura

Xmn è composto da due applicazioni (vedi figura 3.1):

- *Xmnd* (Xen Monitor Network demon) è il demone che viene eseguito all'avvio dai VMM slave, ed ha lo scopo di rimanere in ascolto in attesa dei comandi dall'hypervisor master.

- *Xmn* è il client, eseguito sul VMM master, che invia i comandi da eseguire al demone in esecuzione sui vari VMM slave.

Le due applicazioni comunicano utilizzando una connessione TCP. Il protocollo di comunicazione utilizzato è composto da tre fasi:

1. dopo aver stabilito la connessione, xmn invia al demone xmnd il numero di parametri che deve inviare. In questo modo il demone può allocare le strutture dati adeguate per ricevere il comando.

2. per ogni parametro da inviare xmn trasmette prima la dimensione del parametro e, successivamente, il parametro stesso.

3. una volta ricevuti tutti i parametri xmnd può eseguire il comando richiesto, tramite xm, oppure prepararsi a ricevere i file che xmn vuole inviare.

3.4.2 Gestione di una rete di macchine virtuali

Descriviamo ora l'architettura di una rete privata virtuale per creare una rete di macchine virtuali.
Una VPN di questo tipo permette l'utilizzo di strumenti di amministrazione remota, come quelli descritti nei paragrafi precedenti, grazie ai quali è possibile gestire i virtual machine monitor presenti nella rete virtuale. Innanzitutto, è necessario creare su ciascun nodo fisico una *macchina virtuale firewall*, d'ora in poi semplicemente *firewall*. Ogni firewall è dotato del software necessario per la gestione di una VPN. In questo modo viene creato un ulteriore strato di rete dedicato esclusivamente al traffico che le macchine virtuali scambiano fra loro (vedi figura 3.6).
Così facendo, a scapito di una maggiore richiesta di risorse computazionali necessarie alla gestione della VPN, otteniamo un maggiore livello di sicurezza. Nel caso in cui un attaccante riuscisse ad accedere ad una VM, egli non avrebbe comunque la possibilità di interagire direttamente con i VMM presenti sulla rete.
Essendo il firewall virtuale il punto di contatto fra la rete virtuale e le VM in esecuzione sul relativo nodo fisico, per incrementare il livello di sicurezza, è possibile installare un Network IDS, con lo scopo di monitorare il traffico diretto verso il segmento di rete virtuale presente su ciascun nodo fisico. Inoltre, sui nodi firewall, è possibile configurare ulteriori politiche di filtraggio del traffico, utilizzando ad esempio iptables.

3.4.3 Migrazione di macchine virtuali

La migrazione di una macchina virtuale permette il trasferimento di tale VM su un altro nodo sul quale vi sia in esecuzione un VMM.
Come già visto in precedenza, la migrazione permette di effettuare agevolmente operazioni come, ad esempio, bilanciamento del carico o manutenzione di un server, senza la necessità di interrompere i servizi erogati.
Il requisito principale per effettuare la migrazione, è la possibilità di salvare lo stato di esecuzione della macchina virtuale. Il salvataggio, però, non è formato solamente dal file di configurazione della VM e dalle immagini delle partizioni utilizzate. È necessario, infatti, salvare anche il contenuto della RAM, dei registri della CPU virtuale,

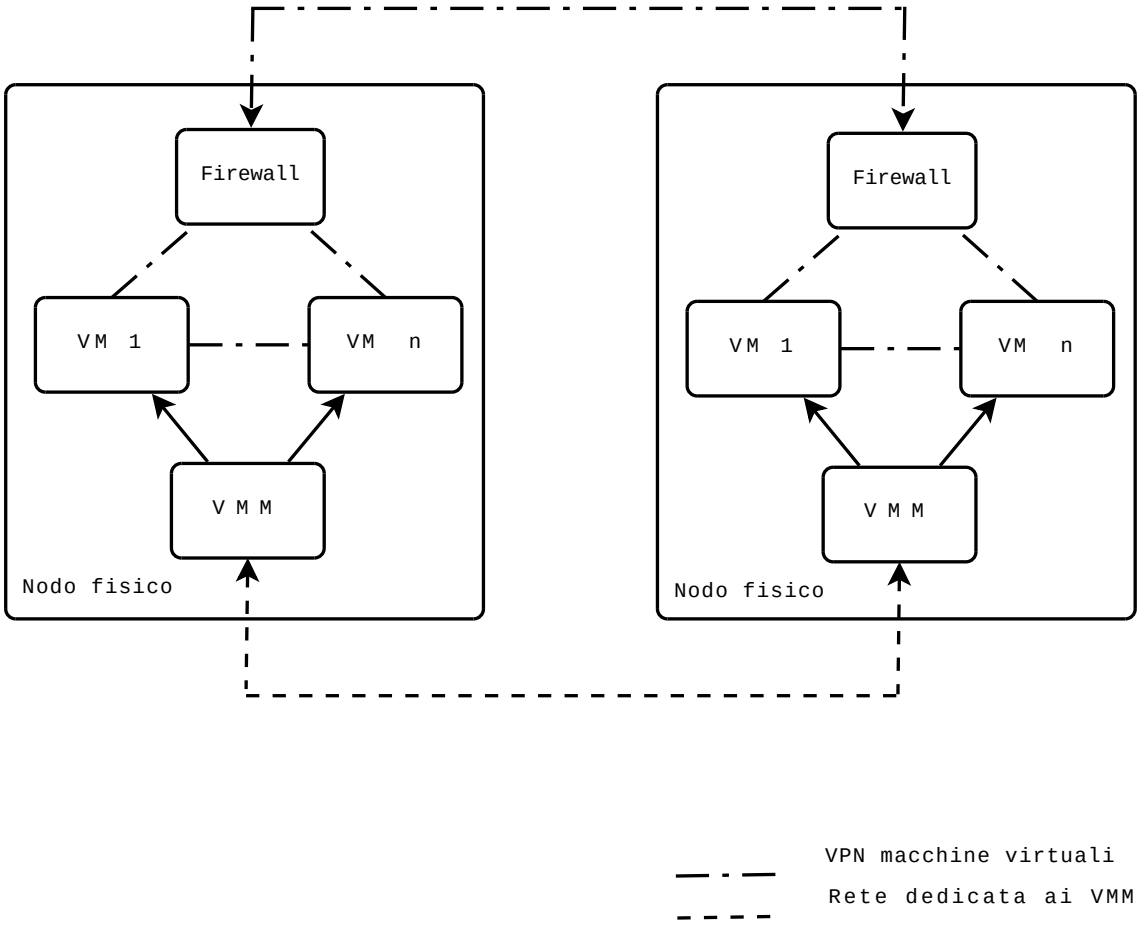

Figura 3.6: La topologia della rete virtuale descritta

lo stato delle periferiche e delle connessioni di rete instaurate. Xen permette di salvare tutte queste informazioni su hard disk, permettendo così di riprendere l'esecuzione della VM su un nodo differente.
Attualmente, è possibile migrare una macchina virtuale Xen in tre modalità differenti:

1. **save and restore**: viene creato un file di checkpoint, contenente lo stato di esecuzione della macchina virtuale. Tale file può essere utilizzato per riprendere l'esecuzione sia sullo stesso nodo, che su un nodo differente.

2. **regular**: l'esecuzione della VM viene fermata, il contenuto della memoria della macchina virtuale viene trasferito dal nodo sorgente al nodo destinazione, dove viene infine ripristinata l'esecuzione. È simile alla migrazione save and restore, con la differenza che il processo di creazione e trasferimento del file di checkpoint è automatizzato. Questo tipo di migrazione, così come il precedente, non è trasparente, in quanto comporta un'interruzione del servizio, e in particolare, causa l'interruzione delle connessioni di rete instaurate.

3. **live**: è il tipo di migrazione da utilizzare nel caso sia necessario ridurre al minimo i tempi di indisponibilità dei servizi offerti. La migrazione live è un procedimento decisamente complesso, in quanto avviene senza interrompere l'esecuzione della VM. Questo comporta un continuo cambiamento dello stato della macchina virtuale durante la migrazione. È necessario, pertanto, utilizzare un algoritmo iterativo: inizialmente viene effettuata una copia intera della macchina virtuale, e, ad ogni iterazione successiva, viene copiata solamente la memoria cambiata nel frattempo. Il processo di migrazione termina quando il numero di pagine di memoria modificate fra un'iterazione e l'altra è sufficientemente basso. Questo tipo di migrazione richiede un tempo dell'ordine di alcuni secondi.

Per ridurre la quantità di informazioni da trasmettere dal nodo sorgente al nodo destinazione, e di conseguenza anche i tempi richiesti dal processo di migrazione, è conveniente avere le immagini delle partizioni accessibili da entrambi i VMM, riducendo così la quantità di informazioni che i nodi coinvolti nella migrazione devono scambiarsi.

Capitolo 4

Test e conclusioni

Questo capitolo descrive i test effettuati per misurare l'overhead dovuto all'utilizzo di una macchina virtuale con le modifiche introdotte per il monitoraggio delle hypercall effettuate.

4.1 Modalità di esecuzione

I vari test sono stati eseguiti sia su una normale macchina virtuale, sia su una macchina virtuale con l'infrastruttura di monitoraggio attiva, in modo tale da poter confrontare i diversi tempi di esecuzione. Il sistema utilizzato per i test ha le seguenti caratteristiche:

- **Processore**: 2.4 GHz 64-bit Intel Core 2 Duo E6600
- **Memoria fisica**: 2 GB
- **Memoria virtuale** allocata al dominio utente: 128 MB
- **Sistema operativo** del dominio 0: Fedora Core 7
- **Sistema operativo** del dominio utente: Debian Etch con kernel 2.6.18-xenU

4.2 LMbench

La prima serie di test è stato effettuata utilizzando LMbench [24], una suite di benchmark creata per sistemi operativi Unix. In particolare, i test eseguiti sono:

- **Memory bandwidth**: misura l'ampiezza di banda della memoria, in lettura e scrittura, eseguendo ripetutamente la somma di array di grosse dimensioni (vedi figura 4.1 e 4.2).
- **File reread bandwidth**: misura l'ampiezza di banda raggiungibile durante la lettura di file memorizzati nella cache di sistema (vedi figura 4.3 e 4.4).
- **Context switch**: misura il tempo necessario ad effettuare il cambi di contesto fra un certo numero di processi in esecuzione (vedi figura 4.5 e 4.6).

Relativamente al benchmark memory bandwidth, l'overhead è abbastanza basso (7%) nel caso di array di dimensioni massime pari a 64 MB. Con array di dimensioni superiori, invece, l'overhead aumenta (11%).
Per quanto riguarda il test di rilettura di file, non si nota una diminuzione apprezzabile delle prestazioni.
Infine, nel caso del benchmark relativo ai cambi di contesto fra processi di varie dimensioni, il monitoraggio delle hypercall introduce un overhead di circa il 50%, indifferentemente dalle dimensioni dei processi stessi.

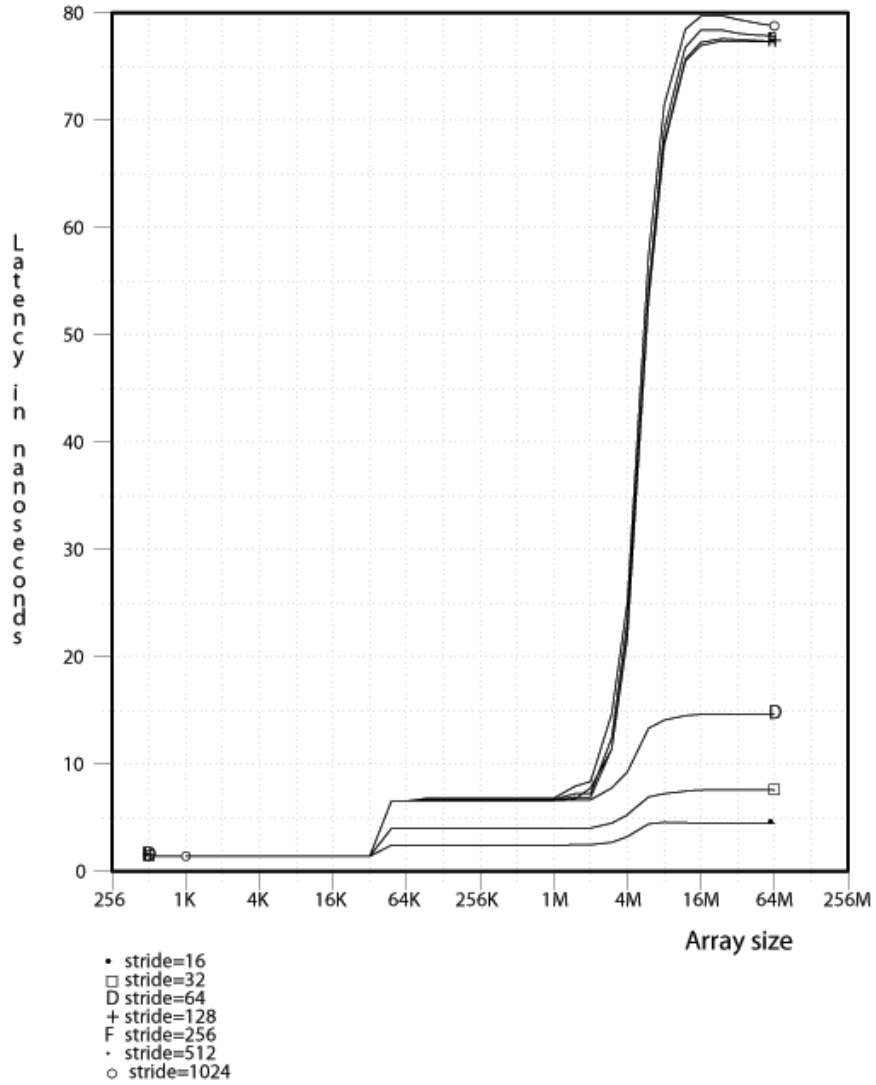

Figura 4.1: Memory bandwidth kernel vanilla

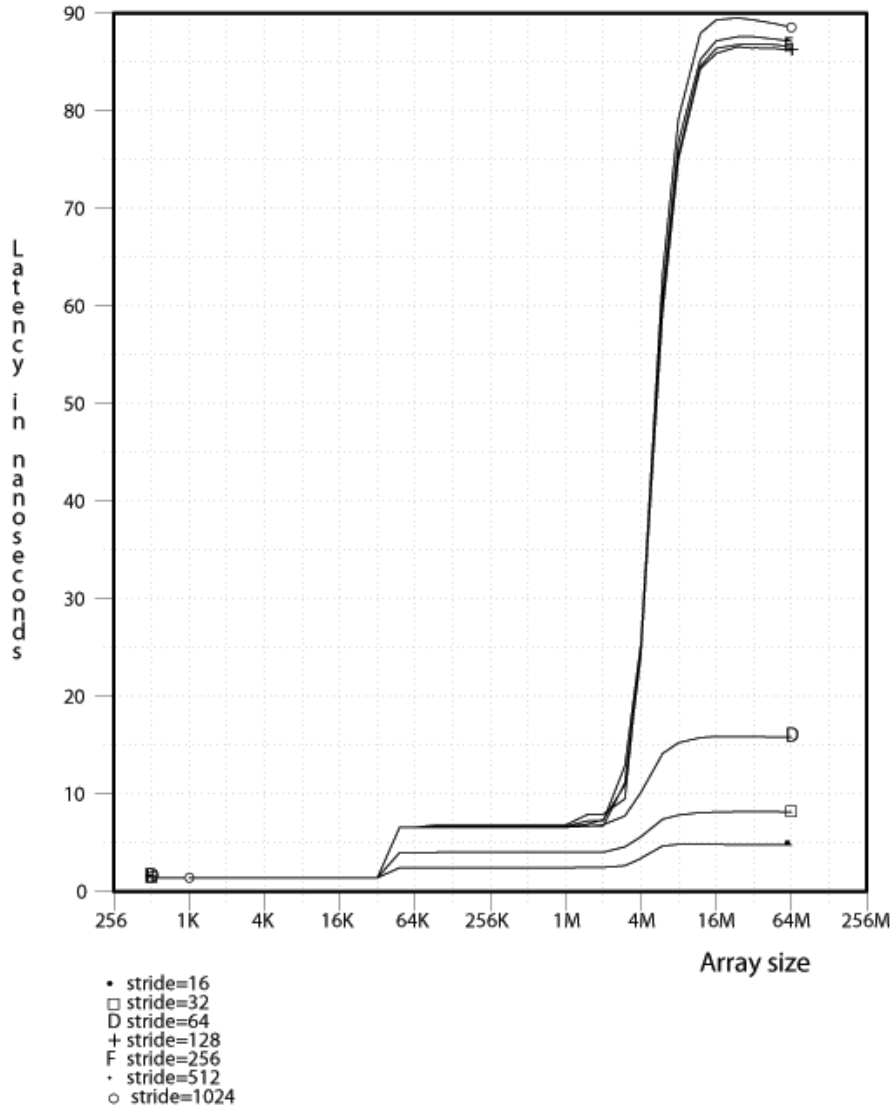

Figura 4.2: Memory bandwidth kernel con patch monitoraggio

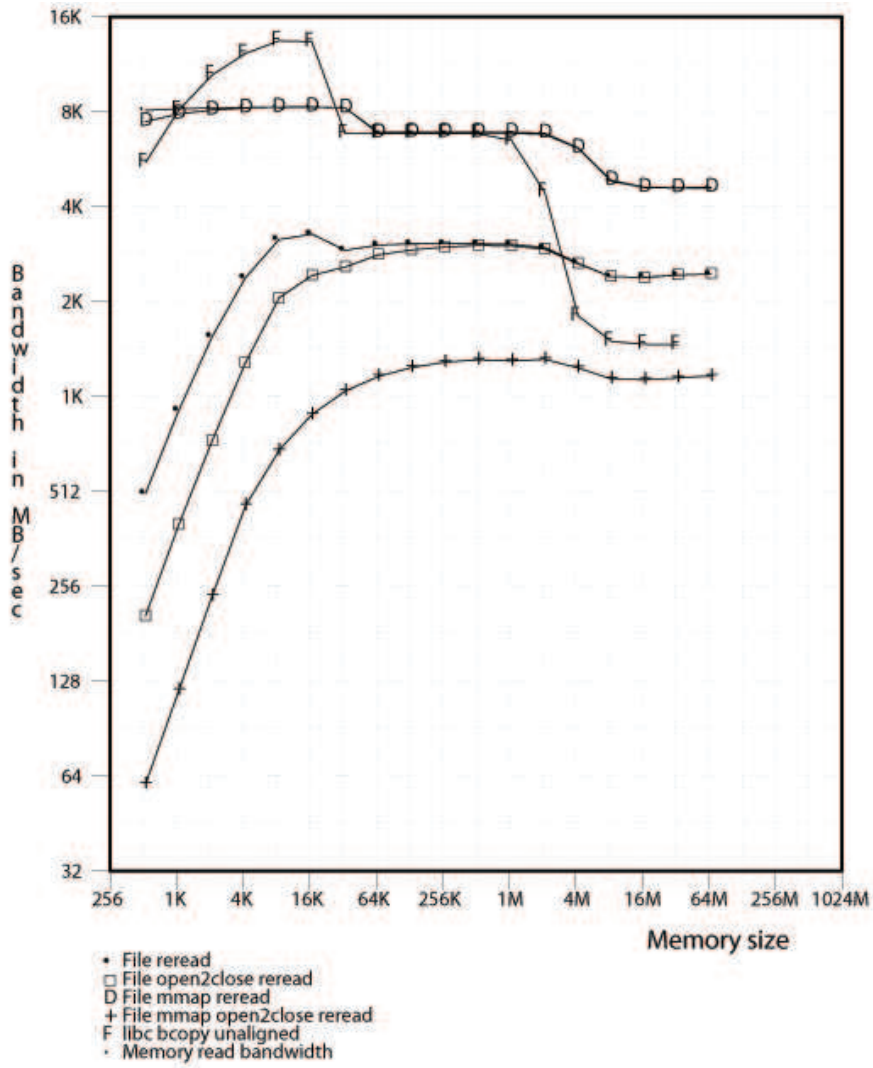

Figura 4.3: File reread bandwidth kernel vanilla

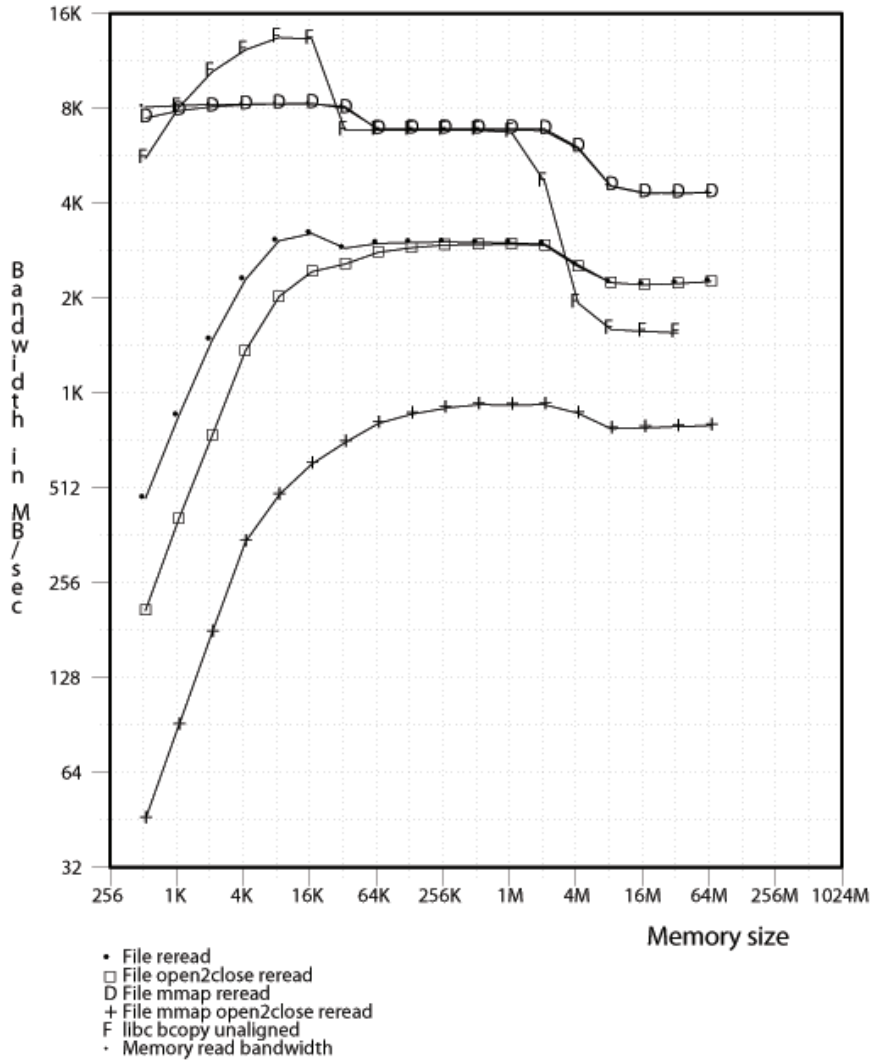

Figura 4.4: File reread bandwidth kernel con patch monitoraggio

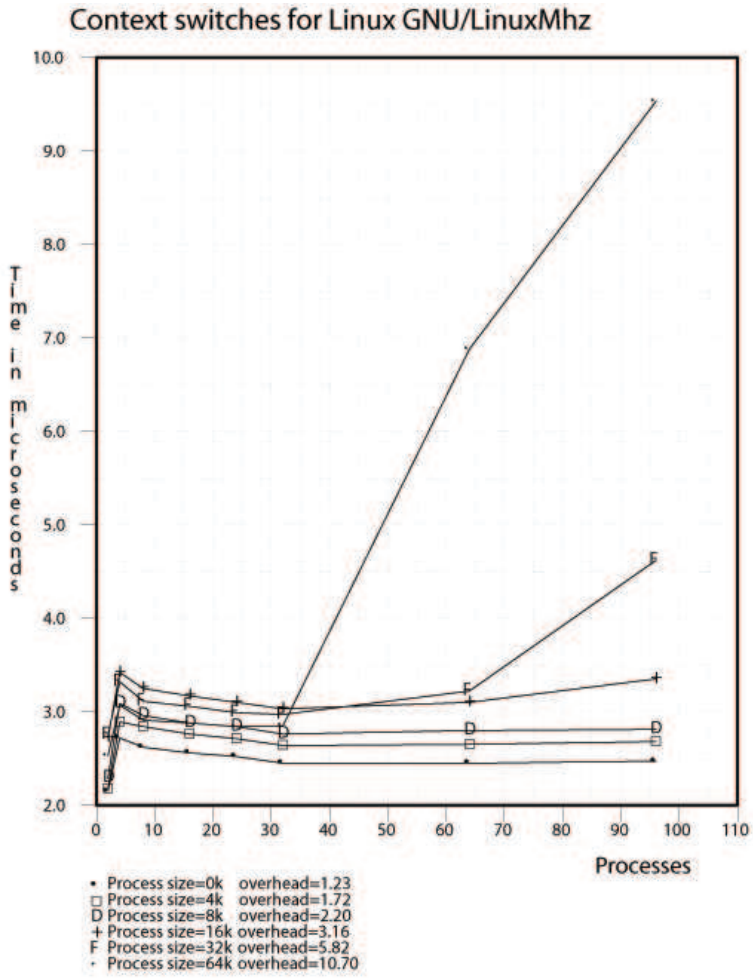

Figura 4.5: Context switch kernel vanilla

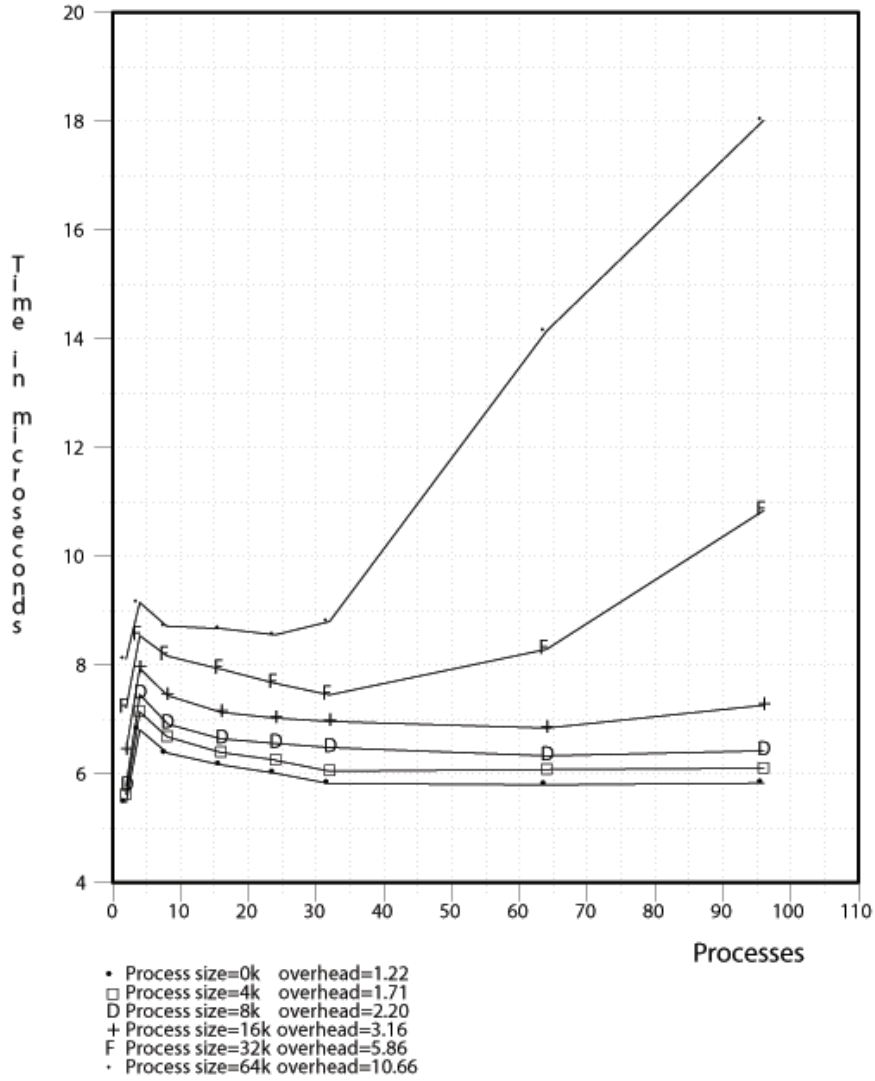

Figura 4.6: Context switch kernel con patch monitoraggio

4.3 IOzone

IOzone [12] è un benchmark multipiattaforma per valutare le prestazioni di file system. IOzone ha lo scopo di testare efficacemente le due principali modalità di accesso ad un file system: sequenziale e casuale. Un esempio del primo tipo si ha effettuando la copia di interi file di grandi dimensioni, mentre il secondo tipo è utilizzato frequentemente dai database relazionali. Sono stati effettuati quattro test differenti:

1. **Write**: misura le prestazioni di scrittura di un nuovo file. Quando un nuovo file viene creato, è necessario scrivere sul file system anche i metadati, ovvero informazioni quali la directory in cui si trova il file, lo spazio occupato dal file... Per questo motivo un test di scrittura è generalmente più lento rispetto ad un test di riscrittura.

2. **Read**: misura le prestazioni di lettura di un file memorizzato.

3. **Random write**: misura le performance di scrittura all'interno di un file in posizioni casuali. I risultati di questo test possono essere influenzati da diversi fattori quali: le dimensioni della cache del sistema operativo, numero di hard disk o ritardo di accesso.

4. **Random read**: misura le performance di lettura di un file in posizioni casuali. Anche i risultati di questo test possono essere influenzati dai medesimi fattori del precedente.

Analizzando i risultati di questi test, si può notare come l'overhead sia generalmente più elevato nel caso di file di dimensioni minori (45-50%), mentre con file di grandi dimensioni il peggioramento di prestazioni scende a livelli più accettabili (10-12%). Questo è probabilmente dovuto al fatto che, con file di grosse dimensioni, tipicamente superiori ai 256MB, la cache della CPU e il buffer di sistema non incidono in modo significativo sulle prestazioni di lettura/scrittura. Di conseguenza, l'attività di I/O richiede un minor numero di hypercall, permettendo così di ridurre l'overhead.

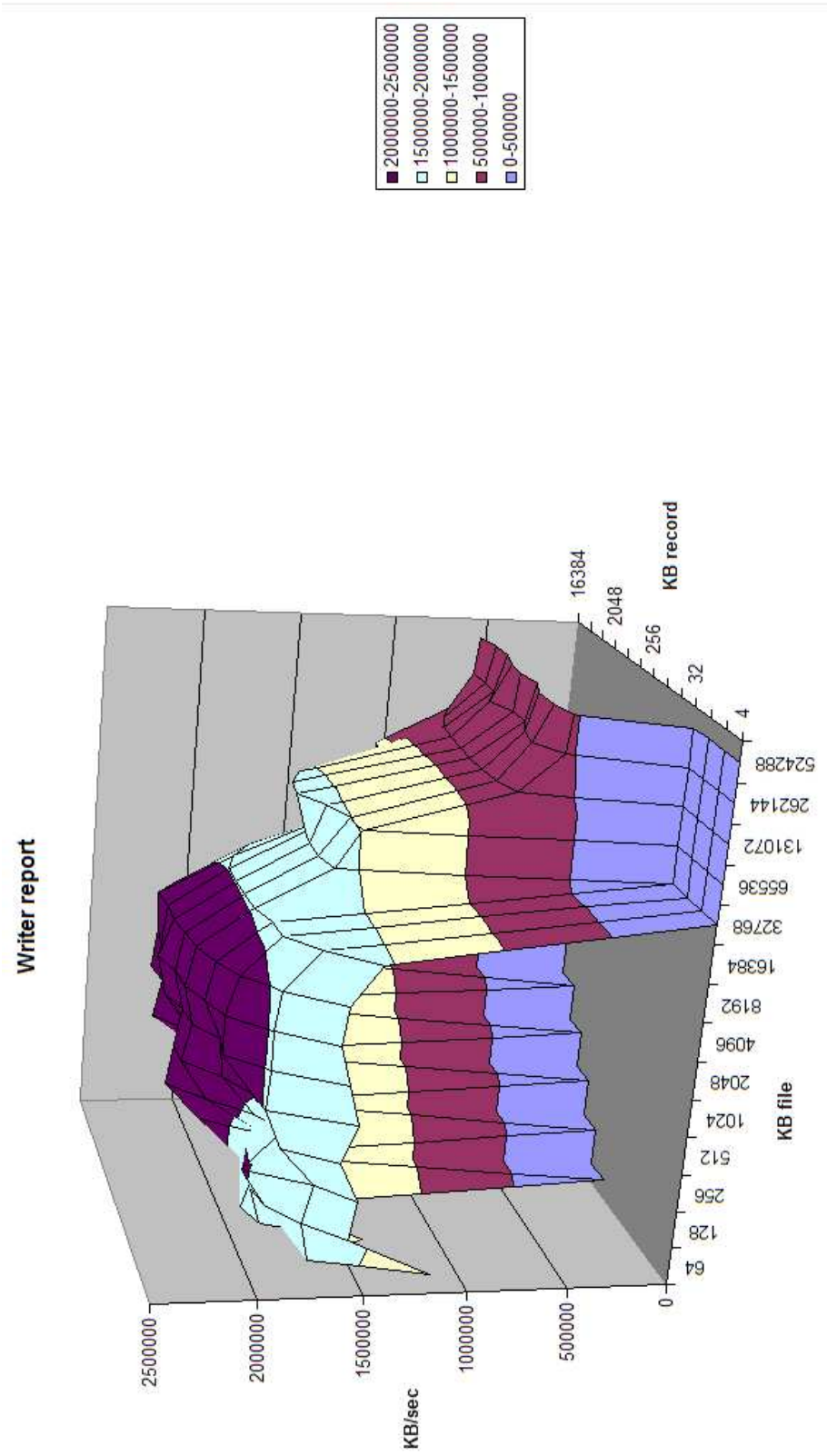

Figura 4.7: IOzone: test di scrittura, kernel vanilla

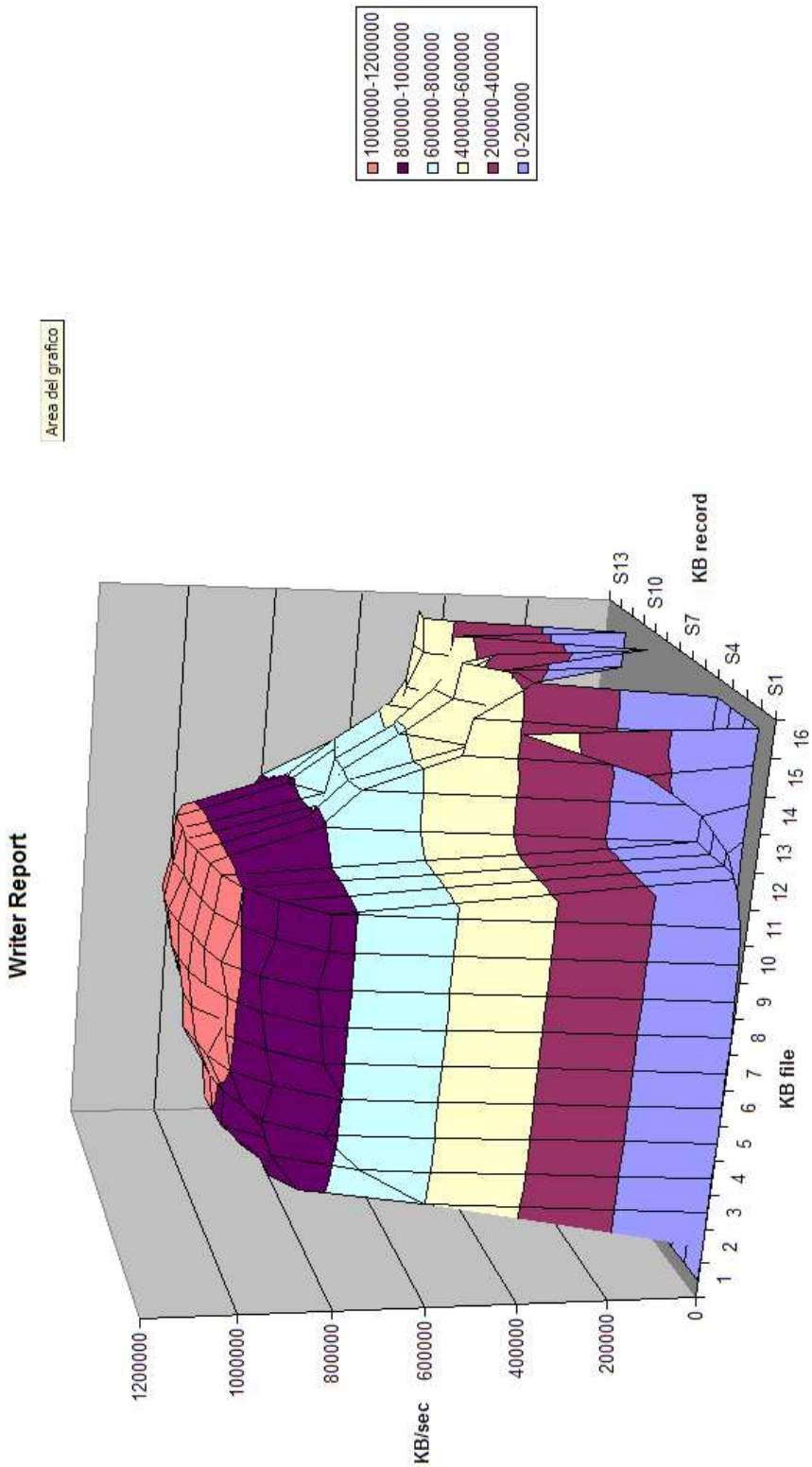

Figura 4.8: IOzone: test di scrittura, kernel con patch monitoraggio

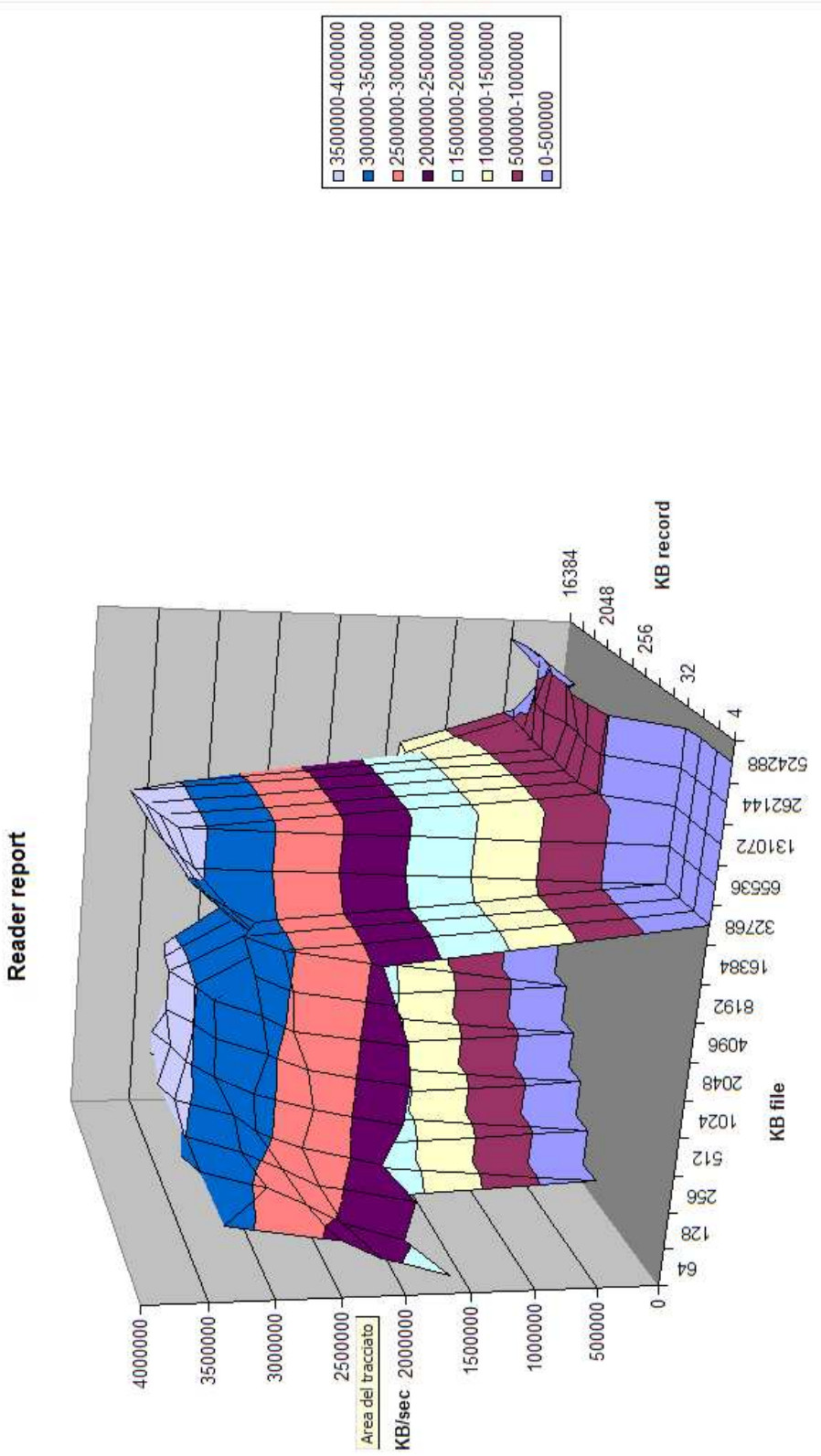

Figura 4.9: IOzone: test di lettura, kernel vanilla

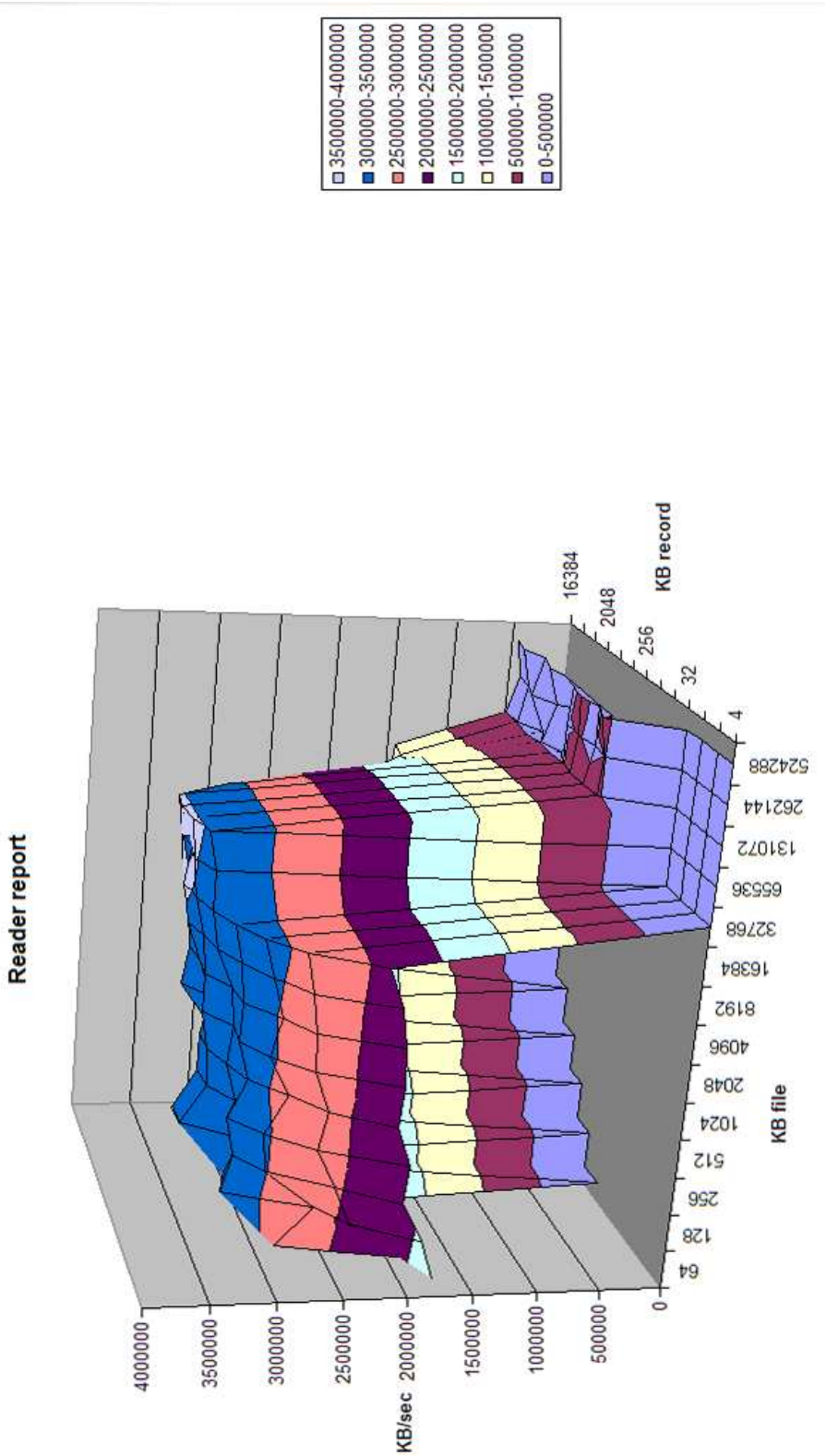

Figura 4.10: IOzone: test di scrittura, kernel con patch monitoraggio

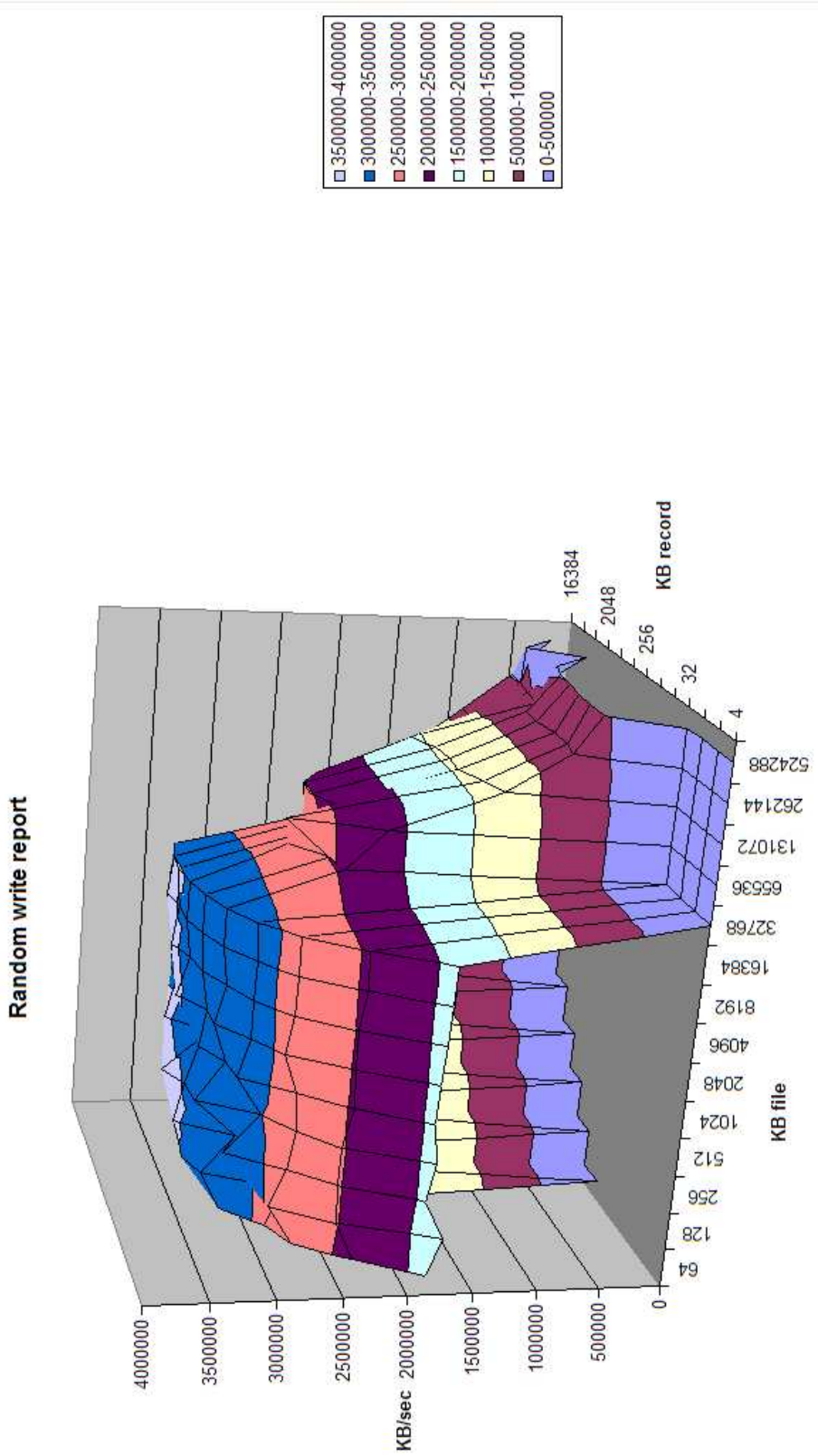

Figura 4.11: IOzone: test di scrittura casuale, kernel vanilla

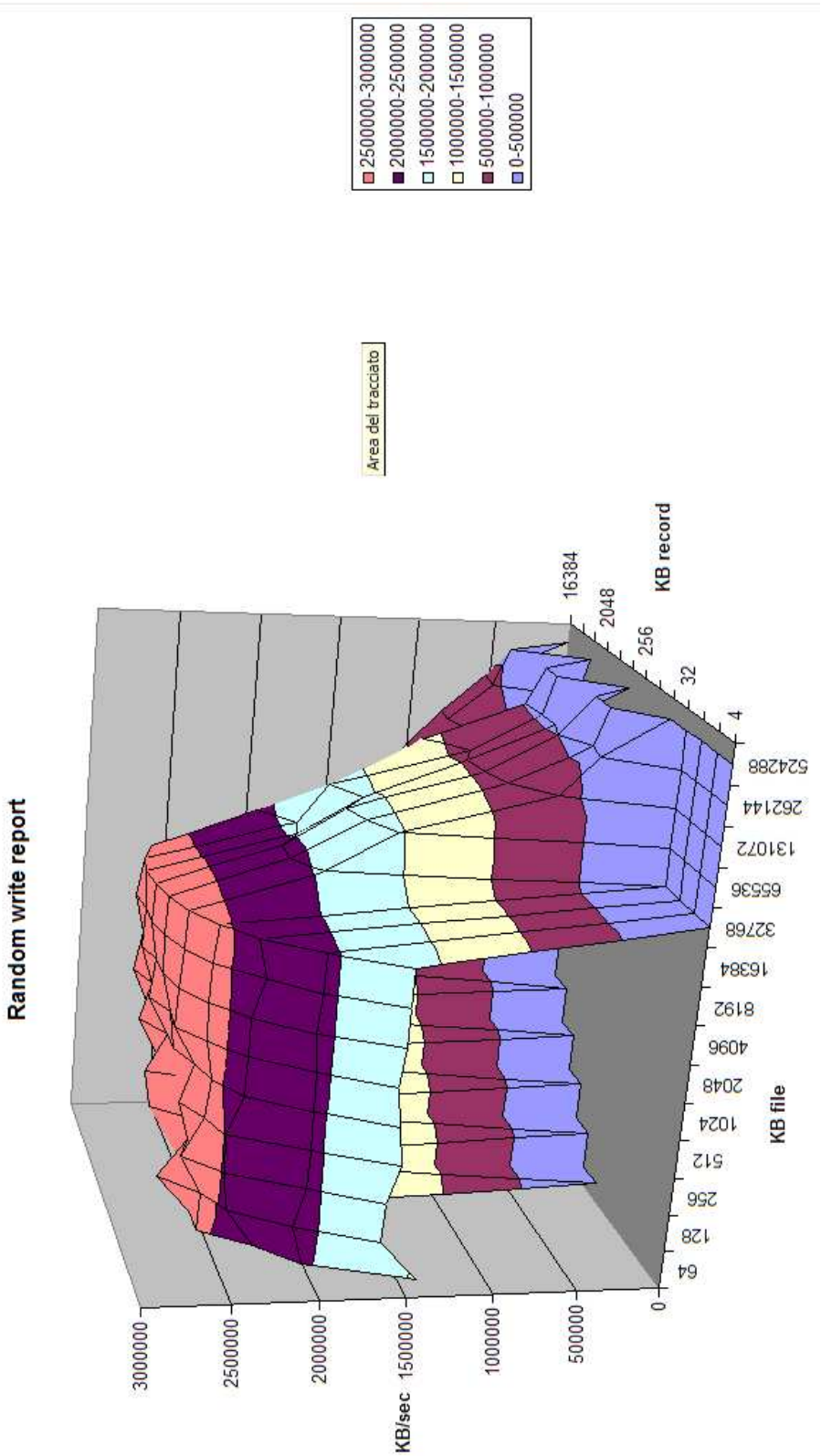

Figura 4.12: IOzone: test di scrittura casuale, kernel con patch monitoraggio

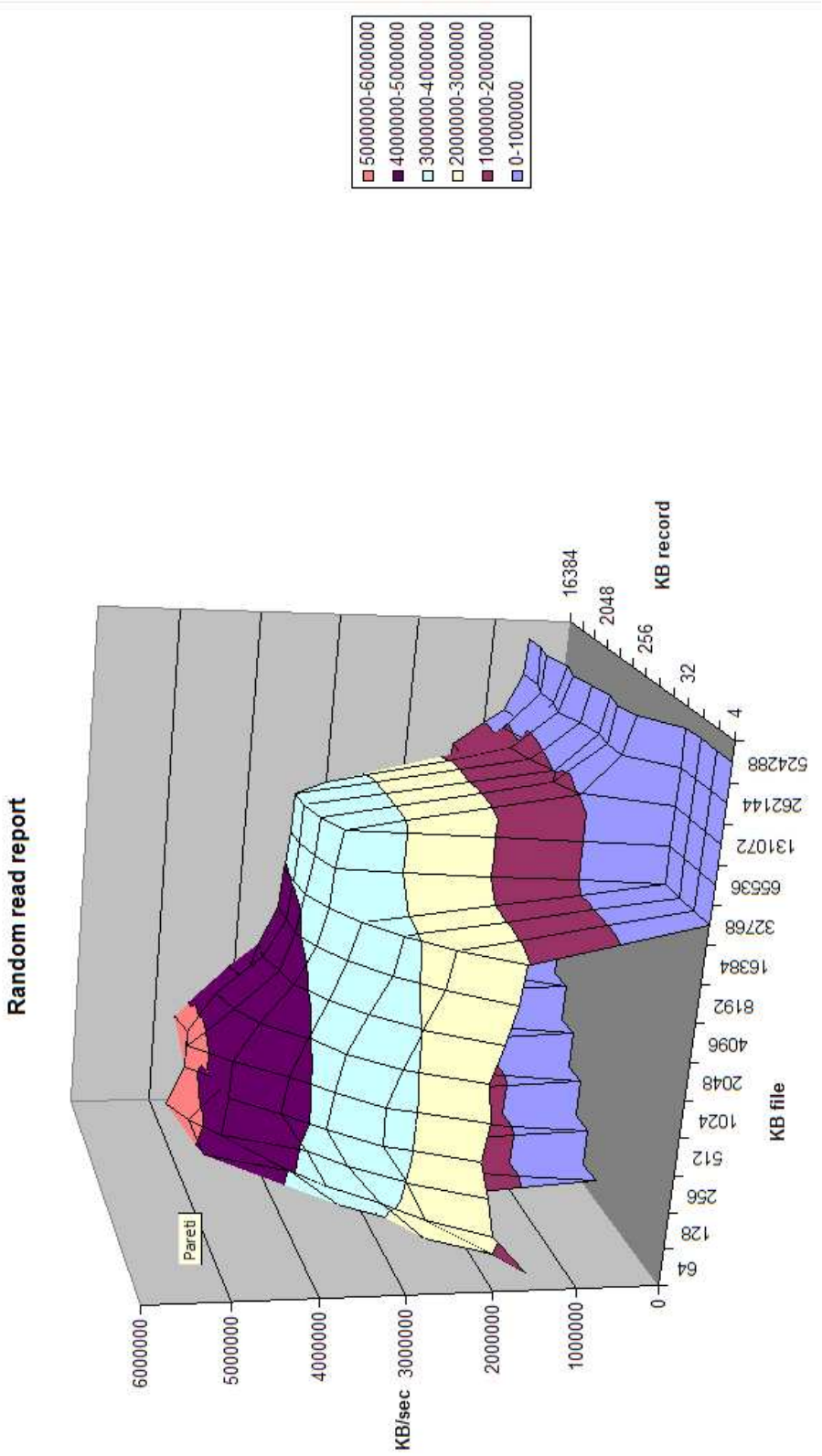

Figura 4.13: IOzone: test di lettura casuale, kernel vanilla

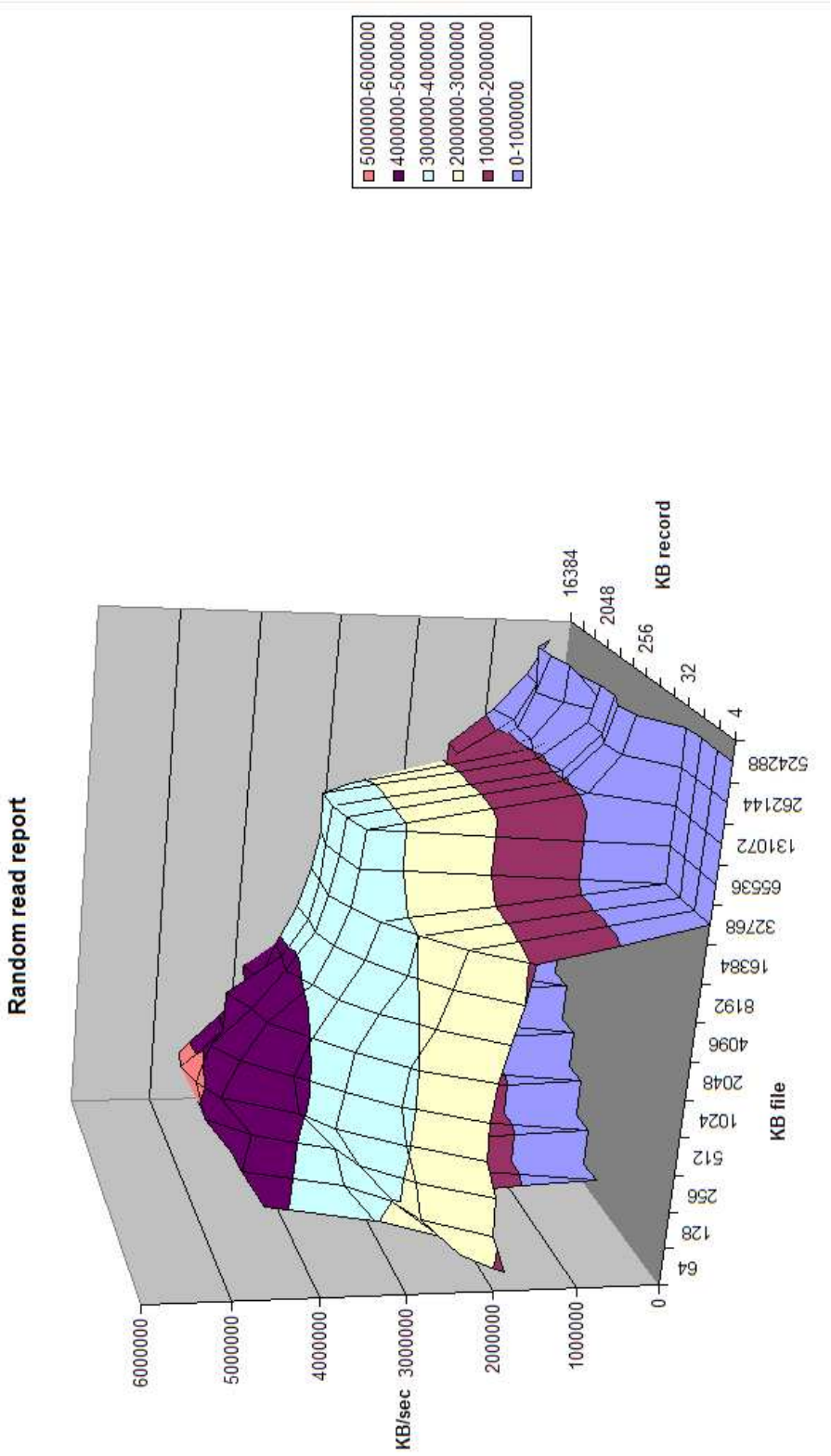

Figura 4.14: IOzone: test di lettura casuale, kernel con patch monitoraggio

4.4 Conclusioni

Il sistema proposto permette di monitorare, e controllare, in modalità remota, un numero arbitrario di macchine virtuali Xen.
La prima parte del lavoro effettuato ha evidenziato le difficoltà nell'adattare il concetto di sense of self di un sistema operativo all'ambito delle macchine virtuali Xen. Il problema principale è dovuto al fatto che le hypercall sono in numero notevolmente inferiore rispetto alle chiamate di sistema messe a disposizione da un sistema operativo. Di conseguenza, le informazioni che si riescono ad estrapolare analizzando le sequenze di hypercall non possono offrire lo stesso livello di dettaglio garantito dalle sequenze di syscall.
Il prototipo, realizzato in C, ha comunque dimostrato la fattibilità dell'architettura proposta per il monitoraggio di macchine virtuali Xen. Il degrado delle prestazioni dovuto all'infrastruttura di monitoraggio è, però, non sempre trascurabile. Pertanto, si può valutare la possibilità di utilizzare il monitoraggio dettagliato delle hypercall effettuate in un ambiente di test, o comunque, che non effettuino frequenti operazioni di I/O su file di piccole dimensioni.

Bibliografia

[1] AMD64 Architecture Tech Docs. AMD64 Architecture Programmer's Manual volume 2: System programming.

[2] Application-specific integrated circuit http://en.wikipedia.org/wiki/Application-specific_integrated_circuit.

[3] C. Hosner. OpenVPN and the SSL VPN Revolution. Technical report, SANS, August 2004.

[4] ncurses http://www.gnu.org/software/ncurses/.

[5] Debian GNU/Linux http://www.debian.org/.

[6] W. Le Febvre. top: a top-CPU Usage Dispaly http://www.groupsys.com/topinfo/, 1993.

[7] Stephanie Forrest, Steven A. Hofmeyr, and Anil Somayaji. A sense of self for unix processes. In *In Proceedings of the 1996 IEEE Symposium on Security and Privacy*, pages 120–128. IEEE Computer Society Press, 1996.

[8] R. P. Goldberg. Architecture of virtual machines. In *Proceedings of the workshop on virtual computer systems*, pages 74–112, 1973.

[9] R. Gupta, R. Gardner, and L. Cherkasova. XenMon: QoS Monitoring and Performance Profiling Tool. Technical report, 2005.

[10] RFC2409 - The Internet Key Exchange (IKE), 1998.

[11] Intel 64 and IA-32 Architectures Software Developer's Manual Volume 1: Basic Architecture.

[12] IOzone Filesystem benchmark, http://www.iozone.org.

[13] RFC2401 - Security Architecture for the Internet Protocol, 1998.

[14] RFC2408 - Internet Security Association and Key Management Protocol (ISAKMP), 1998.

[15] kad. Handling Interrupt Descriptor Table for fun and profit. *Phrack*, (59), 2002.

[16] Hugo Krawczyk. Skeme: A versatile secure key exchange mechanism for internet. In *In Proceedings of the Internet Society Symposium on Network and Distributed Systems Security*, 1996.

[17] P. Kulkarni, M. Arnold, and M. Hind. Dynamic compilation: the benefits of early investing. In *VEE '07: Proceedings of the 3rd international conference on Virtual execution environments*, pages 94–104, New York, NY, USA, 2007. ACM.

[18] Cisco Layer Two Forwarding (Protocol) L2F, 1998.

[19] RFC2661 - Layer Two Tunneling Protocol L2TP, 1999.

[20] L2TP Multi-hop Connection.

[21] RFC3193 - Securing L2TP using IPsec, 2001.

[22] LAN/MAN Standards Committee of the IEEE Computer Society. IEEE standard for local and metropolitan area networks - Virtualbridged local area networks-amendment 2: VLAN classification byprotocol and port. 2001.

[23] Linux kernel 2.6.26 include/asm/unistd_32.h.

[24] LMbench, tools for performance analysis, http://www.bitmover.com/lmbench/whatis_lmbench.html.

[25] RFC 3031 - Multiprotocol Label Switching Architecture, 2001.

[26] Understanding PPTP http://technet.microsoft.com/en-us/library/cc768084.aspx.

[27] Nagios: What is Nagios http://nagios.sourceforge.net/docs/3_0/about.html.

[28] RFC2412 - The OAKLEY Key Determination Protocol, 1998.

[29] Gerald J. Popek and Robert P. Goldberg. Formal requirements for virtualizable third generation architectures. *Commun. ACM*, 17(7):412–421, 1974.

[30] RFC1661 - The Point-to-Point Protocol (PPP), 1994.

[31] RFC 2637 Point-to-Point Tunneling Protocol (PPTP) http://tools.ietf.org/html/rfc2637, 1999.

[32] http://www.phrases.org.uk/bulletin_board/15/messages/643.html.

[33] Tobias Oetiker http://oss.oetiker.ch/rrdtool/.

[34] R. Sites, A. Chernoff, M. Kirk, M. Marks, and S. Robinson. Binary translation. *Commun. ACM*, 36(2):69–81, 1993.

[35] SOCKS: A protocol for TCP proxy across firewalls.

[36] RFC1928 - SOCKS Protocol Version 5, 1996.

[37] RFC4252 - The Secure Shell (SSH) Authentication Protocol, 2006.

[38] Ubuntu Linux http://www.ubuntu.com/.

[39] R. Uhlig, G. Neiger, D. Rodgers, A. Santoni, F. Martins, AV. Anderson, S. Bennett, A. Kagi, F. Leung, and L. Smith. Intel virtualization technology. *Computer*, 38(5):48–56, 2005.

[40] UML. http://user-mode-linux.sourceforge.net/.

[41] M. Varian. Vm and the vm community: Past, present, and future. page 20, August 1997.

[42] David Wagner and Paolo Soto. Mimicry attacks on host-based intrusion detection systems. In *In Proceedings of the 9th ACM Conference on Computer and Communications Security*, pages 255–264, 2002.

[43] GNU wget http://www.gnu.org/software/wget/.

[44] Xen Hypervisor http://www.xen.org/.

[45] xen-3.2.0/xen/include/public/xen.h.

[46] xm - Xen management user interface http://linux.die.net/man/1/xm.