NETWORK VIRTUAL MACHINE (NETVM): PORTABILITÀ ED EFFICIENZA NELL'ELABORAZIONE DI PACCHETTI DI RETE

Loris Degioanni*, Mario Baldi*, Diego Buffa**, Fulvio Risso*, Federico Stirano***, Gianluca Varenni*
* Politecnico di Torino, Dipartimento di Automatica e Informatica, Torino, Italy
**Telecom Italia Labs - System On Chip, Torino, Italy
***Istituto Superiore Mario Boella, Torino, Italy
{mario.baldi,loris.degioanni,fulvio.risso,gianluca.varenni}@polito.it;
stirano@ismb.it; diego.buffa@tilab.com

I moderni apparati di rete devono essere in grado non solo di sostenere velocità sempre maggiori, ma anche prevedere la capacità di essere riprogrammati dinamicamente. Questa funzionalità è essenziale per poter introdurre nuove applicazioni come, ad esempio, firewall e sistemi per l'individuazione di intrusioni (intrusion detection system, IDS) direttamente sui dispositivi di mrete. Questo articolo presenta un nuovo network processor virtuale, chiamato NetVM, che dispone di un insieme di istruzioni ottimizzate per l'elaborazione dei pacchetti. La NetVM si prefigge di fornire uno strato di compatibilità per l'elaborazione di pacchetti (come ad esempio il filtraggio, il conteggio, lo string matching) eseguiti da varie applicazioni (firewall, network monitor, IDS) in modo che possano essere eseguiti da un qualunque apparato di rete, dal router di backbone all'access point domestico. Inoltre, la NetVM permetterà timappare efficacemente le sopracitate funzionalità di elaborazione di pacchetti su unità hardware specifiche (come ad esempio, ASIC, FPGA, ed elementi per l'elaborazione di rete) presenti in molti sistemi hardware, come apparati di rete o loro componenti.

1. Introduzione

Nei nuovi apparati di rete, spesso la velocità non è più il parametro fondamentale di valutazione in quanto sempre maggiore importanza viene data alla capacità di offrire una elaborazione flessibile (e spesso complessa) ai pacchetti in transito. Questo è vero in particolare per applicazioni di sicurezza di rete come firewall, sistemi per l'individuazione di intrusioni (intrusion detection system, IDS) e funzionalità di filtraggio in generale. Da questo punto di vista, l'impiego di ASIC (application specific integrated circuits) specifici per l'elaborazione dei pacchetti è problematico in quanto alla loro altissima velocità non corrisponde ad un'analoga versatilità: un ASIC non può essere riprogrammato e il tempo necessario allo sviluppo (e alla prototipazione) di un nuovo chip può essere estremamente alto.

Una soluzione che potrebbe garantire alti tassi di elaborazione dei pacchetti, una nuova forma di programmabilità e un ridotto tempo di sviluppo si può trovare nei Network Processor. Questi sono chip programmabili le cui

architetture sono rivolte in particolare all'elaborazione dei pacchetti. Le loro prestazioni sono ottenute dalla sinergia di differenti componenti. Primo, una unità centrale di elaborazione basata sul paradigma RISC (pertanto semplice e molto veloce) che include un insieme ridotto di istruzioni, cioè quelle più significative per l'elaborazione dei pacchetti (ad esempio mentre le istruzioni in virgola mobile non sono presenti, ci sono istruzioni speciali per la manipolazione dei bit). Secondo, realizzano un alto grado di parallelismo poichè potrebbero esserci differenti unità di esecuzione, all'interno di quella principale, in grado di eseguire più frammenti di codice in parallelo. Terzo, prevedono alcuni moduli hardware specifici in grado di eseguire compiti complessi solitamente necessari per l'elaborazione dei pacchetti (ad esempio motori di accelerazione per la ricerca in tabelle).

Questo articolo fornisce una prima visione relativa ad un progetto che ambisce a definire un architettura di una macchina virtuale ottimizzata per la programmazione di rete, chiamata Network Virtual Machine (NetVM), sul modello di quello che altra macchine virtuali (principalmente CLR [1] o JAVA [2]) realizzano relativamente ai processori general-purpose. Questo lavoro mira a:

- Semplificare e velocizzare lo sviluppo di applicazioni ottimizzate per l'elaborazione dei pacchetti, come firewall, traffic monitor, router;
- Rimappare effiacamente funzionalità di rete su specifici moduli hardware, eventualmente disponibili nel nodo di esecuzione del programma;
- Fornire un ambiente di programmazione unificato per varie architetture hardware;
- Offrire la portabilità di applicazioni per l'elaborazione dei pacchetti su differenti piattaforme hardware e software;
- Fornire una architettura di riferimento per l'implementazione di sistemi di rete hardware (integrati);
- Fornire un nuovo strumento di specifica, prototipazione rapida, e implementazione di sistemi di rete hardware (integrati) finalizzati a una specifica applicazione di elaborazione dei pacchetti.

La sezione 2 sottolinea le motivazioni e i potenziali benefici che si possono trarre da questo lavoro, ed esamina in maniera ancor più approfondita le implicazione sopracitate. L'architettura proposta della NetVM è spiegata nella sezione 3, mentre le problematiche di prestazioni sono illustrate nella sezione 4. La sezione 5 trae alcune conclusioni e riassume i lavori correnti e futuri.

2. Motivazioni e Benefici

NetVM si prefigge di essere una piattaforma portabile, ma efficiente per le applicazioni che hanno necessità di elaborare pacchetti tra cui particolare rilevanza hanno le applicazioni per la sicurezza di rete.

Il vantaggio di scrivere del codice per la NetVM rispetto al codice nativo per un network processor è da ricercarsi nella complessità di questi ultimi dal punto di vista della programmabilità. Ciascun dispositivo dispone di un proprio ambiente di programmazione che solitamente include un compilatore simil-C, che

differisce non solo da costruttore a costruttore, ma anche da differenti linee di prodotto di un singolo costruttore. Comunque, approcci di alto livello come la programmazione C non sono la soluzione ideale, in quanto il C è stato inventato per la programmazione di uso generale: esso manca di parecchie caratteristiche che possono aiutare lo sviluppo di programmi di rete e include, invece, caratteristiche non necessarie. Per esempio i linguaggi di alto livello forniti difettano di alcune funzioni standard (quelle che non dovrebbero essere correlate all'elaborazione per il networking), mentre ne rendono disponibili di nuove (alcune che sono necessarie per poter sfruttare al meglio alcune caratteristiche del chip); pertanto non esistono garanzie per la portabilità.

Di conseguenza, il modo più proficuo per programmare questo tipo di dispositivi (e ottenere programmi efficiente da quest'ultimi) è quello di usare il linguaggio assembly nativo, che è dispendioso in termini di tempo e soggetto a errori. Ciò richiede non solo un'approfondita conoscenza della macchina, ma anche una non trascurabile quantità di tempo per programmare il dispositivo. Inoltre, portare un programma da una piattaforma ad un'altra (anche se appartente allo stesso costruttore) si traduce in uno sforzo notevole.

L'idea su cui si basa l'architettura della NetVM è di mantenere le alte prestazioni fornite dai network processor e al contempo aggiungere un grado di programmabilità ragionevole. Primo, la NetVM definisce l'architettura di un nuovo network processor, che è l'architettura di riferimento che si prefigge di compiere i compiti più comuni necessari per l'elaborazione dei pacchetti. Inoltre, definisce una metodologia per estendere tale architettura mediante funzionalità addizionali (come ad esempio processori hardware dedicati) al fine di garantirne la personalizzazione. Secondo, definisce un linguaggio assembly necessario a programare questi dispositivi virtuali e un insieme di specifiche relative alla interazione tra ciascun blocco (ad esempio memoria, unità di esecuzione, ecc.) all'interno della NetVM. Terzo, definisce come un'applicazione può interagire con questi componenti, ad esempio come scaricare del codice, come ottenere dei risultati e così via.

Il principale scopo della NetVM è di fornire ai programmatori un riferimento architetturale, tale che possano concentrarsi su cosa fare sui pacchetti, piuttosto che come farlo. Ciascun fornitore può definire un compilatore per la propria piattaforma che traduce questo codice in codice specifico per il dispositivo. Ciò garantisce un alto grado di portabilità del codice su differenti dispositivi, mantenendo, al contempo, le caratteristiche di prestazioni del network processor.

La NetVM, essendo un processore virtuale, può essere emulato su un processore generico, oppure realizzato su un network processor oppure un ASIC dedicato, ossia l'architettura delle macchina virtuale potrebbe essere utilizzata (come base) per il progetto di una architettura hardware per l'elaborazione di rete (ad esempio, un network processor). Nel primo caso, i benefici in termini di velocità sono discutibili rispetto a codice nativo; negli altri casi, invece, le istruzioni della NetVM specifiche per il networking possono essere efficientemente eseguite da unità funzionali ad hoc, con conseguente

guadagno prestazionale. In altre parole, la NetVM potrebbe fornire supporto per prototipazione veloce, alla specifica e all'implementazione di un sistema hardware orientato alla rete.

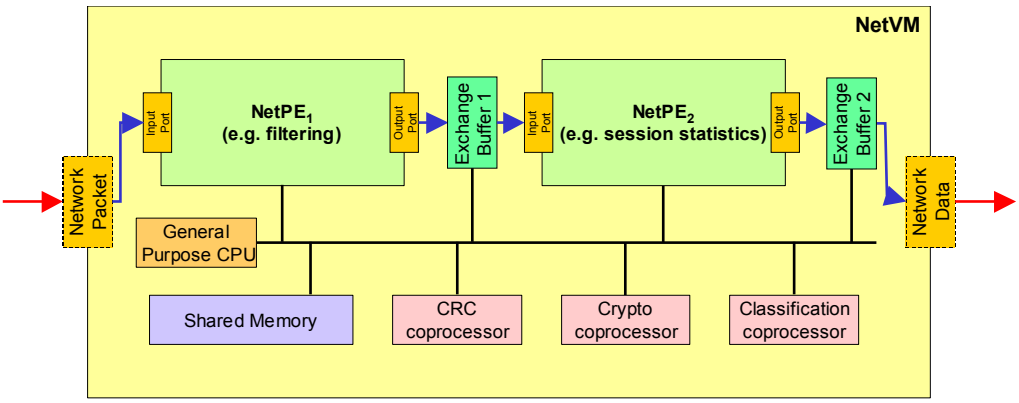

Figura 1. Esempio di impiego di una NetVM.

La Figura 1 illustra un esempio di impiego di un ambiente di NetVM: l'intera architettura è costruita intorno all'elemento di elaborazione. Come mostrato in figura, ciascun NetPE (analogamente a una unità di elaborazione di network processor) isola una specifica funzionalità e usa alcune unità funzionali specializzate (co-processori) e alcune memorie condivise per scambiare dati e pacchetti tra tutti gli elementi della maccina virtuale.

3. Architettura dalla NetVM

I principali obiettivi della NetVM sono **flessibilità**, **semplicità** ed **efficienza**. Tali obiettivi e l'esperienza maturata nel campo delle architetture per network processor hanno determinato le principali scelte architetturali del progetto.

Il processamento dei pacchetti, come ben noto, è adatto a una elaborazione in serie a stadi multipli, o anche in parallelo: infatti, le varie fasi possono essere suddivise facilmente tra piccole unità di processamento di ridotte prestazioni. Per questo, la NetVM ha un'architettura modulare costruita intorno al concetto di *Unità di Elaborazione* (*Processing Element – NetPE*), che virtualizza (o, come sarebbe meglio dire, si ispira a) i micro-elaboratori presenti in un processore di rete reale.

Le singole Unità di Elaborazione (sia quelle integrate che quelle realizzate via software) devono occuparsi solo di compiti ridotti, ma lo devono fare con stringenti vincoli di prestazioni: devono elaborare i dati alla velocità di linea e in tempo reale, devono processare dati con una dimensione variabile (ad esempio il campo dati di un pacchetto IP) e/o dati frammentati (ad esempio un pacchetto IP suddiviso su diverse celle ATM). Di conseguenza, questi semplici elementi devono possedere meccanismi per un'avanzata gestione della memoria ed eseguire algoritmi di schedulazione nel caso le unità abbiano accesso diretto alla memoria. Inoltre, sono responsabili di compiti molto specifici, come ricerche binarie in complesse strutture ad albero e calcolo di CRC (Codice a Ridondanza Ciclica) che devono essere eseguiti in tempi molto ristretti.

La capacità di eseguire operazioni distinte contemporaneamente dovrebbe essere una caratteristica basilare di un processore di rete, perciò bisogna tener conto di questo aspetto durante la fase di progetto dell'architettura: infatti i pacchetti sono sufficientemente indipendenti tra di loro ed è possibile processarli separatamente. Per esempio, uno dei primi network processor – Intel IXP 1200 – è composto da sei unità di elaborazione chiamate "Packet Engines". Maggiore è il numero di unità, maggiore è il grado di parallelismo raggiungibile, dal momento che pacchetti indipendenti possono essere distribuiti sui diversi elementi.

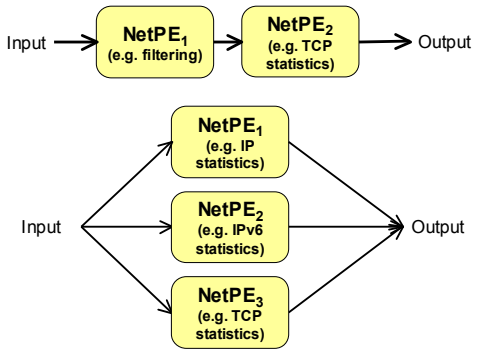

Figura 2. Diversi NetPE organizzati in serie (in alto) o in parallelo (in basso).

Nell'architettura della nostra NetVM, un NetPE è una CPU virtuale (con un opportuno set di istruzioni e una memoria locale) la quale esegue un programma in liguaggio nativo per svolgere una singola funzione all'interno della NetVM ed ha uno stato (registri, stack) privato. Un'applicazione per NetVM è realizzata con la combinazione di vari NetPE (ad esempio, in Figura 1 è rappresentata un'applicazione formata da due NetPE) ognuno dei quali esegue uno specifico compito; strutture più complesse possono essere realizzate collegando diversi NetPE tra di loro. Una tale visione modulare deriva dall'osservazione che la maggior parte delle applicazioni di elaborazione di pacchetti possono essere scomposte in blocchi più semplici, i quali, opportunamente combinati, possono dar luogo a strutture complesse. I singoli blocchi possono lavorare sia in parallelo, sia in serie per raggiungere alte prestazioni. L'approccio modulare non è una novità: altre soluzioni realizzate in software, come *NetFilter* [17] o *Click* [19] hanno dimostrato la loro validità, e molti network processor seguono la stessa strada del parallelismo, basandosi su un elevato numero di semplici micro-processori.

La Figura 2 mostra come è possibile collegare tra loro i NetPE per realizzare strutture di complessità maggiore, sia in serie che in parallelo. Nel primo caso, i pacchetti in uscita da un elemento sono passati all'interfaccia di ingresso del successivo. Nel secondo caso i pacchetto in arrivo da una singola sorgente sono elaborati in parallelo da due o più NetPE.

3.1. NetPE: l'unità di elaborazione di base

Il NetPE è il cuore della NetVM. È l'unità fondamentale che esegue il codice compilato, compiendo l'effettiva elaborazione dei dati in arrivo dalla rete.

Come la maggior parte dei processori virtuali disponibili, la NetVM ha una struttura basata su *stack*. Un processore virtuale basato su stack garantisce portabilità, un insieme di istruzioni compatto e una macchina virtuale semplice. La principale conseguenza di questa scelta è l'assenza di registri generici e tutte le istruzioni che necessitano di memorizzare o elaborare un valore fanno uso dello stack. Ogni NetPE ha il proprio stack indipendente.

Il modello di esecuzione è basato su *eventi*. Ciò significa che l'entrata in funzione di un particolare NetPE è determinata da eventi esterni, ognuno dei quali scatena l'esecuzione di una particolare porzione di codice. Eventi tipici sono l'arrivo di un pacchetto da un'interfaccia di ingresso, la richiesta di un pacchetto sull'interfaccia di uscita o lo scattare di un timer.

Il codice nativo dei NetPE

L'insieme delle istruzioni della NetVM è ricavato da quello utilizzato da un generico processore basato su stack, con aggiunte specifiche dovute alla particolare architettura e destinazione d'uso di questa macchina virtuale. Le istruzioni possono essere suddivise in diverse categorie; le più importanti sono elencate nella TABELLA I.

TABELLA I
ESEMPIO DI ALCUNE ISTRUZIONI PRESENTI NEL CODICE NATIVO DI UN NETPE.

Categoria	**Descrizione**	**Esempio**	**Descrizione**
Inizializzazione	Inizializzazione dell'esecuzione di un'applicazione per NetPE	set.share	Imposta la dimensione della memoria condivisa
Trasferimento dati	Trasferimento di dati all'interno della memoria	dpcopy	Copia un buffer di memoria da una posizione ad un'altra
Ricerca di stringhe	Controllo di un valore in un buffer di memoria con il valore in cima allo stack	field.eq.8	Confronta il valore in cima allo stack con un valore a 8 bit in memoria
Controllo di flusso	Controllo del flusso di esecuzione dell'applicazione	jump	Salto senza condizioni
Gestione dello stack	Gestione dello stack	swap	Scambia i due elementi in cima allo stack
Aritmetica e Logica	Calcolo di semplici operazioni	ror	Rotazione a destra del valore in cima allo stack

Avendo come obiettivo l'esecuzione di applicazioni di rete, l'insieme delle istruzioni della NetVM comprende un gruppo di istruzioni specifiche, non presenti nei tradizionali elaboratori basati su stack. La maggior parte di queste istruzioni riflettono le funzioni che alcuni network processor forniscono per velocizzare l'elaborazione delle intestazioni dei pacchetti. Nella TABELLA II sono evidenziate alcune di queste istruzioni.

Dal momento che, potenzialmente, la NetVM potrebbe essere mappata su sistemi integrati e network processor, l'uso di sistemi di gestione della memoria avanzati quali un *garbage collector* non può essere considerato una soluzione valida. Perciò, il linguaggio nativo ha una visione della memoria direttamente a basso livello. Inoltre, la memoria è allocata *in modo statico* durante la fase di inizializzazione: il programma stesso, grazie a opportune istruzioni, specifica la quantità di memoria necessaria per un corretto funzionamento. Ovviamente,

queste istruzioni possono fallire se non è presente una quantità sufficiente di memoria fisica.

TABELLA II
ESEMPIO DI ISTRUZIONI SPECIFICHE PER L'ELABORAZIONE DI PACCHETTI.

Istruzione	Descrizione
find.bit	Trova il bit impostato. Restituisce la posizione del primo bit impostato a uno all'interno del valore in cima allo stack.
mfind.bit	Trova il bit impostato, con maschera. Restituisce la posizione del primo bit impostato a uno all'interno del valore in cima allo stack, versione con maschera.
clz	Conta gli zeri iniziali. Conta il numero di zeri consecutivi a partire dal bit più significativo nel valore in cima allo stack.
set.bit	Imposta a uno il bit specificato come parametro nel valore in cima allo stack.
clear.bit	Azzera o inverte il bit specificato come parametro nel valore in cima allo stack.
flip.bit	Inverte il bit specificato come parametro nel valore in cima allo stack:
test.bit	Test condizionale su un bit del valore in cima allo stack.
field.*c.t*	Confronta un valore di dimensione *t* nella memoria pacchetto con il valore in cima allo stack. Salta se la condizione *c* è soddisfatta. *t* può valere 8, 16 o 32 (dimensione in bit); *c* può valere eq (uguale a), ne (non uguale a), lt (minore di), le (minore o uguale a), gt (maggiore di), ge (maggiore o uguale a). questa istruzione può essere usata come aiuto nella decodifica di un'intestazione specifica di un protocollo.

La flessibilità persa con questo approccio è bilanciata da un guadagno in efficienza: il programma può accedere alla memoria senza intermediari grazie a opportune istruzioni di caricamento e salvataggio dati. Per lo stesso motivo, istruzioni specifiche per la copia di buffer di memoria (un'operazione ricorrente nell'elaborazione di rete; alcune piattaforma possiedono anche unità hardware dedicate) sono disponibili, sia per copie all'interno della stessa memoria, che tra memorie differenti. Inoltre, la conoscenza della posizione e della dimensione della memoria prima che il programma sia lanciato permette accessi molto veloci quando si usa un compilatore di tipo JIT in quanto gli indirizzi di memoria possono essere pre-calcolati.

I pacchetti sono memorizzati in buffer specifici, chiamati *buffer di scambio*, che sono condivisi tra due NetPE sullo stesso cammino di elaborazione in modo da minimizzare i problemi di condivisione durante lo scambio dei dati. Per ottimizzare la gestione dei pacchetti rendendola il più veloce possible, istruzioni specifiche relative ad operazioni di rete (ad es. ricerca di stringhe) e coprocessori aggiuntivi potrebbero avere accesso diretto a questi buffer. Allo stesso modo sono fornite istruzioni per il trasferimento dei dati (in ingresso e in uscita, o all'interno dei buffer di scambio).

3.2. Coprocessori

Come i network processor integrati aggiungono funzionalità specifiche ai processori generici composti da un singolo chip, analogamente la NetVM aggiunge funzionalità specifiche per l'elaborazione di rete ad un insieme standard di istruzioni per strutture basate su stack. Queste aggiunte specifiche prendono il nome di *coprocessori*.

Questo approccio ha un doppio vantaggio. Primo, i coprocessori possono essere mappati su funzionalità (ove presenti) dei network processor, sfruttando l'hardware avanzato e i coprocessori di cui spesso sono forniti. Questo

permette di incrementare enormemente l'efficienza dei programmi quando la piattaforma da emulare fornisce l'hardware adatto. Per esempio, un programma che utilliza la funzionalità CRC32 di un coprocessore CRC sarà molto più efficiente e semplice su una piattaforma con hardware dedicato per il calcolo del CRC. Secondo, su sistemi generici questi coprocessori permettono l'uso di algoritmi ottimizzati. Essi condividono codice e strutture dati tra moduli differenti, garantendo in questo modo un buon utilizzo delle risorse. Per esempio, in una configurazione con diversi NetPE che utilizzano la funzione CRC32, lo stesso coprocessore può essere utilizzato da tutti i NetPE. Inoltre, se un nuovo algoritmo aumenta l'efficienza della funzione CRC32, tutti i NetPE che utilizzano tale funzione diventano più veloci, migliorando in questo modo l'intera configurazione e non solo quella di un singolo modulo. Infine, compiti quali ricerca di stringhe o classificazione possono condividere strutture dati e tabelle tra moduli differenti per una efficienza ancora migliore e un'ottimizzazione nell'utilizzo di risorse. Un esempio è l'algoritmo Aho-Corasick per la ricerca di stringhe, il quale utilizza un solo automa a stati per ricercare stringhe multiple.

I NetPE comunicano con i coprocessori attraverso interface ben definite. In Figura 1 (parte inferiore) sono rappresentati alcuni coprocessori che possono essere presenti nel progetto di una NetVM base.

4. Valutazione delle prestazioni

Nonostante l'attuale implementazione della NetVM sia ancora in uno stato embrionale, è possibile determinare alcune misure per valutare la bontà dell'architettura proposta.

Questa valutazione preliminare delle prestazioni è stata fatta confrontando la NetVM con il BPF, probabilmente la macchina virtuale più conosciuta nell'ambito dell'elaborazione di rete. La Figura 3 mostra un breve programma nel linguaggio nativo del BPF che, data una trama Ethernet, determina se la trama contiene un pacchetto IP. La Figura 4 mostra un programma simile scritto, però, nel linguaggio nativo della NetVM.

```
(0) ldh  [12]               ; load the ethertype field
(1) jeq  #0x800  jt 2  jf 3 ; if true, jump to (2), else to (3)
(2) ret  #1514              ; return the packet length
(3) ret  #0                 ; return false
```

Figura 3. Esempio di codice che filtra pacchetti IPv4 con la macchina virtuale BPF.

Ad un primo sguardo si nota che il codice nativo della NetVM è decisamente più ricco di quello del BPF, dando una prima idea delle potenzialità del linguaggio della NetVM. Tuttavia, il programma finale è di gran lunga meno compatto (il nucleo del programma è composto da sei istruzioni contro le tre richieste dal linguaggio del BPF). Ciò deriva da una delle più importanti caratteristiche dell'architettura della NetVM: la macchina virtuale basata su stack è meno efficiente di una macchina virtuale basata su registri (come il BPF) in quanto non si può appoggiare su un insieme di registri generici. Non si possono, quindi,

confrontare direttamente le prestazioni ottenute con la NetVM con quelle ottenute con il BPF, senza tener conto di questo fattore.

```
; Push Port Handler
;   triggered when data is present on a push-input port
segment .push

.locals 5
.maxstacksize 10
  pop               ; pop the "calling" port ID
  push 12           ; push the location of the ethertype
  upload.16         ; load the ethertype field
  push 2048         ; push 0x800 (=IP)
  jcmp.eq send_pkt  ; compare the 2 topmost values; jump if true
  ret               ; otherwise do nothing and return

send_pkt:
  pkt.send out1     ; send the packet to port out1
  ret               ; return
ends
```

Figura 4. Esempio di codice che filtra pacchetti IPv4 con la macchina virtuale NetVM.

TABELLA III
VALUTAZIONE DELLE PRESTAZIONI DELLA NETVM.

Macchina Virtuale	Tempo per eseguire il filtro "IPv4" (cicli di clock)
NetVM	600
BPF	124

La TABELLA III mostra il tempo necessario per eseguire i programmi esposti in Figura 3 e Figura 4: come ci si poteva aspettare, il BPF ha prestazioni migliori della NetVM. Inoltre la NetVM è penalizzata non solo dalla presenza di istruzioni aggiuntive, ma anche dal fatto che il codice del BPF è tradotto direttamente nel codice nativo dei processori x86, mentre le istruzioni della NetVM sono interpretate durante l'esecuzione del programma.

Tuttavia, una NetVM va intesa come sistema di riferimento e non ci si aspetta che il suo codice sia eseguito così com'è. Per migliorare le prestazioni, il codice NetVM deve essere tradotto nel codice nativo (attraverso una ricompilazione durante l'esecuzione, la cosiddetta *Ahead-Of-Time compilation*) secondo le caratteristiche della piattaforma finale. Ciò giustifica la scelta di una macchina basata su stack, che è intrinsecamente più lenta, ma le cui istruzioni sono più semplici da tradurre in codice nativo. Ci si aspetta di ottenere prestazioni decisamente superiori dopo una ricompilazione dinamica. L'implementazione di un compilatore di tipo AOT è parte del lavoro futuro sulla NetVM.

5. Conclusioni

Questo articolo presenta l'architettura della Network Virtual Machine (NetVM), una macchina virtuale ottimizzata per la realizzazione di applicazioni fortemente basate sull'elaborazione di pacchetti, tra cui spiccano per importanza le

applicazioni per la sicurezza di rete. Questo articolo discute le motivazioni alla base della definizione di tale architettura e i benefici che si possono trarre dal suo impiego in diverse piattaforme hardware, che includono la semplificazione e velocizzazione dello sviluppo di applicazioni per l'elaborazionde dei pacchetti la cui esecuzione può essere efficientemente delegata a componenti hardware specializzati. In aggiunta, la NetVM fornisce un ambiente di programmazione unificato per diverse architetture hardware offrendo, pertanto, la portabilità di applicazioni per l'elaborazioni dei pacchetti su differenti piattaforme hardware e software. Inoltre, l'architettura proposta può essere usata come architettura di riferimento per l'implementazione di sistemi di rete hardware (integrati). Infine, la NetVM può essere usata come uno strumento innovativo per le specifiche, la prototipazione veloce, e l'implementazione dei sistemi di rete hardware (integrati).

Alcuni risultati preliminari delle prestazioni di un semplice programma NetVM mostrano che altre macchine virtuali più semplici, pensate per applicazioni di rete, superano in prestazioni la NetVM che, a sua volta, fornisce una maggiore flessibilità. L'attività corrente di realizzazione di un compilatore Just In Time (JITTER) per il codice NetVM ha lo scopo si ribaltare o almeno di ridurre la discrepanza in termini di prestazioni.

Poichè scrivere codice nativo (bytecode) per la NetVM non è immediato, il lavoro è stato svolto nella direzione di definire un linguaggio di programmazione di alto livello e di implementare il compilatore corripondente nel bytecode della NetVM. Infine, allo scopo di dimostrare a pieno i benefici, anche in termini di prestazioni, apportati dalla NetVM, attività ulteriori includono l'implementazione di una macchina virtuale e del suo JITTER per un network processor commerciale.

6. Riferimenti bibliografici

[1] ECMA standard 335, Common Language Infrastructure (CLI), 2nd edition, December 2002.

[2] T. Lindholm, F. Yellin, The Java Virtual Machine Specification Second Edition, 1999.

[3] P. Russell et al., Netfilter, http://www.netfilter.org.

[4] R. Morris, E. Kohler, J. Jannotti and M. F. Kaashoek: The Click modular router. Proceedings of the 1999 Symposium on Operating Systems Principles.