Scuola di Scienze Matematiche, Fisiche e Naturali
Corso di Laurea Magistrale in Informatica

QUANTITATIVE ANALYSIS OF SYSTEMS

Experimental Evaluation Project - E06

A. A. 2017-2018

francesco.mucci@stud.unifi.it
FRANCESCO MUCCI
6173140

"Any measurement that you make without any knowledge of the uncertainty is meaningless."
Walter Lewin, Lecture *"For the Love of Physics"* at MIT on May 16 2011. [1]

INDICE

1. Testo del progetto 1

2. Scopo del progetto e misure di interesse 3
2.1. Presentazione di Snort 3
2.2. Scopo del progetto 3
2.3. Misure di interesse 4
2.3.1. Valutare performance di Snort 4
2.3.2. Valutare uso di risorse da parte di Snort 5

3. Setting sperimentale 7
3.1. Scelta dei sistemi operativi 7
3.2. Caratteristiche dei nodi virtuali 7
3.3. Caratteristiche del sistema host 8
3.4. Caratteristiche della connessione internet 9
3.5. Installazione ed utilizzo di Snort 10
3.6. Strumento di monitoraggio 11

4. Workloads e processi di misura 13
4.1. Scelta e descrizione dei workloads 13
4.2. Procedura sperimentale: dropped e received 14
4.2.1. Errori casuali e sistematici presenti 15
4.3. Procedura sperimentale: CPU-load istantanea 16
4.3.1. Errori casuali e sistematici presenti 17
4.4. Struttura della campagna sperimentale 18
4.5. Misurazioni e condizioni di ripetibilitá 19

5. Analisi dei risultati di misura 21
5.1. Stimare la probabilitá di packet loss 21
5.2. Stimare %CPU media usata da Snort 22
5.3. Analisi delle stime ottenute 24
5.3.1. Analisi delle stime ottenute: Ping 24
5.3.2. Analisi delle stime ottenute: Video 25
5.3.3. Analisi delle stime ottenute: Down 27
5.4. Conclusioni 29

A. Un possibile miglioramento 31

1

TESTO DEL PROGETTO

The project aims to observe and measure the performance of Snort as a packet sniffer / packet logger when run on two virtual nodes with two different operating systems, identifying any differences. The delivery of the project consists of:

- A report describing the adopted experimental evaluation methodology and discussing the results;
- An archive containing: log files, any developed code, dumps of a database, and any other additional material that was created throughout the process.

Here below we present the details of the project to clarify the objective. The project requires an experimental evaluation exercise, with application of the methodology described during the course, aimed at characterizing the behavior of Snort and the underlying system. Snort must be set as a packet sniffer / packet logger. It is required to run Snort on two different virtual nodes, with identical assigned resources but with different operating systems. We report the main elements of the project (note that these elements are not necessarily the only ones or absolutely necessary – students have the possibility to make changes, especially on the level of detail):

- Define the two virtual nodes.
- Identify the quantities of interest, and the objectives of the experiments. Mainly, it is important to observe the network using Snort as a packet logger, and the use of resources by the two considered systems (e.g. using the RCL monitoring tool).
- Define an experimental campaign by creating different workloads and stressloads, related to the generation of network load, to stimulate Snort's packet logging activity. The network load can affect both outgoing traffic and incoming traffic.

Tip 1. VirtualBox, distributed free by Oracle, allows you to create virtual nodes, and assign resources (especially memory and processor) to the created virtual nodes.

Tip 2. It requires the use of different workloads, and possibly also stressloads. There are various ways to obtain them, some possible ideas are reported below:

- Perform invocations to websites;
- Respond to incoming connections by an external node;
- Use sites for connection stress-testing;
- Generate a workload in a synthetic way, for example some names for Linux are the stress tools http://people.seas.harvard.edu/ apw/-stress/, netio https://web.ars.de/netio/ .

Tip 3. There are no restrictions on operating systems to be used.

2 SCOPO DEL PROGETTO E MISURE DI INTERESSE

2.1 PRESENTAZIONE DI SNORT

Figura 1.: Logo di Snort

Snort è un software open source che esegue, in tempo reale, monitoraggio ed analisi del traffico su reti IP [2].
Può essere configurato come:

- packet sniffer: legge i pacchetti che transitano sulla rete e li mostra a schermo;
- packet logger: esegue il log dei pacchetti sul disco;
- NIDS (Network-based Intrusion Detection System): legge i pacchetti e genera allarmi sulla base di regole definite dall'utente [3].

2.2 SCOPO DEL PROGETTO

Il testo del progetto ci chiede di "osservare e misurare la performance di Snort come packet sniffer/logger quando è in esecuzione su due nodi virtuali con risorse assegnate identiche, ma con due diversi sistemi operativi", caratterizzando il comportamento dei sistemi sottostanti ed identificando qualsiasi differenza.

Quindi, ci siamo prefissati come obiettivo quello di identificare quale dei due sistemi operativi selezionati, a parità di risorse assegnate e carico di lavoro, garantistica che Snort abbia una migliore performance dal punto di vista del monitoraggio del traffico.

Inoltre, al fine di caratterizzare lo stato del sistema e comprendere i dati raccolti durante la campagna sperimentale, risulterà importante osservare l'uso di risorse da parte dei due sistemi e stimare, se possibile, l'uso di risorse da parte di

Snort.

2.3 MISURE DI INTERESSE

Andiamo adesso ad identificare le misure di interesse, cioè le grandezze che vogliamo stimare e che ci permetteranno di valutare la performance di Snort ed il suo uso di risorse.

2.3.1 *Valutare performance di Snort*

Per capire come poter valutare la performance di Snort, dobbiamo prima capire in dettaglio il suo funzionamento.

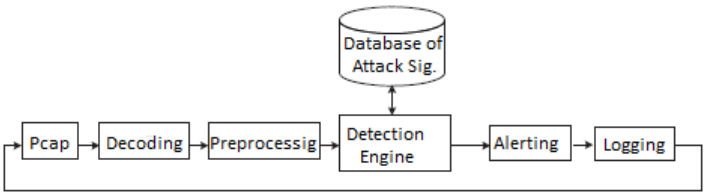

Figura 2.: Componenti software di Snort

Snort è un'applicazione single-threaded costituita dalle componenti software mostrate in Figura 2. Usa libpcap (packet capture library) per accedere ai pacchetti di rete e, per ogni pacchetto così catturato, esegue sequenzialmente tutte le componenti. Solo dopo che tutta la catena sarà stata eseguita per processare un singolo pacchetto, il successivo potrà essere considerato: libpcap metterà in un buffer i pacchetti che arrivano nel mentre. In condizioni di traffico intenso, tale buffer si riempirà rapidamente e per far spazio ai pacchetti in arrivo sarà necessario scartarne altrettanti. Inoltre, in tale situazione di traffico intenso, Snort si bloccherà ripetutamente per eseguire azioni di alerting e/o logging ed il suo tempo di esecuzione si dilaterà notevolmente [4].

Dunque, affinché Snort lavori in modo efficiente, è necessario che riesca a tenere il passo con le sempre maggiori velocità dei link di comunicazione che possono arrivare anche a decine di Gbps.

Per comparare e valutare la performance di Snort si potrebbe, quindi, considerare la seguente misura di interesse:

- $P(\text{packet-loss})$: probabilità che un pacchetto venga scartato.

Risulta importante sottolineare come il numero di pacchetti scartati può essere ricavato direttamente dalle statistiche che Snort ci fornisce a terminazione:

- Received (R): numero totale di pacchetti ricevuti;
- Analyzed (A): numero totale di pacchetti processati;
- Dropped (D): numero totale di pacchetti scartati;
- Outstanding (O): numero totale di pacchetti messi nel buffer.

Inolte, vale la pena evidenziare le seguente relazione:

$$R = A + D. \tag{2.1}$$

Dunque,

$$P(\text{packet-loss}) = \frac{D}{R} \text{ [4]}. \tag{2.2}$$

2.3.2 *Valutare uso di risorse da parte di Snort*

Per capire l'uso di quali risorse dovremmo monitorare, dobbiamo prima sottolineare qualche ulteriore dettaglio riguardo a Snort.

Snort, in base alla modalità selezionata ed al tipo di traffico in arrivo, può essere classificato sia come un processo CPU-bound che I/O-bound [4].

- Processo CPU-bound: sfrutta pesantemente le risorse computazionali del processore (può richiedere il 100% delle risorse per svariati secondi o minuti) e, dunque, il tempo per completare un task è limitato dalla velocità e dalla disponibilità della CPU [5].
- Processo I/O-bound: richiede frequenti operazioni di input/output e, dunque, il tempo per completare una computazione è determinato principalmente dal tempo atteso per il completamento delle operazioni di I/O; la velocità di esecuzione è limitata dalla velocità dei dispositivi di input/output [6].

Ad esempio, come evidenziato nell'articolo "Performance evaluation comparison of Snort NIDS under Linux and Windows Server" [4], in condizioni di traffico normale e senza che sia effettuato il logging, Snort in modalità NIDS si comporterà come un processo CPU-bound in cui la maggior parte del tempo sarà speso processando ed analizzando i pacchetti; in condizioni di traffico malevolo ed in presenza di logging ed alerting, si comporterà, al contrario, come un processo I/O-bound con frequenti operazioni di scrittura su disco o su database dovute al logging ed all'alerting [4].

Noi andremo ad utilizzare Snort come sniffer/logger durante l'esecuzione di diversi workloads che andranno a generare situazioni di traffico di rete più o

meno intenso, ma mai traffico malevolo. Dunque, alla luce di quanto precedente detto, risulterebbe interessante osservare la percentuale di CPU usata ed il numero totale di scritture sul disco. In tal modo, potremmo andare a capire se un numero elevato di pacchetti scartati sia legato al fatto che Snort spenda troppo tempo nell'elaborazione di un singolo pacchetto a causa della scarsa disponibilità e/o velocità della CPU necessaria per processare il pacchetto o a causa delle numerose operazioni di scrittura.

Per una questione pratica, in questo lavoro abbiamo scelto di considerare, come misura di interesse, unicamente la

- %CPU media : percentuale di CPU che, in media, è utilizzata da Snort durante il monitoraggio del traffico.

La nostra misura di interesse dipende dall'intervallo di osservazione del traffico $[t_0, t_f]$ e la possiamo indicare con:

$$\%CPU[t_0, t_f] = \frac{1}{t_f - t_0} \cdot \int_{t_0}^{t_f} \%CPU[t]\, dt \tag{2.3}$$

con %CPU[t] che indica la percentuale di CPU utilizzata all'istante t (%CPU istantanea): questa è la grandezza che dovremmo riuscire a osservare in maniera diretta [1]. Ovviamente, avremo accesso solo ad un numero discreto di osservazioni (assumiamo N):

$$\%CPU[t_0, t_f] \simeq \frac{1}{N} \cdot \sum_{k=1}^{N} \%CPU[k]. \tag{2.4}$$

1 In realtà, in maniera diretta osserveremo la CPU-load istantanea: $\%CPU[t] = 100 \cdot \text{CPU-load}[t]$

3 SETTING SPERIMENTALE

3.1 SCELTA DEI SISTEMI OPERATIVI

Figura 3.: Logo di Ubuntu e Debian

Snort non supporta nessun particolare sistema operativo, anche se ci sono alcune piattaforme su cui è verificata la sua corretta esecuzione [7]; tra queste abbiamo identificato come candidate adatte al nostro studio le seguenti due:

- Ubuntu 14.04.5;
- Debian 8.11.

La scelta è dovuta al fatto che i due sistemi sono abbastanza simili: Debian è Unix-like ed Ubuntu è una distribuzione Linux basata proprio su Debian [8] [9]. Questo ci garantisce che gli stessi software possano essere usati su entrambi e, quindi, i due SO possono essere visti come componenti alternativi per uno stesso uso e tra loro in competizione.

3.2 CARATTERISTICHE DEI NODI VIRTUALI

Per la creazione dei due nodi virtuali ci siamo avvalsi di Oracle VM VirtualBox (versione 5.2), un hosted-hypervisor (software per la creazione ed esecuzione di macchine virtuali) free ed open-source [10].
Ad entrambi i nodi è stato assegnato lo stesso quantitativo di risorse:

- processori: 2 (virtuali);
- RAM: 2048 MB;
- memoria video: 64 MB;
- disco fisso: 100 GB (allocati dinamicamente).

Per quanto riguarda la virtual network card (scheda di rete virtuale) si andrà ad utilizzare l'impostazione di default:

- tipo di scheda: Intel PRO/1000 MT Desktop (82540EM);
- connessa a: NAT.

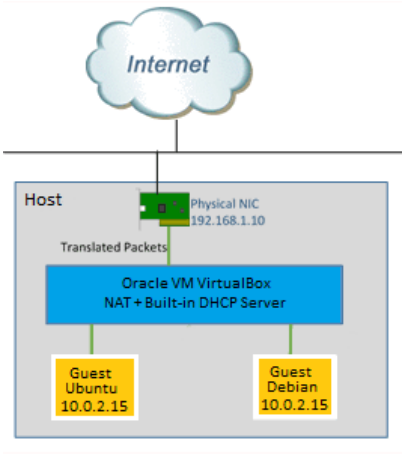

Figura 4.: Configurazione NAT

La modalità "Network Address Translation" consente al sistema guest (il nodo virtuale) di connettersi ad Internet come se fosse un normale computer. La VirtualBox networking engine agisce come un router piazzato tra ogni macchina virtuale e l'host (sistema che ospita i nodi virtuali): ogni nodo riceverà il proprio indirizzo di rete da un DHCP server integrato nella VirtualBox ed ogni frame spedito dal sistema operativo guest verrà ricevuto dal NAT engine della VirtualBox che estrarrà i dati TCP/IP e gli rinvierà usando il sistema operativo host. Ad un computer nella stessa rete dell'host sembrerà che tutti i dati siano stati inviati dall'applicazione VirtualBox usando un IP dell'host. Dunque, lo svantaggio di questa configurazione è che, proprio come una rete privata dietro un router, sulla macchina virtuale non si potrà far girare un server dato che non sarà contattabile dall'esterno a meno che non si imposti il port-forwarding [11].

3.3 CARATTERISTICHE DEL SISTEMA HOST

Come macchina host è stato usato un PC "LENOVO ideapad 310" con sistema operativo Windows 10 Home 64-bit. Le caratteristiche di sistema sono le seguenti:

- processore: Intel Core i5-6200U CPU @ 2.30 GHz (4CPUs)
- RAM: 8.00 GB (8192 MB)
- disco fisso: 420 GB

- scheda video: Intel HD Graphics Family (4048 MB condivisa)
- rendering: NVIDIA GeForce 920MX (4048 MB condivisa)

3.4 CARATTERISTICHE DELLA CONNESSIONE INTERNET

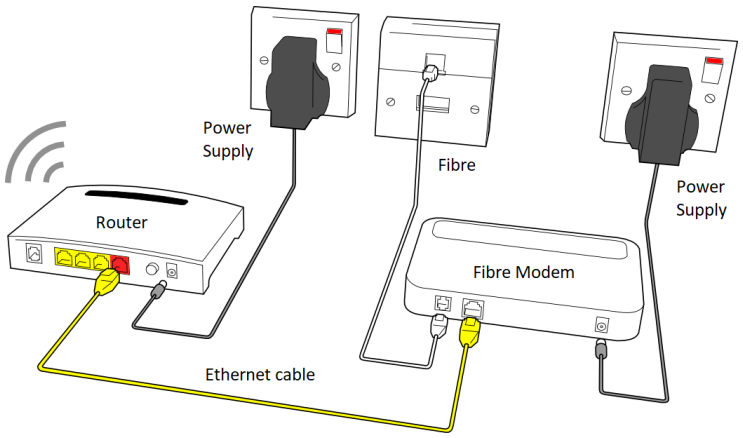

Figura 5.: FTTH router/modem setup

Si utilizzerà una rete di accesso di tipo FTTH (Fiber-to-the-home) con il seguente profilo di velocità nominale:

- in download: 300 Mbps
- in upload: 20 Mbps

ed i seguenti standard minimi di qualità e di prestazioni (QoS garantita):

- Velocità minima garantita (down/up): 180/12 Mbps
- Ritardo massimo trasmissione dati: 50 ms
- Tasso massimo di perdita pacchetti: 0.1 %

La macchina host verrà disposta ad una distanza di poco più di 5 metri dal router di casa (al quale sarà connessa tramite WiFi); inoltre, per tutta la durata degli esperimenti, sarà l'unico dispositivo connesso alla rete.

Con questo setting, sfruttando il servizio `https://fast.com/it/` [1], abbiamo stimato la velocità effettiva della connessione Internet:

- in download: 47.7 Mbps con incertezza di 0.4725 Mbps
- in upload: 18.6 Mbps con incertezza di 0.5513 Mbps

1 Sono state seguite 10 misurazioni, la miglior stima è data dalla media e l'incertezza standard è stimata usando la deviazione standard sperimentale della media.

Si è scelto di utilizzare l'ultima release di Snort 2: Snort 2.9.12. Sebbene fosse già disponibile la Beta di Snort 3.0, si è preferito non avventurarsi in territorio inesplorato; sarebbe comunque interessante ripetere lo studio con questa nuova versione e valutare le differenze (soprattutto alla luce del fatto che la versione 3 supporta threads multipli per processare i pacchetti).

Per installarlo su entrambi i nodi virtuali è stata seguita la guida "Snort 2.9.9.x on Ubuntu 14 and 16 with Barnyard2, PulledPork, and BASE" [12]: tale manuale è tra quelli consigliati sul sito di Snort e, nonostante sia pensato per Ubuntu, è adattabile a Debian senza alcuna modifica. Come richiesto da specifica, Snort dovrà essere utilizzato in modalità sniffer/logger, pertanto sono stati seguiti solamente i passaggi strettamente necessari per utilizzare Snort in tali modalità e nessuna libreria e/o packages in eccesso è stato installato (sono stati trascurati non solo i passi relativi a Barnyard2, PulledPork e BASE, ma anche quelli relativi alla modalità NIDS).

Per lanciare Snort in modalità sniffer verrà usato il seguente comando Bash.

```
sudo snort -dev
```

Snort andrà in esecuzione e stamperà sullo schermo:

- opzione `-v`: gli header IP e TCP/UDP/ICMP dei pacchetti;
- opzione `-d`: i dati a livello applicazione;
- opzione `-e`: gli header a livello di data link.

Per lanciarlo come logger si utilizzerà il seguente comando.

```
sudo snort -dev -l /home/log
```

Aggiungendo l'opzione `-l`, Snort andrà automaticamente in modalità packet logger e salverà ogni pacchetto nella cartella specificata [3].

Diversamente da quanto atteso e da quanto specificato nel manuale di Snort [3], su entrambi i sistemi operativi utilizzati, i pacchetti non verranno salvati di default in ASCII, ma bensì direttamente in formato tcpdump (binario) [2][13]. In questo caso, per salvare in ASCII avremmo dovuto specificare anche l'opzione `-K ascii`; tuttavia, si è preferito mantenere la modalità logger binario dato che questa è più adatta se si deve lavorare con reti ad alta velocità o se si desidera

salvare i pacchetti in un formato più compatto [3].

Dunque, i pacchetti in formato tcpdump verranno salvati in un singolo file binario (e.g. `snort.log.1540808743`) nella directory specificata[3]. Successivamente, sarà possibile processare i pacchetti salvati in una qualunque delle modalità di Snort usando l'opzione `-r` seguita dalla posizione del pacchetto; ad esempio:

```
sudo snort -devr home/log/snort.log.1541241103
```

3.6 STRUMENTO DI MONITORAGGIO

Per osservare a runtime l'utilizzo di CPU ci siamo avvalsi di RCL Monitoring Tool, strumento di monitoraggio (java 8 tool) presentato durante le ore del corso. Tra gli eseguibili disponibili, direttamente scaricabili dal link presente sulle slide di introduzione [14], è stata utilizzata l'ultima versione che, come di consueto, si lancia da terminale con il seguente comando:

```
java -jar RCL_Monitoring_Tool_V3.jar
```

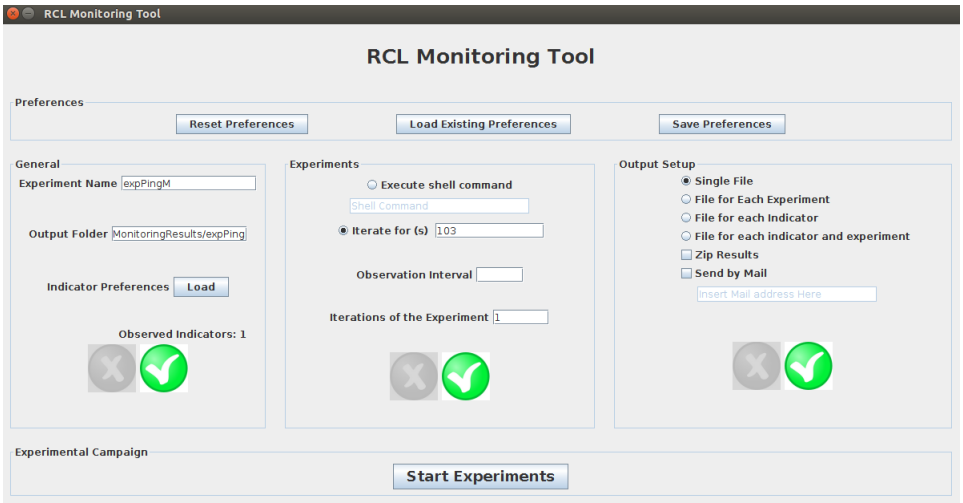

Figura 6.: Interfaccia utente di RCL Monitoring Tool

Lo strumento è utilizzabile attraverso un'interfaccia utente abbastanza intuitiva (Figura 6) ed è in grado di osservare una serie di indicatori elencati nel file di supporto `defaultIndicators.preference`: tra questi, verrà selezionato

2 Nel caso in cui la modalità logger binario non fosse il default, per utilizzarla sarà necessario specificare l'opzione `-b`.

3 Risulta importante evidenziare che, in questo caso, l'intero pacchetto è loggato e risulta superfluo specificare le opzioni `-dev`.

`OperatingSystemImpl.SystemCpuLoad` [4], indicatore a livello di Java Virtual Machine (JVM). La selezione di tale indicatore farà sì che, nel momento in cui faremo partire l'esperimento di monitoraggio, verrà attivata la sonda JVM che, per ogni istante di tempo (ogni secondo), leggerà il dato di interesse: un valore nell'intervallo $[0, 1]$, dove 0 indica che la CPU è inattiva, mentre 1 indica il suo massimo utilizzo. Il tool verrà impostato in modo che, per ogni esperimento, i dati letti vengano scritti in un singolo file `csv`.

4 Il file delle preferenze che conterrà l'indicatore selezionato è `expInd.preferences` ed è contenuto nella cartella `QAS-assignment-E06\RCLMonitoringToolPreferences\Indicators`

4 WORKLOADS E PROCESSI DI MISURA

4.1 SCELTA E DESCRIZIONE DEI WORKLOADS

Sono stati scelti tre diversi carichi di lavoro con cui generare traffico di rete e, di conseguenza, stimolare l'attività di packet sniffing/logging di Snort.

Workload 1 (Ping)**.** Eseguiremo la seguente istruzione sul terminale.

```
sleep 1 && gnome-terminal -e 'sh -c "ping www.google.com -c 100; exec
    bash"'
```

L'istruzione, dopo un secondo di attesa, apre un nuovo terminale ed esegue il comando `ping` (packet internet groper): con tale comando si invia un pacchetto ICMP di tipo echo request ad un target host (nel nostro caso, google.com) e si rimane in attesa di un pacchetto ICMP di tipo echo reply. L'opzione `-c` ci permette di specificare il numero di richieste echo da inviare (aspetta, di default, un secondo prima di inviare la successiva): quindi, invieremo 100 echo request e ci aspetteremo 100 echo reply. Il comando `exec bash` fa sì che, una volta concluso il tutto, il terminale rimanga aperto (in modo tale da aver accesso alle statistiche fornite da `ping`).

Workload 2 (Video)**.** Andiamo a visualizzare un video su youtube eseguendo la seguente istruzione.

```
sleep 1 && gnome-terminal -e 'sh -c "timeout 100 firefox
    http://www.youtube.com/watch?v=tKlRN2YpxRE"'
```

L'istruzione, dopo un secondo dall'avvio, apre, nel browser Firefox, la pagina indicata [1]. Trascorsi 100 secondi, Firefox verrà chiuso.

Workload 3 (Down)**.** Usiamo il comando web get per scaricare (parzialmente) l'immagine ISO di Ubuntu 14 Server.

```
sleep 1 && gnome-terminal -e 'sh -c "timeout 100 wget
    http://releases.ubuntu.com/trusty/ubuntu-14.04.5-server-i386.iso; exec
    bash"'
```

Anche in questo caso il download verrà terminato dopo 100 secondi (il nuovo terminale verrà mantenuto aperto per accedere alle informazioni relative al quantitativo di Byte scaricati).

1 Prima di eseguire il comando sarà necessario impostare le opzioni di riproduzione di youtube affinché il video venga riprodotto sempre con qualità $720p60^{HD}$.

Andiamo a definire la procedura sperimentale che ci consentirà di ottenere le seguenti misure dirette:

- Received (R): numero di pacchetti ricevuti;
- Dropped (D): numero di pacchetti scartati.

Queste sono le grandezze necessarie per poter ricavare la probabilità di packet-loss e, come evidenziato precedentemente, possono essere ottenute direttamente dalle statistiche che Snort ci fornisce a terminazione.

Misureremo tali grandezze per ognuno dei tre workloads, per ogni sistema operativo, prima con Snort in modalità sniffer ed in seguito in modalità logger. Per limitare al minimo l'intervento umano, nel tentativo di ridurre la presenza di errore casuale, ci siamo avvalsi di script Bash che automatizzano l'esecuzione di Snort e dei vari workloads. A titolo d'esempio, riportiamo gli script sviluppati per il workload "Ping" (la struttura è sempre la stessa, varierà solo il workload eseguito):

Script 1 (`expPingSN.sh` [2])**.** Nel caso di Snort in modalità sniffer eseguiremo il seguente script.

```
#!/bin/bash
sleep 1 && gnome-terminal -e 'sh -c "ping www.google.com -c 100; exec
    bash"' &
sudo timeout 102 snort -dev
```

L'utilizzo di `&` dopo la prima istruzione fa sì che il workload venga eseguito in background ed in parallelo rispetto all'istruzione successiva. Dunque, eseguendo lo script, verrà avviato Snort in modalità logger per un tempo pari a 102 secondi e verrà aperto un secondo terminale nel quale, in questo caso, verrà eseguito il comando `ping`. Tale scelta giustifica l'uso di `sleep 1` nel workload: questo consente di avere la certezza che Snort si avvii prima che venga generato qualunque traffico di rete.

Script 2 (`expPingLG.sh`)**.** La struttura dello script è equivalente a quello precedente, con l'unica differenza che Snort viene eseguito come logger.

```
#!/bin/bash
sleep 1 && gnome-terminal -e 'sh -c "ping www.google.com -c 100; exec
    bash"' &
```

2 Ogni script è salvato in un file eseguibile `exp<W><MOD>.sh`, dove `<W>` sarà sostituito dal nome del workload e `<MOD>` sarà una sigla che indica la modalità usata per Snort (`SN` se sniffer e `LG` se logger).

```
sudo timeout 102 snort -dev -l
    /home/osboxes/MonitoringResults/expPing/expPingLG
```

Dunque, descriviamo in modo dettagliato i passaggi della procedura sperimentale seguita.

Procedura Sperimentale 1 (PS1[<W>, <SO>, <MOD>])**.** Passi da seguire per misurare il numero di pacchetti ricevuti e scartati da Snort quando è eseguito in modalità <MOD>, sul sistema operativo <SO>, durante l'esecuzione del workload <W>:

1. aprire un nuovo terminale;
2. spostarsi con `cd` nella cartella contenente lo script `exp<W><MOD>.sh`;
3. per $i = 1 \dots 5$:
   a) eseguire lo script: `./exp<W><MOD>.sh`; attendere la conclusione;
   b) copiare e salvare su file di testo (`exp<W><MOD>i`) le statistiche restituite da Snort; chiudere ogni file e finestra aperti nel processo;
   c) se <W> = `Ping` $\vee$ `Down`: copiare e salvare su file di testo (`exp<W><MOD>i`) le statistiche relative a <W> mostrate sul secondo terminale; chiudere il terminale ed ogni file e finestra aperti nel processo;
4. chiudere il terminale principale.

4.2.1 *Errori casuali e sistematici presenti*

Nella procedura appena descritta è presente errore casuale dovuto al fatto che, nel caso dei workloads Video e Down, non si ha un controllo sul rate con cui vengono generati i pacchetti di rete e sul loro numero. Tuttavia, è possibile aggirare il problema uniformando le informazioni raccolte nell'analisi dati: si può farlo non considerando il numero di pacchetti scartati, ma direttamente la probabilità di scartare un pacchetto (o, alternativamente, la percentuale di pacchetti dropped).

Inoltre, evidenziamo che la procedura potrebbe essere considerata intrusiva. Le performance di Snort (per la precisione, il tempo per processare un singolo pacchetto e quindi il numero di pacchetti scartati) potrebbero essere influenzate dal fatto che il workload venga eseguito sul suo stesso nodo. Tipicamente, Snort è configurato come un'applicazione stand-alone e difficilmente si troverà a condividere cicli di CPU con altre applicazioni [4].

Si discuterà di un possibile approccio che va a risolvere i problemi evidenziati nel capitolo A dell'appendice.

Andiamo a definire la procedura sperimentale che ci consentirà di ottenere le seguenti misure dirette:

- CPU-load[t]: carico di CPU al tempo t (CPU-load istantaneo),
  - per $t \in \{t_0, t_0 + 1, \ldots, t_f\}$;
  - con t_0 = inizio del workload e t_f = conclusione del workload;
  - e CpuLoad[t] $\in [0, 1]$.

Queste sono le grandezze necessarie per poter stimare la %CPU media: percentuale di CPU che, in media, è utilizzata da Snort durante il monitoraggio del traffico.

Come fatto precedentemente, misureremo tali grandezze per ognuno dei tre workloads, per ogni sistema operativo, prima con Snort in modalità sniffer ed in seguito in modalità logger. Anche in questo caso, sfrutteremo degli script per automatizzare l'esecuzione di Snort e del carico di lavoro: nel caso della modalità sniffer, andranno bene quelli precedentemente utilizzati; nel caso della modalità logger, verrà modificata unicamente la cartella in cui salvare i dati di log [3]. Inoltre, sarà necessario utilizzare RCL Monitoring Tool per monitorare l'uso di CPU per tutto il tempo di esecuzione degli script (per l'esattezza, monitorerà il sistema per 103 secondi).

Dunque, descriviamo in modo dettagliato i passaggi della procedura sperimentale seguita.

Procedura Sperimentale 2 (PS2[<W>, <SO>, <MOD>])**.** Passi da seguire per misurare CPU-load istantaeo per ogni istante di tempo (secondo) in cui Snort è eseguito in modalità <MOD>, sul sistema operativo <SO>, durante l'esecuzione del workload <W>:

1. aprire due terminali (saranno indicati con T1 e T2);
2. in T1, spostarsi con `cd` nella cartella contenente lo script `exp<W><MOD>.sh`;
3. in T2, spostarsi con `cd` nella cartella contenente l'eseguibile di RCL Monitoring Tool (RCLMT) e lanciarlo;
4. sull'interfaccia utente di RCLMT, caricare il file di preferenze precedentemente preparato: `exp<W>.preferences` (monitorerà il sistema per 103 secondi);

3 Per distinguerli dagli originali, i nuovi script per la modalità logger sono salvati in un file eseguibile `exp<W>MLG.sh`, dove <W> sarà sostituito dal nome del dato workload.

5. per $i = 1 \ldots 5$:
    a) nell'interfaccia di RCLMT, nominare l'esperimento `exp<W>M<MOD>i`;
    b) in T2, preparare il comando per lanciare lo script, ma non eseguirlo ancora (se `<MOD>` = `LG`, `./exp<W>MLG.sh`; se `<MOD>` = `SN`, `./exp<W>SN.sh`);
    c) nell'interfaccia di RCLMT, avviare l'esperimento premendo su "Start Experiments" e, prontamente, eseguire lo script selezionato premendo invio sul secondo terminale (errore casuale); attendere la conclusione dell'esperimento di monitoraggio;
    d) copiare e salvare sul file di testo (`stat<W>M<MOD>i`) le statistiche restituite da Snort; chiudere ogni file e finestra aperti nel processo;
    e) se `<W>` = `Ping` $\vee$ `Down`: copiare e salvare su file di testo (`stat<W>M<MOD>i` nel primo caso e `<S0>DownStat` nel secondo) le statistiche relative al workload `<W>` mostrate sul terzo terminale; chiudere il terminale ed ogni file e finestra aperti nel processo;
6. chiudere RCLMT ed i terminali ancora aperti.

4.3.1 *Errori casuali e sistematici presenti*

Abbiamo presenza di errore casuale dovuto al fatto che, al passo 5.c, sarà l'operatore umano, subito dopo aver avviato il tool di monitoraggio, a dover eseguire lo script. Questa situazione poteva essere evitata settando RCL Monitoring Tool affinché, automaticamente, lanciasse lo script e monitorasse per il tempo di esecuzione. Teoricamente, questo poteva essere fatto utilizzando l'opzione "Execute shell command"; tuttavia, nel nostro caso, ogni tentativo di sfruttare questa possibilità è risultato vano: l'esperimento di monitoraggio terminava immediatamente (si suppone che il problema sia dovuto al fatto che lo script mandi la prima istruzione in background ed in parallelo rispetto alla seconda).

Risulta importante sottolineare che ci interessa unicamente l'uso di CPU durante il monitoraggio del traffico di rete. Possiamo, quindi, non considerare la presenza di questo errore casuale dato che, nel momento in cui andremo ad analizzare i dati raccolti, trascureremo le osservazioni relative ai primi ed agli ultimi secondi dell'esperimento: in tal modo, saremo sicuri di non considerare l'apporto al carico di CPU dovuto all'avvio ed alla terminazione di Snort.

Inoltre, osserviamo che il sistema di monitoraggio della CPU è intrusivo: viene eseguito sulle spalle del sistema e contribuisce ad aumentare il valore dell'indicatore CPU-load. Stesso discorso vale per il workload. Abbiamo, dunque, un errore sistematico e, per poter stimare la percentuale media di CPU usata da Snort, dobbiamo prima stimare quella usata dal sistema quando vengono eseguiti

unicamente il workload e RCL Monitoring Tool. A tale scopo, ci avvarremo della seguente procedura sperimentale.

Procedura Sperimentale 3 (PS3[<W>, <SO>])**.** Passi da seguire per misurare CPU-load istantaeo per ogni istante di tempo (secondo) in cui è eseguito il workload <W>, sul sistema operativo <SO>:

1. aprire due terminali (saranno indicati con T1 e T2);
2. in T1, spostarsi con `cd` nella cartella contenente lo script `exp<W>M.sh` [4];
3. in T2, spostarsi con `cd` nella cartella contenete l'eseguibile di RCL Monitoring Tool (RCLMT) e lanciarlo;
4. sull'interfaccia utente di RCLMT, caricare il file di preferenze precedentemente preparato: `exp<W>.preferences` (monitorerà il sistema per 103 secondi);
5. per $i = 1 \dots 5$:
   a) nell'interfaccia di RCLMT, nominare l'esperimento `exp<W>Mi`;
   b) in T2, preparare il comando per lanciare lo script (`./exp<W>M.sh`), ma non eseguirlo ancora;
   c) avviare l'esperimento premendo su "Start Experiments" nell'interfaccia di RCLMT e, prontamente, eseguire lo script selezionato premendo invio sul secondo terminale (errore casuale); attendere la conclusione dell'esperimento di monitoraggio;
   d) se `<W>` = `Down`: copiare e salvare su file di testo (`<S0>DownStat`) le statistiche relative al download mostrate sul terzo terminale; chiudere il terminale ed ogni file e finestra aperti nel processo;
   e) se `<W>` = `Ping`: chiudere il terzo terminale aperto;
6. chiudere RCLMT ed i terminali ancora aperti.

4.4 STRUTTURA DELLA CAMPAGNA SPERIMENTALE

Le procedure sperimentali sono state eseguite seguendo questo schema:

- Per ogni workload <W>:
  - per ogni sistema operativo <SO>:
    + procedura sperimentale 3: PS3[<W>, <SO>];
    - per ogni modalità di Snort <MOD>:
      + procedura sperimentale 1: PS1[<W>, <SO>, <MOD>];
      + procedura sperimentale 2: PS2[<W>, <SO>, <MOD>].

4 Tale script conterrà unicamente l'istruzione per l'esecuzione del dato workload.

4.5 misurazioni e condizioni di ripetibilitá

Per ottenere misure precise ci siamo posti in condizioni di ripetibilità:

- sono state sempre usate le procedure sperimentali definite nella sezione precedente;
- tali procedure sono state eseguite dallo stesso operatore;
- il setting sperimentale, descritto nel capitolo 3, è stato mantenuto costante; in particolare, come evidenziato nella sezione 3.4, la macchina host, unico dispositivo connesso alla rete, è stata posizionata sempre alla stessa distanza dal router. Inoltre, ci siamo assicurati che nessun processo non strettamente necessario, oltre quelli di sistema [5], fosse in esecuzione prima dell'avvio di ogni esperimento di monitoraggio (e durante).
- le misurazioni associate ad un dato workload sono state eseguite a breve distanza di tempo le une dalle altre (tutte nella stessa mattina o pomeriggio), in modo da garantire le stesse condizioni operative; per la precisione:
  - Ping: mattina del 13 Novembre 2018,
  - Down: mattina del 18 Novembre 2018,
  - Video: pomeriggio del 20 Novembre 2018 (per CPU-load), mattina del 21 Novembre 2018 (per statistiche di Snort);

5 Per minimizzare ulteriormente l'impatto che le altre attività del sistema potrebbero avere sulle performance di Snort si potrebbe disabilitare ogni servizio in esecuzione in background che non è strettamente necessario eccetto Snort.

5 ANALISI DEI RISULTATI DI MISURA

5.1 STIMARE LA PROBABILITÁ DI PACKET LOSS

Una volta conclusa la campagna sperimentale avremo, per ognuno dei tre workloads, per ogni sistema operativo e per ognuna delle due modalità di Snort analizzate, le seguenti misure dirette:

- R_i: numero totale di pacchetti ricevuti nell'esperimento i;
- D_i: numero totale di pacchetti scartati nell'esperimento i;

con $i = 1 \dots 5$, dato che per ogni caso sono stati eseguiti cinque esperimenti.

Possiamo ricavare in maniera immediata la probabilità di packet-loss relativa all'esperimento i-esimo con:

$$P(\text{packet-loss}_i) = \frac{D_i}{R_i}. \tag{5.1}$$

In questo caso, la miglior stima della probabilità di packet-loss è data dalla media aritmetica delle cinque osservazioni:

$$\overline{P(\text{packet-loss})} = \frac{1}{5} \cdot \sum_{i=1}^{5} P(\text{packet-loss}_i). \tag{5.2}$$

Infine, l'incertezza standard è stata stimata utilizzando la deviazione standard sperimentale della media:

$$u = \sqrt{\frac{s^2}{5}} \tag{5.3}$$

con s^2 che indica la varianza sperimentale delle cinque osservazioni:

$$s^2 = \frac{1}{4} \cdot \sum_{i=1}^{5} \left(P(\text{packet-loss}_i) - \overline{P(\text{packet-loss})}\right)^2 \tag{5.4}$$

Per ogni sistema operativo e per ogni workload (W), abbiamo osservato il sistema tramite RCL Monitoring tool (M) nei seguenti casi:

- mentre viene eseguito il dato workload (W): caso WM;
- mentre viene eseguito il dato workload (W) e Snort come sniffer (SN): caso WMSN;
- mentre viene eseguito il dato workload (W) e Snort come logger (LG): caso WMLG.

Dunque, una volta conclusa la campagna sperimentale, per ognuno dei casi possibili, avremo le seguenti misure dirette:

- $\text{CPU-load}_i[t]$

con $i = 1 \dots 5$, dato che per ogni caso sono stati eseguiti 5 esperimenti e con $t = 1 \dots 103$, dato che il sistema è stato monitorato per 103 secondi.

A partire dalla CPU-load istantanea possiamo ricavare in maniera diretta la percentuale di CPU usata dal sistema al dato istante:

$$\%\text{CPU}_i[t] = 100 \cdot \text{CPU-load}_i[t]. \tag{5.5}$$

Inoltre, dato che ci interessa unicamente l'uso di CPU durante il monitoraggio del traffico di rete, trascureremo le osservazioni relative ai primi ed agli ultimi secondi dell'esperimento: in tal modo, saremo sicuri di non considerare l'apporto al carico di CPU dovuto all'avvio ed alla terminazione di Snort (prendiamo in esame unicamente il suo comportamento a regime). Per la precisione, consideriamo le osservazioni a partire dall'istante $t_0 = 11$ fino all'istante $t_f = 100$.

La %CPU media usata dal sistema durante l'esperimento i-esimo può essere calcolata come:

$$\%\text{CPU}_i[11, 100] = 100 \cdot \text{CPU-load}_i[11, 100]^1 \tag{5.6}$$

con $\text{CPU-load}_i[11, 100]$ che indica il CPU-load medio durante l'intervallo di monitoraggio di rete. Tale valore viene stimato facendo la media aritmetica delle novanta osservazioni selezionate:

$$\text{CPU-load}_i[11, 100] \simeq \frac{1}{90} \cdot \sum_{k=11}^{100} \text{CPU-load}_i[k]. \tag{5.7}$$

1 Per semplicità, da questo momento in poi, considereremo unicamente la CPU-load, dato che passare alla %CPU risulta immediato.

La miglior stima della CPU-load media, per ogni singolo caso, è data dalla media aritmetica delle CPU-load medie dei cinque esperimenti:

$$\overline{\text{CPU-load}[11,100]} = \frac{1}{5} \cdot \sum_{i=1}^{5} \text{CPU-load}_i[11,100]. \tag{5.8}$$

Risulta importante sottolineare che stimare la CPU-load media in tal modo equivale a stimarla facendo un'unica media di tutte le misurazioni riguardo alla CPU-load istantanea nel dato intervallo di interesse:

$$\overline{\text{CPU-load}[11,100]} = \frac{1}{450} \cdot \sum_{i=1}^{5} (\sum_{k=11}^{100} \text{CPU-load}_i[k]). \tag{5.9}$$

Alla luce di ciò, possiamo andare a stimare l'incertezza standard con la deviazione standard sperimentale della media relativa a tutte le osservazioni di interesse per il dato caso:

$$u = \sqrt{\frac{s^2}{450}} \tag{5.10}$$

con s^2 che indica la varianza sperimentale delle 450 osservazioni:

$$s^2 = \frac{1}{449} \cdot \sum_{i=1}^{5} (\sum_{k=11}^{100} (\text{CPU-load}_i[k] - \overline{\text{CPU-load}[11,100]})^2). \tag{5.11}$$

Quindi, per ogni SO e per ogni workload (W), abbiamo tre stime della CPU-load media usata dal sistema (corredate dalle rispettive incertezze) relative ai casi esplicitati poco fa:

- $\overline{\text{CPU-load}_{WM}}$ con incertezza u_{WM};
- $\overline{\text{CPU-load}_{WMSN}}$ con incertezza u_{WMSN};
- $\overline{\text{CPU-load}_{WMLG}}$ con incertezza u_{WMLG}.

Per concludere, possiamo stimare l'apporto al CPU-load medio dovuto a Snort per sottrazione (l'incertezza associata è ottenuta mediante la formula dell'incertezza standard combinata):

- nel caso di Snort come sniffer:

$$\overline{\text{CPU-load}_{SN}} = \overline{\text{CPU-load}_{WMSN}} - \overline{\text{CPU-load}_{WM}}; \tag{5.12}$$

$$u_{SN} = \sqrt{(u_{WM})^2 + (u_{WMSN})^2} \tag{5.13}$$

- nel caso di Snort come logger:

$$\overline{\text{CPU-load}_{LG}} = \overline{\text{CPU-load}_{WMLG}} - \overline{\text{CPU-load}_{WM}}. \tag{5.14}$$

$$u_{LG} = \sqrt{(u_{WM})^2 + (u_{WMLG})^2} \tag{5.15}$$

5.3 ANALISI DELLE STIME OTTENUTE

Andiamo ad analizzare i risultati ottenuti per ogni workload.

5.3.1 *Analisi delle stime ottenute: Ping*

Ping e Snort come sniffer

	Ping					
	Sniffer					
	Ubuntu			Debian		
expNumber	received	dropped	**P(packet-loss)**	received	dropped	**P(packet-loss)**
1	204	0	0	204	0	0
2	204	0	0	204	0	0
3	204	0	0	204	0	0
4	204	0	0	204	0	0
5	204	0	0	204	0	0
MEDIA			**0**			**0**
INCERTEZZA			0			0

Figura 7.: Probabilità di packet-loss per Snort in modalità sniffer con workload Ping.

Come possiamo vedere dalla figura 7, nel caso dell'esecuzione del workload Ping, quando Snort viene lanciato in modalità sniffer nessuno dei pacchetti catturati viene mai scartato.

			CPU-load media	Incertezza
Ping	Ubuntu	WMSN	0,1368420488	0,0012694011
		SN	**0,0447313026**	0,0019676837
	Debian	WMSN	0,2080276487	0,0022029206
		SN	**0,0366917238**	0,0027252396

Figura 8.: CPU-load medio per Snort in modalità sniffer con workload Ping.

Osserviamo in figura 8 il CPU-load medio del sistema durante il monitoraggio di rete: Debian, in media, usa all'incirca il 21% delle risorse della CPU, mentre Ubuntu solamente circa il 14%; tuttavia, quando andiamo a considerare l'apporto al CPU-load dovuto unicamente a Snort vediamo che nel caso di Debian questo è lievemente inferiore.

Ping e Snort come logger

Situazione pressoché identica a quella precedente la abbiamo nel caso di Snort come logger.

	Ping					
	Logger					
	Ubuntu			Debian		
expNumber	received	dropped	**P(packet-loss)**	received	dropped	**P(packet-loss)**
1	204	0	0	204	0	0
2	204	0	0	204	0	0
3	204	0	0	204	0	0
4	204	0	0	204	0	0
5	204	0	0	204	0	0
MEDIA			**0**			**0**
INCERTEZZA			0			0

Figura 9.: Probabilità di packet-loss per Snort in modalità logger con workload Ping.

			CPU-load media	Incertezza
Ping	Ubuntu	WMLG	0,1322485965	0,0015019669
		LG	**0,0401378503**	0,0021251599
	Debian	WMLG	0,2144109703	0,0022441892
		LG	**0,0430750454**	0,0027587056

Figura 10.: CPU-load medio per Snort in modalità logger con workload Ping.

Ping: conclusioni

Nel caso del workload Ping, il rate del traffico generato è moderato: il sistema riesce sempre a processare ogni pacchetto e ad eseguire le operazioni di I/O richieste prima che il buffer di libpcap si riempia.

5.3.2 *Analisi delle stime ottenute: Video*

Risultati più interessanti sono ottenuti nel caso del workload Video.

Video e Snort come sniffer

	Video					
	Sniffer					
	Ubuntu			Debian		
expNumber	received	dropped	**P(packet-loss)**	received	dropped	**P(packet-loss)**
1	25441	19218	0.7553948351	26641	16439	0.6170564168
2	25467	19621	0.7704480308	25807	15770	0.6110745147
3	25413	19514	0.7678747098	26500	17104	0.6454339623
4	25256	19158	0.7585524232	26162	16416	0.6274749637
5	25217	19078	0.7565531189	25281	15649	0.6190024129
MEDIA			**0.7617646236**			**0.6240084541**
INCERTEZZA			0.0030885833			0.005965514

Figura 11.: Probabilità di packet-loss per Snort in modalità sniffer con workload Video.

Con Snort in modalità sniffer, per entrambi i sistemi operativi abbiamo un elevato numero di pacchetti scartati anche se, rispetto ad Ubuntu, su Debian abbiamo una probabilità di packet-loss inferiore.

			CPU-load media	Incertezza
Video	Ubuntu	WMSN	0,7800265996	0,008789875
		SN	**0,0412201391**	0,0107156138
	Debian	MSN	0,924527001	0,0028937953
		SN	**0,0476814633**	0,0048525845

Figura 12.: CPU-load medio per Snort in modalità sniffer con workload Video.

Per quanto riguarda il CPU-load medio, durante il monitoraggio di rete osserviamo che, in media, Debian ha un utilizzo di CPU molto più elevato rispetto ad Ubuntu ($\sim$ 92% contro $\sim$ 78%): tuttavia, si stima che l'apporto al CPU-load dovuto a Snort sia più o meno equivalente per entrambi i SO.

Video e Snort come logger

	Video					
	Logger					
	Ubuntu			Debian		
expNumber	received	dropped	**P(packet-loss)**	received	dropped	**P(packet-loss)**
1	25298	0	0	25825	0	0
2	25833	0	0	26883	0	0
3	25166	0	0	26028	0	0
4	25095	0	0	25359	0	0
5	25036	0	0	25431	0	0
MEDIA			**0**			**0**
INCERTEZZA			0			0

Figura 13.: Probabilità di packet-loss per Snort in modalità logger con workload Video.

Risulta interessante osservare come, invece, quando Snort è eseguito in modalità logger, riesca ad analizzare tutti i pacchetti.

			CPU-load media	Incertezza
Video	Ubuntu	WMLG	0,7814642418	0,0056665
		LG	**0,0426577813**	0,0083469574
	Debian	WMLG	0,9072775275	0,0030481647
		LG	**0,0304319898**	0,0049461938

Figura 14.: CPU-load medio per Snort in modalità logger con workload Video.

Osserviamo che le stime relative alla CPU-load media sono in linea con quanto visto nel caso della modalità sniffer (anche se l'apporto dovuto a Snort risulta lievemente inferiore con Debian).

Video: conclusioni

Il workload Video genera una situazione di traffico intenso e Snort in modalità sniffer non riesce a "tenere il passo": da quanto osservato possiamo concludere che l'elevato numero di pacchetti scartati non sia legato alla scarsa disponibilità e/o velocità della CPU, ma, bensì, ai lunghi tempi di attesa per il completamento delle operazioni di scrittura sul terminale. Dunque, in questo caso, Snort si comporta come un processo I/O-bound.

Probabilmente, stampando un minor numero di informazioni il problema dell'elevata packet-loss si attenuerebbe: sarebbe interessante provare a mantenere, ad esempio, solo l'opzione `-v` stampando unicamente gli header IP e TCP/UDP/ICMP dei pacchetti.

Invece, in modalità logger nessun pacchetto viene scartato: le operazioni di scrittura del file di log binario vengono eseguite più velocemente rispetto alla stampa delle informazioni sul terminale. In questo caso sarebbe interessante andare a ripetere gli esperimenti impostando il logging dei pacchetti in ASCII, dato che ci aspettiamo dei tempi più lunghi in scrittura.

5.3.3 *Analisi delle stime ottenute: Down*

Il workload Down genera una situazione di traffico molto intenso: andiamo a valutare come ciò abbia impattato sul funzionamento di Snort.

Down e Snort come sniffer

	Down					
	Sniffer					
	Ubuntu			Debian		
expNumber	received	dropped	**P(packet-loss)**	received	dropped	**P(packet-loss)**
1	293791	273621	0.9313457526	283788	247183	0.8710128688
2	290101	269527	0.9290798722	283256	248242	0.8763874375
3	292044	272653	0.9336024709	285720	245821	0.8603562929
4	293431	273063	0.9305867478	307247	265557	0.8643111243
5	287924	267846	0.9302663203	323532	280444	0.8668199745
MEDIA			**0.9309762327**			**0.8677775396**
INCERTEZZA			0.0007511834			0.0027622958

Figura 15.: Probabilità di packet-loss per Snort in modalità sniffer con workload Down.

Come visto anche precedentemente, la modalità sniffer non riesce a "tenere il passo": su Debian si ha una probabilità di packet-loss di ~ 0.87 e su Ubuntu addirittura di ~ 0.93.

			CPU-load media	Incertezza
Down	Ubuntu	WMSN	0,5132833831	0,0015742814
		SN	**0,0734137419**	0,0035874773
	Debian	WMSN	0,6517564527	0,0019853355
		SN	**0,1529619345**	0,0030872149

Figura 16.: CPU-load medio per Snort in modalità sniffer con workload Down.

Per quanto riguarda il CPU-load, otteniamo dei valori più moderati rispetto al workload Video, ma, anche in questo caso, il CPU-load medio di Debian è maggiore rispetto a quello di Ubuntu.

Down e Snort come logger

	Down					
	Logger					
	Ubuntu			Debian		
expNumber	received	dropped	**P(packet-loss)**	received	dropped	**P(packet-loss)**
1	282294	0	0	302113	0	0
2	279758	0	0	282901	654	0.0023117628
3	282303	0	0	301306	0	0
4	280775	0	0	277679	1781	0.0064138808
5	280746	0	0	279365	0	0
MEDIA			**0**			**0.0017451287**
INCERTEZZA			0			0.0012500948

Figura 17.: Probabilità di packet-loss per Snort in modalità logger con workload Down.

In modalità logger, praticamente nessun pacchetto è scartato (su Debian abbiamo qualche dropped, ma i numeri sono così poco elevati che sono trascurabili dal punto di vista statistico).

			CPU-load media	Incertezza
Down	Ubuntu	WMLG	0,4946943718	0,0034511152
		LG	**0,0548247306**	0,0047224811
	Debian	WMLG	0,5988167521	0,0024128179
		LG	**0,1000222339**	0,0033780215

Figura 18.: CPU-load medio per Snort in modalità logger con workload Down.

Le osservazioni riguardo alla CPU-load media sono in linea con quelle del caso sniffer (anche se otteniamo dei valori lievemente inferiori).

Down: conclusioni

Valgono le stesse considerazioni che sono state fatte per il workload Video: anche in questo caso Snort si comporta come un processo I/O-bound.

5.4 CONCLUSIONI

Abbiamo osservato che, in situazioni di traffico moderato (Ping), su entrambi i sistemi operativi e qualunque sia la modalità selezionata per Snort, questo riesce sempre a processare tutti i pacchetti.

Al contrario, in presenza di traffico intenso (Video e Down) solamente la modalità logger riesce a "stare dietro" ai pacchetti:

- su entrambi i SO, quando Snort è eseguito come sniffer, si osserva un'elevata probabilità di packet-loss;
- in tal caso, la probabilità di packet-loss rilevata su Debian è tendenzialmente inferiore rispetto a quella su Ubuntu.

Si è concluso che quanto osservato è legato al fatto che, nei casi analizzati, Snort si comporti come un processo I/O-bound: il tempo per processare un pacchetto è determinato principalmente dal tempo atteso per il completamento delle operazioni di I/O. Sarebbe, dunque, interessante ripetere gli esperimenti andando a monitorare anche il numero di scritture su disco effettuate (fattibile attivando l'indicatore `Disk Write` di RCL Monitoring Tool). Inoltre, si potrebbe anche valutare la performance di Snort come logger ASCII dato che dovrebbe comportare tempi più lunghi per le operazioni di scrittura.

Abbiamo osservato che, in quasi tutti i casi, il CPU-load medio su Debian risulta superiore rispetto a quello su Ubuntu (anche se la stima dell'apporto al CPU-load medio dovuto a Snort è molto piccola e, a livello qualitativo, non molto dipendente dalla variazione di SO).

Possiamo concludere che, limitatamente ai casi trattati ed alle misure di interesse selezionate, tra Ubuntu 14.04.05 e Debian 8.11, il sistema operativo che, a parità di risorse assegnate e carico di lavoro, garantisce che Snort abbia una migliore performance (cioè una minor probabilità di packet-loss) è, anche se di poco, Debian 8.11 [2].

2 Per avere una visione più completa sarebbe necessario considerare ulteriori workloads che ci pongono in situazioni di traffico ancora più intenso e, quindi, in grado di mettere in difficoltà anche il logger binario.

A

UN POSSIBILE MIGLIORAMENTO

Abbiamo individuato alcuni aspetti del lavoro svolto che potrebbero essere migliorati:

- la performance di Snort è influenzata dal fatto che il workload venga eseguito sul suo stesso nodo: normalmente, Snort è configurato come un'applicazione stand-alone e difficilmente si troverà a condividere cicli di CPU con altre applicazioni;
- i workloads selezionati non ci consentono di avere un controllo sul rate con cui vengono generati i pacchetti di rete.

Una possibile soluzione ad entrambi i problemi potrebbe essere ottenuta seguendo l'approccio proposto nell'articolo "Performance evaluation comparison of Snort NIDS under Linux and Windows Server" [4]:

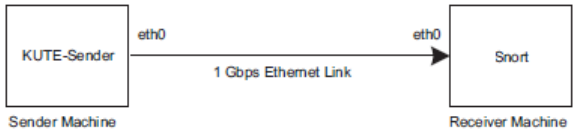

Figura 19.: Possibile miglioramento

- per valutare la performance di Snort vengono utilizzate due macchine, un sender ed un receiver, connesse tramite un cavo Ethernet da 1Gbps;
- sul nodo ricevente sarà installato Snort;
- la macchina sender invierà pacchetti con un dato rate usando KUTE 1.4, un software open-source per la generazione di traffico che non richiede l'installazione di componenti di ricezione aggiuntivi sul ricevente (è non intrusivo).

Questo approccio garantisce un controllo accurato sull'intensità del traffico generato e una minore intrusività.

BIBLIOGRAFIA

[1] Walter Lewin, Lecture "For the Love of Physics", https://www.youtube.com/watch?v=sJG-rXBbmCc, 2011 (Cited on page ii.)

[2] Wikipedia (it), "Snort", https://it.wikipedia.org/wiki/Snort, (Cited on page 3.)

[3] The Snort Project, "SNORT Users Manual 2.9.12", https://www.snort.org/, 2018 (Cited on pages 3, 10, and 11.)

[4] K. Salah, A. Kahtani, "Performance evaluation comparison of Snort NIDS under Linux and Windows Server", *Journal of Network and Computer Applications 33 6–15*, 2010 (Cited on pages 4, 5, 15, and 31.)

[5] Wikipedia (en), "CPU-bound", https://en.wikipedia.org/wiki/I/O_bound (Cited on page 5.)

[6] Wikipedia (en), "I/O bound", https://en.wikipedia.org/wiki/CPU-bound (Cited on page 5.)

[7] Joel Esler, "Snort Supported OSes", https://www.snort.org/documents/snort-supported-oses (Cited on page 7.)

[8] Wikipedia (en), "Debian", https://en.wikipedia.org/wiki/Debian (Cited on page 7.)

[9] Wikipedia (en), "Ubuntu", https://en.wikipedia.org/wiki/Ubuntu (Cited on page 7.)

[10] Wikipedia(en),"VirtualBox",https://it.wikipedia.org/wiki/VirtualBox (Cited on page 7.)

[11] Oracle Corporation, "Oracle VM VirtualBox User Manual Version 5.2.16", http://www.virtualbox.org, 2018 (Cited on page 8.)

[12] Noah Dietrich, "Snort 2.9.9.x on Ubuntu 14 and 16 with Barnyard2, PulledPork, and BASE", http://sublimerobots.com/, 2017 (Cited on page 10.)

[13] William Zereneh, "Snort IDS", *slide per il corso CN8822, Computer Science, Ryerson University*, http://www.scs.ryerson.ca/~zereneh/cn8822/ (Cited on page 10.)

[14] Tommaso Zoppi, "RCL Monitoring Tool", *slide per il corso di "Analisi Quantitativa dei Sistemi 2017-2018", Unifi*, 2017 (Cited on page 11.)