Aalto-yliopisto
Aalto-universitetet
Aalto University

ELEC-E7130
Internet Traffic Measurement and Analysis

From measurements to conclusions

Final Assignment

Aitor Urruticoechea Puig
aitor.urruticoecheapuig@aalto.fi
Student N°101444219
November - December 2023

Acknowledgement

Parts of the code implemented in this work were generated with the assistance of GitHub Copilot [1]. None of its proposals were, however, employed without refinement and necessary tweaks to adapt it to the nuances of the tasks at hand; making it impossible to identify and subsequently mark which lines of code were human or machine-made, for many were the fruit of the combination of both.

Contents

List of Figures

List of Tables

Acronyms

CDF Cumulative Distribution Function. 2, 22, 23, 45

DNS Domain System Name. 13

ECDF Empirical Cumulative Distribution Function. 2, 6, 7, 9, 10, 35, 37

HTTP Hypertext Transfer Protocol. 13, 18

ICT Information and Communication Technologies. 32

IP Internet Protocol. 2, 9, 13, 32, 36

ISP Internet Service Provider. 4, 9, 19, 31, 32

PDF Probabiltiy Density Function. 2, 22, 24, 45

RTT Round Trip Time. 2, 11, 12, 19, 22, 32, 39

STD Standard Deviation. 9, 10, 12

TCP Transmission Control Protocol. 2, 4–7, 32, 34

UDP User Datagram Protocol. 2, 5–7, 13, 34

UI User Interface. 4–6, 34, 35

UTC Coordinated Universal Time. 8

Task 1: Capturing Data

In this task, flow data has been captured for 4 hours in the afternoon and evening of Sunday, 19th of November 2023. It originates from a desktop computer located in Leppävaara, Espoo. It is connected to the internet via Ethernet cable; with the Internet Service Provider (ISP) guaranteeing the connection being Finish provider DNA. During this time, the computer experienced usual usage, ranging from university-related work, to listening to music or watching videos online.

1.1 Acquiring packet capture data

Packet acquisition has been performed using Wireshark [2]. The exact instructions for how this step was performed can be found in Appendix A. When doing a preliminary plot using Wireshark's own User Interface (UI); some insights as to how the data generally looks like can be extracted. Looking at Figure 1; one can clearly see how the 4-hour-long capture of flows has experienced a myriad of conditions, ups and downs, according to its use, ensuring interesting results moving forward.

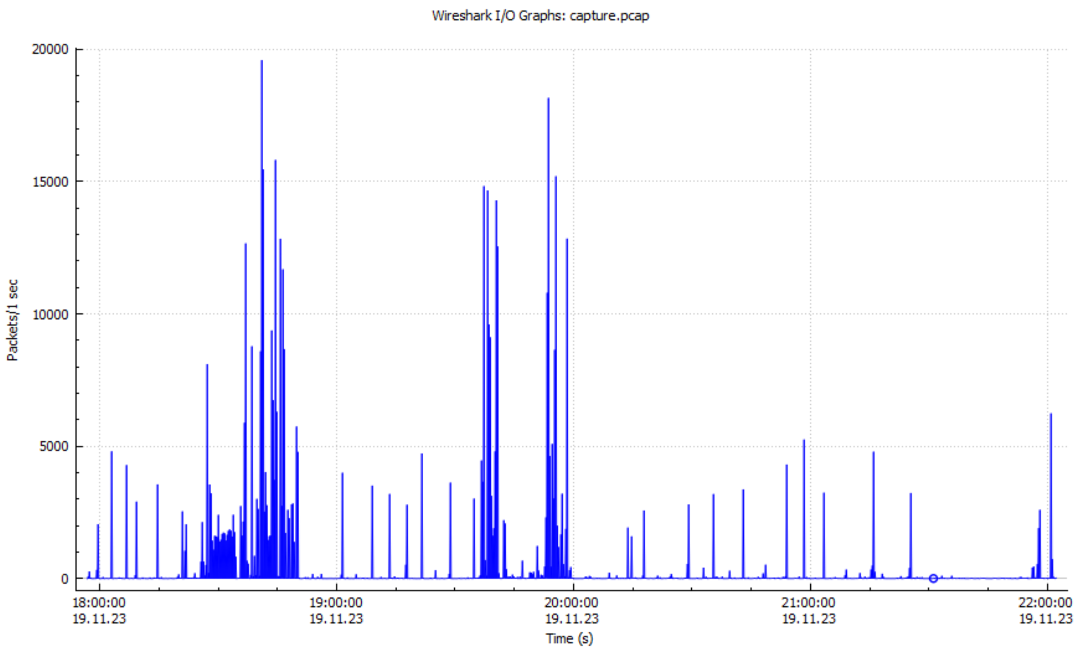


Figure 1: Preliminary plot of the captured data for Task 1 (packets/second against time).

1.2 Data pre-processing

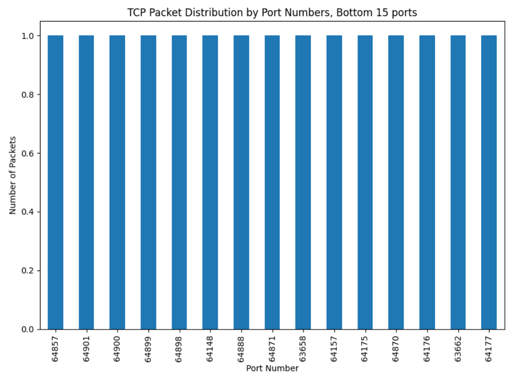
From the captured data, three different data sets can be identified from the different pre-processing techniques applied. First, **PS1** is understood as the raw `.pcap` file. This raw file has then been processed as instructed using Coral Flow; which translates the capture into data flows; thus obtaining **PS2**. On the flip side, one can instead process the raw file using Transmission Control Protocol (TCP) Trace; thus obtaining TCP connection statistics. This results in **PS3**, the third and final data set to be considered for analysis. As usual, detailed instruction on the methods employed for obtaining these datasets are included in Appendix A.

1.3 Data Analysis

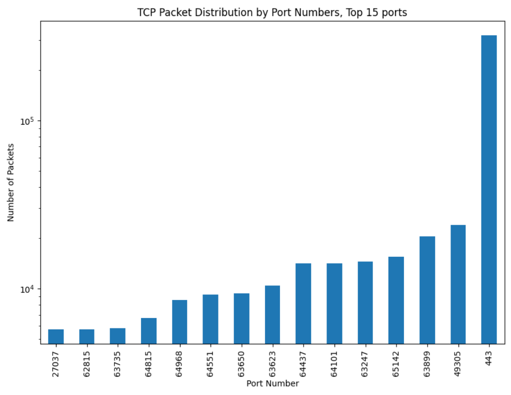
All instructions and relevant technical details stemming from the process of obtaining the results in this section can be found in Appendix A.

1.3.1 Packet data PS1

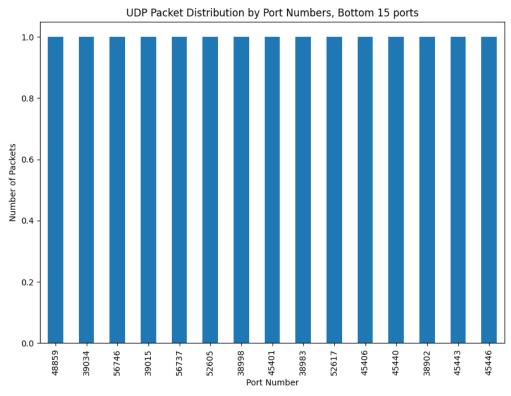
Analysing directly the raw `.pcap` file understood as **PS1** can be quite a heavy load on the chosen programming language, Python [3]. Here, however, Wireshark's own UI comes handy to clean up the data needed for each of the desired figures and relevant statistical values. To begin with, it is of interest to see the port usage for both User Datagram Protocol (UDP) and TCP connections. Since the data file is quite large, only top and bottom 15 ports are shown for each case, which nonetheless shed light into the composition of the data. See Figure 2.



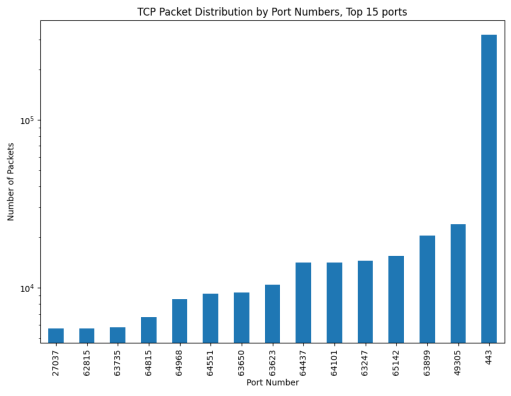
(a) Bottom 15, least used, TCP ports.



(b) Top 15, most used, TCP ports.



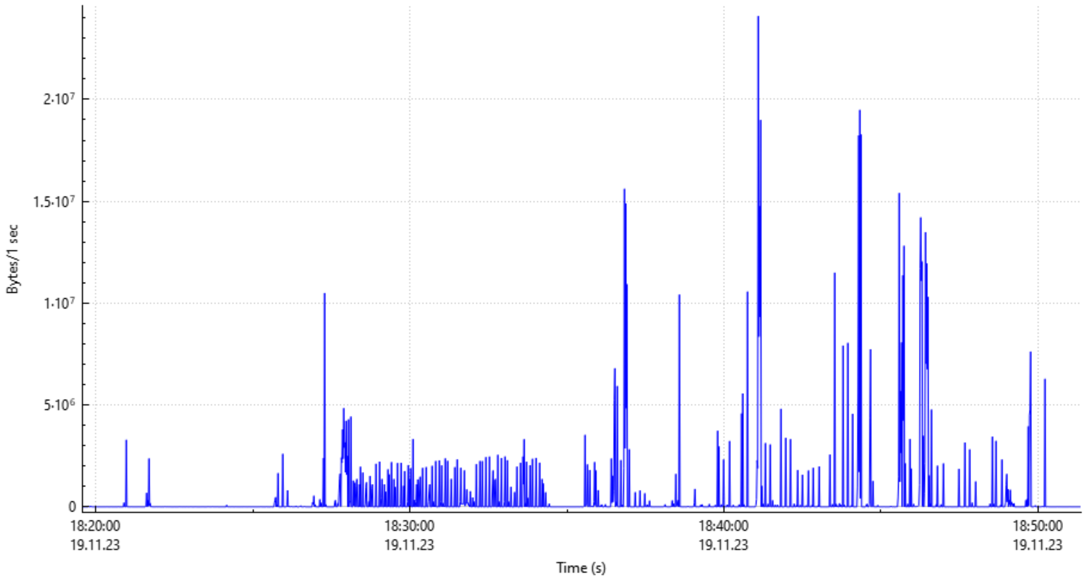
(c) Bottom 15, least used, UDP ports.



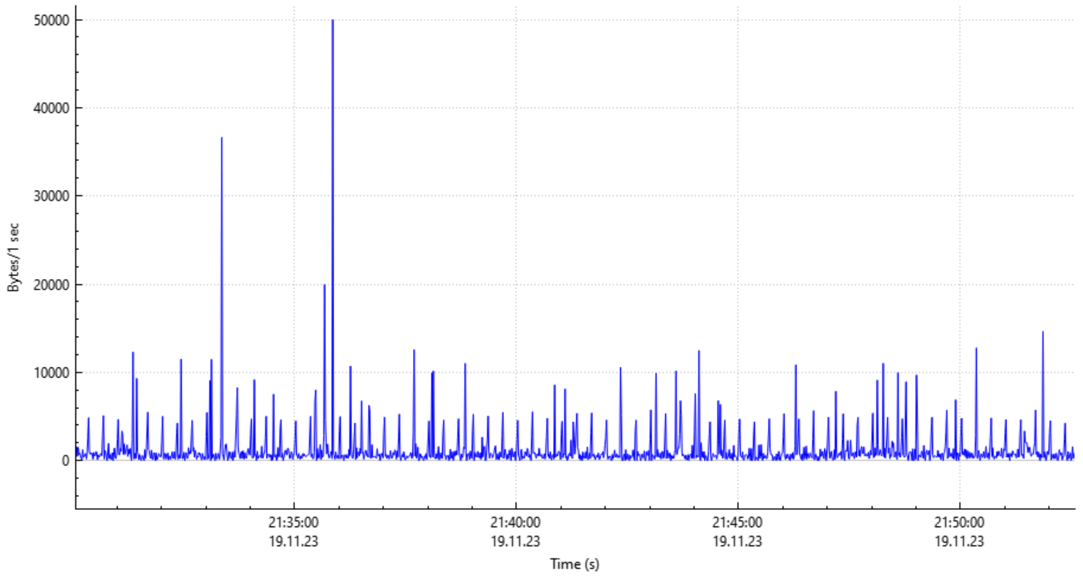
(d) Top 15, most used, UDP ports.

Figure 2: Most and Least used TCP and UDP ports.

Traffic volume as a function of time can be obtained directly from Wireshark's UI. In this case (Figure 3), zoom has been made in the periods of highest and lowest activity, which comprise the time periods between 18:25 to 18:50 and 21:35 to 21:50 respectively.



(a) Highest activity period.

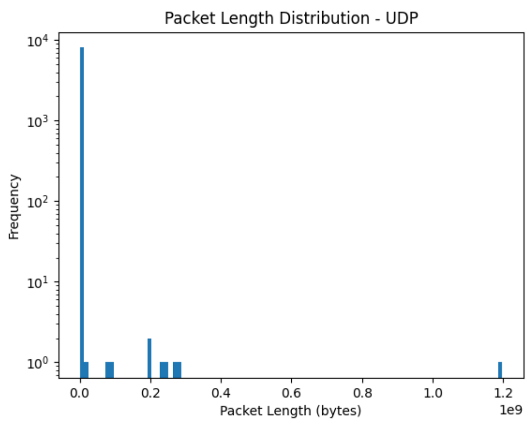


(b) Lowest activity period

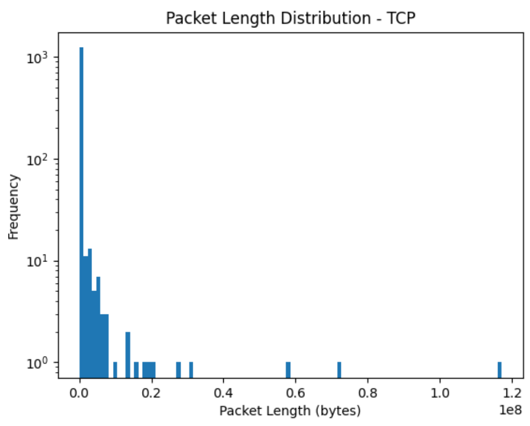
Figure 3: Traffic volume against time, for two separate time periods.

The distribution of packet length can be found, for TCP and UDP in Figure 4; while Figure 5 shows Empirical Cumulative Distribution Function (ECDF) plots for both protocols. To wrap it up, relevant statistical data has been included as obtained from Wireshark's UI:

- **Packets**: 1.547.904
- **Timespan, s**: $14.700, 838$
- **Average pps**: $105, 3$
- **Average packet size, B**: 1134
- **Bytes**: 1.755.492.282
- **Average kbytes/s**: 119
- **Average kbits/s**: 955

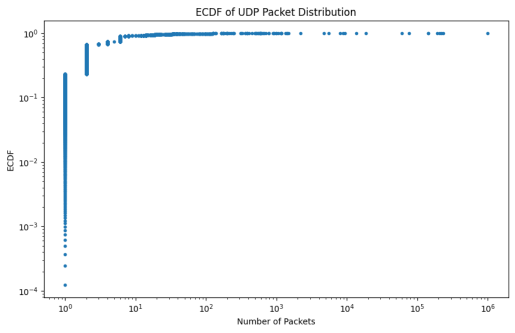


(a) UDP historiogram.

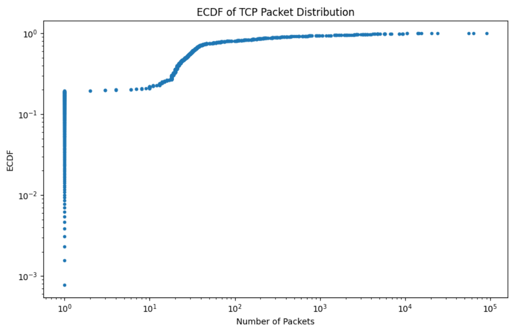


(b) TCP historiogram.

Figure 4: Historiograms for packet length distribution.



(a) UDP ECDF plot.

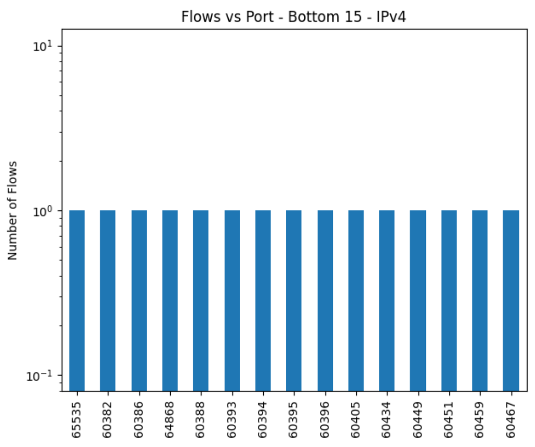


(b) TCP ECDF plot.

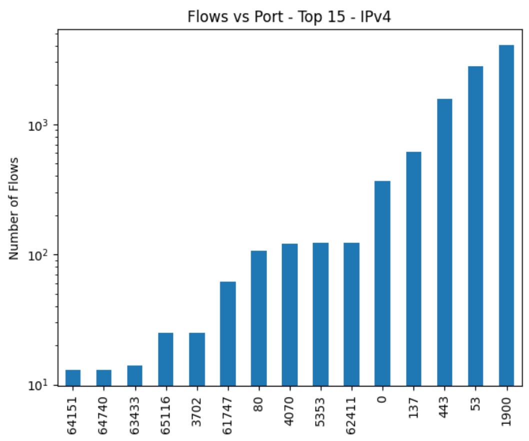
Figure 5: ECDF plots.

1.3.2 Flow data PS2

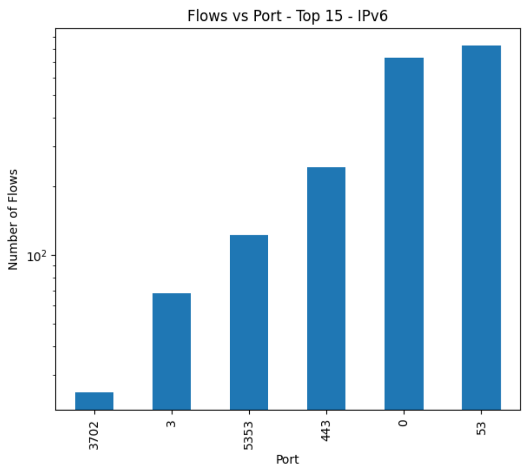
For **PS2**, one will be working with a .t2 file that will have to be treated notably different from PS1. This time, the top 15 and bottom 15 ports are obtained separately by whether they correspond to a IPv4 flow or an IPv6 one. Note, however, that the number of ports used by the latter is not enough to obtain top and bottom 15, so only one such plot is needed. See Figure 6.



(a) Bottom 15, least used, ports by IPv4 flows.



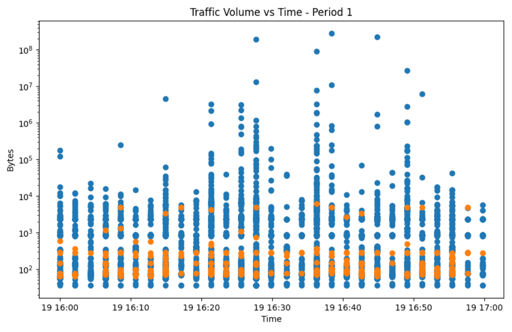
(b) Top 15, most used, ports by IPv4 flows.



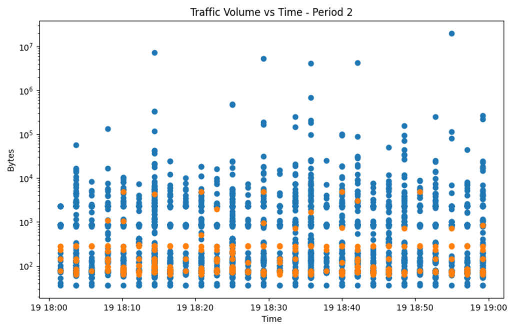
(c) Port usage according to IPv6 flows captured.

Figure 6: Ports against captured flows.

This time, two different time periods have been plotted to showcase the diversity of the captured data. These have been the periods between Coordinated Universal Time (UTC) 16:00 and 17:00; and between UTC 18:00 and 19:00 (Figure 7).



(a) Period between UTC 16:00 and 17:00



(b) Period between UTC 18:00 and 19:00

Figure 7: Traffic volume against time, for two separate (different) time periods.

Having seen that, the goal now shifted to finding the origin of the most popular Internet Protocol (IP)s by number of flows. Using IPLookup [4]; one can easily see the place of origin of a given IP address. After filtering out the most popular ones; and removing the ones that are anonymous and thus cannot be traced to any physical location, the top 5 most popular origins are the ones shown in Table 1. Note that Kansas City, Missouri, appears twice due to it referring to two different servers used by Google.

IP	Number of Flows	ISP	Location	Country of Origin
35.186.224.25	228	Google	Kansas City, Missouri	United States of America
35.186.224.17	204	Google	Kansas City, Missouri	United States of America
2001:14ba:a051:8c00:15a0:90ff:ab54:3867	147	DNA Oyj	Helsinki, Uusimaa	Finland
34.158.0.131	121	Google	Not Available	United States of America
20.199.120.85	62	Microsoft	Paris, Île-de-France	France

Table 1: Top 5 locations by number of flows.

One can now focus instead in IP pairs and their frequency. For this matter, a Zipf plot makes most sense. When plotting both IPv4 and IPv6 connections together, one can easily see the frequency of both types and their differences, though in their own scale both produce similar curves (Figure 8).

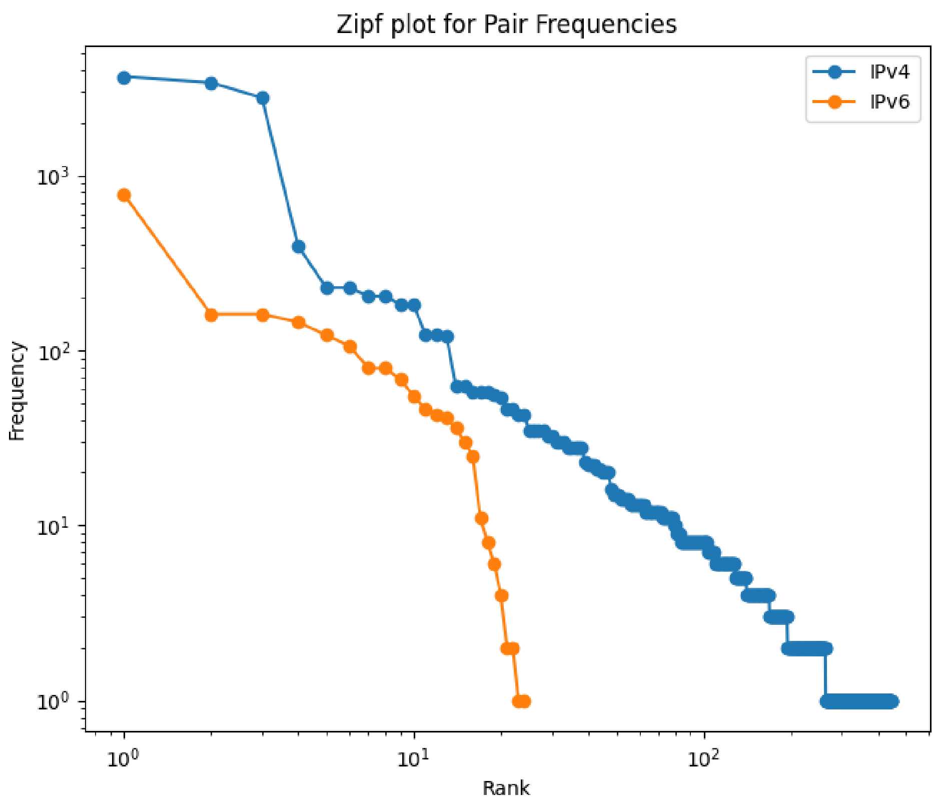


Figure 8: Zipf plot for IP pairs, for both IPv4 and IPv6 connections

Now, relevant statistical parameters and plots can help in painting a clearer picture of the data. The former can be easily consulted in Table 2; which includes the mean, median, Standard Deviation (STD), minimum, and maximum for IPv4 and IPv6 relevant parameters. The latter are shown in Figures 9 and 10, which show flow length distribution and ECDF plots respectively.

	IPv4 Flows		**IPv6 Flows**	
	Flow Length (s)	**Bytes**	**Flow Length (s)**	**Bytes**
Median	218	0	85,0	0
Mean	120.743,86	35,06	215,74	5,66
STD	4.677.312,48	454,97	650,85	29,51
Minimum	36	0	94	0
Maximum	280.519.764	14.592	7.091	256

Table 2: Relevant statistical data for PS2.

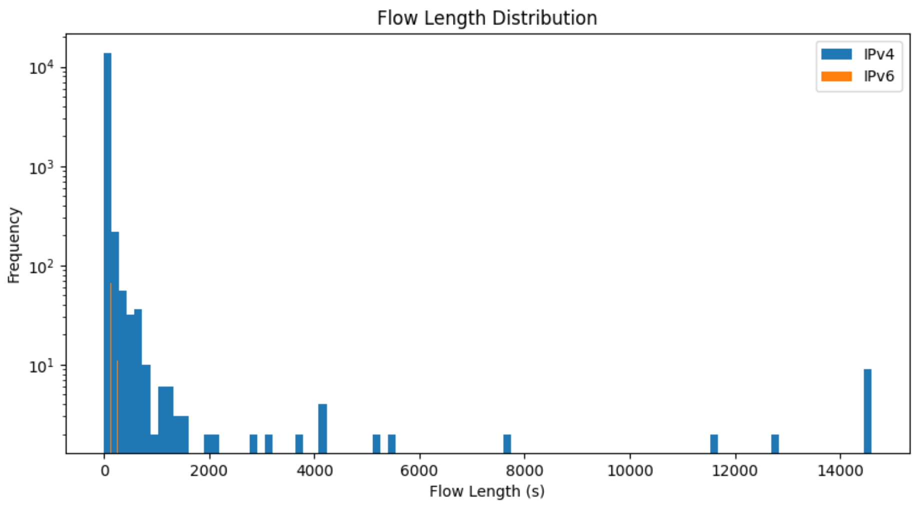
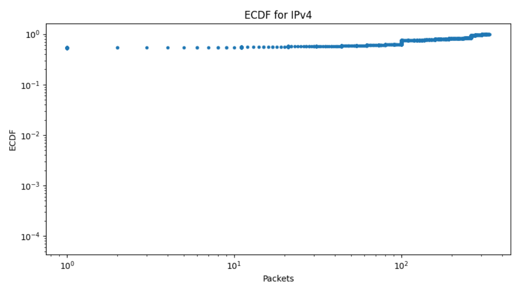
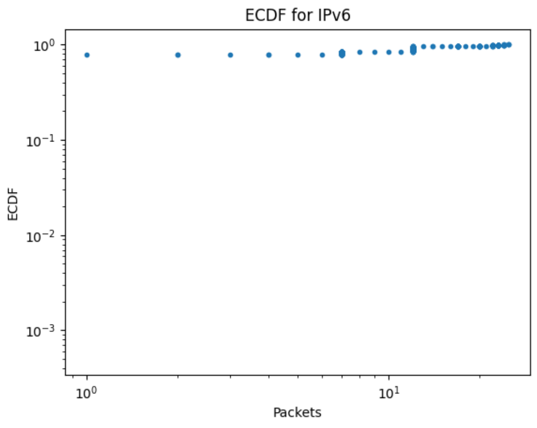


Figure 9: Flow length distribution, in seconds, for both IPv4 and IPv6 packets.



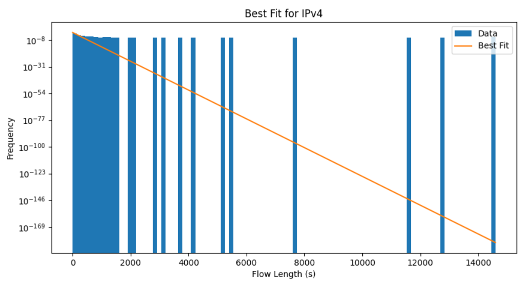
(a) ECDF plot for IPv4 packets.



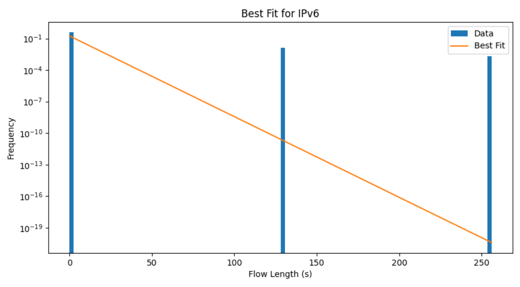
(b) ECDF plot for IPv6 packets.

Figure 10: ECDF plots for PS2.

Focusing on Figure 9; one can be now interested in finding the best statistical distribution to fit that flow length distribution. In the studied case, for both IPv4 and IPv6 the best fitted distribution is an exponential one, though as it can be clearly seen in Figure 11, this kind of distribution loses a lot of accuracy as flow lengths get longer; while providing a very good fit for shorter ones.



(a) Best fit for IPv4 flows.



(b) Best fit for IPv6 flows.

Figure 11: Best fitted distribution for flow length.

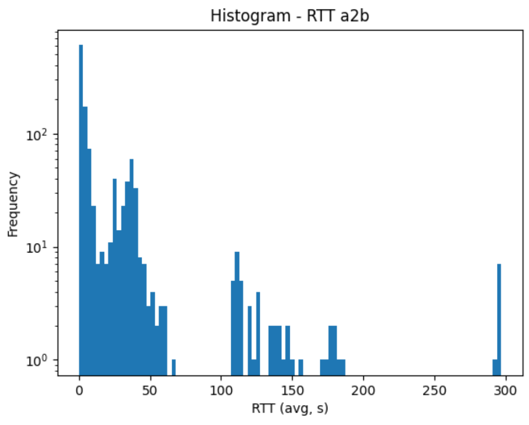
To wrap PS2 up, it is relevant to observe how the conversion from `.pcap` file to flows has affected the data analysed. In this case, the timeout setting has been put on the spotlight. While the used timeout second is 60 s, it can be seen in Table 3 that the number of IPv4 and IPv6 flows can significantly change by altering this setting, ranging from a very high end when using very short timeouts to significantly lower numbers when using longer timeouts.

Timeout (s)	**1**	**10**	**60**	**120**	**1800**
IPv4 flows	52.143	28.961	**14.346**	13.502	12.868
IPv6 flows	2.487	2.329	**2.015**	1.629	1.071

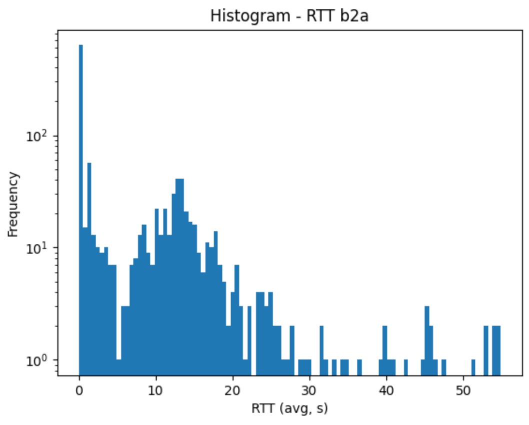
Table 3: Number of flows obtained for each specified timeout.

1.3.3 TCP connection data PS3

When looking at **PS3**, the main focus will necessarily be on Round Trip Time (RTT). For that, it is of interest to see what are the typical values for RTT when transferring data from A to B and vice versa (Figure 12). For extra detail, relevant statistical values of RTT data are included in Table 4.



(a) A to B RTTs.



(b) B to A RTTs.

Figure 12: Historiograms of RTTs.

RTT (ms)	Mean	Median	STD	Minimum	Maximum
A to B	16,398	2,900	35,949	0,0	296,8
B to A	5,997	0,2	9,094	0,0	54,9

Table 4: Relevant statistical parameters for PS3 RTT.

The total volume of data exchanged is available in PS2: 1.732.314.428 bytes for IPv4 connections and 434.566 bytes for IPv6 connections. However, PS3 allows for greater zoom in individual exchanges of information, rather than flows. In using the actual bytes exchanged between A and B, one can plot a new historiogram showcasing the frequency of large and small individual exchanges between pairs (Figure 13).

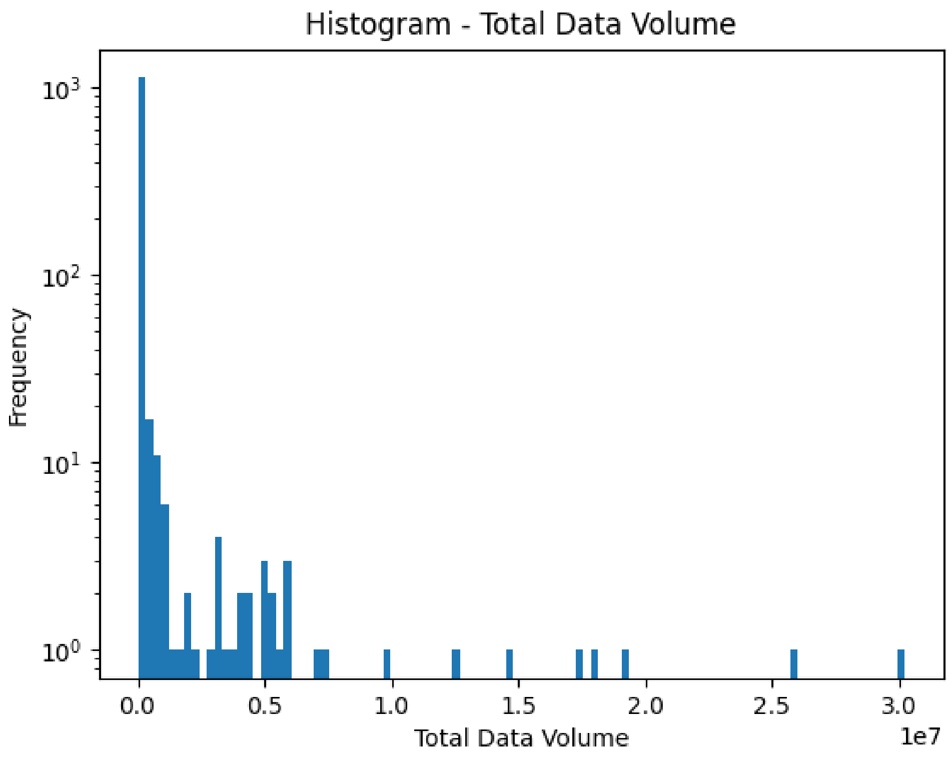


Figure 13: Historiogram: Total Data volume exchanged between A and B and vice versa.

1.3.4 Conclusions

From the traffic volume distribution through time, it is very interesting to see how clear patterns can be identified from the usage the computer got during the capturing of data (Figure 3). The staggering difference between the highest activity period, where files were being downloaded and video was being streamed displays a nigh-and-day difference with the lowest activity period where only music was being streamed, and peaks can even be discerned due to the streaming service fetching more data periodically.

When looking at the most used ports (Figure 6), the potential uses of them can be theorised based on the numbers obtained:

- UDP-based Simple Service Discovery Protocol (port 1900), typically used to discover devices on a local network.
- Domain System Name (DNS) (port 53), used to translate domain names to IP addresses.
- HTTPS (port 443), the secure version of Hypertext Transfer Protocol (HTTP), widely used for web applications.
- NetBIOS Name Service (port 137), used typically to resolve NetBIOS names to IP addresses.
- Initiating client connections (port 0), while typically a reserved port, it is common that the client application uses port 0 to request that the operating system assign an available port number automatically (also known as dynamic port allocation).

From this and the most common locations by number of flows, it is clear that the streaming of video via Google's YouTube is the responsible for the highest activity period and the overusage of ports associated with DNS and secure HTTP. Similarly, the low activity period, where music was being streamed via Spotify, is probably the responsible for the usage of port 1900, for Spotify regularly checks local networks for other devices where the user might want to stream their music other than the active application.

Task 2: Flow Data

2.1 Acquiring flow data

From the whole data set provided, the continuous flow files have been chosen as **FS1**. From there, and according to the final digit of the student's number, only the subnetworks starting with 163.35.116. will be of interest.

2.2 Data pre-processing

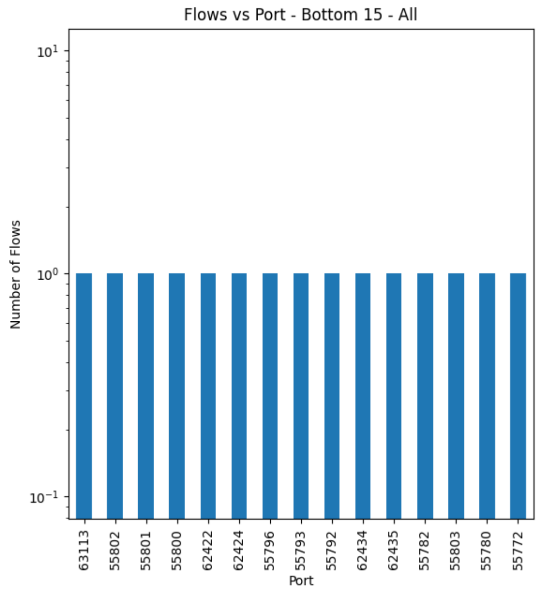
FS2 has been obtained by applying a gawk-type filter to FS1, to only have the flows corresponding to the addresses included in the filter. The proposed method can be consulted in Appendix B; where also other employed methods to then use Python's [3] pandas [5] library for data clean-up are discussed.

2.3 Data Analysis

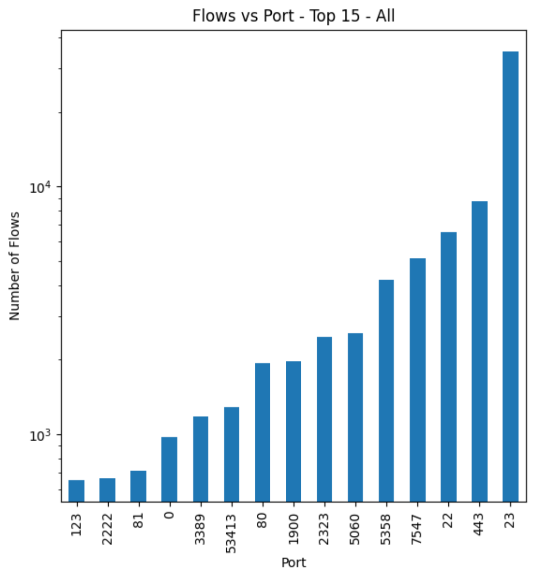
All instructions and relevant technical details stemming from the process of obtaining the results in this section can be found in Appendix B.

2.3.1 Plot traffic volume

For traffic volume, the points 4 and 5 of the previous tasks will be repeated, again, now for this data set. The most and least used ports in the obtained **FS2** data set can be consulted in Figure 14.



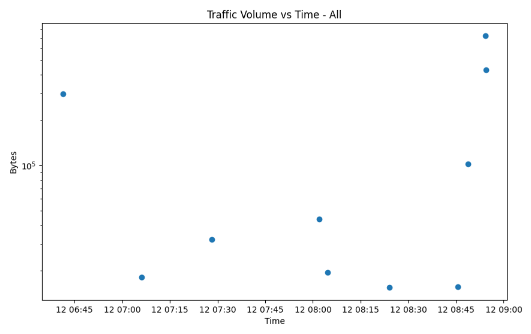
(a) Bottom 15, least used, ports.



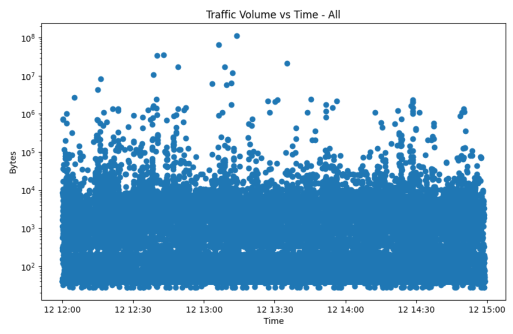
(b) Top 15, most used, ports.

Figure 14: Most and least used ports of FS2.

Traffic volume as a function of time has been visualised in two very distinct time periods. In a very-low activity period between 6 and 9 am on one hand; and in an extremely-high activity period between 12 and 15 hours. See Figure 15.



(a) Between 6:00 and 9:00.



(b) Between 12:00 and 15:00.

Figure 15: Bytes transmitted against time, during two different time periods (FS2).

2.3.2 Per user data volume

For this section, the main focus is to obtain the aggregate data volume for each user. This analysis has been done, and from it a historiogram has been drawn (Figure 16). As it is usual, this is a case of a long-tail, where the vast majority of the data is actually moved by very few users.

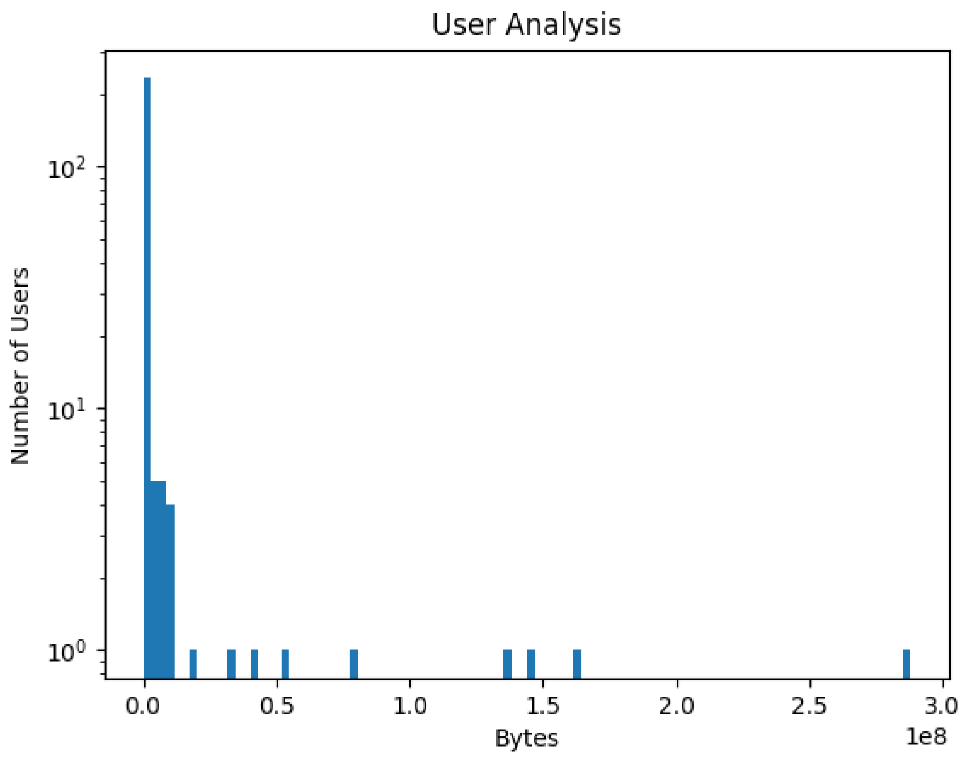
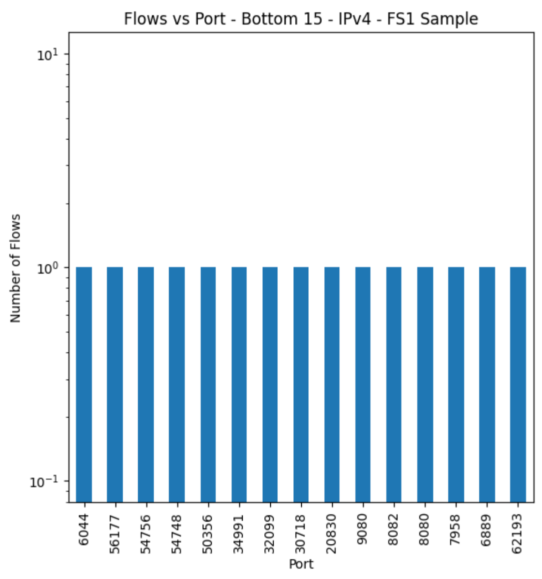


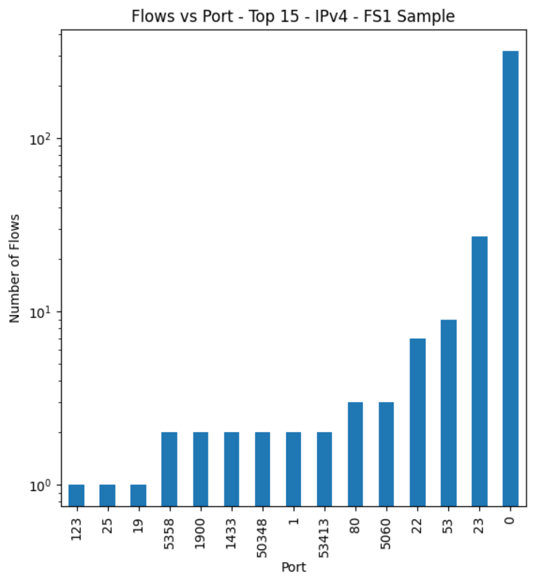
Figure 16: Historiogram of data transmitted per user.

2.3.3 Flow sampling

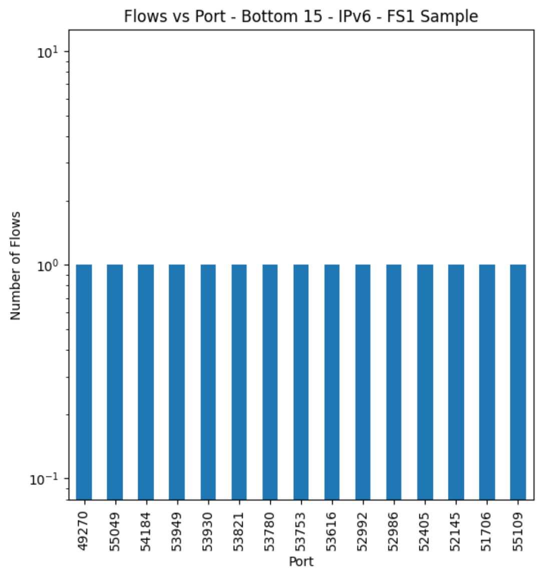
Now the whole dataset of **FS1** is being considered. However, due to its sheer size, only a sample will be analysed. For each hourly file, 50 random flows corresponding to IPv4 and IPv6 flows respectively have been sampled. This corresponds to a total file size quite similar to the size of **FS2**. Then, the same methods to plot traffic volumes have been applied, with the most and least used ports available in Figure 17; and the data flows against time can be consulted in Figure 18.



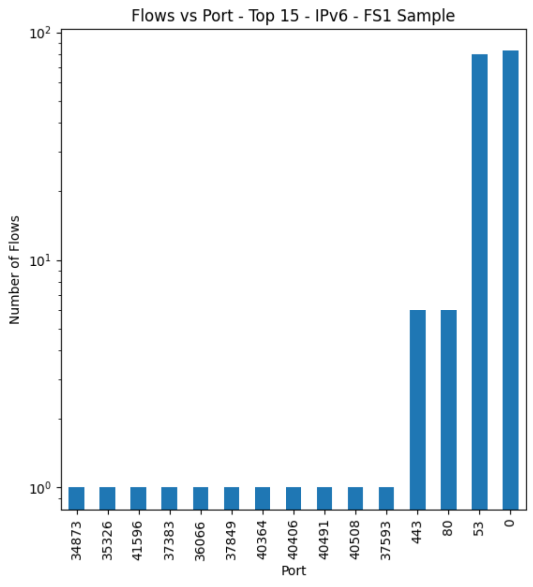
(a) Bottom 15, least used, ports - IPv4.



(b) Top 15, most used, ports - IPv4.

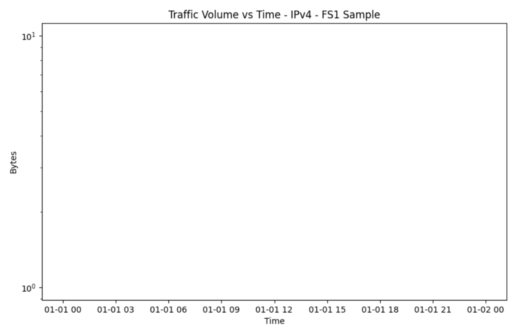


(c) Bottom 15, least used, ports - IPv6.

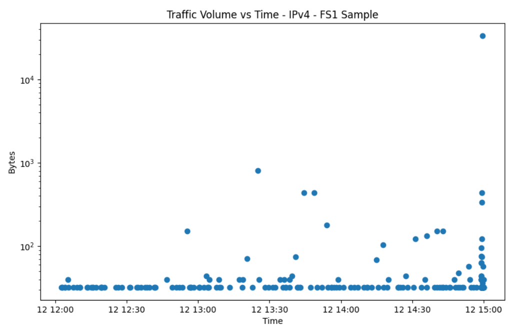


(d) Top 15, most used, ports - IPv6.

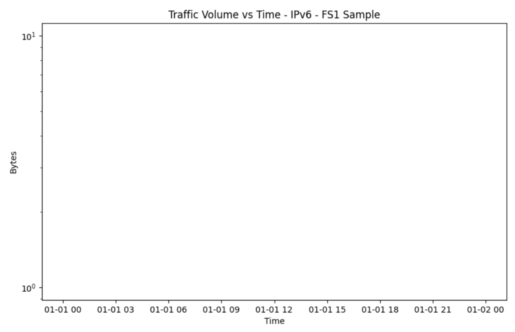
Figure 17: Most and least used ports of the sampled FS1.



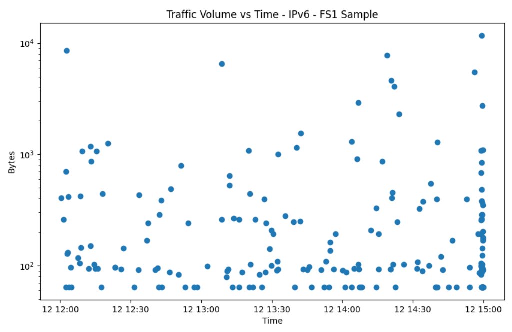
(a) Between 6:00 and 9:00 - IPv4.



(b) Between 12:00 and 15:00 - IPv4.



(c) Between 6:00 and 9:00 - IPv6.



(d) Between 12:00 and 15:00 - IPv6.

Figure 18: Bytes transmitted against time, during two different time periods (FS1 sample).

From all of this, what becomes clear is that the subnetwork is representative enough of the whole data set. It is clear that the most and least used ports follow a similar pattern and, notwithstanding some changes in order of preference for some ports, they do correlate pretty decently. The high and low periods of activity are, for all intents and purposes, the same; and no significant differences in correlation seem to happen between IPv4 and IPv6.

2.3.4 Conclusions

All in all, the traffic volume analysed displays very identifiable patterns for seasonality. The period in the early morning, between 6 and 9, clearly very little activity (notably with absolutely no flows in the sampled data from FS1); while the period in the early afternoon from 12 to 15 displays high levels of activity.

Looking at the most used ports for AS2, the potentially most used applications are:

- **Telnet** (port 23), an "insecure" protocol that allows for remote access and control of a computer.
- **HTTPS** (port 443), the secure version of HTTP.
- **SSH** (port 22), a "secure" protocol that allows for remote access and control of a computer.
- **SAP** NetWeaver Application Server (port 7547), an enterprise resource planning software suite.
- **WSD** (port 5358), a network protocol that allows devices to discover and communicate with each other over a network.

From this port usage, it seems that there is a lot of internet administration work being done by them, due to the high usage of ports 23 and 22. There seems to be also a lot of remote access to other machines due to the other ports used, which also hints to the direction of internet administration or remote work. As said before, this user is somewhat representative of the whole subnetwork, for ports 22 and 23 are also quite common in the sampled data, though they do over-represent port 443 when compared to the sample.

On the complete other end of the spectrum, there is the comparison that can be made with PS2. When revisiting figure 6, it is clear that the most used ports are nothing alike; with the only common one being port 443, arguably one of the most popular ones due to its extensive usage for web browsing.

Task 3: Analysing active measurements

In this task, the data collected during two weeks has been analysed. The data was collected between October, 1st and 16th, 2023; again from a desktop computer located in Leppävara, Espoo; connected to the internet via Wi-Fi, with DNA as the ISP. This raw data has been processed into different data frames according to the method used and the server proved. For latency measurements, there exists 11 subsets of **AS1**:

- **Nameservers** (located in the French Southern and Antarctic Lands)
  - **AS1.d1**: e.ext.nic.fr, using ping.
  - **AS1.d2**: g.ext.nic.fr, using ping.
  - **AS1.d3**: f.ext.nic.fr, using ping.
  - **AS1.n1**: e.ext.nic.fr, using curl.
  - **AS1.n2**: g.ext.nic.fr, using curl.
  - **AS1.n3**: f.ext.nic.fr, using curl.
- **Research Servers**
  - **AS1.r1**: bcn (Barcelona), using ping.
  - **AS1.r2**: mnl (Manila), using ping.
  - **AS1.r3**: hnl (Honolulu), using ping.
- **Iperf Servers**
  - **AS1.i1**: ok1, using ping.
  - **AS1.i2**: sgp1, using ping.

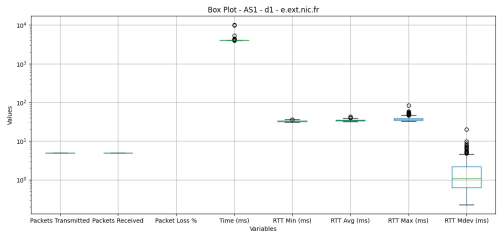
On the flipside, the throughputs measurements include two subsets of **AS2**:

- **AS2.i1**: ok1, iperf server.
- **AS1.i2**. sgp1, iperf server.

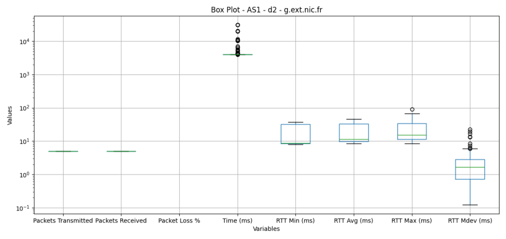
Note that, as per usual, every code and its implementation needed for this task has been discussed in-depth in Appendix C, from the data treatment to its analysis and the obtaining of key results and figures.

3.1 Latency data plots (AS1.x)

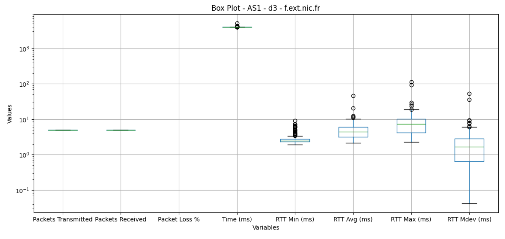
By filtering out the data with packet loss out of every **AS1.x** subset, the first insight of relevance that has been mustered is the box plots corresponding to every variable of relevance of every subset of data (Figure 19). This can be then compared to the same box plots, but now considering also the instances with packet loss (Figure 20). In most instances, it is clear that the packet loss is not significant enough to really make a dent into the rest of the values. It also highlights how the methodology worked correctly for considering packet loss when the data was not available (subsets AS1.nX), where packet loss was considered for total latency times higher than one second. The very few instances of packet loss observed in the subsets AS1.dX reflect the very few instances of packet loss in AS1.nX, hinting that this was indeed a good assumption. Other than that, it is true that they overall seem to be methods that display a lot more variability in the data. When looking at the box plots of AS1.n1 and AS1.n2 total latency and time start transfer, the size of the quartiles is remarkably larger than the RTT values of the equivalent AS1.d1 AS1.d2.



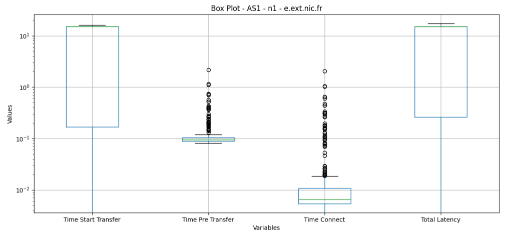
(a) AS1.d1.



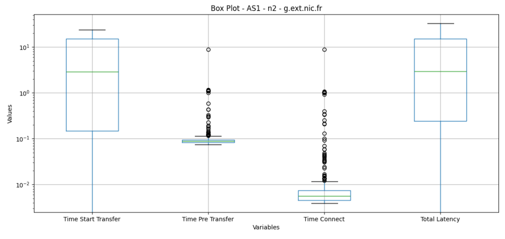
(b) AS1.d2.



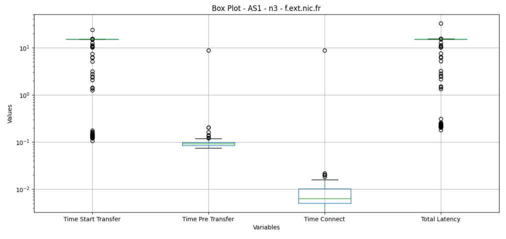
(c) AS1.d3.



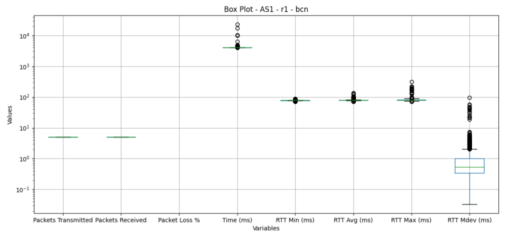
(d) AS1.n1.



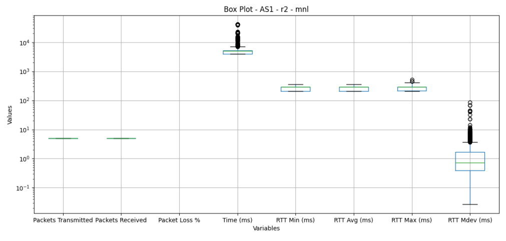
(e) AS1.n2.



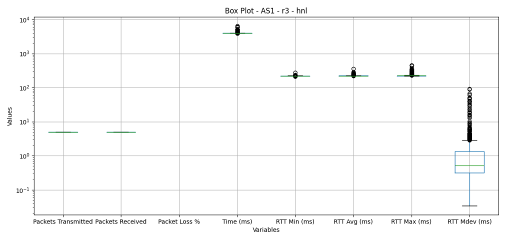
(f) AS1.n3.



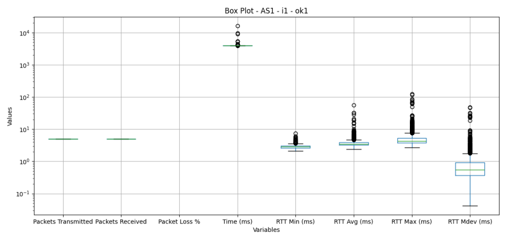
(g) AS1.r1



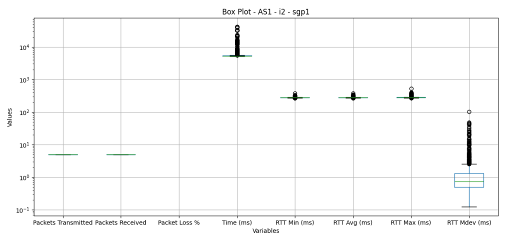
(h) AS1.r2.



(i) AS1.r3.

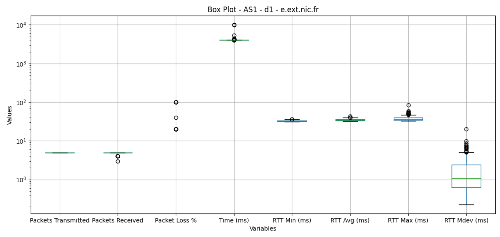


(j) AS1.i1.

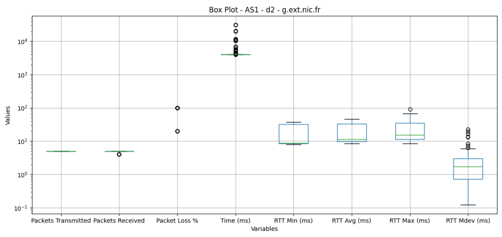


(k) AS1.i2.

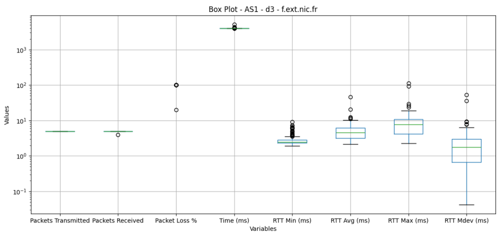
Figure 19: Box plots without considering packet loss.



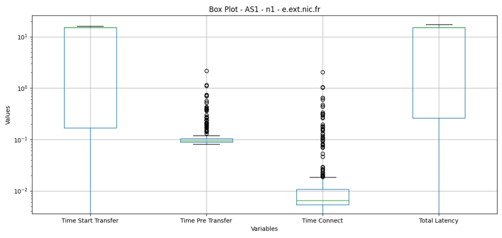
(a) AS1.d1.



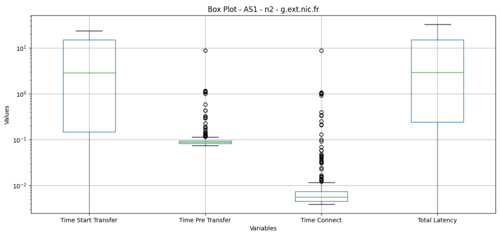
(b) AS1.d2.



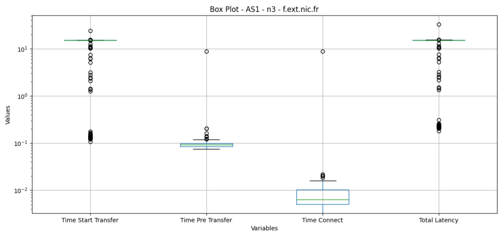
(c) AS1.d3.



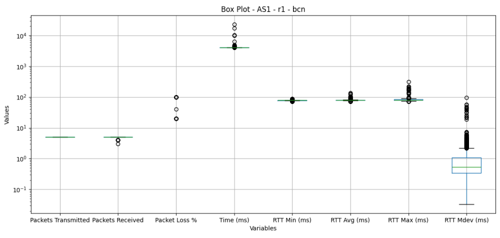
(d) AS1.n1.



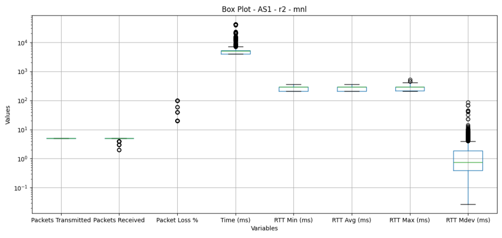
(e) AS1.n2.



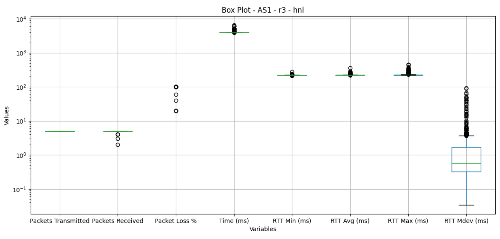
(f) AS1.n3.



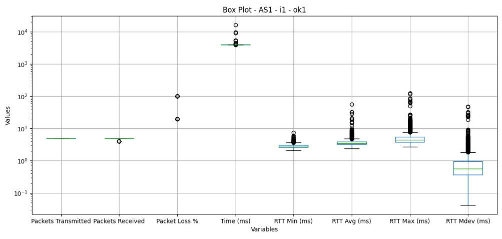
(g) AS1.r1



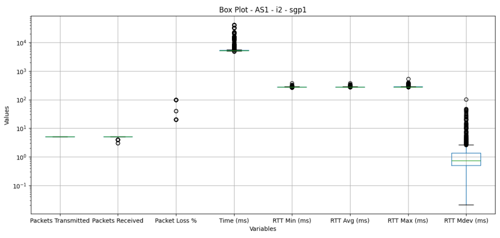
(h) AS1.r2.



(i) AS1.r3.



(j) AS1.i1.



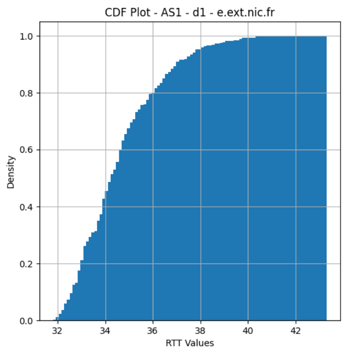
(k) AS1.i2.

Figure 20: Box plots now considering packet loss.

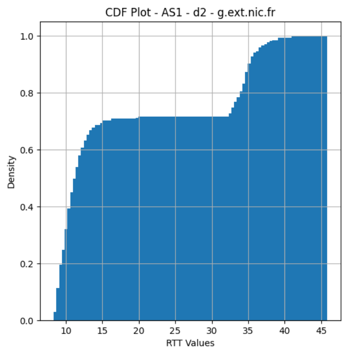
For further analysis, it is of interest to focus on the average RTT and Total Latency values, and see what kind of Cumulative Distribution Function (CDF) and Probabiltiy Density Function (PDF) they display. For that, see Figure 21 and 22. In a similar fashion, a summary of relevant statistical parameters has been included in Table 5.

Dataset	**Server**	**First delay (ms)**	**Mean (ms)**	**Median (ms)**	**Values above 1.5mean (%)**	**Distance between 95th and 50th percentile**
AS1.d1	e.ext.nic.fr	34.01	34.613	34.265	0.0	3.608
AS1.d2	g.ext.nic.fr	32.731	17.714	11.342	28.481	25.108
AS1.d3	f.ext.nic.fr	2.734	4.985	4.398	11.218	4.189
AS1.n1	e.ext.nic.fr	15.287	8.107	15.249	50.920	0.243
AS1.d2	g.ext.nic.fr	15.255	7.605	2.979	46.626	12.282
AS1.d3	f.ext.nic.fr	15.244	12.853	15.241	0.307	0.0458
AS1.r1	bcn	72.633	79.923	79.055	0.155	5.5206
AS1.r2	mnl	291.234	260.513	291.051	0.0	7.381
AS1.r3	hnl	222.787	222.897	222.229	0.054	5.546
AS1.i1	ok1	3.808	3.849	3.495	4.595	2.208
AS1.i2	sgp1	278.95	279.944	279.117	0.0	4.966

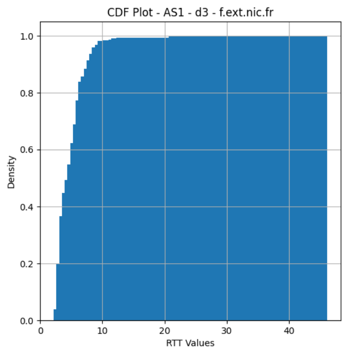
Table 5: Summary table for AS1.x data sets.



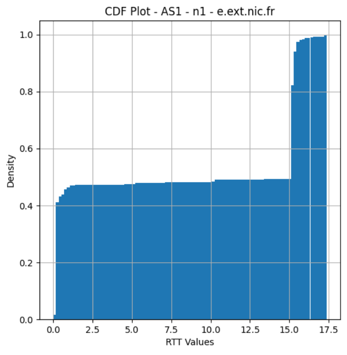
(a) AS1.d1.



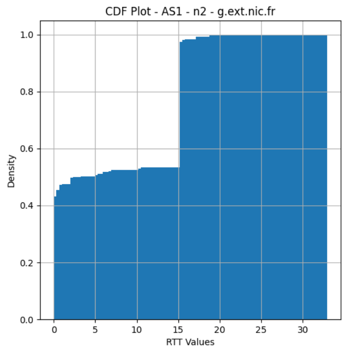
(b) AS1.d2.



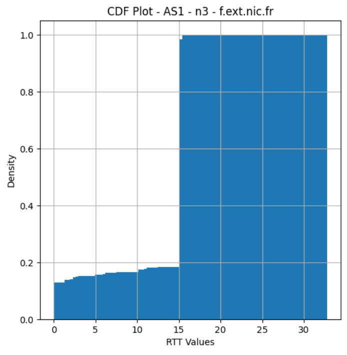
(c) AS1.d3.



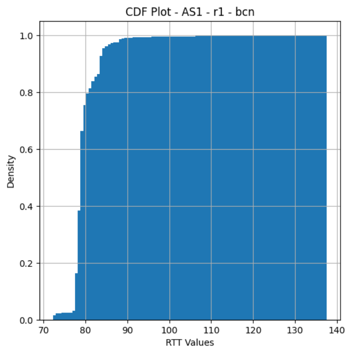
(d) AS1.n1.



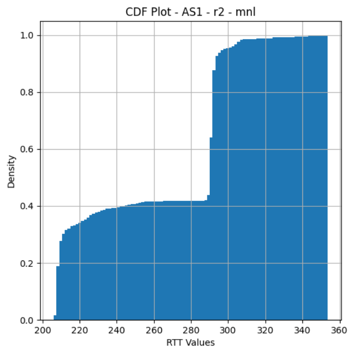
(e) AS1.n2.



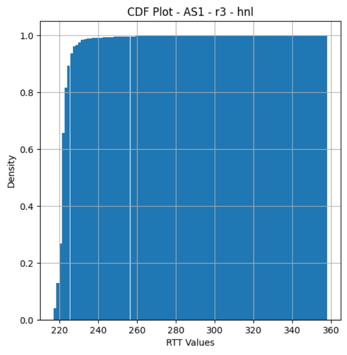
(f) AS1.n3.



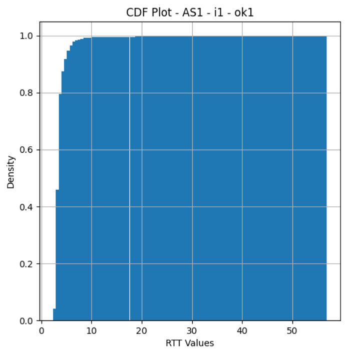
(g) AS1.r1



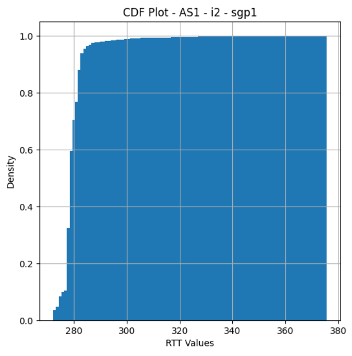
(h) AS1.r2.



(i) AS1.r3.

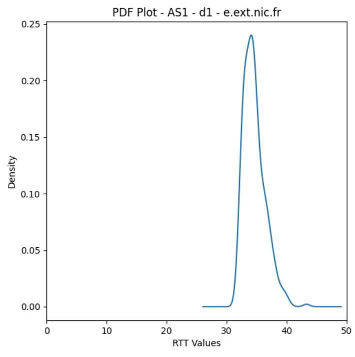


(j) AS1.i1.

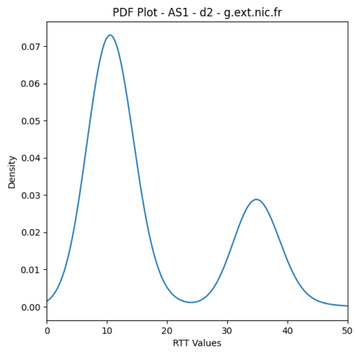


(k) AS1.i2.

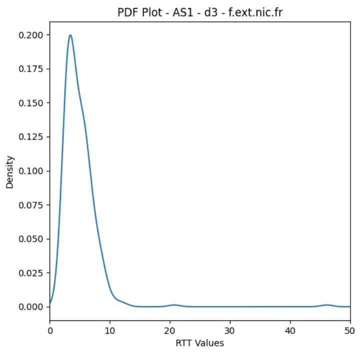
Figure 21: CDF plots for AS1.



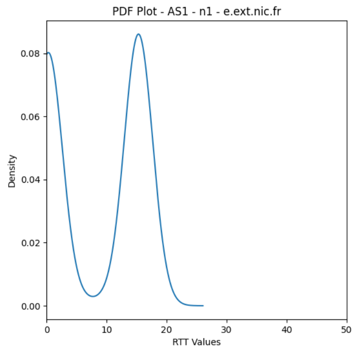
(a) AS1.d1.



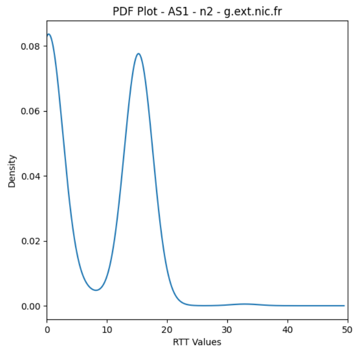
(b) AS1.d2.



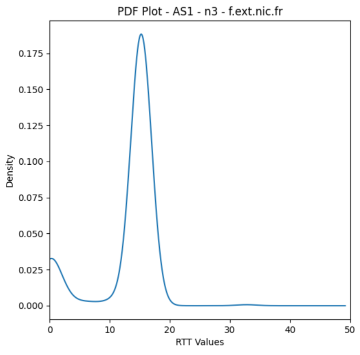
(c) AS1.d3.



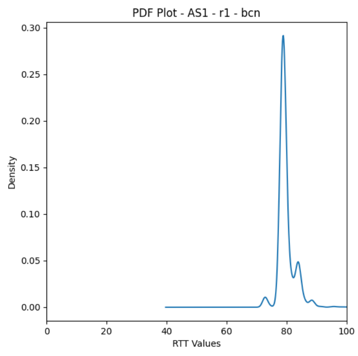
(d) AS1.n1.



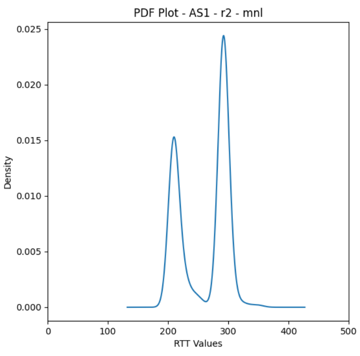
(e) AS1.n2.



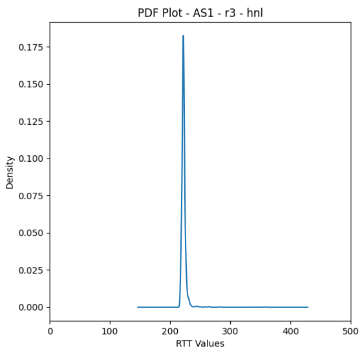
(f) AS1.n3.



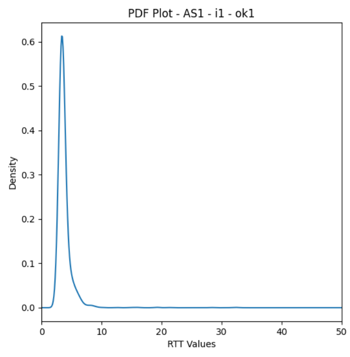
(g) AS1.r1



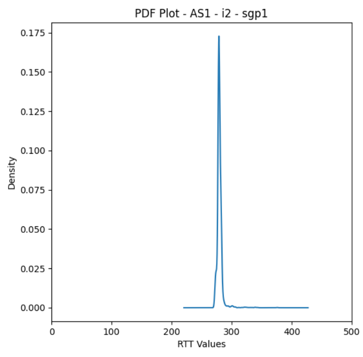
(h) AS1.r2.



(i) AS1.r3.



(j) AS1.i1.



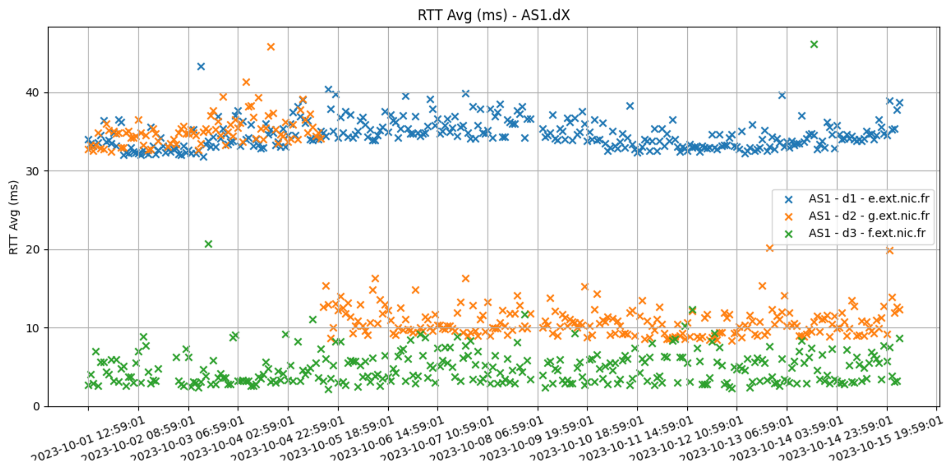
(k) AS1.i2.

Figure 22: PDF plots for AS1.

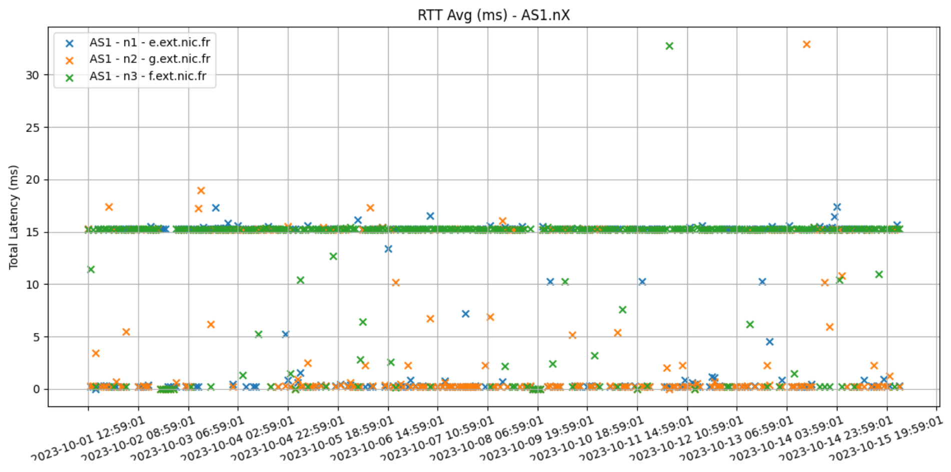
3.2 Latency data time series

Still working with AS1.x data sets, it is of relevance to plot the measured latency values for the analysed time period (Figures 23 and 24). Despite the long duration of the measuring, there seems to be no noticeable variability, day or night cycle, or even weekly events. However, and very interestingly, some servers seem to have received upgrades during the measuring period. This seems to be the case for g.ext.nic.fr (very noticeable in AS1.d2, but not so much in AS1.n2); and mnl (clearly seen in AS1.r2, where in the latter half of the period it outperforms hnl).

Finally, autocorrelation plots of AS1.d1, as1.r1, AS1.i1, and AS1. i2 can be consulted in Figure 25; where the stability observed in the AS1.iX subsets can be confirmed very clearly. On the flip side, the waving nature of AS1.d1 is exaggerated and can be further understood in its autocorrelation plot; while the perceived stability of the AS1.r1 subset can actually be seen with fluctuations that seem to be at least somewhat more periodic than initially assumed.

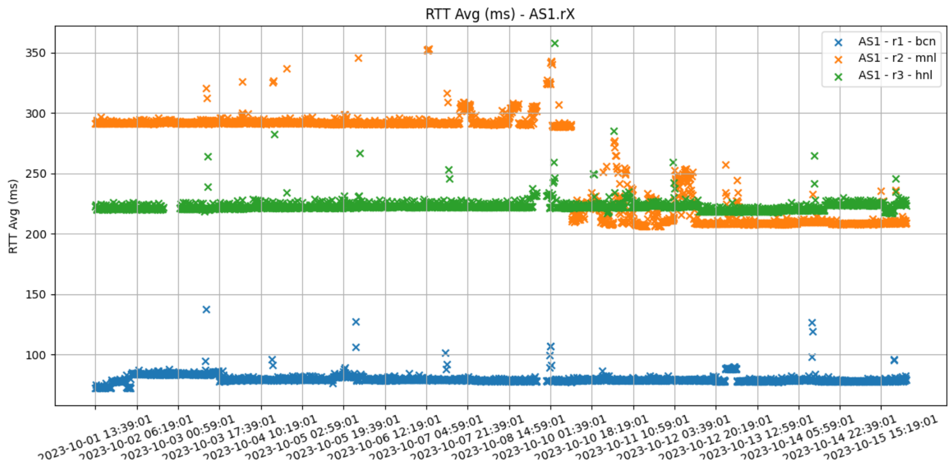


(a) AS1.dX.

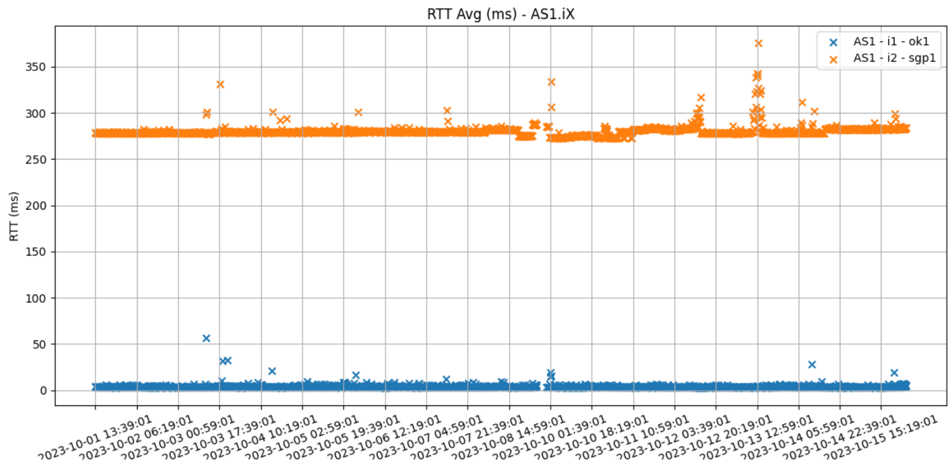


(b) AS1.nX.

Figure 23: Timeseries plots for AS1.dX and AS1.nX.

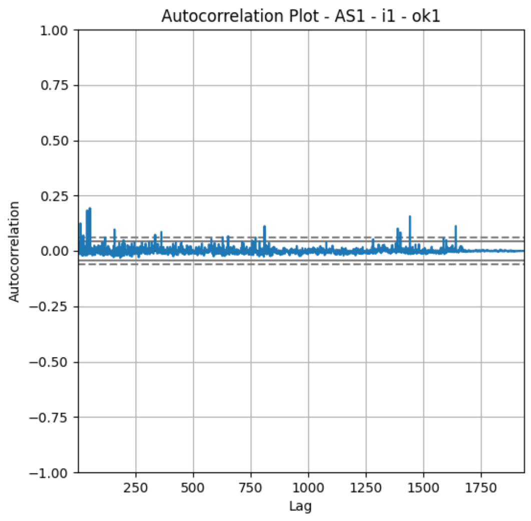


(a) AS1.rX.

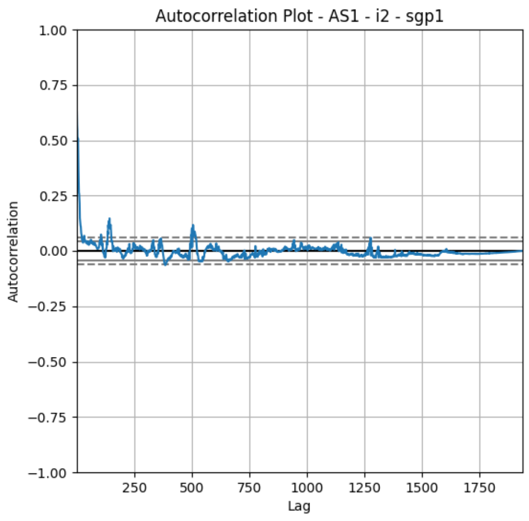


(b) AS1.iX.

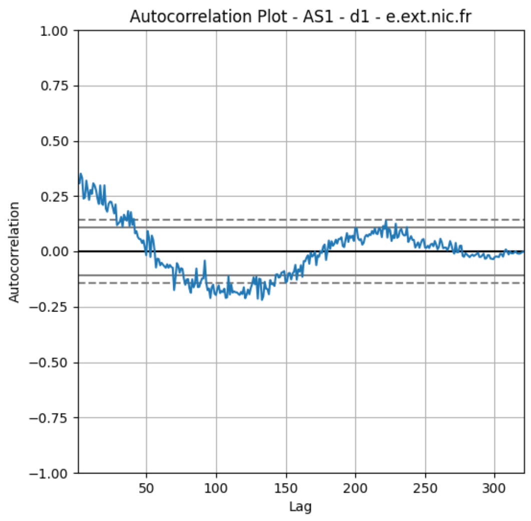
Figure 24: Timeseries plots for AS1.rX and AS1.iX.



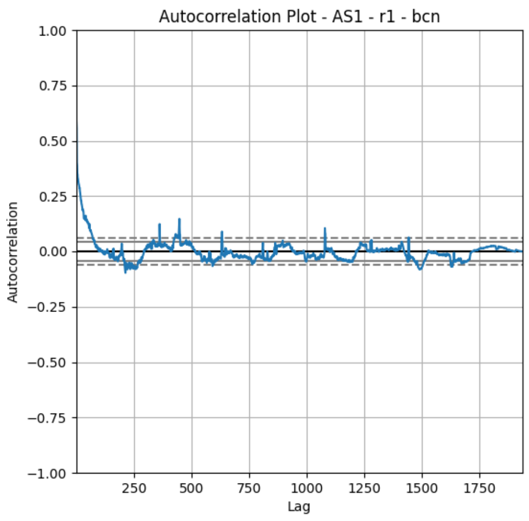
(a) AS1.i1.



(b) AS1.i2.



(c) AS1.d1.



(d) AS1.r1.

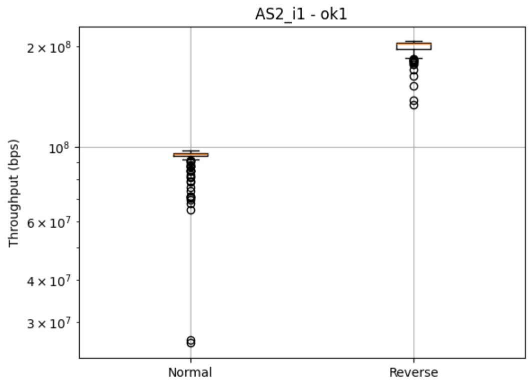
Figure 25: Autocorrelation plots for AS1.

3.3 Throughput

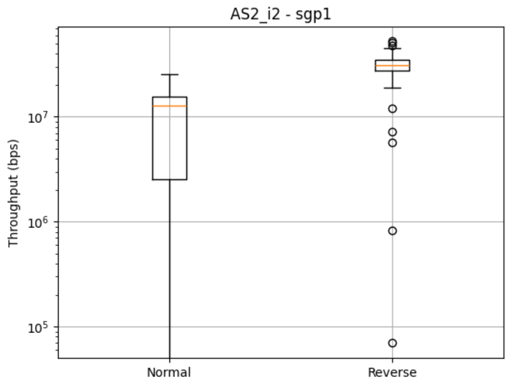
Shifting now the attention to the **AS2** dataset, focused on throughput data, it is of interest to compute relevant statistical parameters that can help in making sense of the data obtained. These are available in Table 6. Similarly, the box plots for AS2.i1 and AS2.i2 are available in Figure 26, with the two possible modes of operation differentiated.

Dataset	Mode	Server	Mean (kbps)	Harmonic mean (kbps)	Geometric mean (kbps)	Median (kbps)
AS2.i1	Normal	ok1	93912.680	92460.035	93420.343	95694.012
AS2.i1	Reverse	ok1	200882.926	200367.011	200643.422	204721.134
AS2.i2	Normal	sgp1	11586.501	4382.480	8380.677	13381.061
AS2.i2	Reverse	sgp1	31716.775	12041.718	30300.799	31861.672

Table 6: Summary table for AS2.x data sets.



(a) AS2.i1.



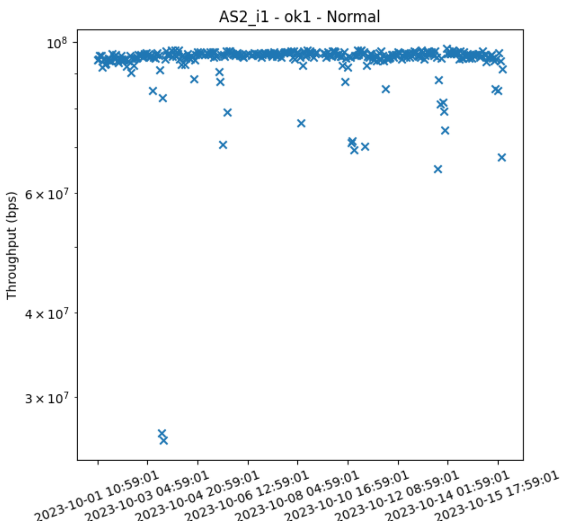
(b) AS2.i2.

Figure 26: Box plots for the throughput measurements from the AS2 dataset.

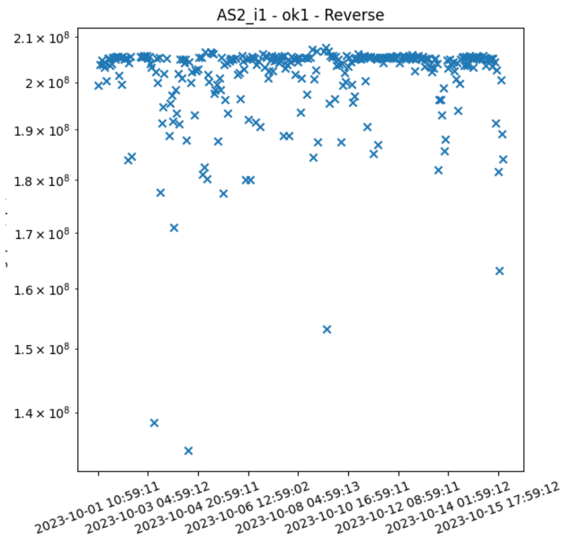
3.4 Throughput time series

The time series for the throughput values analysed from AS2.x are available in Figure 27. In it, it is clear that the most stable connection is achieved with ok1; with remarkable few outliers in the normal mode of operations and a bit more variability (yet still, stable) for the reverse mode. This general trend is maintained in sgp1 for Reverse operations, where despite the slower operations the stability is still as good as it can realistically get. It is in the Normal mode of operations, though, where almost total chaos happens. The connection seems to be notably unstable, and only two general trends can be barely drawn: connections in the order of 10^7 bps and connection in the order of 10^6 bps, with no noticeable seasonality nor pattern to it.

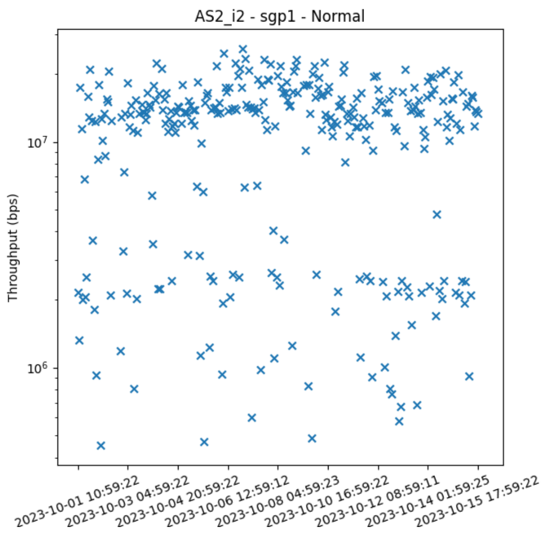
This analysis can be brought a step further by plotting autocorrelation plots of the same data (Figure 28). Where, predictively, the far less predictability observed in normal operations in AS2.i2 results in an autocorrelation plot that tends a lot more towards the 0 value. Interestingly, however, there seems to be a lot less - generally - autocorrelation in this data than the autocorrelation plots obtained before, in Figure 25. AS2.i1 autocorrelation plots showcase somewhat superior autocorrelation values than those in AS1.i1, while the comparison between AS1.i2 and AS1.i2 happens the other way around, with the throughput measurements giving somewhat less autocorrelation.



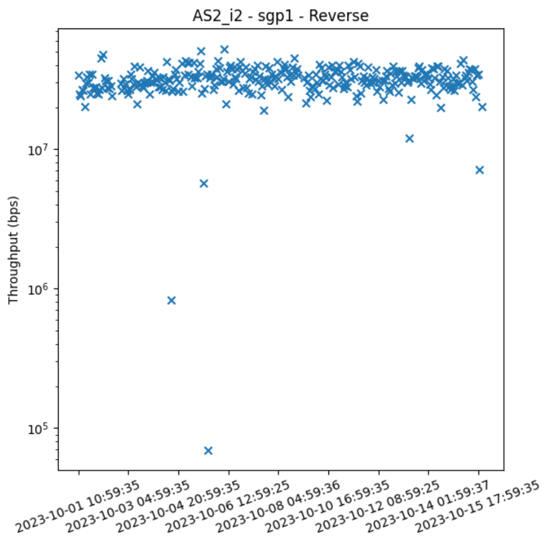
(a) AS2.i1 - Normal mode.



(b) AS2.i1 - Reverse mode.

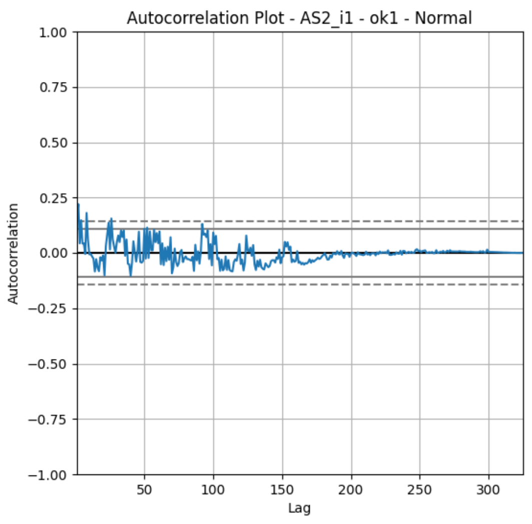


(c) AS2.i2 - Normal mode.

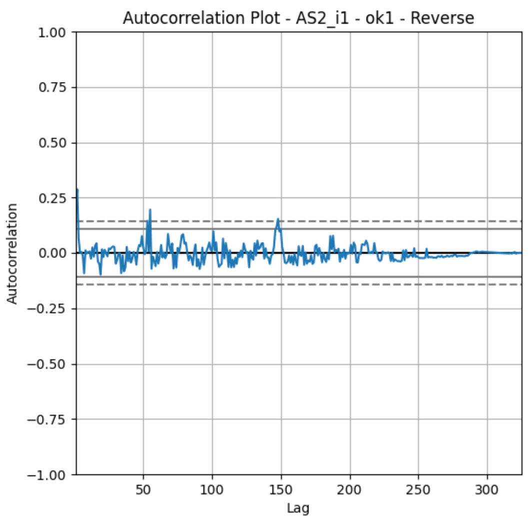


(d) AS2.i2 - Reverse mode.

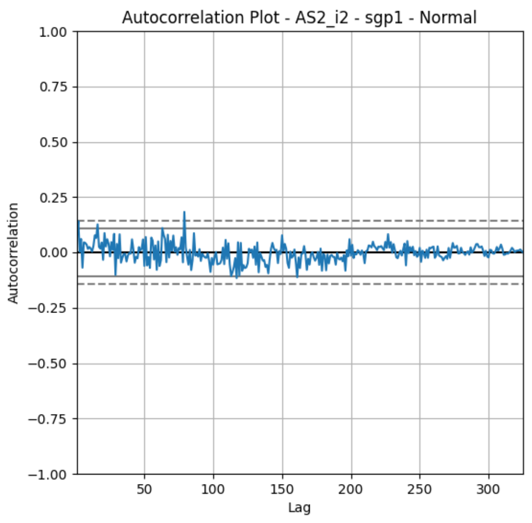
Figure 27: Time series for the throughput measurements from the AS2 dataset.



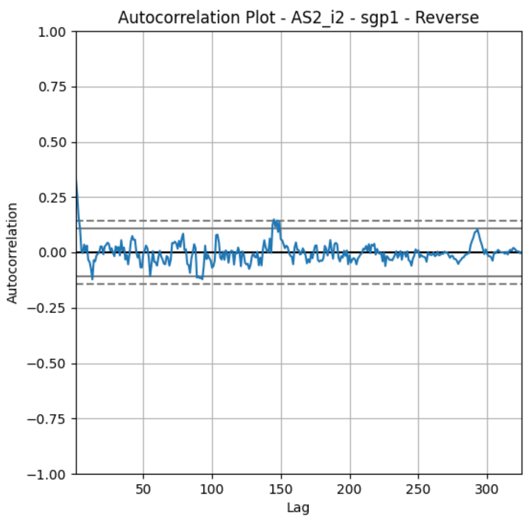
(a) AS2.i1 - Normal mode.



(b) AS2.i1 - Reverse mode.



(c) AS2.i2 - Normal mode.



(d) AS2.i2 - Reverse mode.

Figure 28: Autocorrelation plots for the throughput measurements from AS2.

3.5 Conclusions

The system used for the measurements, as described before, is a desktop computer. This means that typical issues with measurements like having to move the laptop around or getting the connection cut were very rare, as it always stayed in place and with reliable connection. The true challenge was adapting to the way the data was formatted. Since it used the same encoding as used in Assignment 2, its suboptimal form was never corrected and, thus, the same convoluted decoding process was needed to get access to the data for processing.

When looking at the data, it is of interest to see how distance (in jumps) does seem to correlate with time; though changes or upgrades in some servers (as discussed previously), or in the ISP that serves them, seem to have improved this for both g.ext.nic.fr and mnl. Regarding throughput, it is clear some correlation seems to exist between it and latency. The iperf servers analysed clearly display a difference in throughput of an order of magnitude between the two (ok1 versus sgp1). This is in turn also the case for the latency observed, where the connections to sgp1 did take around an order of magnitude more to reach the server when compared with ok1.

Final Conclusions

After a full in-depth analysis of the different data sets, key proposed questions are answered as closing conclusions.

How was your own traffic (Task 1) different from the data provided (Task 2)? What kind of differences can you identify? What could be a reason for that?
The key difference between recorded data in Task 1 and provided data in Task 2, as has been seen in the different conclusions, is the network usage the different users had. While in Task 1 the user used the network for web browsing, video, and music streaming; the user in Task 2 seemed to perform tasks much more closely related to backend, remote machine connections, and network administration. Nonetheless, there were also strong similarities: both data displayed very strong seasonality, with periods of remarkably high and low traffic volume. Plus, as it is usual in these analyses, there is a very strong long tail behaviour for the historigrams on data volume per flow.

Comparing RTT latency about TCP connections (3.1), were active latency measurements around the same magnitude or was another much larger than the other?
When looking at the two-week-long data gathered (Task 3), it is clear that there are very different behaviours for the wide array of servers proved. Geographically (and network-wise) further away servers naturally displayed higher RTT values. Interestingly, the name servers located in the French Southern and Antarctic Lands, despite the geographical distance, displayed latency values comparable to those of the research server in Barcelona. This is probably due to both being, at the end of the day, servers located in European countries, which explains why the network-wise distance is lower than that of Manila or Honolulu (which displayed RTT values an order of magnitude higher).

Discuss how data protection needs to be taken into account if you as a network provider employee were doing similar measurements as in this assignment in a network provider network (traffic generated by customers that may be private persons or companies).
Data protection is paramount, no matter the context, when network measurement work is being done. Nonetheless, this becomes a particularly heavy concern when dealing with traffic generated by diverse customers, including private individuals and companies. As an ISP employee (and as an ISP in general), it is essential to uphold stringent privacy standards and adhere to relevant data protection regulations. Even without the existence of these standards and regulations, however, just for ethic usage of these measurement tools, measures such as IP address anonymization should be implemented. This involves masking or truncating portions of IP addresses to prevent the direct identification of individuals or entities. Additionally, aggregated and anonymized data should be prioritized for analysis to mitigate the risk of unintentional disclosure. Of course, this is added to the obvious company-wide audits and verification processes that need to happen regularly to ensure adherence to good practices and regulations.

Discuss how data protection needs to be taken into account if you as a company Information and Communication Technologies (ICT) support group employee were doing similar measurements as in this assignment in a company network (traffic generated by employees and customers).
Similarly to the previous scenario, as an employee within a company's ICT support group, stringent adherence to data protection principles is imperative. Again, to safeguard privacy, it is crucial to prioritize the anonymization of sensitive information; yet again, employing techniques such as IP anonymization or aggregation to helps prevent the direct identification of individuals or groups. Additionally, explicit consent mechanisms should be in place, ensuring that employees and customers are informed about the nature of the measurements and their potential impact on privacy. Access to the collected data should be restricted to authorized personnel, and robust security measures must

be implemented to safeguard against unauthorized access. This is all, and repeating the answer previously given, added to the necessary regular privacy assessments and compliance checks should be conducted to ensure ongoing alignment with data protection regulations.

How do you rate the complexity of different tasks? Were some tasks more difficult or laborious than others? Did data volume cause any issues with your analysis?
Personally, as someone with moderate experience with Python and its wide range of libraries; and little experience in working with Linux-based systems; I have to say the tasks were not particularly difficult to address. Rather, they were very much time-consuming - specially Task 1 and 3. Even when reusing functions with just slight tweaks, it was a tedious and repetitive process that was rarely satisfying. Probably the same learning outcomes could be achieved without this very disheartening feeling of doing everything many times in a row (first throughout the many assignments, and now several times in this final assignment).

Regarding data volumes, it was very interesting to work around the limitations it provided. Directly treating the data provided for Task 2, for instance, using Python, was basically unworkable with a normal computer. This, in turn, forced the student to work around the problem and use more efficient methods for the task, like the gwak command described in Appendix B. Overall, this concrete experience should be rated positively, for stepping outside of one's comfort zone is always needed and rewarding.

References

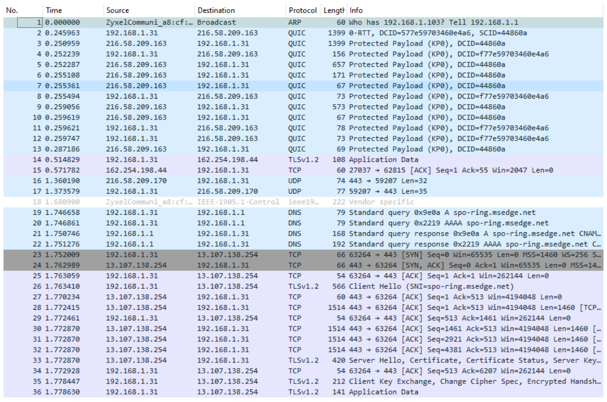
1. *GitHub Copilot* [online]. GitHub, Inc, 2023 [visited on 2023-11-25]. Available from: https://github.com/features/copilot.
2. *WireShark - Go Deep* [online]. WireShark Foundation, 2023 [visited on 2023-10-08]. Available from: https://www.wireshark.org/.
3. *About Python* [online]. Python Foundation, 2022 [visited on 2023-09-20]. Available from: https://www.python.org/about/.
4. *Bulk IP Lookup - Geolocation* [online]. Brand Media Inc, 2023 [visited on 2023-11-20]. Available from: https://www.iplocation.net/bulk-ip-lookup.
5. *pandas - Python Data Analysis Library* [online]. AQR Capital Management, 2023 [visited on 2023-09-20]. Available from: https://pandas.pydata.org/about/.
6. *NumPy* [online]. NumPy, 2023 [visited on 2023-09-20]. Available from: https://numpy.org/.

Appendix A: Task 1 Instructions and Related Code

Data for this task has been collected using Wireshark's UI [2]. Once the .pcap file is obtained, it has been uploaded to Aalto's servers to process and obtain the three different data sets. PS1 is the base .pcap file, but PS2 and PS3 have been obtained employing Coral Flow and TCP Trace as instructed.

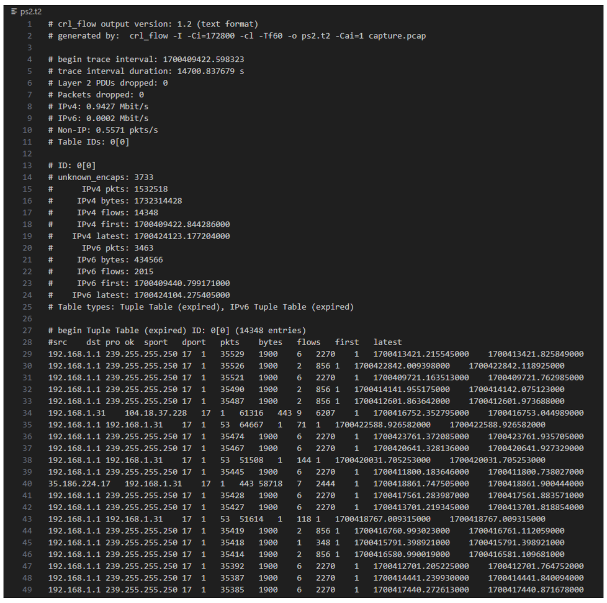
```
urrutia1@brute:/var/tmp/urrutia1$ . /work/courses/unix/T/ELEC/E7130/general/use.sh
urrutia1@brute:/var/tmp/urrutia1$ crl_flow -I -Ci=172800 -cl -Tf60 -o ps2.t2 -Cai=1
  ↪  capture.pcap
urrutia1@brute:/var/tmp/urrutia1$ tcptrace -l -r -n --csv capture.pcap > ps3.csv
urrutia1@brute:/var/tmp/urrutia1$ ls
capture.pcap  ps2.t2  ps3.csv
```

This results in the three data sets having the form shown in Figure 29.



```
No.  Time      Source                Destination          Protocol Length Info
   1 0.000000  ZyxelCommuni_a8:cf:…  Broadcast            ARP        60 Who has 192.168.1.103? Tell 192.168.1.1
   2 0.245963  192.168.1.31          216.58.209.163       QUIC     1399 0-RTT, DCID=577e59703460e4a6, SCID=44860a
   3 0.250959  216.58.209.163        192.168.1.31         QUIC     1399 Protected Payload (KP0), DCID=44860a
   4 0.252239  192.168.1.31          216.58.209.163       QUIC      156 Protected Payload (KP0), DCID=f77e59703460e4a6
   5 0.252287  216.58.209.163        192.168.1.31         QUIC      657 Protected Payload (KP0), DCID=44860a
   6 0.255108  216.58.209.163        192.168.1.31         QUIC      171 Protected Payload (KP0), DCID=44860a
   7 0.255361  216.58.209.163        192.168.1.31         QUIC       67 Protected Payload (KP0), DCID=44860a
   8 0.255494  192.168.1.31          216.58.209.163       QUIC       73 Protected Payload (KP0), DCID=f77e59703460e4a6
   9 0.259056  216.58.209.163        192.168.1.31         QUIC      573 Protected Payload (KP0), DCID=44860a
  10 0.259619  216.58.209.163        192.168.1.31         QUIC       67 Protected Payload (KP0), DCID=44860a
  11 0.259621  192.168.1.31          216.58.209.163       QUIC       78 Protected Payload (KP0), DCID=f77e59703460e4a6
  12 0.259747  192.168.1.31          216.58.209.163       QUIC       73 Protected Payload (KP0), DCID=f77e59703460e4a6
  13 0.287186  216.58.209.163        192.168.1.31         QUIC       69 Protected Payload (KP0), DCID=44860a
  14 0.514829  192.168.1.31          162.254.198.44       TLSv1.2   108 Application Data
  15 0.571782  162.254.198.44        192.168.1.31         TCP        60 27037 → 62815 [ACK] Seq=1 Ack=55 Win=2047 Len=0
  16 1.360190  216.58.209.170        192.168.1.31         UDP        74 443 → 59207 Len=32
  17 1.373579  192.168.1.31          216.58.209.170       UDP        77 59207 → 443 Len=35
  18 1.680900  ZyxelCommuni_a8:cf:…  IEEE-1905.1-Control  ieee19…   222 Vendor specific
  19 1.746658  192.168.1.31          192.168.1.1          DNS        79 Standard query 0x9e0a A spo-ring.msedge.net
  20 1.746861  192.168.1.31          192.168.1.1          DNS        79 Standard query 0x2219 AAAA spo-ring.msedge.net
  21 1.750746  192.168.1.1           192.168.1.31         DNS       168 Standard query response 0x9e0a A spo-ring.msedge.net CNAM…
  22 1.751276  192.168.1.1           192.168.1.31         DNS       192 Standard query response 0x2219 AAAA spo-ring.msedge.net C…
  23 1.752009  192.168.1.31          13.107.138.254       TCP        66 63264 → 443 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=256 S…
  24 1.762989  13.107.138.254        192.168.1.31         TCP        66 443 → 63264 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=14…
  25 1.763059  192.168.1.31          13.107.138.254       TCP        54 63264 → 443 [ACK] Seq=1 Ack=1 Win=262144 Len=0
  26 1.763410  192.168.1.31          13.107.138.254       TLSv1.2   566 Client Hello (SNI=spo-ring.msedge.net)
  27 1.770234  13.107.138.254        192.168.1.31         TCP        60 443 → 63264 [ACK] Seq=1 Ack=513 Win=4194048 Len=0
  28 1.772415  13.107.138.254        192.168.1.31         TCP      1514 443 → 63264 [ACK] Seq=1 Ack=513 Win=4194048 Len=1460 [TCP…
  29 1.772461  192.168.1.31          13.107.138.254       TCP        54 63264 → 443 [ACK] Seq=513 Ack=1461 Win=262144 Len=0
  30 1.772870  13.107.138.254        192.168.1.31         TCP      1514 443 → 63264 [ACK] Seq=1461 Ack=513 Win=4194048 Len=1460 […
  31 1.772870  13.107.138.254        192.168.1.31         TCP      1514 443 → 63264 [ACK] Seq=2921 Ack=513 Win=4194048 Len=1460 […
  32 1.772870  13.107.138.254        192.168.1.31         TCP      1514 443 → 63264 [ACK] Seq=4381 Ack=513 Win=4194048 Len=1460 […
  33 1.772870  13.107.138.254        192.168.1.31         TLSv1.2   420 Server Hello, Certificate, Certificate Status, Server Key…
  34 1.772928  192.168.1.31          13.107.138.254       TCP        54 63264 → 443 [ACK] Seq=513 Ack=6207 Win=262144 Len=0
  35 1.778447  192.168.1.31          13.107.138.254       TLSv1.2   212 Client Key Exchange, Change Cipher Spec, Encrypted Handsh…
  36 1.778630  192.168.1.31          13.107.138.254       TLSv1.2   141 Application Data
```

(a) Snippet of PS1.



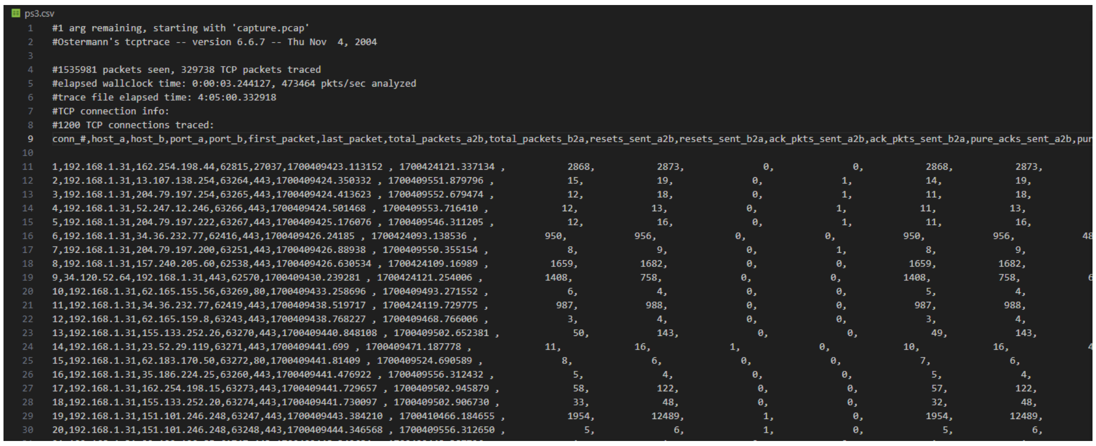
```
ps2.t2
# crl_flow output version: 1.2 (text format)
# generated by:  crl_flow -I -Ci=172800 -cl -Tf60 -o ps2.t2 -Cai=1 capture.pcap

# begin trace interval: 1700409422.598323
# trace interval duration: 14700.837679 s
# Layer 2 PDUs dropped: 0
# Packets dropped: 0
# IPv4: 0.9427 Mbit/s
# IPv6: 0.0002 Mbit/s
# Non-IP: 0.5571 pkts/s
# Table IDs: 0[0]

# ID: 0[0]
# unknown_encaps: 3733
#      IPv4 pkts: 1532518
#     IPv4 bytes: 1732314428
#     IPv4 flows: 14348
#     IPv4 first: 1700409422.844286000
#    IPv4 latest: 1700424123.177204000
#      IPv6 pkts: 3463
#     IPv6 bytes: 434566
#     IPv6 flows: 2015
#     IPv6 first: 1700409440.799171000
#    IPv6 latest: 1700424104.275405000
# Table types: Tuple Table (expired), IPv6 Tuple Table (expired)

# begin Tuple Table (expired) ID: 0[0] (14348 entries)
#src    dst pro ok  sport   dport   pkts    bytes   flows   first   latest
192.168.1.1 239.255.255.250 17  1   35529   1900    6   2270    1   1700413421.215545000    1700413421.825849000
192.168.1.1 239.255.255.250 17  1   35526   1900    2   856 1   1700422842.009398000    1700422842.118925000
192.168.1.1 239.255.255.250 17  1   35521   1900    6   2270    1   1700409721.163513000    1700409721.762985000
192.168.1.1 239.255.255.250 17  1   35490   1900    2   856 1   1700414141.955175000    1700414142.075123000
192.168.1.1 239.255.255.250 17  1   35487   1900    2   856 1   1700412601.863642000    1700412601.973688000
192.168.1.31    104.18.37.228   17  1   61316   443 9   6207    1   1700416752.352795000    1700416753.044989000
192.168.1.1 192.168.1.31    17  1   53  64667   1   71  1   1700422588.926582000    1700422588.926582000
192.168.1.1 239.255.255.250 17  1   35474   1900    6   2270    1   1700423761.372085000    1700423761.935705000
192.168.1.1 239.255.255.250 17  1   35467   1900    6   2270    1   1700420641.328136000    1700420641.927329000
192.168.1.1 192.168.1.31    17  1   53  51508   1   144 1   1700420031.705253000    1700420031.705253000
192.168.1.1 239.255.255.250 17  1   35445   1900    6   2270    1   1700411800.183646000    1700411800.738027000
35.186.224.17   192.168.1.31    17  1   443 58718   7   2444    1   1700418861.747505000    1700418861.900444000
192.168.1.1 239.255.255.250 17  1   35428   1900    6   2270    1   1700417561.283987000    1700417561.883571000
192.168.1.1 239.255.255.250 17  1   35427   1900    6   2270    1   1700413701.219345000    1700413701.818854000
192.168.1.1 192.168.1.31    17  1   53  51614   1   118 1   1700418767.009315000    1700418767.009315000
192.168.1.1 239.255.255.250 17  1   35419   1900    2   856 1   1700416760.993023000    1700416761.112059000
192.168.1.1 239.255.255.250 17  1   35418   1900    1   348 1   1700415791.398921000    1700415791.398921000
192.168.1.1 239.255.255.250 17  1   35414   1900    2   856 1   1700416580.990019000    1700416581.109681000
192.168.1.1 239.255.255.250 17  1   35392   1900    6   2270    1   1700412701.205225000    1700412701.764752000
192.168.1.1 239.255.255.250 17  1   35387   1900    6   2270    1   1700414441.239930000    1700414441.840094000
192.168.1.1 239.255.255.250 17  1   35385   1900    6   2270    1   1700417440.272613000    1700417440.871678000
```

(b) Snippet of PS2.



```
ps3.csv
#1 arg remaining, starting with 'capture.pcap'
#Ostermann's tcptrace -- version 6.6.7 -- Thu Nov  4, 2004

#1535981 packets seen, 329738 TCP packets traced
#elapsed wallclock time: 0:00:03.244127, 473464 pkts/sec analyzed
#trace file elapsed time: 4:05:00.332918
#TCP connection info:
#1200 TCP connections traced:
conn_#,host_a,host_b,port_a,port_b,first_packet,last_packet,total_packets_a2b,total_packets_b2a,resets_sent_a2b,resets_sent_b2a,ack_pkts_sent_a2b,ack_pkts_sent_b2a,pure_acks_sent_a2b,pur

1,192.168.1.31,162.254.198.44,62815,27037,1700409423.113152 , 1700424121.337134 ,          2868,       2873,          0,          0,       2868,       2873,
2,192.168.1.31,13.107.138.254,63264,443,1700409424.350332 , 1700409551.879796 ,              15,         19,         0,         1,         14,         19,
3,192.168.1.31,204.79.197.254,63265,443,1700409424.413623 , 1700409552.679474 ,              12,         18,         0,         1,         11,         18,
4,192.168.1.31,52.247.12.246,63266,443,1700409424.501468 , 1700409553.716410 ,              12,         13,         0,         1,         11,         13,
5,192.168.1.31,204.79.197.222,63267,443,1700409425.176076 , 1700409546.311205 ,              12,         16,         0,         1,         11,         16,
6,192.168.1.31,34.36.232.77,62416,443,1700409426.24185 , 1700424093.138536 ,            950,        956,        0,         0,        950,        956,        48
7,192.168.1.31,204.79.197.200,63251,443,1700409426.88938 , 1700409550.355154 ,             8,          9,         0,         1,          8,          9,
8,192.168.1.31,157.240.205.60,62538,443,1700409426.630534 , 1700424109.16989 ,           1659,       1682,         0,         0,       1659,       1682,
9,34.120.52.64,192.168.1.31,443,62570,1700409430.239281 , 1700424121.254006 ,          1408,        758,        0,         0,       1408,        758,         6
10,192.168.1.31,62.165.155.56,63269,80,1700409433.258696 , 1700409493.271552 ,             6,          4,         0,         0,          5,          4,
11,192.168.1.31,34.36.232.77,62419,443,1700409438.519717 , 1700424119.729775 ,           987,        988,         0,         0,        987,        988,
12,192.168.1.31,62.165.159.8,63243,443,1700409438.768227 , 1700409468.766006 ,             3,          4,         0,         0,          3,          4,
13,192.168.1.31,155.133.252.26,63270,443,1700409440.848108 , 1700409502.652381 ,             50,        143,         0,         0,         49,        143,
14,192.168.1.31,23.52.29.119,63271,443,1700409441.699 , 1700409471.187778 ,            11,         16,        1,         0,        10,         16,         4
15,192.168.1.31,62.183.170.50,63272,80,1700409441.81409 , 1700409524.690589 ,             8,          6,         0,         0,          7,          6,
16,192.168.1.31,35.186.224.25,63260,443,1700409441.476922 , 1700409556.312432 ,              5,          4,         0,         0,          5,          4,
17,192.168.1.31,162.254.198.15,63273,443,1700409441.729657 , 1700409502.945879 ,             58,        122,         0,         0,         57,        122,
18,192.168.1.31,155.133.252.20,63274,443,1700409441.730097 , 1700409502.906730 ,             33,         48,         0,         0,         32,         48,
19,192.168.1.31,151.101.246.248,63247,443,1700409443.384210 , 1700410466.184655 ,          1954,      12489,          1,          0,       1954,      12489,
20,192.168.1.31,151.101.246.248,63248,443,1700409444.346568 , 1700409556.312650 ,             5,          6,          1,          0,          5,          6,
```

(c) Snippet of PS3.

Figure 29: Snippets of PS1, PS2, and PS3.

A.1 PS1 analysis

Using PS1, one can get the data filtered by port by navigating Wireshark's UI. Going to Statistics, then Endpoints, is a straight-forward path to obtain a .csv file for each protocol, UDP and TCP, that contains the data relevant for the port plots. This can then be processed using Python [3] and its library Pandas [5].

```python
# Load the data
dfudp = pd.read_csv('ps1-udp.csv')
dftcp = pd.read_csv('ps1-tcp.csv')

# Group by 'Port' and sum 'Packets'
groupudp = dfudp.groupby('Port')['Packets'].sum().sort_values()
grouptcp = dftcp.groupby('Port')['Packets'].sum().sort_values()

top15udp = groupudp.tail(15)
bottom15udp = groupudp.head(15)
top15tcp = grouptcp.tail(15)
bottom15tcp = grouptcp.head(15)
```

The historiogram for packet length distribution can be similarly obtained:

```python
plt.hist(dfudp['Bytes'], bins=range(min(dfudp['Bytes']), max(dfudp['Bytes']) + 1, 1))
```

This, however, can easily crash most machines. Changing the bins parameters to a more digestible number like 100 can help in easing processing. The final plots regarding traffic volume, as well as statistically relevant values, have been again obtained from Wireshark's UI. The first in Statistics, then I/O Graphs, and the second one in Statistics, Captured File properties. Finally, to compute the ECDF, a quick Python function employing NumPy [6] has been devised.

```python
def ecdf(data):
    n = len(data)
    x = np.sort(data)
    y = np.arange(1, n+1) / n
    return x, y
```

A.2 PS2 analysis

The data treatment used to obtain PS2 results in a .t2 file that contains all flows and relevant related data in two separate categories: IPv4 and IPv6. In order to read that file and translate into two Python objects that can be operated with, a function has been devised:

```python
def extract_data(file_content, table_type):
    start_pattern = f"# begin {table_type} ID: 0[0] ("
    end_pattern = "# end of text table"
    start_index = file_content.find(start_pattern)
    end_index = file_content.find(end_pattern, start_index)
    table_data = file_content[start_index +
    ↪  len(start_pattern):end_index].strip().split('\n')[1:]
    # Extracting column names
    columns = table_data[0].split()
    # Extracting data rows
    data_rows = [re.split(r'\t+', row) for row in table_data[2:]]

    return columns, data_rows
```

To get the top and bottom 15 ports by number of flows, the obtained columns and data rows are converted into a pandas object, then sorted and grouped accordingly by a separate function; in a very similar manner to that used for PS1 analysis:

```python
def plot_flows_vs_dport(columns, data_rows, title):
    # Create a DataFrame
    df = pd.DataFrame(data_rows, columns=columns)
    df['dport'] = pd.to_numeric(df['dport'])
    df['flows'] = pd.to_numeric(df['flows'])
    # Group by dport and sum flows, get only top and bottom 15
    grouped = df.groupby('dport')['flows'].sum().sort_values()
    top15 = grouped.tail(15)
    bottom15 = grouped.head(15)
    # Plotting
    bottom15.plot(kind='bar')
    plt.xlabel('Port')
    plt.ylabel('Number of Flows')
    plt.yscale('log')
    plt.title('Flows vs Port - Bottom 15 - ' + title)
    plt.show()
    top15.plot(kind='bar')
    plt.xlabel('Port')
    plt.ylabel('Number of Flows')
    plt.yscale('log')
    plt.title('Flows vs Port - Top 15 - ' + title)
    plt.show()
    return df
```

Then, to plot relevant time periods, the next function has been coded. Note that $start_time$ and end_time need to be datetime objects, as well as the objects in the 'first' column. The 'bytes' column needs to be int or float as well.

```python
def plot_timebytes_period(df_ipv4, df_ipv6, start_time, end_time, title):
    df_ipv4_filtered = df_ipv4[(df_ipv4['first'] >= start_time) & (df_ipv4['first'] <=
    ↪   end_time)]
    df_ipv6_filtered = df_ipv6[(df_ipv6['first'] >= start_time) & (df_ipv6['first'] <=
    ↪   end_time)]
    plt.figure(figsize=(10, 6))
    plt.scatter(df_ipv4_filtered['first'], df_ipv4_filtered['bytes'], label='IPv4')
    plt.scatter(df_ipv6_filtered['first'], df_ipv6_filtered['bytes'], label='IPv6')
    plt.title('Traffic Volume vs Time - ' + title)
    plt.xlabel('Time')
    plt.ylabel('Bytes')
    plt.yscale('log')
    plt.show()
```

Top IPs by number of flows, to then be translated to locations of the IPs, have been obtained very similarly:

```python
top10dest_ipv4 = df_ipv4.groupby('dst')['flows'].sum().sort_values().tail(10)
top10dest_ipv6 = df_ipv6.groupby('dst')['flows'].sum().sort_values().tail(10)
```

For the Zipf plot, though the process is not complicated, a function dedicated to that has been coded in order to be able to call it at any time if the need arises:

```python
def zipf_plot(df_ipv4, df_ipv6):
    df_ipv4['pair'] = df_ipv4['#src'] + ' -\n ' + df_ipv4['dst']
    df_ipv6['pair'] = df_ipv6['#src'] + ' -\n ' + df_ipv6['dst']

    pair_freq_ipv4 = df_ipv4['pair'].value_counts().values
    pair_freq_ipv6 = df_ipv6['pair'].value_counts().values

    rank_ipv4 = np.arange(1, len(pair_freq_ipv4)+1)
    rank_ipv6 = np.arange(1, len(pair_freq_ipv6)+1)

    plt.figure(figsize=(10, 5))
    plt.loglog(rank_ipv4, pair_freq_ipv4, marker="o", label='IPv4')
    plt.loglog(rank_ipv6, pair_freq_ipv6, marker="o", label='IPv6')
    plt.title('Zipf plot for Pair Frequencies')
    plt.xlabel('Rank')
    plt.ylabel('Frequency')
    plt.legend()
    plt.show()
```

Now moving to flow length distribution, a function dedicated to that has been devised, which returns the modified data frames with the flow_length parameter for later use:

```python
def flow_length_dist(df_ipv4, df_ipv6):
    df_ipv4['flow_length'] = (df_ipv4['latest'] - df_ipv4['first']).dt.total_seconds()
    df_ipv6['flow_length'] = (df_ipv6['latest'] - df_ipv6['first']).dt.total_seconds()

    plt.figure(figsize=(10, 5))
    plt.hist(df_ipv4['flow_length'], bins=100, label='IPv4')
    plt.hist(df_ipv6['flow_length'], bins=100, label='IPv6')
    plt.title('Flow Length Distribution')
    plt.xlabel('Flow Length (s)')
    plt.ylabel('Frequency')
    plt.yscale('log')
    plt.legend()
    plt.show()
    return df_ipv4, df_ipv6
```

Similarly, for ECDF plots and relevant statistical properties, functions have been implemented so they can be repeatedly called for different uses, as seen in the report.

```python
def ecdf(data):
    x = np.sort(data)
    y = np.arange(1, len(x)+1) / len(x)
    return x, y

def relevant_statistics(data):
    try:
        mean = np.mean(data)
        print('Mean: ' + str(mean))
    except:
        pass
    median = np.median(data)
    print('Median: ' + str(median))
    std = np.std(data)
    print('Standard Deviation: ' + str(std))
    min = np.min(data)
    print('Min: ' + str(min))
    max = np.max(data)
    print('Max: ' + str(max))
```

Finally, to explore which statistical distributions fit better the flow length distribution, the following two functions are provided:

```python
def best_fit(data):
    # Distributions to fit
    dist_names = ['expon', 'gamma', 'lognorm', 'norm']
    dist_results = []
    params = {}
    for dist_name in dist_names:
        dist = getattr(stats, dist_name)
        param = dist.fit(data)
        params[dist_name] = param
        # Applying the Kolmogorov-Smirnov test
        D, p = stats.kstest(data, dist_name, args=param)
        dist_results.append((dist_name, p))
    # Select the best fitted distribution
    best_dist, best_p = (max(dist_results, key=lambda item: item[1]))
    # Store the name of the best fit and its p value
    return best_dist, best_p, params[best_dist]

def plot_bestfit(data, best_dist, params, title):
    # Plotting the best fit
    plt.figure(figsize=(10, 5))
    plt.hist(data, bins=100, density=True, label='Data')
    # Generating the x values
    x = np.linspace(np.min(data), np.max(data), 100)
    # Generating the PDF
    pdf = getattr(stats, best_dist).pdf(x, *params)
    plt.plot(x, pdf, label='Best Fit')
    plt.title(title)
    plt.xlabel('Flow Length (s)')
    plt.ylabel('Frequency')
    plt.yscale('log')
    plt.legend()
    plt.show()
```

To wrap PS2 up, different timeouts have been tested to see how the number of flows changes. This has been done by slightly altering the command used to transform the .pcap file to flows:

```
urrutia1@brute:/var/tmp/urrutia1$ crl_flow -I -Ci=172800 -cl -Tf1 -o ps2-1.t2 -Cai=1
↪ capture.pcap
urrutia1@brute:/var/tmp/urrutia1$ crl_flow -I -Ci=172800 -cl -Tf10 -o ps2-1.t2 -Cai=1
↪ capture.pcap
urrutia1@brute:/var/tmp/urrutia1$ crl_flow -I -Ci=172800 -cl -Tf120 -o ps2-1.t2 -Cai=1
↪ capture.pcap
urrutia1@brute:/var/tmp/urrutia1$ crl_flow -I -Ci=172800 -cl -Tf1800 -o ps2-1.t2 -Cai=1
↪ capture.pcap
```

A.3 PS3 analysis

For PS3, the readout is very straightforward using Python's pandas library, for the .csv generated has only a few lines that need skipping at the very top:

```python
df = pd.read_csv('ps3.csv', header=1, skiprows=range(1, 8), skip_blank_lines=True)
```

From there on, it is of interest to reuse the previously defined relevant$_s$tatistics function, as well as using already shown methods to plot the historiogram of RTT average column. To finish this appendix, all that is left is to showcase the definition of total data volume, that is comprised of a sum between the actual bytes sent from A to B and from B to A.

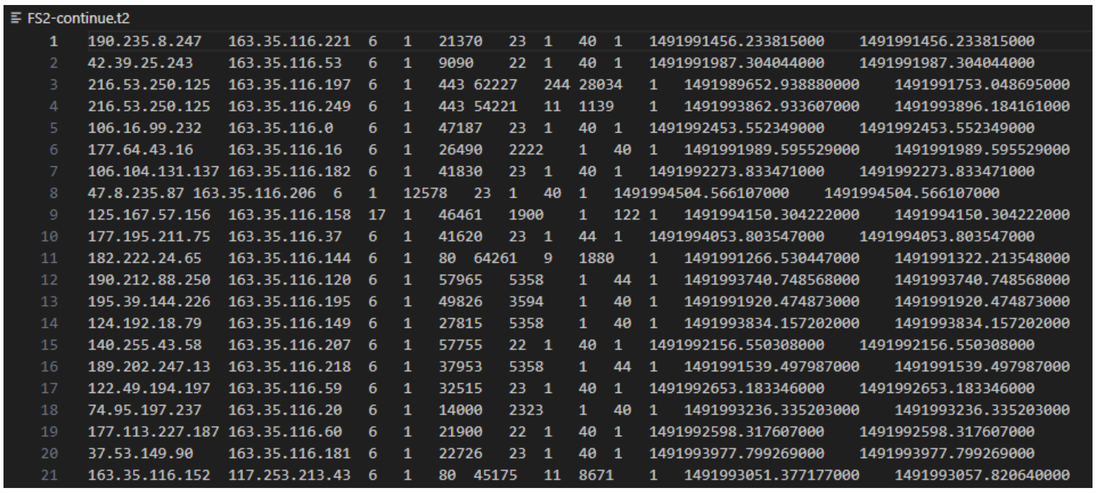
```python
df['total_data_volume'] = df['actual_data_bytes_a2b'] + df['actual_data_bytes_b2a']
```

Appendix B: Task 2 Instructions and Related Code

In order to transform the chosen FS1 file into a FS2 type of file, the gawk command has been used as suggested. Since this is a flow – continue type of file, however, the filter has been applied to the first and second columns instead:

```
output_file=~/FS2-continue.t2
> $output_file
for file in $(find /var/tmp/urrutia1/flow-continue -type f -name "*.t2"); do
    gawk '$1~/^163\.35\.116\./||$2~/^163\.35\.116\./' $file >> $output_file
done
```

This results in a FS2 file with the aspect shown in Figure 30.



```
FS2-continue.t2
190.235.8.247   163.35.116.221  6   1   21370   23  1   40  1   1491991456.233815000    1491991456.233815000
42.39.25.243    163.35.116.53   6   1   9090    22  1   40  1   1491991987.304044000    1491991987.304044000
216.53.250.125  163.35.116.197  6   1   443 62227   244 28034   1   1491989652.938880000    1491991753.048695000
216.53.250.125  163.35.116.249  6   1   443 54221   11  1139    1   1491993862.933607000    1491993896.184161000
106.16.99.232   163.35.116.0    6   1   47187   23  1   40  1   1491992453.552349000    1491992453.552349000
177.64.43.16    163.35.116.16   6   1   26490   2222    1   40  1   1491991989.595529000    1491991989.595529000
106.104.131.137 163.35.116.182  6   1   41830   23  1   40  1   1491992273.833471000    1491992273.833471000
47.8.235.87 163.35.116.206  6   1   12578   23  1   40  1   1491994504.566107000    1491994504.566107000
125.167.57.156  163.35.116.158  17  1   46461   1900    1   122 1   1491994150.304222000    1491994150.304222000
177.195.211.75  163.35.116.37   6   1   41620   23  1   44  1   1491994053.803547000    1491994053.803547000
182.222.24.65   163.35.116.144  6   1   80  64261   9   1880    1   1491991266.530447000    1491991322.213548000
190.212.88.250  163.35.116.120  6   1   57965   5358    1   44  1   1491993740.748568000    1491993740.748568000
195.39.144.226  163.35.116.195  6   1   49826   3594    1   40  1   1491991920.474873000    1491991920.474873000
124.192.18.79   163.35.116.149  6   1   27815   5358    1   40  1   1491993834.157202000    1491993834.157202000
140.255.43.58   163.35.116.207  6   1   57755   22  1   40  1   1491992156.550308000    1491992156.550308000
189.202.247.13  163.35.116.218  6   1   37953   5358    1   44  1   1491991539.497987000    1491991539.497987000
122.49.194.197  163.35.116.59   6   1   32515   23  1   40  1   1491992653.183346000    1491992653.183346000
74.95.197.237   163.35.116.20   6   1   14000   2323    1   40  1   1491993236.335203000    1491993236.335203000
177.113.227.187 163.35.116.60   6   1   21900   22  1   40  1   1491992598.317607000    1491992598.317607000
37.53.149.90    163.35.116.181  6   1   22726   23  1   40  1   1491993977.799269000    1491993977.799269000
163.35.116.152  117.253.213.43  6   1   80  45175   11  8671    1   1491993051.377177000    1491993057.820640000
```

Figure 30: Snippet of TS2.

B.1 FS2 Analysis

To begin with, the function used in Task 1 to plot the most and least used ports has been adapted to the reality of this dataset.

```
def plot_flows_vs_dport(df, title):
    df['dport'] = pd.to_numeric(df['dport'])
    df['flows'] = pd.to_numeric(df['flows'])
    # Group by dport and sum flows, get only top and bottom 15
    grouped = df.groupby('dport')['flows'].sum().sort_values()
    top15 = grouped.tail(15)
    bottom15 = grouped.head(15)
    # Plotting
    bottom15.plot(kind='bar')
    plt.xlabel('Port')
    plt.ylabel('Number of Flows')
    plt.yscale('log')
    plt.title('Flows vs Port - Bottom 15 - ' + title)
    plt.show()
    top15.plot(kind='bar')
    plt.xlabel('Port')
    plt.ylabel('Number of Flows')
    plt.yscale('log')
```

```python
    plt.title('Flows vs Port - Top 15 - ' + title)
    plt.show()
```

Similarly, the function to visualise data volumes for two different time periods has as well received very minor tweaks for the analysis of FS2.

```python
def plot_timebytes_period(df, start_time, end_time, title):
    df_zoom = df[(df['first'] >= start_time) & (df['first'] <= end_time)]
    plt.figure(figsize=(10, 6))
    plt.scatter(df_zoom['first'], df_zoom['bytes'])
    plt.title('Traffic Volume vs Time - ' + title)
    plt.xlabel('Time')
    plt.ylabel('Bytes')
    plt.yscale('log')
    plt.show()

```

When dealing with user activity, a new function has been devised from scratch, where a new list is done with the aggregate of total data moved by each user, and then a historiogram is plotted from that.

```python
def user_analysis(df):
    aggregate = []
    for user in range(0, 255):
        user = '163.35.116.' + str(user)
        df_user = df[(df['src'] == user) | (df['dst'] == user)]
        aggregate.append(df_user['bytes'].sum())
    plt.hist(aggregate, bins=100)
    plt.xlabel('Bytes')
    plt.ylabel('Number of Users')
    plt.title('User Analysis')
    plt.yscale('log')
    plt.show()
    return aggregate

```

B.2 FS1 Sampling

In order to perform a sampling of the whole FS1 dataset, a bash script has been devised so that it gets only 50 lines per flow type per file:

```bash
#!/bin/bash

dir="/var/tmp/urrutia1/flow-continue/"
output_file_ipv4="FS1_sample_ipv4.txt"
output_file_ipv6="FS1_sample_ipv6.txt"

> "$output_file_ipv4"
> "$output_file_ipv6"

for file in "$dir"*
do
    # Print the filename (helps in tracking progress)
    echo "$file"
    # Separate lines corresponding to IPv4 and IPv6
    grep -v '^#' "$file" | grep '\.' | shuf -n 50 >> "$output_file_ipv4"
    grep -v '^#' "$file" | grep ':' | shuf -n 50 >> "$output_file_ipv6"
done
done
```

Appendix C: Task 3 Instructions and Related Code

C.1 Data collection

Data collection was done using the same scripts developed for Assignment 2. These are bash scripts, one for each type of server and measurement. For instance, this is the bash file regarding the latency measurements of research servers. The rest can be consulted in the attached .zip file with all the codes.

```
date
echo "bcn-es"
ping bcn-es.ark.caida.org -c 5 -O -D | fgrep -e packets -e rtt
echo "mnl-ph"
ping mnl-ph.ark.caida.org -c 5 -O -D | fgrep -e packets -e rtt
echo "hnl-us"
ping hnl-us.ark.caida.org -c 5 -O -D | fgrep -e packets -e rtt
```

These were then called at regular intervals using crontab, in Windows Subsystem Linux.

C.2 AS1.x - Latency log processing

Due to the structure used for storing the two-week-long data set, careful data translation needs to be performed so that a pandas data frame can be obtained. Then, in order to obtain the datasets for each connection, a simple filter can be done to separate the data frame into individual ones for each server. Included here is the process done for the nameservers proved via ping, but a very similar process has been done for the rest of the cases.

```
with open(nameservers_path, 'r') as file:
    lines = file.readlines()
    now_timestamp = None
    line_index = 0
    for line in lines:
        line = line.strip()
        try:
            line = line.replace('EEST', '')
            now_timestamp = (datetime.strptime(line, '%a %b %d %H:%M:%S
            ↪  %Y')).strftime('%Y-%m-%d %H:%M:%S')
        except:
            pass
        if line.startswith('f.') or line.startswith('g.') or line.startswith('e.'):
            now_nameserver = line
            timestamps.append(now_timestamp)
            nameserver_list.append(now_nameserver)
            # If connection is not possible, the next two lines will not include ping data
            try:
                if not("packets" in lines[line_index+1]):
                    packets_transmitted.append(5)
                    packets_received.append(0)
                    packet_loss.append(100)
                    timems.append(float('inf'))
                if not("rtt" in lines[line_index+2]):
                    rtt_min.append(float('inf'))
                    rtt_avg.append(float('inf'))
                    rtt_max.append(float('inf'))
                    rtt_mdev.append(float('inf'))
```

```python
            except:
                packets_transmitted.append(5)
                packets_received.append(0)
                packet_loss.append(100)
                timems.append(float('inf'))
                rtt_min.append(float('inf'))
                rtt_avg.append(float('inf'))
                rtt_max.append(float('inf'))
                rtt_mdev.append(float('inf'))
        elif "packets" in line:
            ping = line.split (',')
            packets_transmitted.append(int(ping[0].strip(' packets transmitted')))
            packets_received.append(int(ping[1].strip(' packets received')))
            packet_loss.append(float(ping[2].strip('% packet loss')))
            timems.append(int(ping[3].strip('time ms')))
        elif "rtt" in line:
            rtt = line.strip('rtt min/avg/max/mdev = ms').split('/')
            rtt_min.append(float(rtt[0]))
            rtt_avg.append(float(rtt[1]))
            rtt_max.append(float(rtt[2]))
            rtt_mdev.append(float(rtt[3]))
        elif "CURL" in line:
            dig = line.strip('CURL: ').split(' ')
            time_starttransfer.append(float(dig[0]))
            time_pretransfer.append(float(dig[1]))
            time_connect.append(float(dig[2]))
            total_latency.append(float(dig[0])+float(dig[1]))
        line_index += 1

nameservers_pingdata = pd.DataFrame({
    'Timestamp': timestamps,
    'Nameserver': nameserver_list,
    'Packets Transmitted': packets_transmitted,
    'Packets Received': packets_received,
    'Packet Loss %': packet_loss,
    'Time (ms)' : timems,
    'RTT Min (ms)': rtt_min,
    'RTT Avg (ms)': rtt_avg,
    'RTT Max (ms)': rtt_max,
    'RTT Mdev (ms)': rtt_mdev
})

AS1_d1 =
↪  nameservers_pingdata[nameservers_pingdata['Nameserver'].str.contains("e.ext.nic.fr")]
AS1_d2 =
↪  nameservers_pingdata[nameservers_pingdata['Nameserver'].str.contains("g.ext.nic.fr")]
AS1_d3 =
↪  nameservers_pingdata[nameservers_pingdata['Nameserver'].str.contains("f.ext.nic.fr")]
```

Once the different **AS1.x** data frames have been generated, the box plots have been generated by devising two functions, one where packet loss is not included, and one where every data is taken into consideration. Note that for the case where packet loss is not one of the columns of the data frame, total latency is considered instead; and a packet is consider lost if the latency is more than 1 seconds (or 1000 miliseconds).

```python
def create_boxplot_success(dataframe, title):
    try:
        dataframe = dataframe[dataframe['Packet Loss %'] == 0]
    except:
        try:
            dataframe = dataframe[dataframe['Packet Loss %'] == 100]
        except:
            dataframe = dataframe[dataframe['Total Latency'] < float(1000)]
    plt.figure(figsize=(14, 6))
    dataframe.boxplot()
    plt.title('Box Plot - ' + title)
    plt.yscale('log')
    plt.xlabel('Variables')
    plt.ylabel('Values')
    plt.savefig(own_path + '\\boxplots-success\\' + title + '.png')
    plt.show()

def create_boxplot_all(dataframe, title):
    plt.figure(figsize=(14, 6))
    dataframe.boxplot()
    plt.title('Box Plot - ' + title)
    plt.yscale('log')
    plt.xlabel('Variables')
    plt.ylabel('Values')
    plt.savefig(own_path + '\\boxplots-all\\' + title + '.png')
    plt.show()
```

Similarly, the PDF and CDF plots also have their own functions. Notably, the PDF one takes the upper limit into consideration so that the plot can be adjusted to the different ranges the data might need.

```python
def create_pdf_plot(dataframe, title, uplim=50):
    dataframe = dataframe[dataframe != float('inf')]
    dataframe = dataframe[dataframe != float('NaN')]
    plt.figure(figsize=(6, 6))
    dataframe.plot.kde()
    plt.title('PDF Plot - ' + title)
    plt.xlabel('RTT Values')
    plt.ylabel('Density')
    plt.xlim(0, uplim)
    plt.savefig(own_path + '\\pdf-plots\\' + title + '.png')
    plt.show()

def create_cdf_plot(dataframe, title):
    dataframe = dataframe[dataframe != float('inf')]
    dataframe = dataframe[dataframe != float('NaN')]
    plt.figure(figsize=(6, 6))
    dataframe.hist(cumulative=True, density=1, bins=100)
    plt.title('CDF Plot - ' + title)
    plt.xlabel('RTT Values')
    plt.ylabel('Density')
    plt.savefig(own_path + '\\cdf-plots\\' + title + '.png')
    plt.show()
```

The data for the summary table has been obtained using pandas' own functions for statistical values:

```python
def table_data(dataframe, title):
    dataframe = dataframe[dataframe != float('inf')]
    dataframe = dataframe[dataframe != float('NaN')]
    print(title)
    try:
        print('First delay: ' + str(dataframe[0]))
    except:
        print('First delay: ' + str(dataframe.iloc[0]))
    print('Mean: ' + str(dataframe.mean()))
    print('Median: ' + str(dataframe.median()))
    max_delay = dataframe.mean()*1.5
    proportion_max = dataframe[dataframe > max_delay].count()/dataframe.count()
    print('Percentage of values above 1.5*mean: ' + str(proportion_max*100) + '%')
    distance_quantiles = dataframe.quantile(0.95) - dataframe.quantile(0.5)
    print('Distance between 95th and 50th percentile: ' + str(distance_quantiles))
    print('')
```

Finally, the autocorrelation plots can also be done directly using this time matplotlib:

```python
def plot_autocorrelation(dataframe, title):
    dataframe = dataframe[dataframe != float('inf')]
    dataframe = dataframe[dataframe != float('NaN')]
    plt.figure(figsize=(6, 6))
    pd.plotting.autocorrelation_plot(dataframe)
    plt.title('Autocorrelation Plot - ' + title)
    plt.xlabel('Lag')
    plt.ylabel('Autocorrelation')
    plt.savefig(own_path + '\\autocorrelation-plots\\' + title + '.png')
    plt.show()
```

C.3 AS2.x - Throughput log processing

In a very similar way to the one employed for AS1.x; the method to read .json files that make up the throughput log have been analysed following the same method that was employed for Assignment 2. In this case, basically a for loop like the one displayed here has been employed, and afterwards the data has been transformed into a pandas data frame.

```python
for filename in ok1Nfiles:
    with open(filename, "r") as json_file:
        data = json.load(json_file)
        match = re.search(r'ok1N(\d+)\.json', filename)
        timestamp_seconds = int(match.group(1))
        timestamp = datetime.utcfromtimestamp(timestamp_seconds).strftime('%Y-%m-%d
        ↪  %H:%M:%S')
        if "streams" in data["end"] and data["end"]["streams"]:
            sender_bps = data["end"]["streams"][0]["sender"]["bits_per_second"]
            receiver_bps = data["end"]["streams"][0]["receiver"]["bits_per_second"]
        else:
            sender_bps = 0
            receiver_bps = 0
        ok1Ndata.append({"Timestamp": timestamp, "Sender_bps": sender_bps,"Receiver_bps":
        ↪  receiver_bps})
```

Notably, in order to create AS2.i1 and AS2.i2, the “normal” and “reverse” measurements for each server have been combined into a single data frame so that AS2.x contains indeed all the required information.

```python
ok1Ndata = pd.DataFrame(ok1Ndata)
ok1Rdata = pd.DataFrame(ok1Rdata)
sgp1Ndata = pd.DataFrame(sgp1Ndata)
sgp1Rdata = pd.DataFrame(sgp1Rdata)

ok1Ndata['Mode'] = 'N'
ok1Rdata['Mode'] = 'R'
sgp1Ndata['Mode'] = 'N'
sgp1Rdata['Mode'] = 'R'

AS2_i1 = pd.concat([ok1Ndata, ok1Rdata])
AS2_i2 = pd.concat([sgp1Ndata, sgp1Rdata])

AS2_i1 = AS2_i1.sort_values(by='Timestamp').reset_index(drop=True)
AS2_i2 = AS2_i2.sort_values(by='Timestamp').reset_index(drop=True)
```

Now that the subsets of AS2 have been defined and obtained, the statistical data is quite straightforward to get employing numpy. Importantly, for the geometric and arithmetic means, some operations were needed to ensure that the values were not zero nor infinity. The latter was actually a common result due to overflow, and for that logarithmic sums were used in lieu of typical multiplication operations.

```python
def statistics_analysis(data):
    non_zero_data = [x for x in data if x != 0 and x != float('inf') and x != float('NaN')]
    mean = np.mean(non_zero_data)
    harmonic_mean = len(non_zero_data) / sum([1 / x for x in non_zero_data])
    log_sum = sum(np.log(non_zero_data))
    geometric_mean = np.exp(log_sum / len(non_zero_data))
    median = np.median(non_zero_data)
    print(f"Mean: {mean}, Harmonic Mean: {harmonic_mean}, Geometric Mean: {geometric_mean},
    ↪  Median: {median}")
    return mean, harmonic_mean, geometric_mean, median
```

For the different plots, only slight modifications were needed from the functions defined for AS1, just to account for the different modes the data has (normal and reverse)

```python
def boxplot(data, title):
    N = data[data["Mode"] == "N"]
    R = data[data["Mode"] == "R"]
    plt.boxplot([N["Sender_bps"], R["Sender_bps"]], labels=["Normal", "Reverse"])
    plt.ylabel("Throughput (bps)")
    plt.yscale("log")
    plt.grid()
    plt.title(title)
    plt.show()

def plot_throughput(data, title):
    plt.scatter(data["Timestamp"], data["Sender_bps"], label="Throughput", marker='x')
    plt.xticks(np.arange(350,step=40),rotation=20)
    plt.xlabel("Timestamp")
    plt.ylabel("Throughput (bps)")
    plt.yscale("log")
    plt.title(title)
    plt.show()

def plot_autocorrelation(dataframe, title):
    dataframe = dataframe["Sender_bps"]
    dataframe = dataframe[dataframe != float('inf')]
    dataframe = dataframe[dataframe != float('NaN')]
    plt.figure(figsize=(6, 6))
    pd.plotting.autocorrelation_plot(dataframe)
    plt.title('Autocorrelation Plot - ' + title)
    plt.xlabel('Lag')
    plt.ylabel('Autocorrelation')
    plt.show()
```