Networking Lab – Group Lab
Analysis with Python of Tstat network passive traces – REPORT 4

Network monitoring has a key role in understanding telecommunication networks. Today, monitoring traffic has become a key element to characterize network usage and users' activities, to understand how complex applications work and to identify anomalous or malicious behaviors. In this lab, we will analyze the output of Tstat, **a free open-source passive network monitoring tool**.
Tstat automates the collection of packet statistics of traffic aggregates, using real-time monitoring features. Developed in ANSI C for efficiency purposes, today Tstat is a powerful tool that allows sophisticated multi-Gigabit per second traffic analysis to be run live using common hardware.
Being a passive tool, the typical usage scenario is live monitoring of Internet links, in which all transmitted packets are observed. Fig. 1 sketches the common setup for a probe running Tstat: on the left there is the network to monitor, e.g., a network that is connected to the Internet through an access link that carries all packets originated from and destined to terminals in the monitored network. In our case, the monitored network is the internal network of Politecnico.

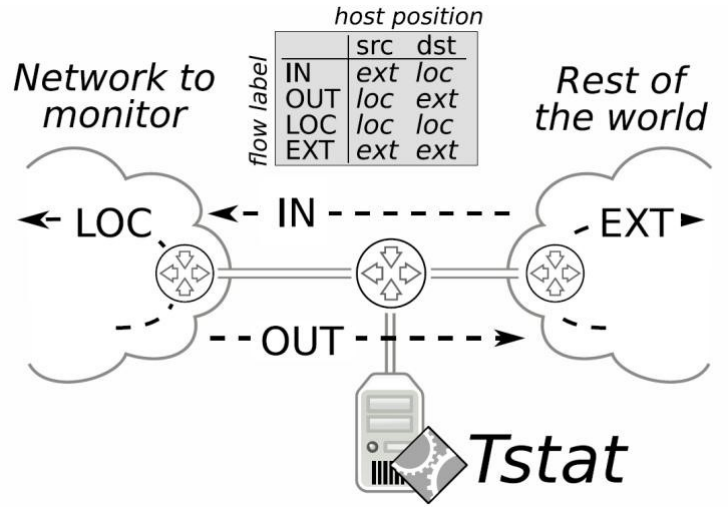


Figure 1: Tstat location in a network

The basic objects that passive monitoring tools consider are the IP packets that are transmitted on the monitored link. Grouping IP packets together, according to some rules, we derive ***flows***. A common choice is to consider:

flow ID = (ipProtocolType, ipSourceAddress, sourcePort, ipDestinationAddress, destinationPort)

so that TCP and UDP flows are automatically considered, and tracked. For example, in case of TCP, Tstat identifies a new flow start when it observes a TCP three-way handshake. Similarly, it identifies a TCP flow

end either when it sees the TCP connection teardown, or when it doesn't observe packets for some time (*idle* time). Similarly, in case of UDP, a new flow is identified when the first packet is observed, and it is ended after an idle time.

As Internet conversations are generally bidirectional, the two opposite unidirectional flows (i.e., having symmetric source and destination addresses and ports) are typically grouped together and tracked as ***connections***. This allows for gathering separate statistics for client-to-server and server-to-client flows, e.g., the size (in number of bytes) of an HTTP client request and an HTTP server response. Furthermore, the origin of the packets can be distinguished, so that it is possible to separate local hosts from remote hosts in the Internet.
As depicted in Fig. 1, there are four possible classes:

- **incoming traffic**: the source is remote and the destination is local;
- **outgoing traffic**: the source is local and the destination is remote;
- **local traffic**: both source and destination are local;
- **external traffic**: both source and destination are remote.

The local and external traffic cannot be observed at the edge, where Tstat is located.

Tstat produce traffic traces, extracting as much information as possible from the pure passive observation of packets. It first rebuilds flows, and, for each flow, a number of statistics are collected. Once the flow is terminated, Tstat logs it in a simple text file, in which each column is an attribute of the flow. Several "log files" are then created, one per type of flow (e.g., TCP log, UDP log, multimedia, log etc.). **A log file is arranged as a simple table where each column is associated to specific information and each row reports the two unidirectional flows of a connection.** The log information is a summary of the connection or flow properties. For instance, in the TCP log we can find columns like the starting time of a TCP connection, its duration, the number of sent and received packets, the observed Round Trip Time.

In the file "log_tcp_complete_description.pdf" you can find the description of each field and the classification of "Connection Type". More detailed statistics can be found on the Tstat web site at http://www.tstat.polito.it.

Analysis of TCP logs

In this lab you will perform analysis of TCP passive traces produced by Tstat. You will use Python and Pandas within a Jupyter notebook. You can use Python, Pandas and Jupyter on your favorite operating system (Linux, Mac OS X, Windows) . You can install them in different ways, but I suggest to use **Anaconda**, a cross-platform Python distribution for data analytics and scientific computing. Alternatively, Python, Pandas and Jupyter are already installed and configured in the iso file of Ubuntu already shared with you.

For this lab, we will use the file "*log_tcp_complete.xz*" that **must be downloaded** from the http://didattica.polito.it portal under the folder "Lab_Report_4_Material". This log has been collected from an operative network during 2015 and contains all TCP flows observed considering a one-hour long time

interval. The file has been compressed using *xz*, an efficient compression program, which reduced its size to 491MB from the original 2.1GB.

The file can be easily accessed with Pandas in Python by using the command below. The command *read_csv* automatically infers the format of the file (in this case *xz* compressed). To debug and tune your code, you can limit to a part of the log, by specifying the *nrows* parameter in the *read_csv* command:

```
df = pd.read_csv("Data/log_tcp_complete.xz", sep=" ", nrows = 500000)
df
```

	#15#c_ip:1	c_port:2	c_pkts_all:3	c_rst_cnt:4	c_ack_cnt:5	c_ack_cnt_p:6	c_bytes_uniq:7	c_pkts_data:8	c_bytes_all:9	c_pkts_retx:10	...	s_last_handsha
0	146.0.77.146	26775	3	0	0	0	0	0	0	2	...	
1	151.34.18.161	55077	22	0	21	14	2869	6	2869	0	...	
2	67.32.97.36	64114	9	0	8	4	375	3	375	0	...	5
3	67.32.127.193	61054	6	0	5	2	0	2	2	2	...	
4	67.32.127.193	61060	6	0	5	2	0	2	2	2	...	
...	...	...	...	...	...	...	...	...	...	...	...	
499995	67.32.82.90	61756	29	0	28	21	2477	6	2477	0	...	
499996	67.32.82.90	61843	23	0	22	17	940	4	940	0	...	
499997	67.32.82.90	61645	39	0	38	26	8073	11	8073	0	...	
499998	47.20.38.47	47967	5	0	4	2	68	1	68	0	...	
499999	67.32.49.145	51924	12	0	11	7	1728	3	1728	0	...	

500000 rows × 131 columns

For the final results, you need to analyze the whole trace. When analyzing the whole trace, you might have to read the file in chunks as in the example below. For each chunk, you can filter the set of flows that are targeted in the exercise, e.g., all HTTP traffic, or *facebook.com* flows coming from a given IP address, etc. In the end, you have to group together the information from all chunks.

Warm up -Exercises

Given the TCP log file "*log_tcp_complete.xz*":

1. Compute the total number of TCP connections
2. Compute the number of TCP connections whose client is "local"
3. Compute the number of TCP connections whose client is "local" and uses HTTP as L7-protocol
4. Compute the number of TCP connections whose client is "local", uses HTTP as L7-protocol and are directed to the service "*facebook". Repeat the computation for HTTPS flows
5. Compute the number of TCP connections whose client is "local", uses HTTP as L7-protocol, are directed to the service "*facebook" separated for each server IP address. Then plot them as a barplot.
6. Plot the barplot of point 5, this time sorting the servers by decreasing number of TCP connections.

Warm up - Solutions

Note: These solutions are limited to the first 500 000 rows. In your report you will need to analyze all the rows.

1. Count the number of rows in the file:

Total TCP connections

```
tot_conn = len(df)
print(f"The number of total TCP connections is: {tot_conn}")
```

```
The number of total TCP connections is: 500000
```

2. This time we need to filter those flows whose client is marked as “internal”

TCP connections whose client is local

```
num_local = len(df[df["c_isint:38"] == 1])
#Another way to calculate the same thing:
num_local2 = df[df["c_isint:38"] == 1].shape[0]

print(f"The number of TCP flows initiated by internal clients is: {num_local}. I also calculated it another way: {num_local2}")
```

```
The number of TCP flows initiated by internal clients is: 435767. I also calculated it another way: 435767
```

3. We need to consider those lines in which the 38th field takes the value of 1 (client internal), and then 42nd connection type field takes the value of 1 (HTTP)

TCP connections whose client is “local” and use HTTP as L7-protocol

```
df_filtered = df[(df["c_isint:38"] == 1) & (df["con_t:42"] == 1)]
num_local_http = len(df_filtered)
print(f"The number of HTTP flows initiated by internal clients is {num_local_http}")
```

```
The number of HTTP flows initiated by internal clients is 73822
```

4. As above, but consider only those lines matching with “facebook”. Differentiate between HTTP and HTTPS flows

Name of the services may appear in different columns/fields:

- colunm 116 - c_tls_SNI - Server Name Indication (SNI) - This is the name of the server requested fbhy the client when negotiating the TLS encrytpted channel -- "I'd like to contact mail.google.com"
- column 117 - s_tls_SCN - Subject Common Name (SCN) - This is the name the server is certified. It can be more generic than the requested SNI -- for instance a google server may be certified for *google.com
- column 127 - The Fully Qualified Domain Name (fqdn) the client requested to the DNS when resolving the hostname into an IP address
- column 131 - http_hostname - The hostname as seen in the HTTP requests.

Not all of them are always present (e.g., there is http_hostname only on HTTP traffic, while SNI/SCN may be there only on HTTPS traffic). In the following case, I assume it is enough if it appears in (at least) one of the 4 fields.

TCP connections whose client is “local”, use HTTP or HTTPS as L7-protocol and are directed to Facebook

```
#Local client, HTTP, directed to facebook
str_fb = "/*facebook*"
df_fb_http = df[
    (df["c_isint:38"] == 1) & (df["con_t:42"] == 1) &
    ( (df["http_hostname:131"].str.contains(str_fb)) |
      (df["fqdn:127"].str.contains(str_fb)) |
      (df["c_tls_SNI:116"].str.contains(str_fb)) |
      (df["s_tls_SCN:117"].str.contains(str_fb))
    )

]
print(f"The number of HTTP connections to Facebook, initiated by internal clients is {len(df_fb_http)}.")
```

```
The number of HTTP connections to Facebook, initiated by internal clients is 142.
```

```
#Local client, HTTPS, directed to facebook
df_fb_https = df[
    (df["c_isint:38"] == 1) & (df["con_t:42"] == 8192) &
    ( (df["http_hostname:131"].str.contains(str_fb)) |
      (df["fqdn:127"].str.contains(str_fb)) |
      (df["c_tls_SNI:116"].str.contains(str_fb)) |
      (df["s_tls_SCN:117"].str.contains(str_fb))
    )

]
print(f"The number of HTTP connections to Facebook, initiated by internal clients is {len(df_fb_https)}.")
```

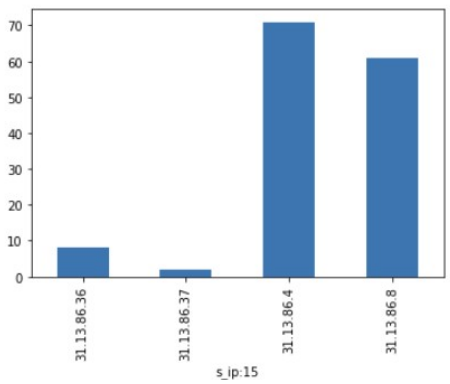
```
The number of HTTP connections to Facebook, initiated by internal clients is 10781.
```

5. This time we need to count the number of flows per single IP addresses. We can use the groupby function to count the number of flows directed to a given IP address. The server IP address will be used as “index” of the array.

```
fb_servers = df_fb_http.groupby("s_ip:15").size()
fb_servers
```

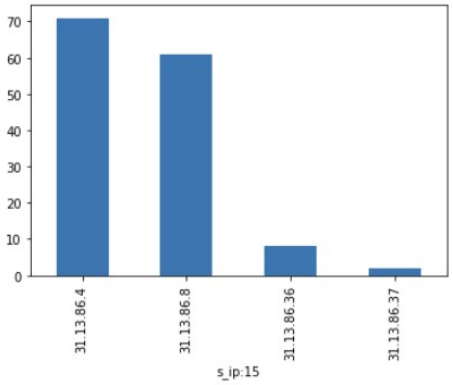
```
s_ip:15
31.13.86.36     8
31.13.86.37     2
31.13.86.4     71
31.13.86.8     61
dtype: int64
```

```
barplot = fb_servers.plot(kind="bar")
```



6. Sort the resulting series in descending order

```
barplot = fb_servers.sort_values(ascending=False).plot(kind="bar")
```



Help: Example of reading the whole file with chunks (you may need it for the final report)

```
df_iter = pd.read_csv("Data/log_tcp_complete.xz", sep=" ", iterator=True, chunksize=1000)
nlines = 0
for chunk in df_iter:
    nlines += chunk.shape[0]

print(f"The number of TCP flows in the whole trace is: {nlines}")
```

```
The number of TCP flows in the whole trace is: 4281135
```

Tasks – Part 1

When debugging scripts, you can run it on a smaller portion of the file. Specify the *nrows* parameter when reading the initial file, as shown in the examples above. **In the final report you will need to analyze all the rows.**

1. Compute the number of TCP connections whose client is "local" and are directed towards the services matching with "*facebook.com", separated for each complete service name. . Then plot them as a barplot, ordered by decreasing number of TCP connections (if they are more than 20, plot the top-20).

The match should be done considering column 116, 117,127, and 131 as in the example 4. Report as complete service name, the string in column 127. If this does not match "*facebook.com", report the string in column 131. If this does not match "*facebook.com", report the string in column 116. If this does not match "*facebook.com", report the string in column 117.
Help: Create a new column called *hostname* where you report the complete service name, according to the rules above.

1a) What happens if you match "*facebook*" instead of "*facebook.com"? Discuss the usefulness of the regex concept.

2. What are the top-15 most used services in general (without any filter on local client or service), in terms of TCP connections? Plot them as a barplot, ordered by decreasing number of TCP connections.
2a) What happens if you repeat the exercise considering only HTTPS connections?

3. Count the fraction of connections for different "connection type" (without any filter on local client or service). Plot the results as a barplot.

4. Divide the rows of the file in sequential groups of 100000 connections (or smaller when not working on the whole trace). Do not use any filter on local client or service. Count the fraction of HTTP flows for each of these groups. Then plot the series of these fractions (number of the group on the x-axis, fraction of HTTP connections on the y-axis).
Then report on the previous plot also the fraction of HTTPS connections for each group.
Finally report on the previous plot also the fraction of other protocol flows for each group.
Comment on the plot. Are there more HTTP or HTTPS flows? Are they stable in time?

Tasks – Part 2

Each group has to focus on one web service, as shown in the Googlesheet file (ask the professor). The goal is to perform a study of the selected service, by showing statistics about their usage and the infrastructure each service is using.

Consider **your assigned webservice**. First, define the list of domain names that belong to your webservice. To do this, you can use your domain knowledge or you can match substrings in the service name in the trace. You can also run Wireshark on your PC while accessing those services, and look at the names of the

servers being contacted. Use ingenuity here! No need to cover 100% of cases, but don't stop at the main service name too.
Comment on the filters you apply in order to obtain the assigned webservice.

Given your assigned webservice:

5. Which protocols are used? Count the fraction of connections for different “connection type”. Plot the results as a histogram.

6. Which sub-services are used? [e.g., www.google.com and js.google.com and www.google.it are all subservices of google]. How many are they? What is their name? Plot the total number of connections per each sub-service, ordered by decreasing number of connections (If subservices are more than 20, plot the top-20).

7. Which IP addresses of servers are used? How many are they? Plot the total number of connections per server IP address, ordered by decreasing number of connections (If IP addresses are more than 20, plot the top-20).

8. How are sub-services mapped to server IPs?
Plot the number of distinct server IPs for each subservice, in decreasing order (If subservices are more than 20, plot the top-20).
Let’s discuss why there is more than one server IP address per sub-service. Are they from the same network? Try to find the geolocation of these IP addresses. What is their Autonomous System?

How are IPs mapped to sub-services?
Plot the number of distinct sub-services for each server IP, in decreasing order (If server IPs are more than 20, plot the top-20).

9. Plot the total number of connections per each combination of sub-service and server IP. (If the combinations are more than 30, plot the top-30)

10. How many distinct clients access the service in total?
How many distinct clients access each server IP address ? Plot in decreasing order the number of clients for each IP server address (If server IPs are more than 20, plot the top-20).
How many distinct clients access each subservice? Plot in decreasing order the number of clients for each subservice (If subservices are more than 20, plot the top-20).

11. How many bytes were exchanged from the clients to the servers (uplink) in total? How many bytes were exchanged from the servers to the clients (downlink) in total?
Which fraction of the clients exchanged with the service more than 1MB in total (uplink+downlink)?

12. Count the total number of bytes handled by each server IP address. Compute this separately for the uplink (client to server) and the downlink (server to client). Plot these two metrics, ordered by decreasing number of bytes (If server IPs are more than 20, plot the top-20).
Repeat the same for sub-services.

13. For each server IP, extract the minimum of the minimum RTT and average of the average RTT from the client to the server. Plot them, sorted from the one handling the most clients (see exercise 10), to the least. (If server IPs are more than 20, plot the top-20) Can you estimate at which distance the servers are? How?

14. Plot a scatter plot, considering a point for each sub-service, characterized by the number of connections and the number of exchanged bytes each. Are there subservices that handle a lot of connections, but little data? Comment on the plot (you might use log scale for the bytes).

15. Plot a heatmap representing the number of bytes exchanged between the top-20 most active clients (in terms of number of connections) and the top-20 most contacted sub-services (in terms of number of connections). Select a proper colormap and scale and annotate the cells with whole numbers. Comment on the plot.

16. For each sub-service, plot a boxplot of the distribution of the fraction of retransmitted packets from the client to the server and vice-versa. Is there a sub-service with a lot of retransmitted packets? What could be possible reasons for a large number of retransimissions?

17. Draw the empirical Probability Distribution Function (PDF) and the empirical Cumulative Distribution Function (CDF) of the RTT (server to client, you can choose between minimum, average or maximum). The PDF should be estimated either with a histogram or with a proper density estimation. The CDF should be computed precisely (see lecture slides).

Bonus task:

18. Play with the other columns. Which analysis do you think would be interesting? (I) Propose some analyses, (ii) show its results, (iii) comment on the findings.

How to write and submit the report

In your report, you must answer to all questions above in Part 1 and 2 of the Tasks, report the code the that you have written, and show the output plots and results. You are required to comment each instruction (or group of instructions) - i.e., what it the goal of the piece of code and what are the findings. You must follow the order in which questions and exercises are posed.

Your report must be a **Jupyter notebook ".ipynb" file**. It is directly generated from a Jupyter notebook as a single file with both code, comments (markdown) and output results. Please export also the notebook as a **PDF file.**

HOW: The report (jupyter and pdf files) must be uploaded on the Teaching Portal course page didattica.polito.it under the "Elaborati" section

Naming convention: The file must be named with the following schema:

- Group<XX>_Lab4.ipynb
- Group<XX>_Lab4.pdf

For example, if you are group 12, the files must be named: Group12_Lab4.ipynb and Group12_Lab4.pdf
Reports with a wrong file name will NOT be considered.

WHEN: Fourth report deadline: **19/12/2021 h23:59**. After submission, we grade them and send them back to you by email, so that you understand how to possibly improve them.
In any case all reports must be submitted by the deadline to book the exam - Same day of exam booking date – h23:59