

Network Sniffer - Design Document

Introduction

This is a console application built using Python Scapy to continuously monitor the live HTTP traffic of the system. The application also has a built-in alerting system which sends out alerts in the case if traffic crosses a set threshold.

Design & Modus Operandi

Main.py:

Driver code to run the console application. It schedules three threads - to sniff the traffic continuously, the thread to display traffic information and the thread to check for the alert logic.

Sniffer.py:

This class implements a packet sniffer using Python Scapy to monitor the HTTP traffic.

Method:

```
def sniff_urls(self, packet): Thread to get packets and populate data structures to be used for displaying the traffic data. It also logs the request data in traffic file.
```

Alert.py:

This class implements the alerting logic of the application. Config.py mentions the static average value considered as the threshold for high traffic. Whenever the traffic goes above the threshold an alert is generated for "high traffic". When the system recovers from the high traffic another alert is generated to display that it has "recovered" from the high load. High traffic related information is logged for future inspection purposes.

Method:

```
def examine_traffic_behaviour(self): This method runs as a thread and checks after every 10 seconds if the traffic encountered over the last two minutes was greater than the average value. If yes, then it generates a high traffic alert otherwise if already in the high traffic phase, checks if the system has recovered.
```

Display.py:

This class contains the display format of the 10 seconds thread that displays the 3 most URL sections hit along with their counts. It also shows other traffic data like the most hit IP addresses, the traffic encountered over the last minute, and the rate of traffic of the system.

Methods:

```
def display_most_hits(self): Displays the url sections which were hit the most and
other traffic related data and checks for high traffic and shows results accordingly.

def generate_high_traffic_alert(self, url_hits, time_of_alert): Generates high traffic
alert

def generate_recovered_high_traffic_alert(self, recovery_time): generates recovery
alert
```

Config.py

This file stores the configurable parameters such as - dev(device) for the sniffer, the threshold to distinguish high traffic, and the name of the log file to log the packet details for later inspection

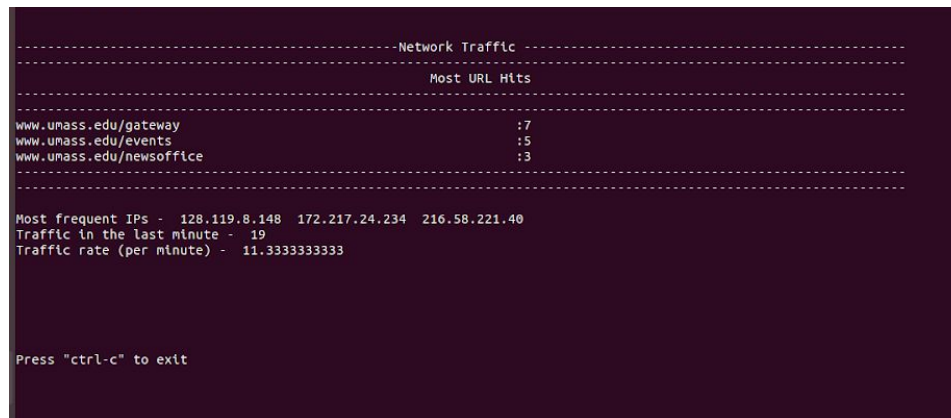
How to run:

```
sudo python main.py
```

To exit - **press ctrl + c**

Console UI Screenshots

The screen refreshes every 10 seconds(without scrolling) to display the traffic data. Whenever there is high traffic an alert is displayed(in red) mentioning the number of URL hits that caused the alert and the time the alert took place. When the traffic for the last two minutes goes below the average value, a recovery message is displayed(in green). The console keeps displaying the last alert and recovery time (for historical purposes) unless there is another high traffic alert.



```
-----Network Traffic -----
-----
Most URL Hits
-----
www.umass.edu/gateway          :7
www.umass.edu/events           :5
www.umass.edu/newsoffice       :3
-----

Most frequent IPs - 128.119.8.148 172.217.24.234 216.58.221.40
Traffic in the last minute - 19
Traffic rate (per minute) - 11.3333333333

Press "ctrl-c" to exit
```

Fig 1: Displaying the website with most hits and other traffic-related data

```
-----Network Traffic -----
-----
Most URL Hits
-----
www.umass.edu/gateway           :42
www.nyu.com/careers             :36
www.umass.edu/events            :5
-----

Most frequent IPs - 128.119.8.148 69.172.201.153 172.217.31.2
Traffic in the last minute - 0
Traffic rate (per minute) - 44.8333333333

HIGH TRAFFIC GENERATED AN ALERT
URL HITS : 90
TRIGGERED AT : 2019-01-18 14:56:06.290938

Press "ctrl-c" to exit
```

Fig 2: High alert generated(for the past two mins)

```
-----Network Traffic -----
-----
Most URL Hits
-----
www.umass.edu/gateway           :42
www.nyu.com/careers             :36
www.umass.edu/events            :5
-----

Most frequent IPs - 128.119.8.148 69.172.201.153 104.198.143.177
Traffic in the last minute - 0
Traffic rate (per minute) - 33.625

HIGH TRAFFIC GENERATED AN ALERT
URL HITS : 90
TRIGGERED AT : 2019-01-18 14:56:06.290938

RECOVERED FROM HIGH TRAFFIC AT 2019-01-18 14:58:02.134403

Press "ctrl-c" to exit
█
```

Fig 3: Recovered from the high traffic alert

```
-----Network Traffic -----
-----
Most URL Hits
-----
www.umass.edu/gateway           :91
www.nyu.com/careers             :36
www.umass.edu/events            :15
-----

Most frequent IPs - 128.119.8.148 69.172.201.153 172.217.31.2
Traffic in the last minute - 71
Traffic rate (per minute) - 32.52

HIGH TRAFFIC GENERATED AN ALERT
URL HITS : 73
TRIGGERED AT : 2019-01-18 14:59:12.587815

Press "ctrl-c" to exit
█
```

Fig 4: High traffic generated(for the last 2 min) again

Unit test to test the alert logic:

To Run the test : `sudo python alert_test.py`

The test checks for the alerting mechanism in the code. The alert class has two flags `high_traffic_flag` and `recovered_flag`. The test sends 30 requests when the threshold set in `config.py` is 20. The test checks if the `high_traffic_flag` is true. After this, no requests are sent and thus after 120 seconds the traffic should be 0 (for the past two minutes) and the traffic would have definitely recovered. Then the test code checks for the `recovered_flag` being true (and `high_traffic_flag` being false).

```
akanksha@akanksha-VirtualBox:~/network-sniffer/test$ sudo python alert_test.py
setup done
starting test
sending requests...
high_traffic_flag True
recovered_flag True
exiting test
.
-----
Ran 1 test in 178.120s

OK
akanksha@akanksha-VirtualBox:~/network-sniffer/test$ `
```

The test was passed.

Suggestions/ Improvements :

- The sniffer is created using Scapy which sniffs only HTTP traffic and not HTTPS traffic. A different sniffing framework can be used to capture even HTTPS traffic.
- The average is a static value which can be made dynamic by learning from the traffic trends in the system. If the consumption of the system increases for a long time and should not be treated as an anomaly, the newly computed average can help with that.
- The alerting mechanism thread runs every 10 secs(which can be changed in the code). So there may be an instance that sum of URL hits displayed on the console may be greater than the set average value of traffic but it will be updated only after the alert thread generates an alert by setting the `high_traffic_flag`.
- More data can be logged from the packets for inspection
- More information can be inferred about the traffic from the packets like - type of data downloaded, amount of data downloaded, the time of the day when the traffic is highest etc.
- Separate logs can be maintained for different classes of data
- Another design for the application could be to store the traffic data as a pcap file and then run the alert and display mechanism from that (could introduce lag in producing statistics). This way more information could be stored(if needed) regarding the traffic packets. I chose

to sniff live packets and handle only the required fields of the traffic which could have made the application a little restricted to only what was asked in the question.

- Data can be logged in a cleaner way by separating in different files. As of now the high traffic alerts and URLs are being logged in just one file