UNIVERSITÀ DI PISA

Traffic Flow
Gestione di Reti A.A. 2019/20

Sacco Giuseppe

28 agosto 2020

Indice

1 Introduzione

L'applicazione sviluppata per questo progetto ha l'obiettivo di monitorare i flussi di rete.
L'obiettivo è di mostrare la quantità di pacchetti e di bytes inviati e ricevuti, suddivisi in base al protocollo utilizzato e di identificare, attraverso la porta sorgente/destinazione del pacchetto, qual è il processo che ha inviato il pacchetto o che lo ha ricevuto.

Il progetto è stato realizzato in Python3 con l'ausilio della libreria Scapy che permette la manipolazione di pacchetti (creazione e modifica) e non solo; è usata anche per fare lo scann della rete, test, tracerouting, attacchi e network discovery.

1.1 Descrizione

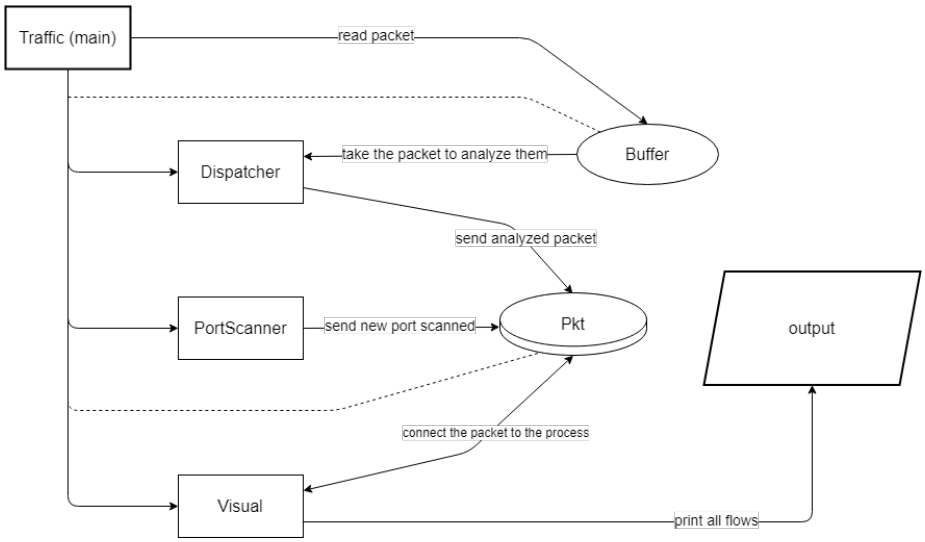
La funzione principale è quella di mostrare, dal momento dell'avvio, tutti i flussi in entrata e uscita facendo anche vedere quale processo ha inviato/ricevuto questi flussi. Nella realizzazione dell'applicazione sono state fatte alcune precisazioni sui pacchetti catturati:

- vengono a far parte dei flussi solo quei pacchetti che hanno un protocollo TCP/UDP dato che dominano il traffico di rete;
- durante la scansione dei pacchetti potremmo ricevere anche pacchetti di broadcast o di altri protocolli che non ci interessano al fine del progetto, per cui questi pacchetti verranno definiti come "*unknown*";
- inoltre potrebbe verificarsi che (al momento dell'interruzione del programma) alcuni pacchetti non siano ancora stati catalogati in base al processo che li ha inviati/ricevuti, e vengono definiti come "*unknown process*".

Dunque il conteggio totale dei pacchetti non riguarda solo quelli analizzati appartenenti ai flussi, ma anche quelli "unknown" e "unknown process".
L'ultima precisazione da fare è che l'applicazione è pensata per un uso in un arco di tempo limitato. Dunque, l'idea è di far vedere tutti i flussi catturati mostrandoli anche se le porte o i processi sono ormai chiusi.

2 Funzionamento

Per comprendere il funzionamento e come interagiscono tra di loro i vari pezzi di codice faremo uso di uno schema.



Tutto parte da Traffic che oltre ad effettuare lo sniff dei pacchetti, crea i threads: Dispatcher, PortScanner, Visual che faranno utilizzo delle due strutture dati inizializzate da Traffic per comunicare tra di loro: un Buffer e varie liste di python per catalogare i pacchetti, contenute nella classe Pkt.

2.1 Strutture dati

Nella realizzazione dell'applicazione sono state sviluppate due classi per gestire i dati: Buffer, Pkt.

2.1.1 Buffer

Questa classe mette a disposizione una lista per far comunicare il thread main che sniffa i pacchetti e il thread Dispatcher che li preleva per fare una pre-analisi.
Mette a disposizione una lock "*_mutex*" per l'accesso in mutua esclusione al buffer, e due funzioni per aggiungere in coda alla lista e per prelevare l'elemento dalla coda.

2.1.2 Pkt

Questa classe, invece, serve per mettere in comunicazione tra loro i threads Dispatcher, PortScanner e Visual.

Per il thread Dispatcher mette a disposizione l'unica funzione implementata: *insertPkt* che si occupa di inserire il pacchetto o nella lista *unanalizedPkt* oppure nella lista *unknown*; nella prima vengono messi quei pacchetti che rispettano tutte le condizioni per essere collegati ad un processo attivo su una porta, nella seconda quelli di cui non ci interessa il relativo protocollo.
L'accesso alla lista unanalizedPkt è gestito in mutua esclusione tramite la lock "*_lockPkts*".
Inoltre vi è una lista *connection* dove vengono inseriti tutti i processi attivi che saranno scoperti dal thread PortScanner e che successivamente il thread Visual userà per creare i collegamenti con i pacchetti da analizzare. Poi nella lista *pktsInformations* vengono salvati tutti i flussi che saranno riconosciuti tramite il thread Visual che si occupa di gestire i nuovi flussi da aggiungere e di aggiornare le statistiche dei vecchi.
Infine, viene messo a disposizione un dizionario definito con *chiave:valore* dove verranno inserite le informazioni relative a un pacchetto e successivamente tornerà utile per descrivere i flussi usando i campi *packets* e *bytes*.
Struttura del pktStyle iniziale:
pktStyle={"Name": None, "PID": None, "protocol": None, "bytes": None,"packets": None,"IP src": None,"port src": None,"IP dst": None,"port dst": None, "in/out": None}

2.2 Codice

L'applicazione è composta quindi da quattro threads che cooperano tra loro:

- Traffic (main)
- Dispatcher
- PortScanner
- Visual

2.2.1 Traffic

Come prima cosa, viene chiesto all'utente su quale interfaccia di rete si vuole applicare la scansione. Come anticipato, il thread main (Traffic) crea gli altri tre threads: Dispatcher, PortScanner, Visual. In seguito utilizza la funzione sniff di Scapy per catturare i pacchetti e attraverso uno dei parametri *prn* è possibile definire una funzione che gestisce il singolo pacchetto appena viene catturato. La funzione creata per gestire i pacchetti si chiama *traffic_handler*, la quale acquisisce la variabile di mutua esclusione di accesso al buffer e inserisce in coda il pacchetto.
Alla ricezione di un SIGINT (Ctrl+C) viene automaticamente bloccata la cattura dei pacchetti, dunque viene interrotto il lavoro della funzione sniff.
A questo punto il main dovrà notificare a tutti gli altri thread un segnale di arresto settando la variabile chiamata *term* (di tipo threading.Event()) a tutti gli altri threads. Successivamente attenderà la loro terminazione e concluderà

il suo funzionamento stampando le informazioni relative ai pacchetti catturati, nello specifico: il numero di pacchetti che sono "unknown", "unknown process" e il numero totale di pacchetti catturati.

2.2.2 Dispatcher

Nel momento in cui il thread main inserisce pacchetti nel buffer, entra in gioco il thread Dispatcher.
Inizialmente, accede in maniera concorrente al buffer prendendo, uno alla volta, i pacchetti in testa alla lista. Successivamente ricava dalla seconda struttura dati, Pkt, un esempio di informazioni che si vogliono salvare da un pacchetto catturato (*pktStyle*).
Da qui in poi parte una pre-analisi del pacchetto.
Verranno salvati i dati relativi a:

- protocollo utilizzato;
- grandezza del pacchetto;
- se il pacchetto è stato inviato/ricevuto;
- indirizzo IP sorgente e di destinzazione;
- porta sorgente e di destinazione

Da precisare che, durante l'analisi del protocollo usato si fa la differenza tra TCP e UDP tramite la funzione *haslayer* di Scapy e se nessuno di questi due layer è stato trovato, allora il campo protocollo sarà segnato come "unknown".
Infine viene chiamata la funzione *insertPkt* della classe Pkt che si occuperà di inserire nella coda corretta le informazioni del pacchetto. Il ciclo di analisi ricomincia fin quando non verrà ricevuto il segnale di terminazione dal thread main.

2.2.3 PortScanner

Il lavoro svolto dal thread scanner è semplice ma fondamentale. Raccoglie in una lista tutti i processi attivi che sono in ascolto su una porta per ricevere o inviare pacchetti. L'utilizzo di questo risultato è di far vedere il nome del processo e il suo PID nel momento in cui si andrà a far vedere il flusso di pacchetti in entrata o in uscita da quel processo. Ad ogni iterazione modifica la lista delle connessioni situata nella struttra Pkt.
Il thread svolge periodicamente il test sui processi attivi (ogni mezzo secondo), per trovare nuovi processi e collegarli ai pacchetti catturati in rete non ancora messi in relazione al processo che li invia/riceve.
Termina quando verrà ricevuto il segnale di terminazione dal thread main.

2.2.4 Visual

Il thread Visual è responsabile nel collegare i pacchetti al loro processo e di creare l'output dei flussi.
Ad ogni iterazione, recupera tutta la lista dei pacchetti non analizzati (popolata dal thread Dispatcher) facendone una copia; poi recupera la lista dei processi attualmente attivi.
Quindi, per ogni pacchetto, controlla se nella lista *pktsInformations* esiste già un flusso con quel processo collegato a quella porta. Se è così viene aumentato il conteggio dei pacchetti e di bytes catturati; se un flusso non esiste allora ne viene creato uno nuovo. Al termine dell'analisi di un gruppo di pacchetti viene stampato a schermo il resoconto dei flussi individuati.
Termina quando verrà ricevuto il segnale di terminazione dal thread main.

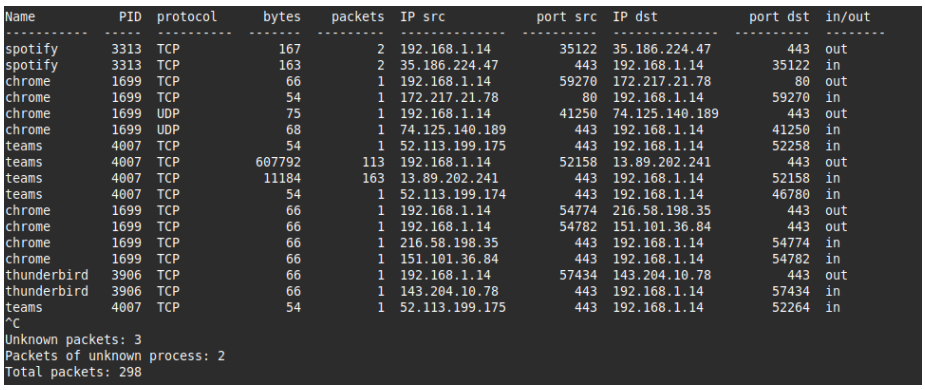
2.3 Percorso di un pacchetto

Un pacchetto appena sniffato da **Traffic** viene aggiunto al **buffer**.
Da lì, il thread **Dispatcher** preleva i pacchetti e utilizzando il dizionario pktStyle (messo a disposizione dalla classe *Pkt*) aggiunge le principali informazioni relative al pacchetto e viene smistato nella relativa lista in **Pkt**:

- se il pacchetto ha un protocollo TCP o UDP viene messo nella lista per essere collegato al suo processo;
- se ha un protocollo che non ci interessa viene messo nella lista dei pacchetti "unknown".

Intanto il thread **PortScanner** continua periodicamente a mandare le relative informazioni riguardo i processi che sono in ascolto sulle porte.
Fa uso della libreria *psutil* della quale si utilizza la funzione *psutil.net_connections()* che restituisce una lista di tuple riguardanti le varie socket connections di sistema.
La lista che viene mandata è suddivisa in base alle porte in ascolto (eg. un processo come "chrome" può avere più sottoprocessi in ascolto su porte diverse, quindi comparirà più volte la coppia "chrome" con il suo relativo PID, ognuna riguardante sottoprocessi diversi collegati a diverse porte).
Infine il thread **Visual** ha il compito di collegare i pacchetti al relativo processo e controllare se esiste o meno un flusso al quale il pacchetto potrebbe appartenere. Se c'è, viene incrementato il numero di pacchetti del flusso e viene sommato il numero di bytes del pacchetto, se non c'è viene creato un nuovo flusso.

2.4 Output



Name	PID	protocol	bytes	packets	IP src	port src	IP dst	port dst	in/out
spotify	3313	TCP	167	2	192.168.1.14	35122	35.186.224.47	443	out
spotify	3313	TCP	163	2	35.186.224.47	443	192.168.1.14	35122	in
chrome	1699	TCP	66	1	192.168.1.14	59270	172.217.21.78	80	out
chrome	1699	TCP	54	1	172.217.21.78	80	192.168.1.14	59270	in
chrome	1699	UDP	75	1	192.168.1.14	41250	74.125.140.189	443	out
chrome	1699	UDP	68	1	74.125.140.189	443	192.168.1.14	41250	in
teams	4007	TCP	54	1	52.113.199.175	443	192.168.1.14	52258	in
teams	4007	TCP	607792	113	192.168.1.14	52158	13.89.202.241	443	out
teams	4007	TCP	11184	163	13.89.202.241	443	192.168.1.14	52158	in
teams	4007	TCP	54	1	52.113.199.174	443	192.168.1.14	46780	in
chrome	1699	TCP	66	1	192.168.1.14	54774	216.58.198.35	443	out
chrome	1699	TCP	66	1	192.168.1.14	54782	151.101.36.84	443	out
chrome	1699	TCP	66	1	216.58.198.35	443	192.168.1.14	54774	in
chrome	1699	TCP	66	1	151.101.36.84	443	192.168.1.14	54782	in
thunderbird	3906	TCP	66	1	192.168.1.14	57434	143.204.10.78	443	out
thunderbird	3906	TCP	66	1	143.204.10.78	443	192.168.1.14	57434	in
teams	4007	TCP	54	1	52.113.199.175	443	192.168.1.14	52264	in

^C
Unknown packets: 3
Packets of unknown process: 2
Total packets: 298

Nella figura vi è un possibile output del programma, ogni riga fa riferimento ad un flusso.

3 Utilizzo

Per rendere semplice l'utilizzo è stato creato un makefile.
Eseguendo da terminale:
permette di installare *Scapy* e altre dipendenze come *tabulate* per la stampa

```
make install
```

in output.
Per l'esecuzione:

```
make test
```

4 Testing

Il codice è stato testato su due macchine sia con cavo lan sia via wi-fi.
Le due macchine montano entrambe linux:

- Linux Mint 20 (Ulyana)
- Pop!_OS 20.04