Aalto-yliopisto
Aalto-universitetet
Aalto University

ELEC-E7130

Internet Traffic Mesurement and Analysis

Basic Programming and Processing Data

Assignment #1

Aitor Urruticoechea Puig
aitor.urruticoecheapuig@aalto.fi
Student N°101444219
September 2023

Contents

List of Figures

Task 1: Programming Tools

For the first task, the proposed questions have been answered.

1.1 What is the function of the command awk? How does the awk command work? Could you give at least three examples highlighting its usefulness?

awk command is a Linux tool very useful for extracting data and doing basic processing with it. It is specially useful for very large files that would otherwise take too much space [1].

Its structure is quite simple. To use it, it can be called from the terminal like so:

```
urrutia1@lyta:/assginment1$ awk -F: '# awk_code' filename
```

where #awk_code should be changed by the code to be run and filename is the data one wants to analyse using awk. Alternatively, it is possible to store the awk code in a file ending in .awk to execute longer code or code that the user wants to execute multiple times, for multiple files, or just wants to have stored. This is done like

```
urrutia1@lyta:/assginment1$ awk -f code.awk filename
```

where code.awk is the file with the stored awk commands and filename is again the file to be analysed.

Its basic functioning has been summarized in this example awk code file. Note that # is used for comments.

```
BEGIN{
    #The code located here will be executed before anything else.
    count = 0 #This creates a variable with a value
}
$3>1{
    #The code here will be executed when the condition previously established is met.
    #In this example, the variable is counting how many times the value in column 3
    ↪  ($3) is more than 1
    count += 1
}
# More condition checks can be put here, or alternatively code with no condition, to be
↪  executed with every line of the file.
END{
    #The code located here will be executed at the very end.
    print (count) # This will display the value of the count.
}
```

The awk command can be very useful, for instance, to get quick statistics from a given file (column averages or sums of certain values), peek at specific values that need to be checked without the need of more heavy processing tools, or text search and pattern matching (where matches, patterns, and trends are the point of interest).

1.2 Compare the similarities and differences between Python and R, and explain in which situations Python is more suitable and in which situations R is more suitable. Provide three examples for each.

Python and R have many things in common, like wide communities backing them with multiple libraries, capacity for large scale data analysis (and subsequent visualization of the data), and many related applicabilities in statistics and machine learning. Both of them are open-source projects, making them easier to manipulate to one's advantage, and both communities benefit a lot from this to develop and implement shared tools for the world to use [2, 3].

However, it can be argued that they have more differences than things in common. On one hand, Python is a General Purpose Language. It is meant as an almost do-it-all. This situation makes it a bit of a jack of all trades, master of none, for it can be used for many applications, from web development to gaming and from data analysis to automation. Its syntax is more readable, which makes it easier to learn and adopt; and has higher usage rates because of all of these reasons. Python is more adequate for Machine Learning training and deployment, data pipelines, data integration to websites and servers, and every other not data-heavy application.

On the other hand, R is a purpose-built language for data analysis and visualization. It excels at statistical analysis and research, and allows for highly-customizable data visualization options that can be tailored to every need. When dealing with complex statistical models, R is also the go-to language, for its optimization in that regard is not comparable with what the user can get when using a General Purpose Language like Python.

1.3 What are three commonly used data analysis libraries in Python and R? Provide a brief description of the functionality of each library.

For Python, common libraries are:

- **pandas**, commonly imported as `pd`, is a fast and efficient DataFrame meant for data manipulation. It has flexible reshaping of data, along with labelling, indexing, and optimized performance; which makes it one of the top choices for this type of work with Python [4].
- **numpy**, commonly imported as `np`, adds a new array of tools to Python, mainly for numerical computation, vectorization, and optimization for working with n-dimensional arrays [5].
- **matplotlib**, commonly imported as `plt`, is meant to help with visualizing data with Python. It provides with a set of tools which allow for the creation of interactive figures compatible with a vast array of formats and intercompatible with other Python tools like Jupyter [6].

While for R, common libraries are:

- **dplyr** is a grammar set used mostly for data manipulation. Interestingly, it also offers a group-by function which allows for applying the desired transformations to full groups [7].
- **ggplot2** is a widely used library for data visualization. It provides with tools to create aesthetic plots and graphs that, while customizable, give the user the option to "forget about the details" [8].
- **readr** is used to quickly access table-type data from an array of different types of files, from `csv` to `tsv`. It also provides for quick reporting on potential errors in the data for easier troubleshooting [9].

1.4 How would you personally define latency and throughput based on your understanding? Please provide two methods for measuring latency and two methods for measuring throughput.

Latency can be understood as the time it takes for a packet of data to travel between sender and receiver, and it is thus measured in time units, typically ms (milliseconds). As it measures a "delay" between communication, it is something to be minimized. It can be measured from using the commands

- **ping**, which will measure the time for a message to go to the required server and return. This round trip is the typical way to measure latency.
- **tracert** will record the time it takes the message to hope between internet servers from the pc to the server, and not the full round trip. Instead, it is useful to see if some hopes along the route are the cause of connectivity issues.

Throughput can be defined as the amount of data that a system can transmit over a period of time. In plain language, it is commonly understood as "speed", for it typically measured in some multiple of bps (bytes per second). It can be measured either with

- **iperf**, a command line that needs to be run in both server and client, and after executing will return the throughput value between the two in bps.
- **Dedicated Analytic Webpages**, if the user prefers easier tests and does not want to run commands in both server and client, the general throughput connection with the wider internet can be tested in many specialized webpages that will execute such tests automatically from the web browser.

Task 2: Processing CSV data using awk

For this task, the file log_tcp_complete, located in the course's server, has been analysed using the awk command.

2.1 How can you peek at a file if it is too large to fit into memory?

Small bits of a way-too-large-file can be peeked into using either the command head or tail. Both work very similarly, for the user need only indicate the number of rows they want to display, either beginning from the top of the file using head, or alternatively tail if the way to begin from the end of the document. As an example, the tail command has been used to print the last 3 lines of the file log_tcp_complete.

```
1 urrutia1@lyta:/work/courses/unix/T/ELEC/E7130/general/trace/tstat/2017_04_11_18_00.out$ tail -2 log_tcp_complete
2 163.35.93.244 58938 435 0 434 260 228163 173 229537 1 1374 1 1 1 94.47.215.181 80 498 0 498 97 541009 399 541009 0 0 0 1 1 1492008873135.538086 1492008899419.040039 26283.502000 250.674000 1932.930000 20343.815000 21236.893000 249.043000 489.623000 1 0 0 0 0 0 0 239.759755 236.695000 277.526000 5.421766 98 56 56 17.494759 9.597000 111.072000 16.236706 215 50 50 0 0 0 0 0 0 1 1 5 1 0 1386 1374 70 131904 65535 0 59575 603 603 0 0 1 0 1 0 0 0 0 1 1 7 1 2 1460 1374 62 178176 28960 0 112668 565 13740 0 0 0 0 0 0 0 0 0 0 0 --- 12 18 - - 0 0 0 0.000000 0.000000 0.000000 0.000000 0 0 - - 0.0 0.0
3 59.44.95.176 47121 3 0 0 0 0 0 0 2 2 0 3 0 203.246.140.57 22 6 0 6 0 0 0 0 5 5 0 6 0 1492008874871.737061 1492008898868.857910 23997.121000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0 1 0 0 0 0 0 0.694931 0.695000 0.695000 0.000000 1 54 54 0.000000 0.000000 0.000000 0.000000 0 63 63 0 0 0 0 0 0 1 1 7 1 0 1460 0 0 3737600 29200 0 0 0 0 0 0 0 0 0 0 2 0 0 1 1 1 1 0 1460 0 0 131070 65535 0 0 0 0 5 0 0 0 0 0 0 0 0 0 0 --- 0 0 - - 0 0 0 0.000000 0.000000 0.000000 0.000000 0 0 - - 0.0 0.0
4 187.82.166.205 12876 9 0 8 6 213 1 213 0 0 0 1 1 163.35.11.105 80 7 0 7 2 4858 4 4858 0 0 0 1 0 1492008898985.959961 1492008899933.167969 947.208000 320.934000 328.491000 320.934000 328.497000 320.920000 326.132000 0 1 0 0 0 0 0 7.829406 7.557000 8.259000 0.376310 3 53 53 308.565629 306.855000 312.661000 2.417251 5 58 58 0 0 0 0 0 0 1 1 7 1 0 1460 213 213 66560 63616 0 213 213 213 0 0 0 0 0 0 0 0 0 1 1 3 1 0 1460 1448 514 131768 65535 0 4858 1448 4858 0 0 0 0 0 0 0 0 0 0 0 --- 1 0 - - 0 0 0 0.000000 0.000000 0.000000 0.000000 0 0 - - 0.0 0.0
```

2.2 Print the first line (i.e. headers of the columns 3, 7, 10, 17, 21, 24)

To do that, the NR variable in awk has been used with the restriction to print only the data in the first row, using NR == 1. Note, however, that this could have also been achieved with the head command.

```
urrutia1@lyta:/work/courses/unix/T/ELEC/E7130/general/trace/tstat/2017_04_11_18_00.out$ awk -F: 'NR==1{print $0}' log_tcp_complete
#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 c_bytes_retx:11 c_pkts_ooo:12 c_syn_cnt:13 c_fin_cnt:14 s_ip:15 s_port:16 s_pkts_all:17 s_rst_cnt:18 s_ack_cnt:19 s_ack_cnt_p:20 s_bytes_uniq:21 s_pkts_data:22 s_bytes_all:23 s_pkts_retx:24 s_bytes_retx:25 s_pkts_ooo:26 s_syn_cnt:27 s_fin_cnt:28 first:29 last:30 durat:31 c_first:32 s_first:33 c_last:34 s_last:35 c_first_ack:36 s_first_ack:37 c_isint:38 s_isint:39 c_iscrypto:40 s_iscrypto:41 con_t:42 p2p_t:43 http_t:44 c_rtt_avg:45 c_rtt_min:46 c_rtt_max:47 c_rtt_std:48 c_rtt_cnt:49 c_ttl_min:50 c_ttl_max:51 s_rtt_avg:52 s_rtt_min:53 s_rtt_max:54 s_rtt_std:55 s_rtt_cnt:56 s_ttl_min:57 s_ttl_max:58 p2p_st:59 ed2k_data:60 ed2k_sig:61 ed2k_c2s:62 ed2k_c2c:63 ed2k_chat:64 c_f1323_opt:65 c_tm_opt:66 c_win_scl:67 c_sack_opt:68 c_sack_cnt:69 c_mss:70 c_mss_max:71 c_mss_min:72 c_win_max:73 c_win_min:74 c_win_0:75 c_cwin_max:76 c_cwin_min:77 c_cwin_ini:78 c_pkts_rto:79 c_pkts_fs:80 c_pkts_reor:81 c_pkts_dup:82 c_pkts_unk:83 c_pkts_fc:84 c_pkts_unrto:85 c_pkts_unfs:86 c_syn_retx:87 s_f1323_opt:88 s_tm_opt:89 s_win_scl:90 s_sack_opt:91 s_sack_cnt:92 s_mss:93 s_mss_max:94 s_mss_min:95 s_win_max:96 s_win_min:97 s_win_0:98 s_cwin_max:99 s_cwin_min:100 s_cwin_ini:101 s_pkts_rto:102 s_pkts_fs:103 s_pkts_reor:104 s_pkts_dup:105 s_pkts_unk:106 s_pkts_fc:107 s_pkts_unrto:108 s_pkts_unfs:109 s_syn_retx:110 http_req_cnt:111 http_res_cnt:112 http_res:113 c_pkts_push:114 s_pkts_push:115 c_tls_SNI:116 s_tls_SCN:117 c_npnalpn:118 s_npnalpn:119 c_tls_sesid:120 c_last_handshakeT:121 s_last_handshakeT:122 c_appdataT:123 s_appdataT:124 c_appdataB:125 s_appdataB:126 fqdn:127 dns_rslv:128 req_tm:129 res_tm:130
```

2.3 Calculate the average of the columns 3, 7, 10, 17, 21, 24

To do this task in an orderly manner, the file 2dot3.awk has been created in the student's personal folder. In this file, the data in each column is added independently, which allows for later calculation of the average of each column, as well as the calculation of the global average value between the data of all six columns.

```
# File 2dot3.awk
# Aitor Urruticoechea 2023

BEGIN{
    n_total=0
    sum_total=0
    sum_3 =0
    sum_7 =0
    sum_10 =0
    sum_17 =0
    sum_21 =0
    sum_24 =0
}
{sum_total = sum_total + $3 + $7 + $10 + $17 + $21 + $24}
{num_total= num_total+1}
{sum_3 = sum_3+ $3}
{sum_7 = sum_7+ $7}
```

```
{sum_10 = sum_10+ $10}
{sum_17 = sum_17+ $17}
{sum_21 = sum_21+ $21}
{sum_24 = sum_24+ $24}
END{
    print FILENAME
    print "AVERAGES"
    print sum_3/num_total
    print sum_7/num_total
    print sum_10/num_total
    print sum_17/num_total
    print sum_21/num_total
    print sum_24/num_total
    print sum_total/(num_total*6)
}
```

This .awk file is then called from the terminal to analize the log_tcp_complete file. This outputs the desired averages, starting from the average of column 3 and ending with the average of column 24 in line 9. Line 10 shows the average of all 6 columns combined.

```
urrutia1@lyta:/work/courses/unix/T/ELEC/E7130/general/trace/tstat/2017_04_11_18_00.out$ awk -f /m/home/home2/26/urrutia1/unix/assignment1/2dot3.awk log_tcp_complete
log_tcp_complete
AVERAGES
93.573
36741.1
0.491826
178.716
214087
2.2859
41850.6
```

2.4 Calculate the percentage of records where column10/column7 exceeds a) 0.01, b) 0.10, c) 0.20

Similarly to the previous exercise, a file named 2dot4.awk has been created. In this case, it is important to both check that the seventh column is neither empty (meaning having data with length zero), nor with data equal to zero, for in both cases it will result in an attempted division by zero. This condition is checked before any attempted division is done, as well the necessary filtering out of the first row where the header is.

```
# File 2dot4.awk
# Aitor Urruticoechea 2023

BEGIN{
    sum_a = 0
    sum_b = 0
    sum_c = 0
    sum_total = 0
}
NR>1 && length($7)!=0 && $7!=0{
sum_total += 1;
if ($10/$7 > 0.01)
    sum_a += 1;
```

```
    if ($10/$7 > 0.1)
        sum_b += 1;
        if ($10/$7 > 0.2)
            sum_c += 1;
}
END{
    print FILENAME
    print "PERCENTAGES"
    print (sum_a/sum_total)*100
    print (sum_b/sum_total)*100
    print (sum_c/sum_total)*100
}
```

When called, the script prints out the desired percentages in order. First, the percentage that exceeds 0,01 (1,96%), then the percentage that exceeds 0,10 (0,57%), and finally the percentage that exceeds 0,20 (0,35%).

```
urrutia1@lyta:/work/courses/unix/T/ELEC/E7130/general/trace/tstat/2017_04_11_18_00.out$ awk
↪ -f /m/home/home2/26/urrutia1/unix/assignment1/2dot4.awk log_tcp_complete
log_tcp_complete
PERCENTAGES
1.9574
0.567982
0.346482
```

2.5 Calculate the maximum of each column: 3, 9, 17, 23, 31

Again, a file named 2dot5.awk has been generated to have the script in a single place. The script will ignore the header row and check in the rest whether a larger number than previously seen exists, and store it.

```
# File 2dot5.awk
# Aitor Urruticoechea 2023
BEGIN{
    max3 = 0
    max9 = 0
    max17 = 0
    max23 = 0
    max31 = 0
}
NR!=1 {
if ($3 > max3)
    max3 = $3;
if ($9 > max9)
    max9 = $9;
if ($17 > max17)
    max17 = $17;
if ($23 > max23)
    max23 = $23;
if ($31 > max31)
    max31 = $31;
}
END{
    print FILENAME
    print "MAXIMUMS"
```

```
25      print max3
26      print max9
27      print max17
28      print max23
29      print max31
30  }
```

When called, it prints the maximums in the column's order, starting by the maximum of the third column and ending with the maximum of the thirty-first column.

```
1  urrutia1@lyta:/work/courses/unix/T/ELEC/E7130/general/trace/tstat/2017_04_11_18_00.out$ awk
   ↪ -f /m/home/home2/26/urrutia1/unix/assignment1/2dot5.awk log_tcp_complete
2  log_tcp_complete
3  MAXIMUMS
4  1700014
5  313220528
6  2791706
7  3835783508
8  72901107.834000
```

Task 3: Processing throughput and latency data

3.1 Latency data using ping

You need to complete the following points:

1. *Load the CSV into a DataFrame and change the timestamp to date format.*
2. *Plot the average RTT over a time series from the provided CSV file as a first approach to the data and analysis of this.*
3. *Generate another CSV file or dataframe with the values calculated of the average of successful RTTs, the maximum of RTTs, and the percentage of packet loss every hour.*
4. *Plot another time series to observe the behavior of the RTTs (average and maximum) according to the measurements calculated in each hour.*
5. *Can you make any conclusions of stability and latency based on data?*

Using Pyhton [2] and its library Pandas [4], the required data has been loaded into a DataFrame. Importantly, using seconds as a base, the timestamp data has been converted into something actually readable. Having done that, the average RTT can be easily plotted against time to have a general idea of what kind of data is this (Figure 1). There, it can be seen how during the first hours of measurement there is a mostly constant RTT except around the second hour, where a gap is clearly observed. Then, it is after midnight when important spikes in average RTT are observed, that last all the way to the end of the measurements.

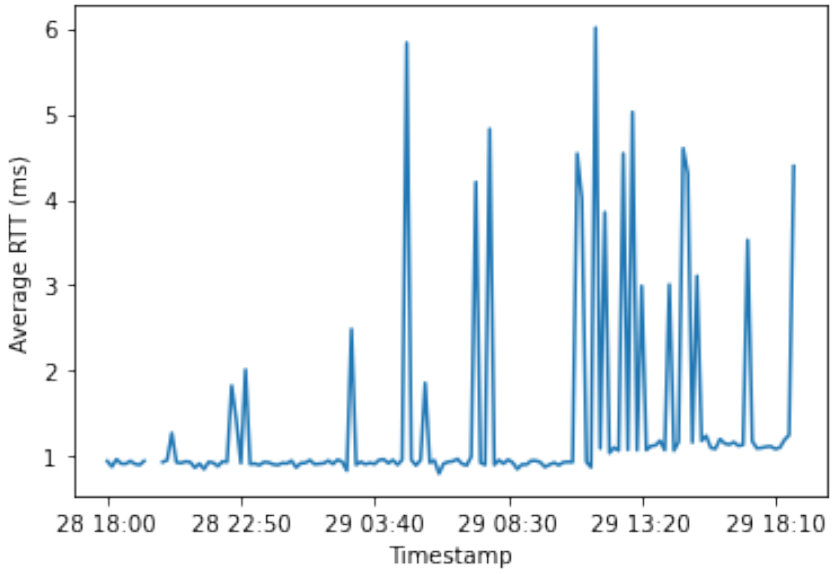


Figure 1: Average RTT over time series, unclean data.

Now, to get hourly data averages, etc; a `for` loop has been used. This is suboptimal when using Pandas' DataFrames and in order to optimize somewhat the computations data is accessed but not directly edited in the DataFrame. Instead, it is stored as a list and only when the loop ends is the list actually transformed into a DataFrame for easier handling. These new DataFrames can now be plotted to get the data of interest, namely the hourly average and maximum RTT (Figure 2) and average hourly packet loss (Figure 3). This, now treated, data, can clearly show some of the issues observed with the raw data. From the packet loss, it is clear that the gap observed in the second hour is due to the loss of over 50% of packets (maybe due to connectivity issues?) an issue that is

nonetheless solved by the end of the hour as for the rest of the measurements the packet loss is basically zero.

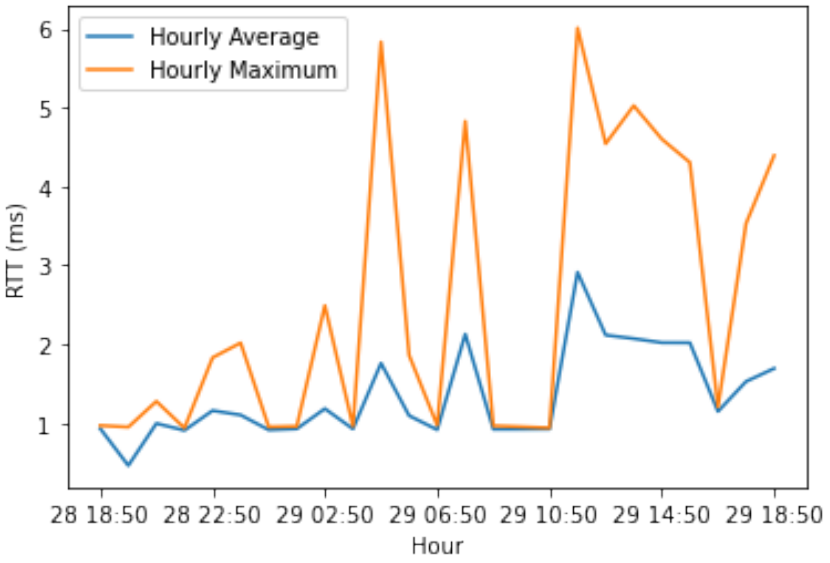


Figure 2: Average and Maximum Hourly RTT.

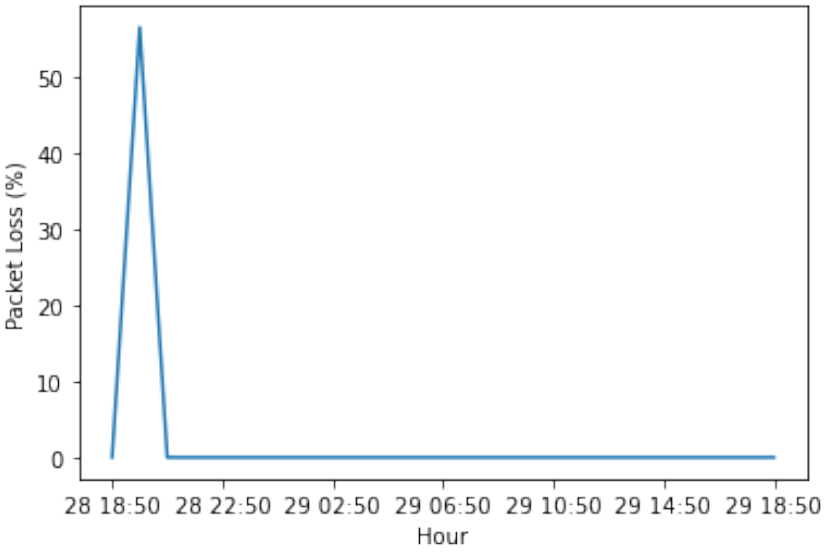


Figure 3: Average Hourly Packet Loss

Regarding RTT, it is interesting how, quite clearly, the increases in average hourly RTT are rarely due to general increase of RTT. Rather, they are clearly correlated to an increase of the maximum RTT recorded that hour. Since the measurements are every 10 minutes, it would be possible to hypothesize that these sudden increases in maximum RTT and subsequentially increases in average RTT could be due to punctual network congestion. Alternatively, since this spikes happen clearly after 3 am, and do not seem to return to normal conditions at the same hour next day, another possible hypothesis is that due to a change in routing policy, latency has been increased for the user being analysed. This does not have to be a permanent change, but there is a chance that the measurements just caught the transition moment between one (faster) route and the new (slower) one.

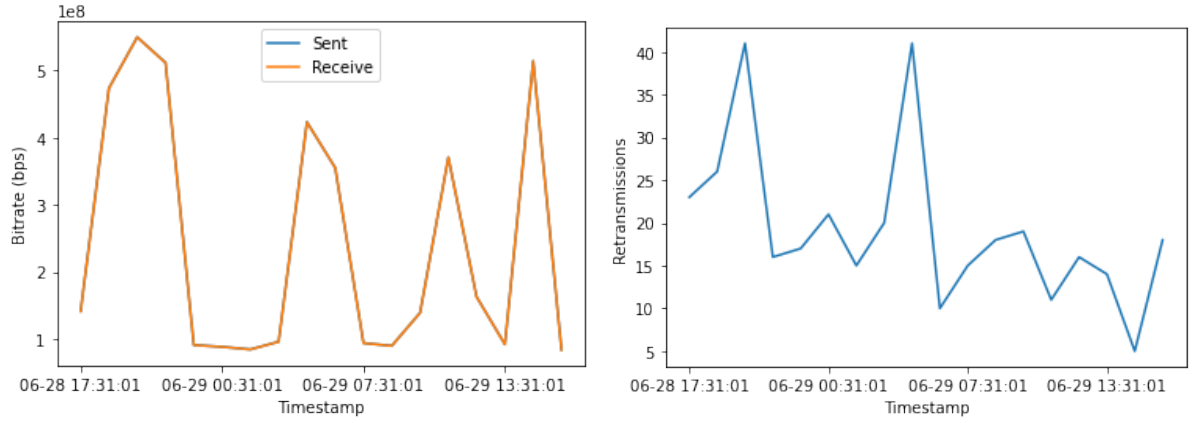
3.2 Throughput data using iperf3

You need to complete the following exercises:

1. *Load the CSV into a DataFrame and change the timestamp to date format.*
2. *Remove the rows with values "-1" and classify the mode sent based on the column 'Mode', where '0' is normal (client-server) and '1' refers to reverse (server-client)*
3. *Plot comparing bitrate and TCP retransmissions over a time series (one for normal direction and one for reverse)*
4. *Create a scatter plot to observe the relationship between TCP retransmissions and bitrate (one for normal direction and one for reverse)*
5. *Can you make any conclusions of stability based on data and relationship between bitrate and TCP retransmissions?*

In the same way as before, Python and Pandas have been used, and the data has been loaded into a DataFrame with the timestamps corrected taking seconds as a base. Again as well, a `for` loop has been used to clean the data despite it not being the most efficient way. Since these files are not big enough for this to be a noticeable problem, this path has been chosen for simplicity's sake. In a single loop, the DataFrame is read and the indexes corresponding to rows with values of "-1", "0", and "1" are recorded. Then, the rows with values equal to "-1" can be safely deleted for a cleaner global DataFrame; while two new DataFrames can be created with the rows with "0" (normal operations, meaning client-server) and "1" (reverse operations, meaning server-client).

It is of interest now to plot the bitrate and TCP retransmissions over time, with differentiated plots in both directions (see Figure 4 for normal operations and Figure 5 for reverse operations). Note that in both operation types, for bitrate data, both sent and received data has been plotted (Figures 4a and 5a). Since they represent very similar values, however, it is hardly possible to differentiate them.

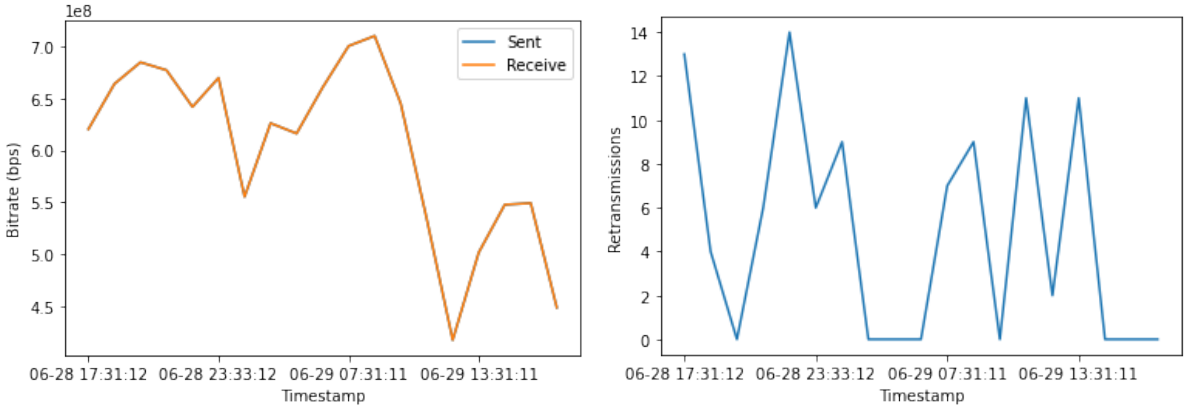


(a) Bitrate (bps) against time.

(b) TCP Retransmissions against time.

Figure 4: Bitrate and TCP Retransmissions against time for normal operations.

Nothing specially notable occurs in these plots, for without information about what kind of operations were intended behind this connection are unknown it is not possible to assess if this connection is performing as expected. In general, nothing out of the ordinary is observed. It is then, of interest, to try to search for some correlation between bitrate and TCP retransmissions, leaving time aside. This can potentially point towards performance issues, connectivity problems, and the like.

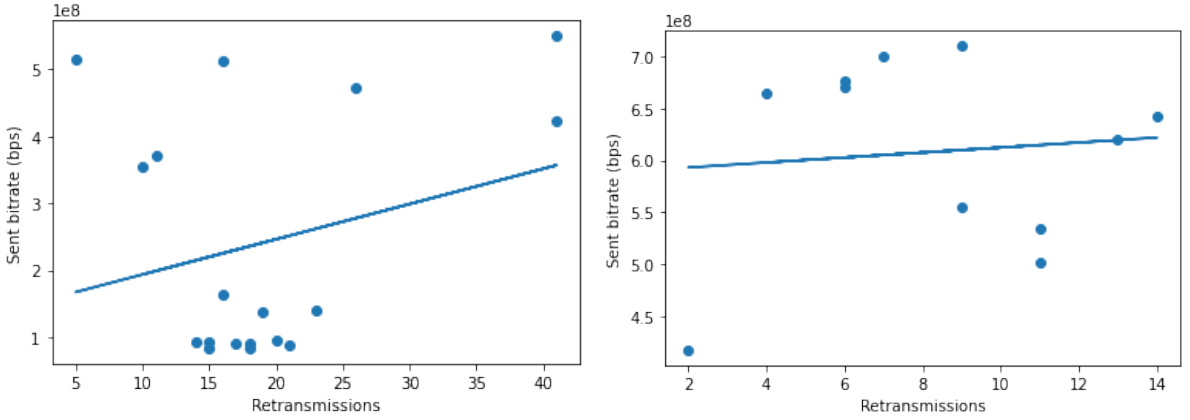


(a) Bitrate (bps) against time.

(b) TCP Retransmissions against time.

Figure 5: Bitrate and TCP Retransmissions against time for reverse operations.

These new plots (Figure 6), show no correlation between the two parameters. Please note that for Figure 6b, bitrates with TCP retransmissions equal to zero did exist in the given data, and have naturally been removed. This has been further showcased by plotting a fitted polynomial of first order, which clearly cannot be used to draw any conclusions for the data is not even remotely correctly approximated by these lines. Note that this (lack of) correlation continues regardless of the scale (scalar or logarithmic) used horizontally and/or vertically.



(a) Bitrate (bps) against TCP Retransmissions, normal operations.

(b) Bitrate (bps) against TCP Retransmissions, reverse operations.

Figure 6: Scatter plots of Bitrate (bps) against TCP Retransmissions, for both types of operations.

This lack of correlation seems to indicate a certain stability exists in the network, with resources to deal with sudden transmissions without congesting the network and/or appropriate congestion control. This should also mean general low package loss and subsequent lack of need for further retransmissions; or alternatively good and reliable error correction methods on the network and/or client side to avoid saturation of the connection despite package loss. That is all assuming, of course, that the data given is truthful and has not been contaminated by the presence of firewalls or similar contraptions that might make it look like there is no correlation between them when analysed.

References

1. *GAWK: Effective AWK Programming: A User's Guide for GNU Awk* [online]. 2009. [visited on 2023-09-20]. Available from: https://www-zeuthen.desy.de/dv/documentation/unixguide/infohtml/gawk/gawk.html.
2. *About Python* [online]. Python Foundation, 2022 [visited on 2023-09-20]. Available from: https://www.python.org/about/.
3. *What is R?* [online]. R Foundation, 2023 [visited on 2023-09-20]. Available from: https://www.r-project.org/about.html.
4. *pandas - Python Data Analysis Library* [online]. AQR Capital Management, 2023 [visited on 2023-09-20]. Available from: https://pandas.pydata.org/about/.
5. *NumPy* [online]. NumPy, 2023 [visited on 2023-09-20]. Available from: https://numpy.org/.
6. *Matplotlib - Visualization with Python* [online]. The Matplotlib development team, 2023 [visited on 2023-09-20]. Available from: https://matplotlib.org.
7. *A Grammar of Data Manipulation* [online]. dplyr, 2023 [visited on 2023-09-20]. Available from: https://dplyr.tidyverse.org/.
8. *Create Elegant Data Visualizations Using the Grammar of Graphics* [online]. ggplot2, 2023 [visited on 2023-09-20]. Available from: https://ggplot2.tidyverse.org/.
9. *Read Rectangular Text Data* [online]. readr, 2023 [visited on 2023-09-20]. Available from: https://readr.tidyverse.org/.