UNIVERSITÀ DI PISA

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

Sacco Giuseppe

27 agosto 2020

Indice

1 Introduzione **3**
1.1 Descrizione 3

2 Funzionamento **4**
2.1 Percorso di un pacchetto 4
2.2 Output 5

3 Utilizzo **5**

4 Testing **6**

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

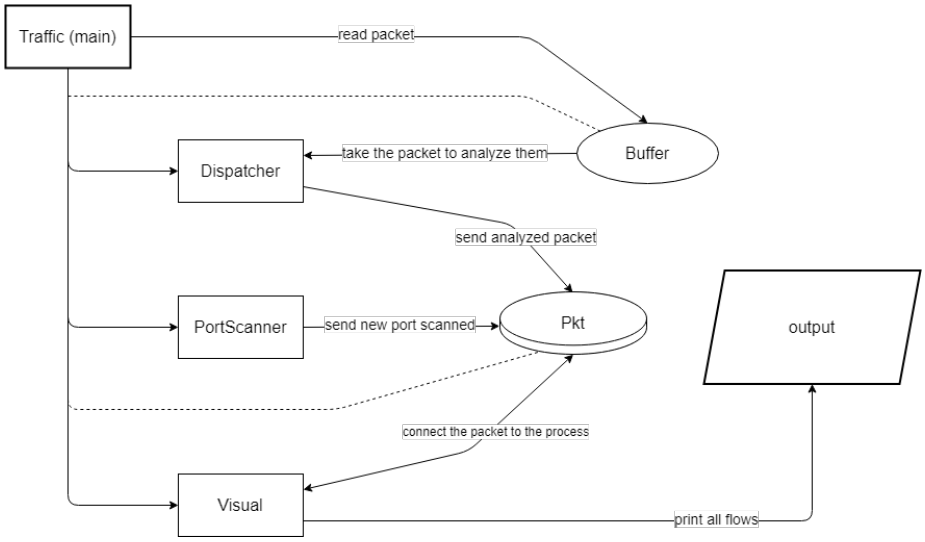
La funzione principale è quella di mostrare, dal momento dell'avvio, tutti i flussi in entrata e uscita facendo anche vedere quale processo ha inviato/ricevuto questi flussi. Nella realizzazione dell'applicazione sono state fatte alcune precisazioni sui pacchetti catturati:

- 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".

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 Percorso di un pacchetto

Un pacchetto appena sniffato da **Traffic** viene aggiunto al **buffer**.
Da lì il thread **Dispatcher** preleva i pacchetti e crea un dizionario (struttra di python) per memorizzare 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.

2.2 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