Francesco Soncina

DOCUMENTO FINALE PROGETTO VSD

2014

Indice

1 UMsandnet **5**
1.1 Descrizione 5
1.2 Funzionamento 5
1.3 Possibili utilizzi 6
1.4 Sviluppi futuri 6

2 Gvde **7**
2.1 Descrizione 7
2.2 Il sistema utilizzato 7
2.3 Gvde vs vde 8
2.3.1 I test effettuati 8
2.3.2 I risultati 8
2.3.3 Analisi 8
2.4 Vde_pcapplug2 9
2.4.1 I test effettuati 10
2.4.2 I risultati 10
2.4.3 Analisi 11
2.4.4 Possibili sviluppi 12

Capitolo 1

UMsandnet

1.1 Descrizione

UMview permette di creare delle macchine virtuali a livello di syscalls, ovvero l'esecuzione dentro alla macchina è identica all'esecuzione fuori, tranne per le chiamate a sistema, che vengono intercettate dal core di *UMview* e delegate ad eventuali moduli e sottomoduli per gestirne la virtualizzazione. *UMsandnet* è un modulo per *UMview* che permette di supervisionare in modo interattivo la connettività di un singolo processo ed i suoi discendenti, ovvero la possibilità di decidere se consentire traffico in uscita od entrata. Il funzionamento del modulo si basa sull'intercettazione di determinate socketcalls, lasciando la decisione della loro effettiva esecuzione all'utente, permettendo quindi di avere una sorta di firewall prototipale in userspace per singoli processi.

1.2 Funzionamento

Le socketcalls principali interessate sono le seguenti:

- *int socket(int domain, int type, int protocol)*: è possibile approvare/negare traffico di basso livello, i.e. domain == AF_PACKET || type == SOCK_RAW, che altrimenti sfuggerebbe alle socketcalls seguenti.

- *int connect(int sockfd, const struct sockaddr *addr, socklen_t addrlen)*: è possibile approvare/negare tutte le connect() oppure solo per certi ip.

- *int bind(int sockfd, const struct sockaddr *addr, socklen_t addrlen)*: è possibile approvare/negare tutte o le singole bind().

1.3 Possibili utilizzi

Questo modulo è stato pensato per dotare l'utente di un controllo basilare sulla connettività di un singolo processo sfruttando la virtualizzazione parziale offerta da *UMview*, ovvero intercettando le socketcalls prima che vengano effettivamente gestite dal kernel, permettendole o negandole a seconda dei casi. Per quanto riguarda la rete, il processo eseguito dentro la macchina virtuale parziale non ha un accesso diretto con il kernel e quindi si possono avere tutti i vantaggi di aver installato un firewall nel sistema, ma eseguendo solo codice utente e solo per quella specifica macchina virtuale. Se si vuole eseguire un processo senza che abbia la possibilità di usufruire della rete, dovremmo, se per esempio non si ha a disposizione un firewall, staccare la rete a tutta la macchina virtuale, per essere sicuri che quel processo non riesca in nessun modo ad utilizzare la rete, poiché esso ha un contatto diretto col kernel come tutti gli altri processi attivi; al contrario la virtualizzazione parziale ci fornisce un metodo comodo (solo codice userspace) ed efficiente (viene virtualizzato solo l'accesso alla rete) per separare la connettività di un processo (la macchina virtuale) da quella del sistema host. Si potrebbe anche essere interessati a consentire tutto il traffico di un processo tranne verso certi ip, senza dover negare il traffico verso quell'ip a tutto il sistema attraverso *iptables*.

1.4 Sviluppi futuri

UMsandnet si dedica solo al controllo della rete, quindi in futuro potrebbe trovare un ruolo complementare ad altri moduli scritti con lo stesso obbiettivo che però potrebbero controllare la creazione, la modifica e la rimozione di files oppure l'esecuzione di altre syscalls critiche, in modo da poter trasformare UMview in una sorta di sandbox interattiva dove l'utente possa decidere, prima che lo faccia il kernel, se un processo debba, o no, eseguire certe syscalls che in date situazioni potrebbero essere ritenute critiche.

Capitolo 2

Gvde

2.1 Descrizione

Gvde è una versione modificata di *vde* che utilizza memoria condivisa invece dei sockets per far comunicare le macchine virtuali connesse allo switch. Si è quindi ritenuto necessario confrontare la nuova versione con la precedente al fine di valutare l'entità di eventuali benefici dovuti alla nuova implementazione.

2.2 Il sistema utilizzato

Tutti i test sono stati eseguiti su una distribuzione debian stable (wheezy)[1] utilizzando per *gvde* e *libvdeplug* il codice fornito e per *vde* l'ultima versione disponibile sul repository ufficiale con la *libvdeplug* fornita dal sistema. I software utilizzati, tutti presi dai repository debian, sono stati:

- *Linux kernel* versione 3.2.0-4-686-pae (3.2.51-1).
- *Qemu-kvm* versione 1.1.2.
- *Libvdeplug* versione 2.3.2-4.
- *Iperf* versione 2.0.5 (08 Jul 2010) pthreads, per le misurazioni dell'ampiezza di banda e dei pacchetti persi.
- *Libpcap* versione 1.3.0-1.

Il computer utilizzato è munito di processore Intel i7 2600K (quadcore con hyperthreading e vanderpool) e 8 GB di memoria ram.

[1] http://snapshot.debian.org/archive/debian/20140123T042010Z/

2.3 Gvde vs vde

La principale differenza tra *gvde* e *vde* sta nella gestione dello smistamento dei pacchetti: in *vde* lo switch è incaricato di ricevere ed inviare i dati alle macchine virtuali, partecipando quindi attivamente al trasferimento dei dati; in *gvde* lo smistamento viene effettuato direttamente da *libvdeplug* mediante memoria condivisa, quindi attraverso la libreria caricata dalla macchina virtuale, riducendo quindi l'utilizzo del processore da parte dello switch a zero.

2.3.1 I test effettuati

Sono state prese in esame le configurazioni composte da due, quattro, sei ed otto macchine virtuali *qemu-kvm*, comunicanti due a due tra di loro attraverso l'utility *iperf* pensata specificatamente per analizzare le prestazioni di rete: prima viene eseguito in modalità TCP per stimare l'ampiezza di banda B, poi in modalità UDP, che necessita come input la velocità d'invio, per analizzare la quantità di pacchetti persi. In particolare, per la modalità UDP è stato generato traffico in ambo le direzioni con ampiezza $\frac{B}{2*\#VMs}$ per singolo flusso.

2.3.2 I risultati

Legenda:

- *#VMs*: numero di istanze *qemu-kvm*.
- *BANDA MASSIMA*: somma flussi due a due unidirezionali misurati con *iperf* in modalità TCP.
- *PACKETS LOSS*: media delle percentuali di pacchetti persi segnalate da ogni istanza di *iperf* in modalità UDP bidirezionale.
- *CPU KVM*: massimo utilizzo del processore tra tutte le istanze di *qemu-kvm*.
- *CPU VDE*: utilizzo del processore di *vde*.

2.3.3 Analisi

Come possiamo notare dalle tabelle 2.1 e 2.2, *gvde* si mostra più performante e scalabile, con una crescita di pacchetti persi proporzionale al numero di macchine virtuali connesse ed un guadagno consistente in termini di

#VMs	BANDA MASSIMA	PACKETS LOSS	%CPU KVM	%CPU VDE
2	390Mbits/s	0.45%	100%	24%
4	540Mbits/s	0.37%	90%	25%
6	550Mbits/s	0.35%	88%	26%
8	555Mbits/s	0.20%	76%	26%

Tabella 2.1: Vde

#VMs	BANDA MASSIMA	PACKETS LOSS	%CPU KVM
2	420Mbits/s	2.4%	100%
4	560Mbits/s	0.65%	100%
6	720Mbits/s	0.77%	90%
8	720Mbits/s	1%	77%

Tabella 2.2: Gvde

ampiezza di banda disponibile all'aumentare dei flussi di trasferimento fino al raggiungimento di una costante, la quale è stata raggiunta con sei flussi contemporanei generati da altrettante macchine. Per quanto riguarda *vde*, il limite di trasferimento viene già sfiorato con solo quattro istanze connesse e la percentuale di pacchetti persi sembra dipendere più dall'ampiezza del singolo flusso che dalla loro somma (una volta che questa ha raggiunto il limite), infatti la percentuale di pacchetti persi ha un andamento decrescente. Curioso notare una percentuale di pacchetti persi molto alta con *gvde* e solo due macchine virtuali, ripetibile sperimentalmente, che potrebbe indicare un limite massimo per la velocità di trasferimento per un singolo flusso in questa modalità.

2.4 Vde_pcapplug2

Vde_pcapplug2 permette di connettere la rete virtuale gestita dallo switch ad un'interfaccia di rete e, quindi, ad una rete fisica esistente, trasmettendo tutto ciò che riceve da un capo della connessione all'altro, ovvero inoltrando tutto il traffico proveniente dallo switch sulla rete fisica e viceversa. La particolarità di *vde_pcapplug2* sta nel sfruttare socket mmappati per ricevere e trasmettere pacchetti invece di utilizzare la libreria libpcap, ovvero condividendo due porzioni di memoria con il kernel attraverso la syscall *mmap()*, le quali saranno trattate come due buffer circolari, uno per la ricezione ed uno per l'invio: il processo utente per inviare dati riempirà il buffer op-

portuno, contrassegnando di volta in volta le posizioni scritte in modo che il kernel autonomamente possa processarle e ricontrassegnarle come vuote; analogo funzionamento per la ricezione dove sarà il kernel a riempire il buffer e l'utente a leggere e liberare le posizioni. Il vantaggio di questa modalità è l'eliminazione delle syscalls *read()*, *write()* ed affini per ricevere ed inviare, in modo da ridurre i *context switches* per eseguire le varie syscalls.

2.4.1 I test effettuati

Sono stati effettuati test analizzando il traffico passante tra una macchina virtuale collegata a *gvde* ed una macchina fisica connessa direttamente via cavo all'interfaccia *eth1*, separata dalla rete principale per non aver interferenze sui dati raccolti, con i protocolli TCP ed UDP. Si è poi proceduto alla ricerca della causa dello strano comportamento di *vde_pcapplug2*, effettuando modifiche al codice per controllare tempistiche di smistamento e modalità di funzionamento dei buffer circolari, ovvero modalità di visita del buffer e parametri di inizializzazione dei buffer stessi. Si è inoltre utilizzata anche l'utility *valgrind* per un controllo generale, ma sistematico, del codice.

2.4.2 I risultati

Legenda:

- *DIREZIONE*: con *IN* si intende che il flusso è verso la macchina con lo switch attivo, con *OUT* il contrario, con *IN/OUT* flusso bidirezionale.
- *BANDA MASSIMA*: somma flussi misurati con *iperf* in modalità TCP.
- *BANDA RICHIESTA*: velocità d'invio passata in input ad *iperf* in modalità UDP.
- *PACKETS LOSS*: media delle percentuali di pacchetti persi segnalate da ogni istanza di *iperf* in modalità UDP.

DIREZIONE	BANDA MASSIMA
IN	17.1Mbits/s
OUT	93.9Mbits/s
IN/OUT	558Kbits/s - 93.8Mbits/s

Tabella 2.3: Vde_pcapplug2 con flussi TCP (socket mmappati)

DIREZIONE	BANDA MASSIMA
IN	15.3Mbits/s
OUT	94Mbits/s
IN/OUT	1.1Mbits/s - 93.9Mbits/s

Tabella 2.4: Vde_pcapplug2 con flussi TCP (libpcap)

DIREZIONE	BANDA RICHIESTA	PACKETS LOSS
IN	50Mbits/s	0%
IN	95Mbits/s	0%
OUT	50Mbits/s	0%
OUT	95Mbits/s	0%
IN/OUT	50Mbits/s	0% - 0%
IN/OUT	95Mbits/s	0.04% - 0.005%

Tabella 2.5: Vde_pcapplug2 con flussi UDP (socket mmappati)

2.4.3 Analisi

Come si può banalmente evincere dalle tabelle 2.5 e 2.6, non si tratta di un problema di perdita di pacchetti. Dai dati raccolti sperimentalmente si nota un comportamento quasi bipolare cambiando protocollo di trasporto: utilizzando UDP ed impostando manualmente la velocità di trasmissione, non ci sono problemi, ricezione ed invio sembrano funzionare, anche con flussi bidirezionali; con TCP *vde_pcapplug2* pare comportarsi in modo inspiegabile, ovvero per quanto riguarda il traffico in uscita dallo switch verso la rete fisica non ci sono problemi mentre quello in entrata sembra essere rallentato, se non addirittura completamente compromesso in prestazioni se si utilizza un flusso bidirezionale. Questa stranezza comunque pare non essere causata da *vde_pcapplug2*, poiché alla luce di modifiche al codice per osservare le modalità di lettura del buffer circolare dedicato al traffico in entrata e quantificare i tempi di invio allo switch dei dati ricevuti non si è trovato nulla di anomalo nel codice: se ad ogni risveglio del processo dopo la syscall *poll()* durante in trasferimento con protocollo TCP viene controllato l'intero buffer dedicato alla ricezione, si può notare come esso stia sempre praticamente vuoto, ovvero con una o due posizioni occupate al massimo, come se fosse proprio il kernel stesso a non rendere disponibili i pacchetti, perché ci si aspetterebbe, in caso di traffico lento, una lentezza nello svuotare il buffer oppure un problema di pacchetti persi, esclusi con l'analisi del traffico UDP.

DIREZIONE	BANDA RICHIESTA	PACKETS LOSS
IN	50Mbits/s	0%
IN	95Mbits/s	0%
OUT	50Mbits/s	0%
OUT	95Mbits/s	0%
IN/OUT	50Mbits/s	0% - 0%
IN/OUT	95Mbits/s	0.02% - 0%

Tabella 2.6: Vde_pcapplug2 con flussi UDP (libpcap)

2.4.4 Possibili sviluppi

In futuro sarebbe interessante provare ad integrare in *vde_pcapplug2* il supporto al framework *netmap* [2] di Luigi Rizzo dell'università di Pisa, utilizzato nello suo switch virtuale *VALE*, il quale è un framework dedicato all'I/O su rete ad alta velocità, basato però su un modulo kernel per migliorare l'efficienza di *linux* nella gestione di *packet capturing* a livello utente.

[2] http://info.iet.unipi.it/~luigi/netmap/