UNIVERSITÀ DI PISA

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

Sacco Giuseppe

14 settembre 2020

Indice

1 Introduzione

L'applicazione sviluppata per questo progetto ha l'obiettivo di monitorare i flussi di rete.
L'obiettivo è di mostrare i flussi di rete: 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 al quale il flusso appartiene.

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

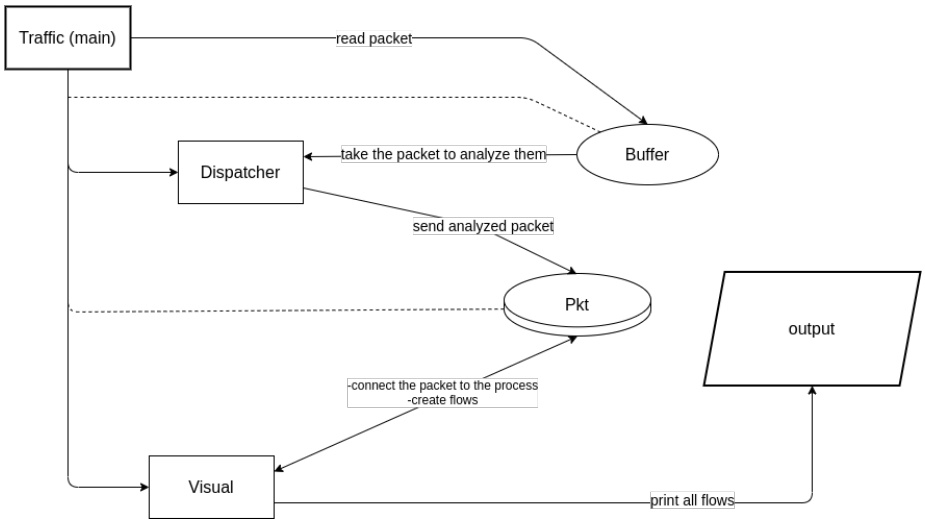
La funzione principale è quella di mostrare, dal momento dell'avvio, tutti i flussi bidirezionali facendo anche vedere a quale processo il flusso appartiene. Nella realizzazione dell'applicazione sono state fatte alcune precisazioni sui pacchetti catturati:

- 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".
Infine quando un flusso raggiunge i 15 secondi di inattività viene rimosso dalla lista dei flussi attivi.

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 e 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 e Visual.
Vengono messe a disposizione due funzioni:

- *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;

- *tryUpdateFlow*: la quale cerca all'interno di tutti i flussi uno che ha il socket locale e quello remoto uguale al pacchetto preso in analisi e restituisce 1 se viene inserito altrimenti 0.

L'accesso alla lista unanalizedPkt è gestito in mutua esclusione tramite la lock "*_lockPkts*".
Poi nella lista *flows* vengono salvati tutti i flussi bidirezionali che saranno riconosciuti tramite il thread Visual, che si occupa di gestire i nuovi flussi da aggiungere e di aggiornare le statistiche di quelli già esistenti.
Infine, vengono messi a disposizione due dizionari definiti con *chiave:valore* per gestire le informazioni relative rispettivamente ad un singolo pacchetto e a un flusso.
I due dizionari sono così definiti:

- 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}

- flowStyle={ "Name": None, "PID": None, "protocol": None, "bytes in": None, "bytes out": None, "packets": None,"local IP": None,"local port": None, "remote IP": None, "remote port": None, "last update": None}

2.2 Codice

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

- Traffic (main)

- Dispatcher

- 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 due threads: Dispatcher e Visual. In seguito utilizza la funzione sniff di Scapy per catturare i pacchetti e attraverso il parametro *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 agli altri thread un segnale di arresto

settando la variabile chiamata *term* (di tipo threading.Event()). 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 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 tramite la funzione *psutil.net_connections()* che restituisce una lista di tuple riguardanti le varie socket connections di sistema.
Quindi, per ogni pacchetto, chiama la funzione *tryUpdateFlow* di *Pkt*, la quale controlla se nella lista dei flussi (*flows*) esiste già un flusso con quel processo collegato a quella porta. Se è così viene aumentato il conteggio dei pacchetti e di bytes (in uscita o in entrata) catturati; se un flusso non esiste allora ne viene creato uno nuovo. Prima di visualizzare i flussi in output, ad ogni iterazione

viene controllato se alcuni di questi flussi sono rimasti inattivi per più di 15 secondi. Se così fosse allora verrà fatto il purge di questi flussi. 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.

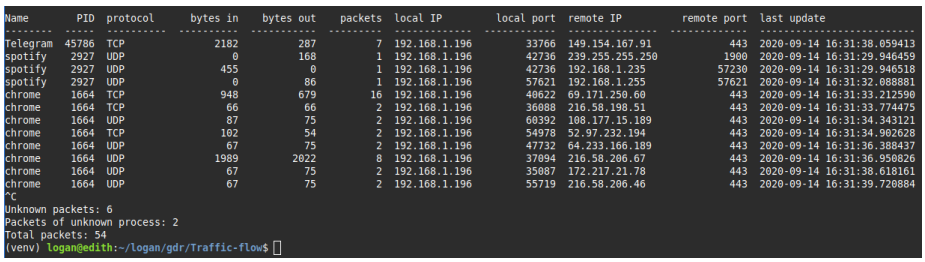
2.3 Percorso di un pacchetto

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".

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 (in entrata o in uscita) 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 in    bytes out    packets  local IP         local port  remote IP          remote port  last update
--------  -----  ----------  ----------  -----------  ---------  -------------  ------------  ---------------  -------------  --------------------------
Telegram  45786  TCP               2182          287          7  192.168.1.196         33766  149.154.167.91           443  2020-09-14 16:31:38.059413
spotify    2927  UDP                  0          168          1  192.168.1.196         42736  239.255.255.250         1900  2020-09-14 16:31:29.946459
spotify    2927  UDP                455            0          1  192.168.1.196         42736  192.168.1.235          57230  2020-09-14 16:31:29.946518
spotify    2927  UDP                  0           86          1  192.168.1.196         57621  192.168.1.255          57621  2020-09-14 16:31:32.088881
chrome     1664  TCP                948          679         16  192.168.1.196         40622  69.171.250.60            443  2020-09-14 16:31:33.212590
chrome     1664  TCP                 66           66          2  192.168.1.196         36088  216.58.198.51            443  2020-09-14 16:31:33.774475
chrome     1664  UDP                 87           75          2  192.168.1.196         60392  108.177.15.189           443  2020-09-14 16:31:34.343121
chrome     1664  TCP                102           54          2  192.168.1.196         54978  52.97.232.194            443  2020-09-14 16:31:34.902628
chrome     1664  UDP                 67           75          2  192.168.1.196         47732  64.233.166.189           443  2020-09-14 16:31:36.388437
chrome     1664  UDP               1989         2022          8  192.168.1.196         37094  216.58.206.67            443  2020-09-14 16:31:36.950826
chrome     1664  UDP                 67           75          2  192.168.1.196         35087  172.217.21.78            443  2020-09-14 16:31:38.618161
chrome     1664  UDP                 67           75          2  192.168.1.196         55719  216.58.206.46            443  2020-09-14 16:31:39.720884
^C
Unknown packets: 6
Packets of unknown process: 2
Total packets: 54
(venv) logan@edith:~/logan/gdr/Traffic-flow$
```

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

3 Utilizzo

Per usare l'applicazione è necessario installare Scapy:

```
pip install --pre scapy[basic]
```

In seguito è necessario installare *tabulate*, un tool per visualizzare in tabella i flussi.

```
pip3 install tabulate
```

Per eseguire l'app è necessario usare *sudo* per dare i diritti a Scapy per catturare i pacchetti:

```
sudo python3 Traffic.py
```

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