**ELEC-E7130 Internet Traffic Measurements and Analysis**

**Final Assignment: From measurements to conclusions**

Name: Xingji Chen

Student ID: 101659554

E-mail: xingji.chen@aalto.fi

Task 1: Capturing data

* Acquiring packet capture data
* What kind of trace file and tool/s you are using to perform the packet capture.

Wireshark.

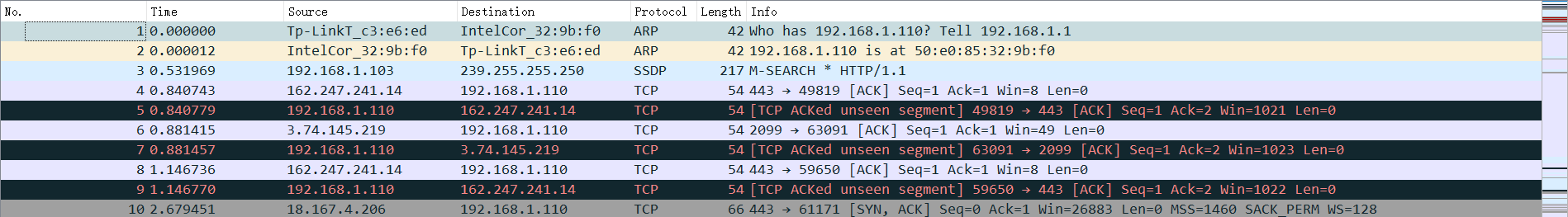
* Date, time, duration, measurement setting (in terms of profile if you are using the Wireshark) or file name if you are using the some public traces.

Date, time, duration: 2023.11.20 from 19:00 to 20:00.

Measurement setting: Default measurement setting of Wireshark.

File name: final.pcap

Provide a short sample (10 lines or so) of the data taken from your capture file.



* Data pre-processing
* Commands or code that is used in pre-processing for each case.

Converting packet trace to flow data (PS2)

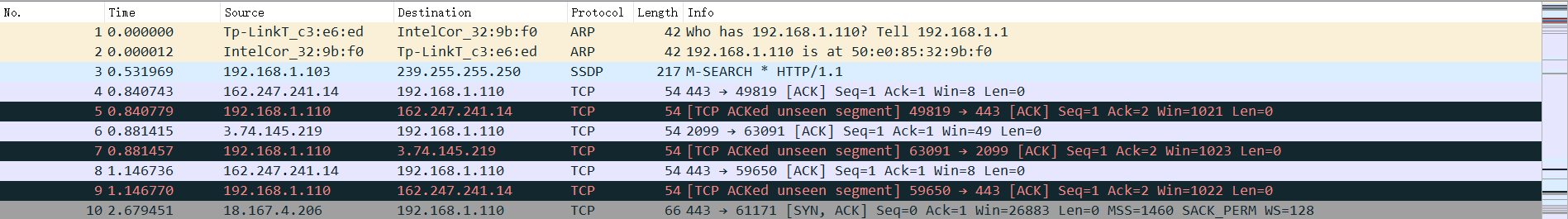
tshark -r finalr.pcap -q -z conv,tcp > final.txt

TCP connection statistics (PS3)

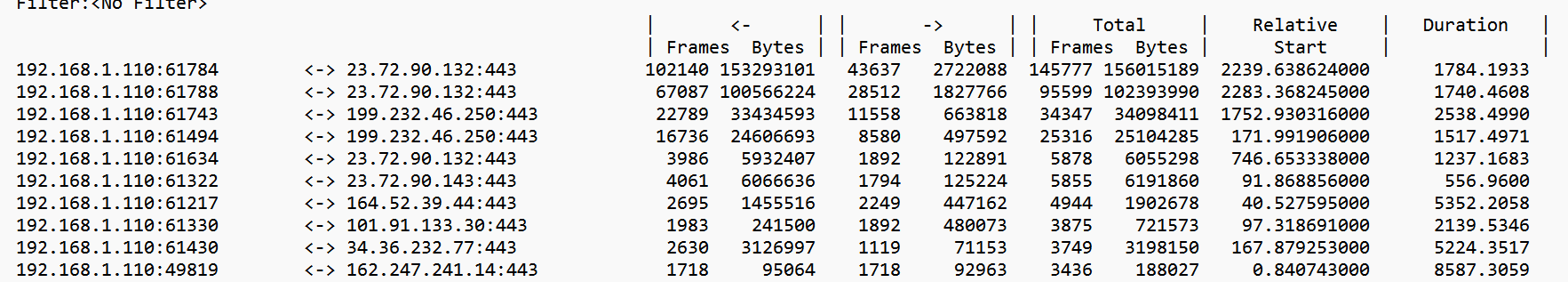
tcptrace -l -r -n --csv final.pcap > finaltcp.csv

* Short samples (10 lines or so) of the distilled data in each case (for PS3, one connection summary is enough).

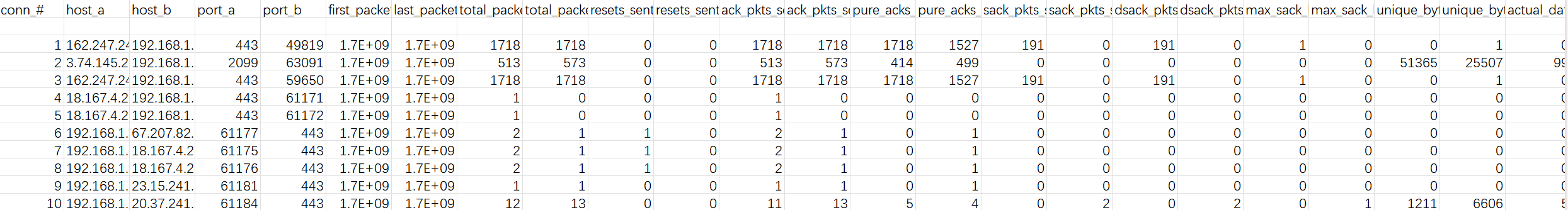
PS1:



PS2:



PS3:

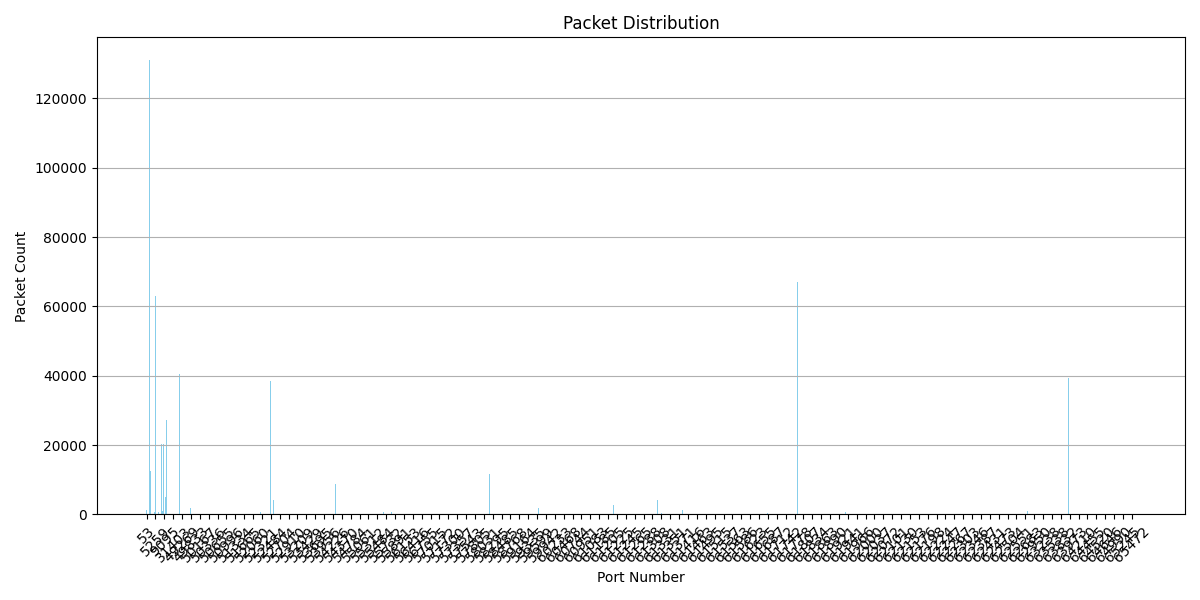


* Packet data PS1

1. Visualize packet distribution by port numbers.

import pyshark  
import matplotlib.pyplot as plt  
from collections import Counter  
  
  
def capture\_packets(pcap\_path, tshark\_path):  
 *"""  
 Captures packets from a pcap file and returns a pyshark FileCapture object.  
 """* return pyshark.FileCapture(pcap\_path, tshark\_path=tshark\_path, keep\_packets=True)  
  
  
def count\_port\_usage(packet\_capture):  
 *"""  
 Counts the usage frequency of destination ports in the captured packets.  
 """* port\_counter = Counter()  
 for packet in packet\_capture:  
 if 'TCP' in packet or 'UDP' in packet:  
 transport\_layer = packet.tcp if 'TCP' in packet else packet.udp  
 port\_counter[transport\_layer.dstport] += 1  
 return port\_counter  
  
  
def plot\_port\_distribution(ports, counts):  
 *"""  
 Plots a bar chart showing the distribution of packet counts per port number.  
 """* plt.figure(figsize=(12, 6))  
 plt.bar(ports, counts, color='skyblue')  
 plt.xlabel('Port Number')  
 plt.ylabel('Packet Count')  
 plt.title('Packet Distribution Across Ports')  
 plt.grid(axis='y')  
  
 # Display every nth label to avoid clutter  
 label\_step = 20  
 plt.xticks([port for i, port in enumerate(ports) if i % label\_step == 0], rotation=45)  
 plt.tight\_layout()  
 plt.show()  
  
  
# Paths to the tshark executable and pcap file  
tshark\_executable = 'D:\\wireshark\\tshark.exe'  
pcap\_file = 'files/final.pcap'  
  
# Capture and count port usage  
packet\_capture = capture\_packets(pcap\_file, tshark\_executable)  
port\_usage = count\_port\_usage(packet\_capture)  
packet\_capture.close()  
  
# Prepare data for plotting  
sorted\_ports, packet\_counts = zip(\*sorted(port\_usage.items(), key=lambda x: int(x[0])))  
  
# Plot distribution of packets across ports  
plot\_port\_distribution(sorted\_ports, packet\_counts)

This code is designed to analyze network traffic in a pcap (packet capture) file, with a particular focus on the distribution of traffic across different network ports. It first reads and parses the pcap file, identifying the destination ports of TCP and UDP packets. Then, the code calculates the number of packets for each port. After processing, it visualizes this data in the form of a bar chart, displaying the packet count for each port.



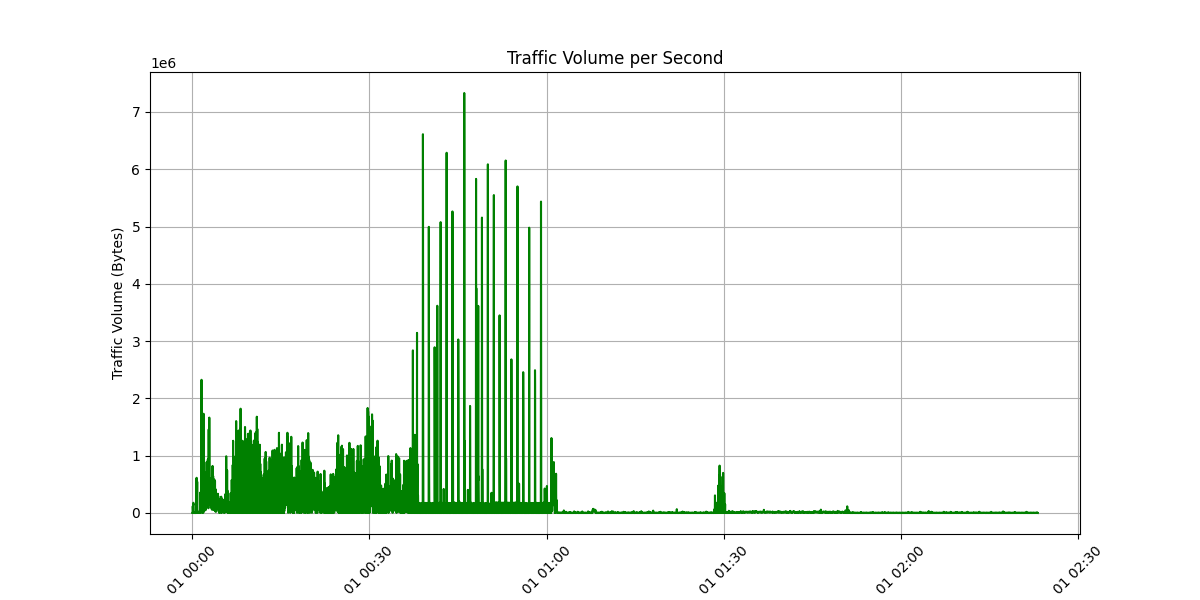
The chart exhibits several ports with notably high traffic, and these peaks likely correspond to commonly used, well-known ports typically employed for popular services or protocols. The packet count for the majority of port numbers is very low. This indicates that only a few ports are heavily utilized within this network. The distribution shows that network traffic is concentrated on common ports, which is typical in most network environments where certain services (such as HTTP, HTTPS, FTP, SMTP, etc.) dominate the traffic.

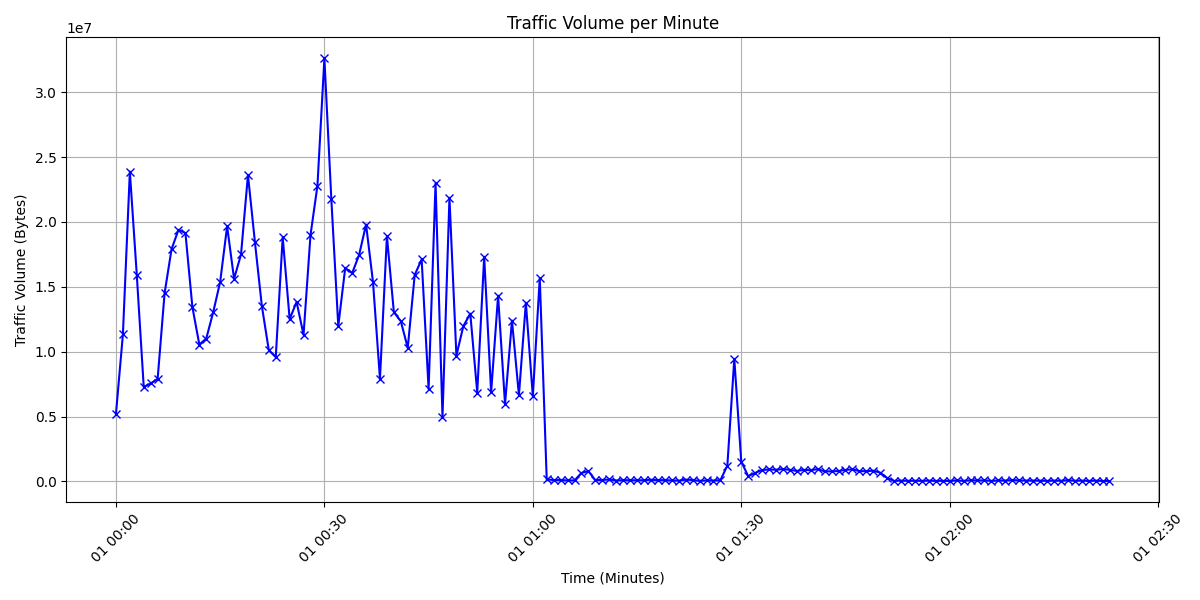
1. Plot traffic volume as a function of time with at least two sufficiently different time scales.

import pandas as pd  
import matplotlib.pyplot as plt  
  
def read\_and\_process\_traffic\_data(file\_path):  
 *"""  
 Reads traffic data from a CSV file and converts the 'Time' column to datetime.  
 """* data = pd.read\_csv(file\_path)  
 data['Time'] = pd.to\_datetime(data['Time'], unit='s')  
 return data  
  
def aggregate\_traffic(data, resample\_period):  
 *"""  
 Aggregates traffic data by summing the 'Length' column for the given resampling period.  
 """* return data.groupby('Time')['Length'].sum().reset\_index().set\_index('Time').resample(resample\_period).sum().fillna(0)  
  
def plot\_traffic(traffic, title, color, marker):  
 *"""  
 Plots the traffic volume as a function of time.  
 """* plt.figure(figsize=(10, 6))  
 plt.plot(traffic.index, traffic['Length'], color=color, marker=marker)  
 plt.grid(True)  
 plt.title(f'Traffic Volume per {title}')  
 plt.xlabel(f'Time ({title})')  
 plt.ylabel('Traffic Volume (Bytes)')  
 plt.xticks(rotation=45)  
 plt.tight\_layout()  
  
csv\_file\_path = 'files/final.csv'  
  
traffic\_data = read\_and\_process\_traffic\_data(csv\_file\_path)  
  
traffic\_data\_by\_second = aggregate\_traffic(traffic\_data, 'S')  
traffic\_data\_by\_minute = aggregate\_traffic(traffic\_data, 'T')

plot\_traffic(traffic\_data\_by\_second, 'Second', 'green', 'o')  
plot\_traffic(traffic\_data\_by\_minute, 'Minute', 'blue', 'x')  
  
plt.show()

This code reads a CSV file that contains timestamps and lengths of data packets. It then resamples this data by second and by minute to plot graphs showing how the traffic volume changes over time.





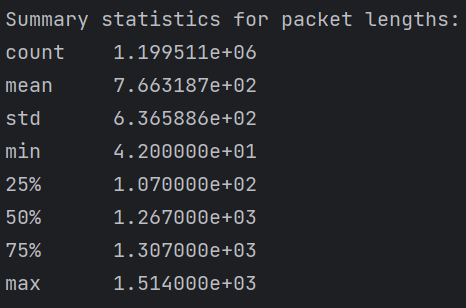
The graph with a second time scale shows a very detailed picture of traffic fluctuations. It is evident that at certain specific moments, the traffic reaches very high peaks. These peaks might indicate sudden surges in traffic, potentially caused by bursts of data packets or concentrated network activities. The volatility of the traffic is quite intense, suggesting that the network may face congestion and instability at these points in time.

When presented with a minute time scale, the same data is shown in a much smoother view. At this scale, the instantaneous peaks in traffic are less pronounced, but the patterns of fluctuation are still identifiable. The appearance of high traffic peaks at intervals suggests that the increase in traffic might have a periodic characteristic.

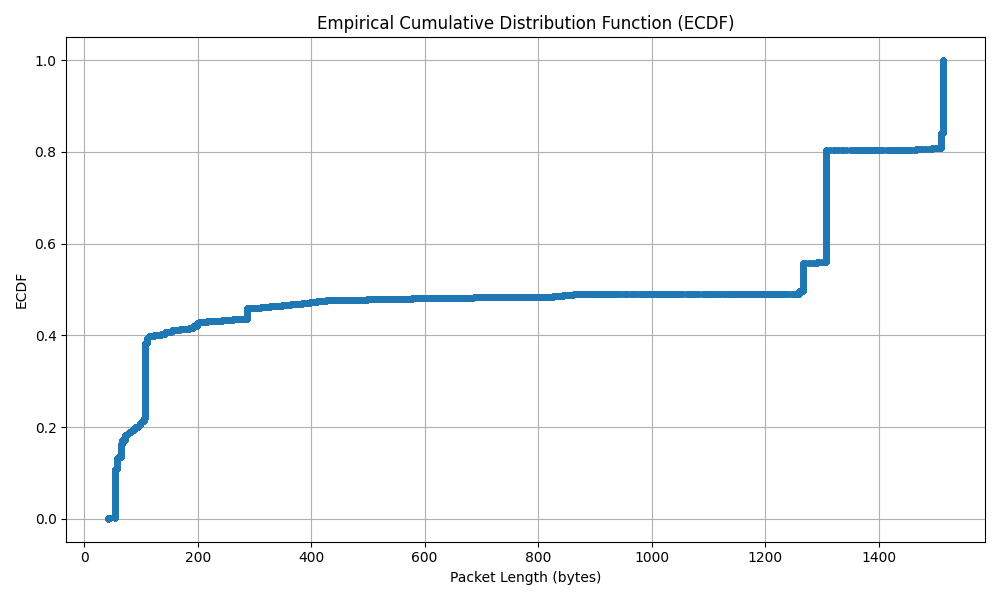
1. Plot packet length distribution (use bins of width 1 byte), its empirical cumulative distribution function and key summary statistics.

import pandas as pd  
import matplotlib.pyplot as plt  
import numpy as np  
  
def load\_data(file\_path):return pd.read\_csv(file\_path)  
  
def plot\_histogram(data, column, bin\_size):plt.figure(figsize=(10, 6))  
 bins\_range = range(min(data[column]), max(data[column]) + bin\_size, bin\_size)  
 plt.hist(data[column], bins=bins\_range, color='blue', alpha=0.7, log=True)  
 plt.grid(axis='y')  
 plt.title('Packet Length Distribution')  
 plt.xlabel('Packet Length (bytes)')  
 plt.ylabel('Frequency (Log Scale)')  
 plt.show()  
  
def plot\_ecdf(data, column):plt.figure(figsize=(10, 6))  
 sorted\_data = np.sort(data[column])  
 y\_values = np.arange(1, len(sorted\_data) + 1) / len(sorted\_data)  
 plt.plot(sorted\_data, y\_values, marker='.', linestyle='none', color='blue')  
 plt.grid(True)  
 plt.title('Empirical Cumulative Distribution Function (ECDF)')  
 plt.xlabel('Packet Length (bytes)')  
 plt.ylabel('ECDF')  
 plt.tight\_layout()  
 plt.show()  
  
def print\_summary\_statistics(data, column):print(f"Summary statistics for {column}:")  
 print(data[column].describe())  
  
# Main code  
csv\_file\_path = 'files/final.csv'  
packet\_data = load\_data(csv\_file\_path)  
  
# Plot the histogram of packet lengths  
plot\_histogram(packet\_data, 'Length', bin\_width=1)  
  
# Plot the ECDF of packet lengths  
plot\_ecdf(packet\_data, 'Length')  
  
# Print summary statistics for packet lengths  
print\_summary\_statistics(packet\_data, 'Length')

This code is primarily used for analyzing and visualizing the distribution of network packet lengths. It achieves this by presenting the data in different aspects through a histogram and an Empirical Cumulative Distribution Function (ECDF) plot. Additionally, it provides some basic descriptive statistics about the data.







The first graph presents the distribution of packet lengths, allowing us to observe the spread of packet sizes in the measured network traffic. The logarithmic scale highlights that certain packet lengths are more common than others, with noticeable peaks at specific lengths.

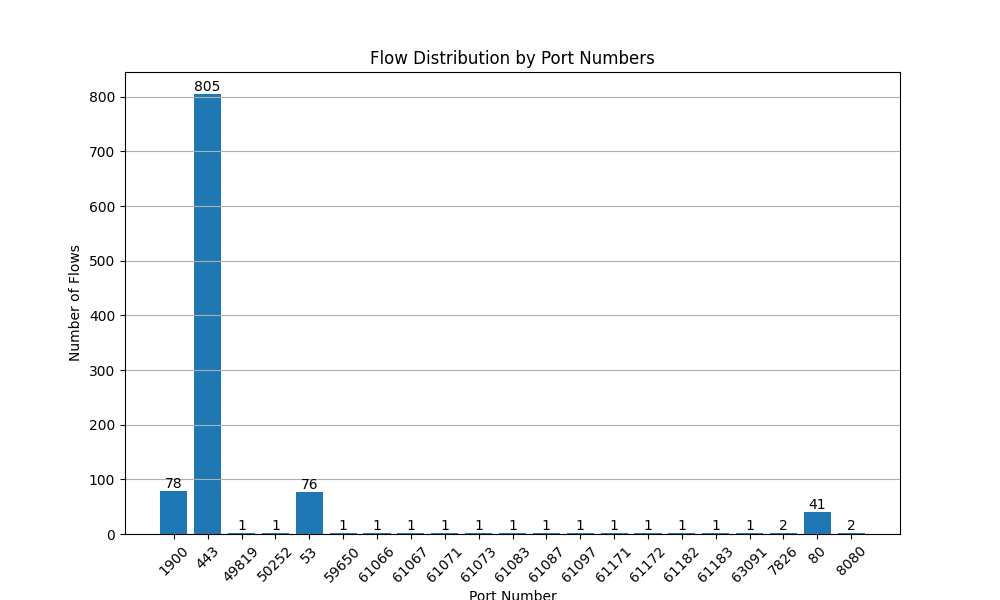
The second graph illustrates the empirical cumulative distribution function (ECDF) of packet lengths. This function provides a cumulative total that indicates the proportion of packets that are less than a certain length. The ECDF shows sharp increases at certain points, corresponding with the peaks observed in the packet length distribution graph.

* Flow data PS2

1. Visualize flow distribution by port numbers.

import pandas as pd  
import matplotlib.pyplot as plt  
  
def convert\_to\_bytes(row):  
 units = {'bytes': 1, 'kb': 1024, 'mb': 1024\*\*2}  
  
 try:  
 load\_bytes\_unit = str(row.get('LD\_Bytes\_Unit', 'bytes')).lower()  
 load\_factor = units[load\_bytes\_unit]  
 load\_bytes = row.get('LD\_Bytes', 0) \* load\_factor  
  
 receive\_bytes\_unit = str(row.get('RD\_Bytes\_Unit', 'bytes')).lower()  
 receive\_factor = units[receive\_bytes\_unit]  
 receive\_bytes = row.get('RD\_Bytes', 0) \* receive\_factor  
  
 total\_bytes\_unit = str(row.get('Total\_Bytes\_Unit', 'bytes')).lower()  
 total\_factor = units[total\_bytes\_unit]  
 total\_bytes = row.get('Total\_Bytes', 0) \* total\_factor  
  
 return pd.Series({  
 'Load\_Bytes': load\_bytes,  
 'Receive\_Bytes': receive\_bytes,  
 'Total\_Bytes': total\_bytes,  
 'Server\_IP': row.get('Destination\_IP', '')  
 })  
  
 except KeyError as e:  
 print(f"Error processing row {row}: {e}")  
 raise ValueError("Invalid unit. Supported units are 'bytes', 'kb', 'mb.")  
  
# Reading and preparing data  
data\_file = 'files/final.txt'  
data\_columns = ["Source\_IP", "Arrow", "Destination\_IP", "LD\_Frames", "LD\_Bytes", "LD\_Bytes\_Unit",  
 "RD\_Frames", "RD\_Bytes", "RD\_Bytes\_Unit", "Total\_Frames", "Total\_Bytes", "Total\_Bytes\_Unit",  
 "Start", "Duration"]  
  
df = pd.read\_csv(data\_file, sep='\s+', skiprows=5, header=None, skipfooter=1, engine='python')  
df.columns = data\_columns  
df = df.assign(\*\*df.apply(convert\_to\_bytes, axis=1))  
df['Port'] = df['Destination\_IP'].str.split(':').str[1].astype(str)  
  
port\_flow\_distribution = df.groupby('Port').size()  
  
plt.figure(figsize=(10, 6))  
bar\_chart = plt.bar(port\_flow\_distribution.index, port\_flow\_distribution.values)  
  
for bar in bar\_chart:  
 height = bar.get\_height()  
 plt.text(bar.get\_x() + bar.get\_width() / 2, height, int(height), va='bottom', ha='center')  
  
plt.xlabel('Port Number')  
plt.ylabel('Number of Flows')  
plt.title('Flow Distribution by Port Numbers')  
plt.xticks(rotation=45)  
plt.grid(axis='y')  
plt.show()

This code is primarily used for analyzing and visualizing network traffic data. Initially, it reads data from a file and converts it into an appropriate format, such as converting the size of data packets from various units into bytes. Then, the code groups this data by port number and calculates the amount of traffic on each port. Finally, it uses the matplotlib library to draw a bar chart, displaying the distribution of traffic across different ports.

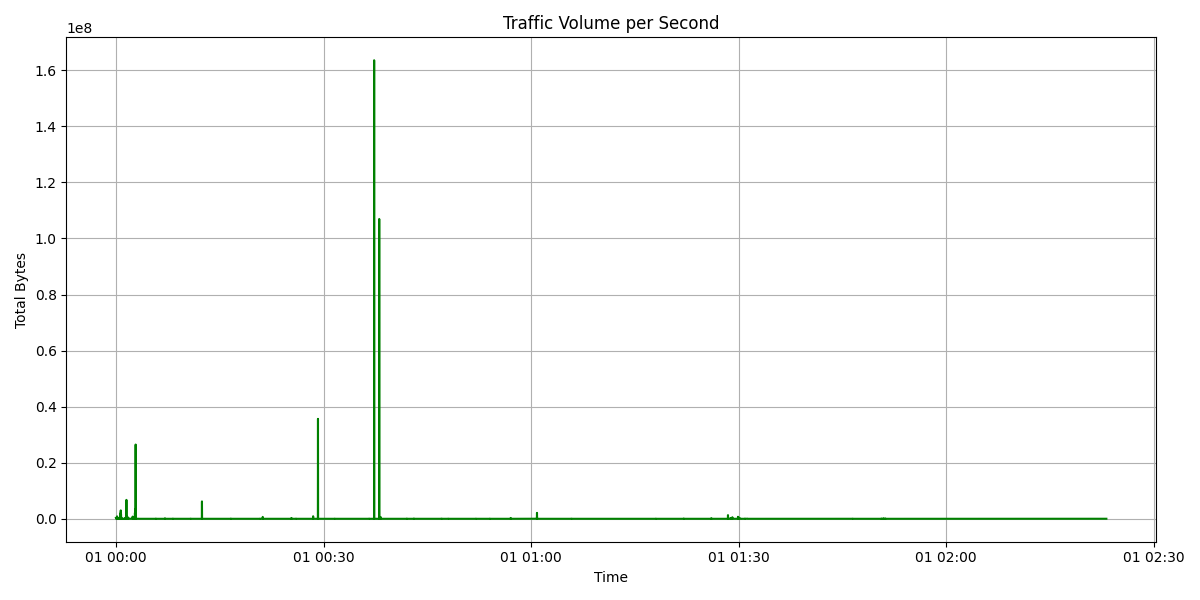


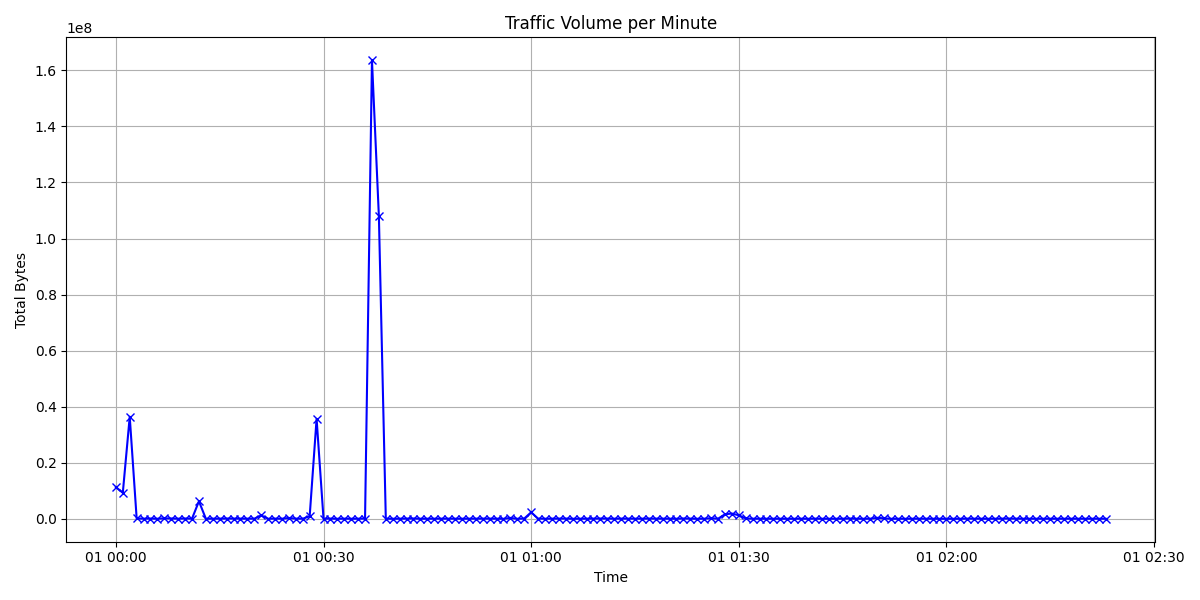
This chart illustrates the distribution of network flows categorized by port numbers. It is evident that the traffic on certain ports is significantly higher than on others. Most notably, port 443 shows over 800 flows, which is commonly used for secure web transmission (HTTPS), suggesting a substantial amount of secure web traffic. Port 80 also displays a higher volume, with about 41 flows, typically associated with HTTP service, indicative of unencrypted web traffic. Other ports such as 22, usually used for SSH, and 3389, for Remote Desktop Protocol, have comparatively lower traffic, yet their presence indicates remote administration activities on the network. The majority of other ports have only 1 or 2 flows, pointing to specific, non-standard services or sporadic network communications.

1. Plot traffic volume as a function of time with at least two sufficiently different time scales.

import pandas as pd  
import matplotlib.pyplot as plt  
import re  
import datetime  
  
def parse\_line(line):  
 match = re.search(r'(\d+\.\d+\.\d+\.\d+:\d+)\s+<->\s+(\d+\.\d+\.\d+\.\d+:\d+)\s+(\d+)\s+(\d+\s+\w+)\s+(\d+)\s+(\d+\s+\w+)\s+(\d+)\s+(\d+\s+\w+)\s+(\d+\.\d+)', line)  
 if match:  
 return {  
 "Source\_IP": match.group(1),  
 "Destination\_IP": match.group(2),  
 "Upload\_Frames": int(match.group(3)),  
 "Upload\_Bytes": match.group(4),  
 "Download\_Frames": int(match.group(5)),  
 "Download\_Bytes": match.group(6),  
 "Total\_Frames": int(match.group(7)),  
 "Total\_Bytes": match.group(8),  
 "Start": float(match.group(9))  
 }  
 else:  
 return None  
  
# Function to convert the total bytes to a uniform unit (bytes)  
def convert\_bytes(byte\_str):  
 number, unit = byte\_str.split()  
 number = float(number)  
 unit = unit.lower()  
 if unit == 'kb':  
 return number \* 1024  
 elif unit == 'mb':  
 return number \* 1024 \* 1024  
 elif unit == 'gb':  
 return number \* 1024 \* 1024 \* 1024  
 else:  
 return number  
  
# Parsing the entire file  
file\_path = 'files/final.txt' # Replace with your file path  
parsed\_full\_data = []  
with open(file\_path, 'r') as file:  
 for \_ in range(5): # Skipping the first five lines  
 next(file)  
 for line in file: # Parsing each line in the file  
 parsed\_line = parse\_line(line)  
 if parsed\_line:  
 parsed\_line["Total\_Bytes"] = convert\_bytes(parsed\_line["Total\_Bytes"])  
 parsed\_full\_data.append(parsed\_line)  
  
# Convert the parsed data to a DataFrame  
full\_df = pd.DataFrame(parsed\_full\_data)  
  
# Convert the 'Start' column to a datetime format using a reference date  
reference\_date = datetime.datetime(1970, 1, 1)  
full\_df['Start'] = pd.to\_datetime(full\_df['Start'], unit='s', origin=reference\_date)  
  
# Resampling data to seconds  
df\_seconds = full\_df.resample('1S', on='Start').sum()  
  
# Plotting the data for second-wise  
plt.figure(figsize=(12, 6))  
plt.plot(df\_seconds.index, df\_seconds['Total\_Bytes'], color='green', marker='o')  
plt.title('Traffic Volume per Second')  
plt.xlabel('Time')  
plt.ylabel('Total Bytes')  
plt.grid(True)  
plt.tight\_layout()  
plt.show()  
  
df\_minutes = full\_df.resample('1T', on='Start').sum()  
  
plt.figure(figsize=(12, 6))  
plt.plot(df\_minutes.index, df\_minutes['Total\_Bytes'], color='blue', marker='x')  
plt.title('Traffic Volume per Minute')  
plt.xlabel('Time')  
plt.ylabel('Total Bytes')  
plt.grid(True)  
plt.tight\_layout()  
plt.show()

This code functions to read network traffic data from a CSV file and convert it into a time series format. Subsequently, it groups the data by timestamps and calculates the total traffic for each time period, measured in seconds and minutes. Finally, the code uses the matplotlib library to plot the network traffic variations per second and per minute, respectively.





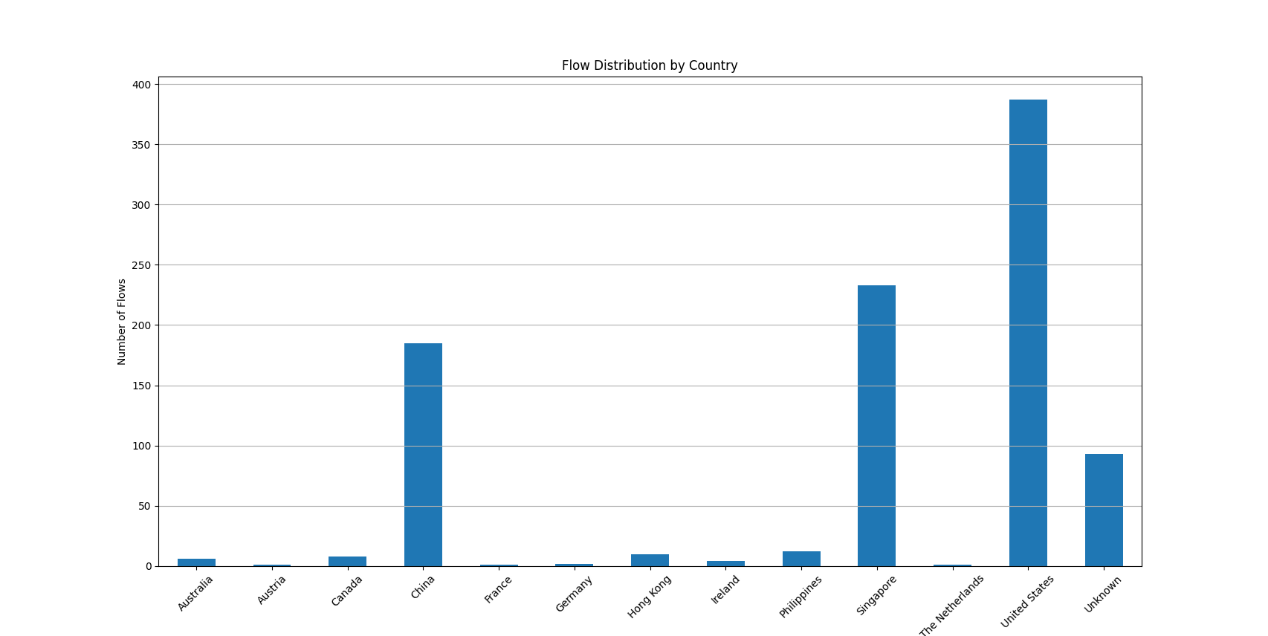
In the first graph, we observe a dramatic increase in traffic volume at specific seconds, indicating moments of high activity or data transfer. The most significant peak is close to 1.6e8 bytes, suggesting a momentary but substantial surge in traffic. Aside from these peaks, the pattern shows relatively low traffic volume, indicating that such high volumes are not sustained over long periods.

The second graph provides a minute-by-minute aggregation of traffic volume. Here, peaks are less pronounced due to the aggregation over each minute, but we still observe peaks at specific intervals, including one that corresponds to the highest peak seen in the per-second graph. This minute-scale view smooths out the extreme fluctuations seen on a per-second basis, yet still highlights moments of intense activity.

1. Visualize flow distribution by country.

import pandas as pd  
import matplotlib.pyplot as plt  
from geoip2.database import Reader  
  
def convert\_bytes(row):  
 units = {'bytes': 1, 'kb': 1024, 'mb': 1024\*\*2}  
  
 try:  
 upload\_unit = str(row['Upload\_Bytes\_Unit']).lower()  
 upload\_factor = units[upload\_unit]  
 upload\_bytes = row['Upload\_Bytes'] \* upload\_factor  
  
 download\_unit = str(row['Download\_Bytes\_Unit']).lower()  
 download\_factor = units[download\_unit]  
 download\_bytes = row['Download\_Bytes'] \* download\_factor  
  
 total\_unit = str(row['Total\_Bytes\_Unit']).lower()  
 total\_factor = units[total\_unit]  
 total\_bytes = row['Total\_Bytes'] \* total\_factor  
  
 return pd.Series({  
 'Upload\_Bytes': upload\_bytes,  
 'Download\_Bytes': download\_bytes,  
 'Total\_Bytes': total\_bytes,  
 'Server\_IP': row['Destination\_IP']  
 })  
 except KeyError as e:  
 print(f"Error processing row {row}: {e}")  
 raise ValueError("Invalid unit. Supported units are 'bytes', 'kb', 'mb.")  
  
data\_file = 'files/final.txt'  
column\_names = [  
 "Source\_IP", "Arrow", "Destination\_IP", "Upload\_Frames", "Upload\_Bytes", "Upload\_Bytes\_Unit",  
 "Download\_Frames", "Download\_Bytes", "Download\_Bytes\_Unit", "Total\_Frames", "Total\_Bytes", "Total\_Bytes\_Unit",  
 "Start", "Duration"  
]  
  
traffic\_data = pd.read\_csv(data\_file, sep='\s+', skiprows=5, header=None, skipfooter=1, engine='python')  
traffic\_data.columns = column\_names  
traffic\_data = traffic\_data.assign(\*\*traffic\_data.apply(convert\_bytes, axis=1))  
geoip\_file = 'others/GeoLite2-Country.mmdb'  
geoip\_reader = Reader(geoip\_file)  
  
def get\_country(ip):  
 try:  
 response = geoip\_reader.country(ip)  
 return response.country.name  
 except:  
 return "Unknown"  
  
traffic\_data['Country'] = traffic\_data['Destination\_IP'].str.split(':').str[0].apply(get\_country)  
  
country\_traffic\_distribution = traffic\_data.groupby('Country').size()  
plt.figure(figsize=(11, 7.5))  
country\_traffic\_distribution.plot(kind='bar')  
plt.grid(axis='y')  
plt.xlabel('Country')  
plt.ylabel('Number of Flows')  
plt.title('Flow Distribution by Country')  
plt.xticks(rotation=45)  
plt.show()

This code is primarily focused on analyzing and visualizing network traffic data, with a specific emphasis on the geographical distribution of traffic. It reads network traffic data from a text file, including the size of data packets and their destination IP addresses. The code then uniformly converts the size of data packets from various units into bytes and uses a GeoIP database to map each IP address to its corresponding country. Finally, the code uses the matplotlib library to create a bar chart, displaying the distribution of network traffic across different countries.



The bar chart visualizes the distribution of network flows by country. The United States has the highest number of flows, followed by the Netherlands and China, indicating a higher level of internet traffic activity or more network interactions originating from or destined to these countries.

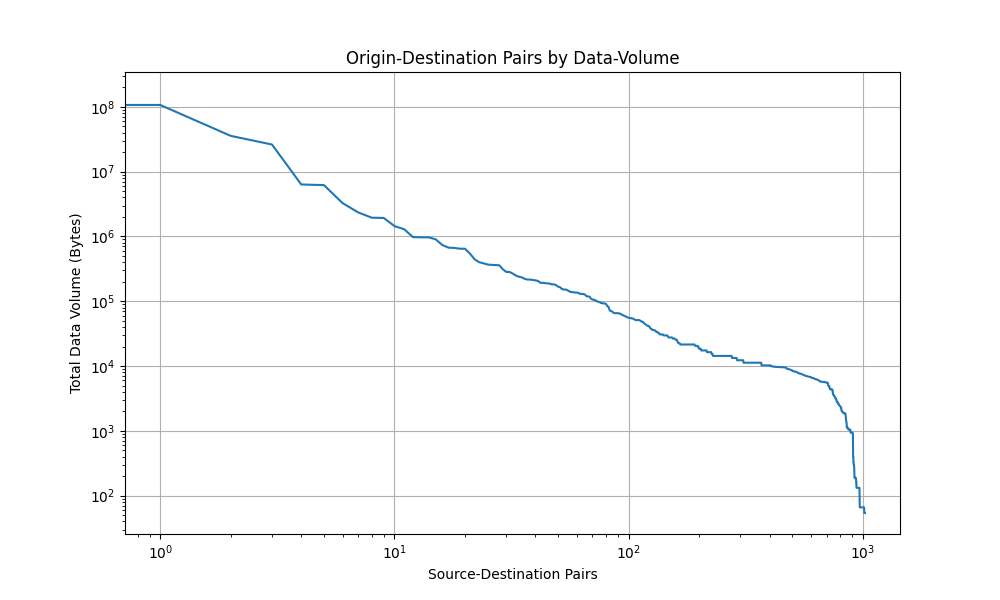
1. Plot origin-destination pairs both by data volume and by flows (Zipf type plot).

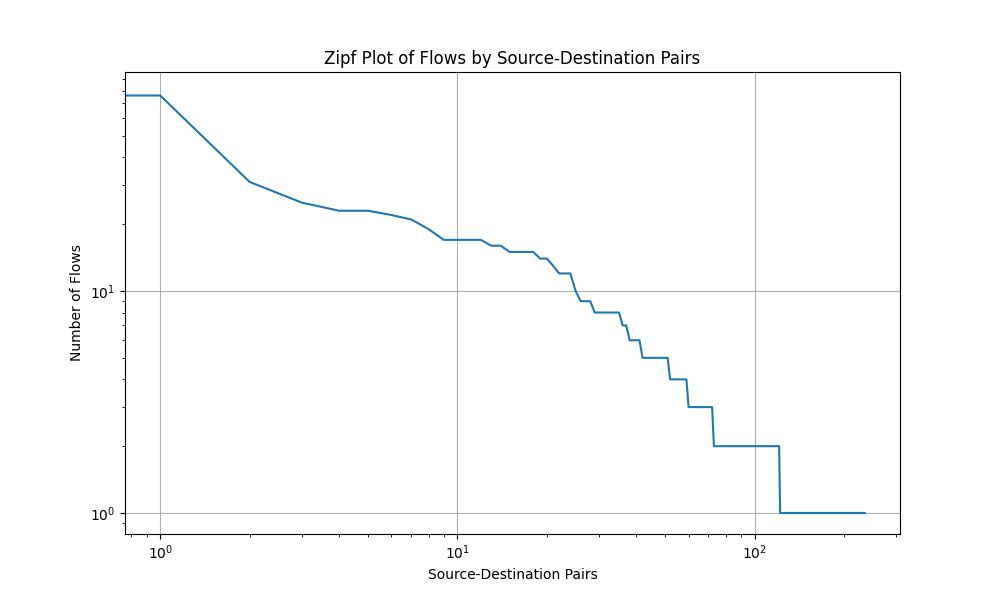
import pandas as pd  
import matplotlib.pyplot as plt  
  
def convert\_units\_to\_bytes(row):  
 units = {'bytes': 1, 'kb': 1024, 'mb': 1024\*\*2}  
  
 try:  
 upload\_unit = str(row['Upload\_Unit']).lower()  
 upload\_factor = units[upload\_unit]  
 upload\_bytes = row['Upload\_Bytes'] \* upload\_factor  
  
 download\_unit = str(row['Download\_Unit']).lower()  
 download\_factor = units[download\_unit]  
 download\_bytes = row['Download\_Bytes'] \* download\_factor  
  
 total\_unit = str(row['Total\_Unit']).lower()  
 total\_factor = units[total\_unit]  
 total\_bytes = row['Total\_Bytes'] \* total\_factor  
  
 return pd.Series({  
 'Upload\_Bytes': upload\_bytes,  
 'Download\_Bytes': download\_bytes,  
 'Total\_Bytes': total\_bytes,  
 'Destination\_IP': row['Destination\_IP']  
 })  
 except KeyError as e:  
 print(f"Error processing row {row}: {e}")  
 raise ValueError("Invalid unit. Supported units are 'bytes', 'kb', 'mb.")  
  
data\_file = 'files/final.txt'  
column\_names = [  
 "Source\_IP", "Arrow", "Destination\_IP", "Upload\_Frames", "Upload\_Bytes", "Upload\_Unit",  
 "Download\_Frames", "Download\_Bytes", "Download\_Unit", "Total\_Frames", "Total\_Bytes", "Total\_Unit",  
 "Start\_Time", "Duration"]  
  
network\_data = pd.read\_csv(data\_file, sep='\s+', skiprows=5, header=None, skipfooter=1, engine='python')  
network\_data.columns = column\_names  
network\_data = network\_data.assign(\*\*network\_data.apply(convert\_units\_to\_bytes, axis=1))  
  
network\_data['Src\_Dst\_Pair'] = network\_data['Source\_IP'] + ' - ' + network\_data['Destination\_IP']  
  
pairwise\_traffic\_data = network\_data.groupby('Src\_Dst\_Pair')['Total\_Bytes'].sum()  
pairwise\_flow\_counts = network\_data.groupby('Src\_Dst\_Pair').size()  
sorted\_traffic = pairwise\_traffic\_data.sort\_values(ascending=False)  
sorted\_flows = pairwise\_flow\_counts.sort\_values(ascending=False)  
  
plt.figure(figsize=(10, 6))  
plt.plot(sorted\_traffic.values)  
plt.grid(True)  
plt.xlabel('Origin-Destination Pairs')  
plt.ylabel('Data Volume (Bytes)')  
plt.title('Origin-destination pairs by data volume')

plt.yscale('log')  
plt.xscale('log')  
plt.show()  
  
plt.figure(figsize=(10, 6))  
plt.plot(sorted\_flows.values)  
plt.grid(True)  
plt.xlabel('Origin-Destination Pairs')  
plt.ylabel('Flow Counts')  
plt.title('Origin-destination pairs by flows')

plt.yscale('log')  
plt.xscale('log')  
plt.show()

This code functions to read network traffic data from a text file and processes it for analysis. Initially, it converts the size of data packets from various units into bytes and associates each packet with its corresponding source-destination IP pair. The code then calculates the total data volume and the number of flows for each source-destination pair, sorts this data, and plots two distributions: one showing the data volume distribution sorted by source-destination pairs, and another depicting the distribution of the number of flows.





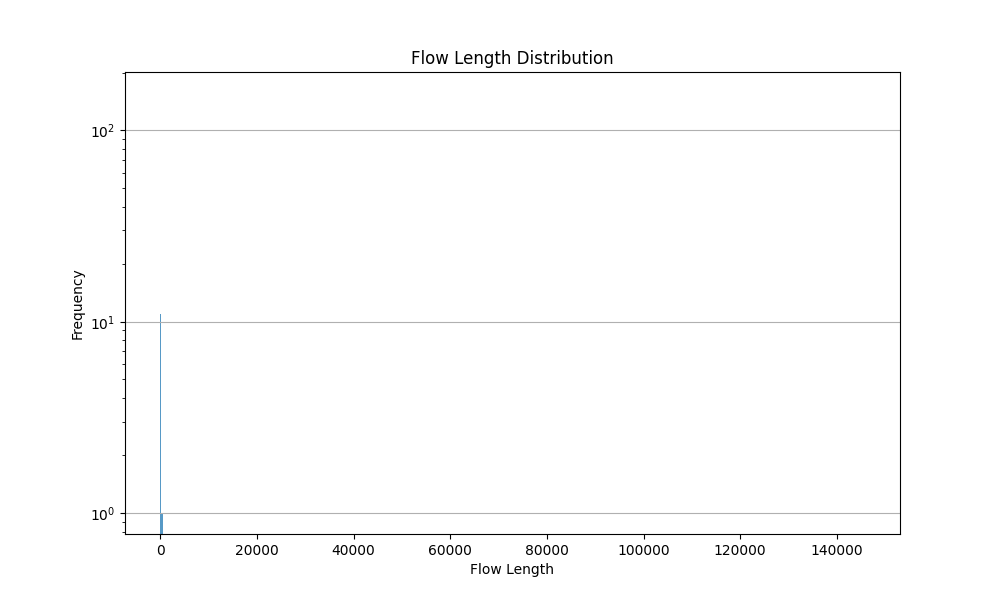
The first graph displays the distribution of network traffic among origin-destination pairs by data volume, where a small number of pairs account for a large portion of the total data volume, indicating that the majority of data transmission in the network is concentrated among a few pairs. The second graph, plotted by the number of flows, shows a distribution that also follows a Zipf-like pattern. This demonstrates that a few origin-destination pairs have a high number of flows, while the majority of pairs have relatively few. This is a typical characteristic of networks where a small number of hosts or services manage most connections.

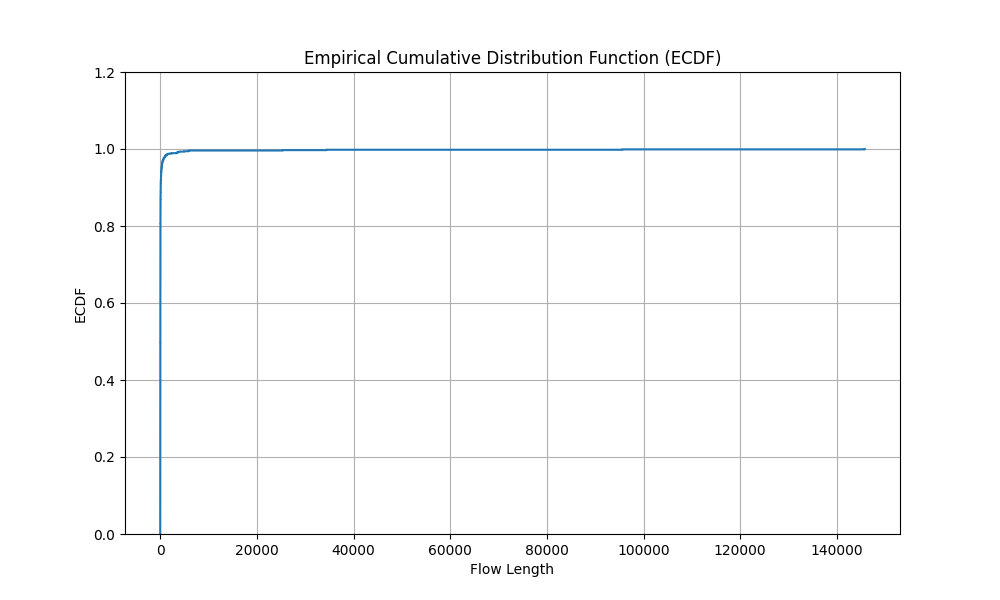
1. Plot flow length distribution, its empirical cumulative distribution function and key summary statistics.

import pandas as pd  
import matplotlib.pyplot as plt  
import seaborn as sns  
  
def convert\_to\_kilobytes(row):  
 units = {'bytes': 1, 'kb': 1024, 'mb': 1024 \*\* 2}  
 converted\_data = {}  
  
 for column\_prefix in ['ld', 'rd', 'total']:  
 bytes\_value = row[f'{column\_prefix}\_bytes']  
 unit = str(row[f'{column\_prefix}\_bytes\_unit']).lower()  
  
 try:  
 factor = units[unit]  
 except KeyError:  
 raise ValueError(  
 f"Invalid unit '{unit}' for {column\_prefix}\_bytes. Supported units are 'bytes', 'kb', 'mb'.")  
  
 converted\_data[f'{column\_prefix}\_kilobytes'] = bytes\_value \* factor  
  
 return pd.Series({\*\*converted\_data, 'server\_ip': row['second\_ip']})  
  
  
# Read and process data  
data\_file = 'files/final.txt'  
data\_columns = ["first\_ip", "->", "second\_ip", "ld\_frames", "ld\_bytes", "ld\_bytes\_unit",  
 "rd\_frames", "rd\_bytes", "rd\_bytes\_unit", "total\_frames", "total\_bytes", "total\_bytes\_unit",  
 "start\_time", "duration"]  
  
df = pd.read\_csv(data\_file, sep='\s+', skiprows=5, header=None, skipfooter=1, engine='python')  
df.columns = data\_columns  
  
df = df.assign(\*\*df.apply(convert\_to\_kilobytes, axis=1))  
  
plt.figure(figsize=(10, 6))  
sns.histplot(df['total\_frames'], kde=False)  
plt.title('Flow Length Distribution (Total Frames)')  
plt.xlabel('Flow Length (Frames)')  
plt.ylabel('Frequency')  
plt.yscale('log')  
plt.grid(axis='y')  
plt.show()  
  
plt.figure(figsize=(10, 6))  
sns.ecdfplot(df['total\_frames'])  
plt.title('ECDF of Flow Length')  
plt.xlabel('Flow Length (Frames)')  
plt.ylabel('ECDF')

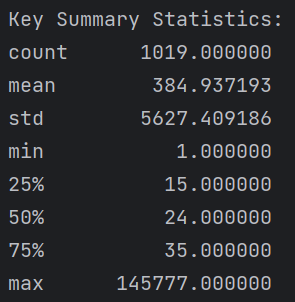
plt.ylim(0, 1.2)  
plt.grid(True)  
plt.show()  
  
print("Key Summary Statistics (Total Frames):")  
print(df['total\_frames'].describe())

This code primarily focuses on reading and processing network traffic data, converting the byte units in the data to kilobytes, and performing a visual analysis. It begins by reading traffic data from a text file and then converts it into a unified kilobyte unit. Following that, the code utilizes histograms and Empirical Cumulative Distribution Function (ECDF) plots to display the distribution of the total number of traffic frames, and it prints out key statistical information about the total frame count.





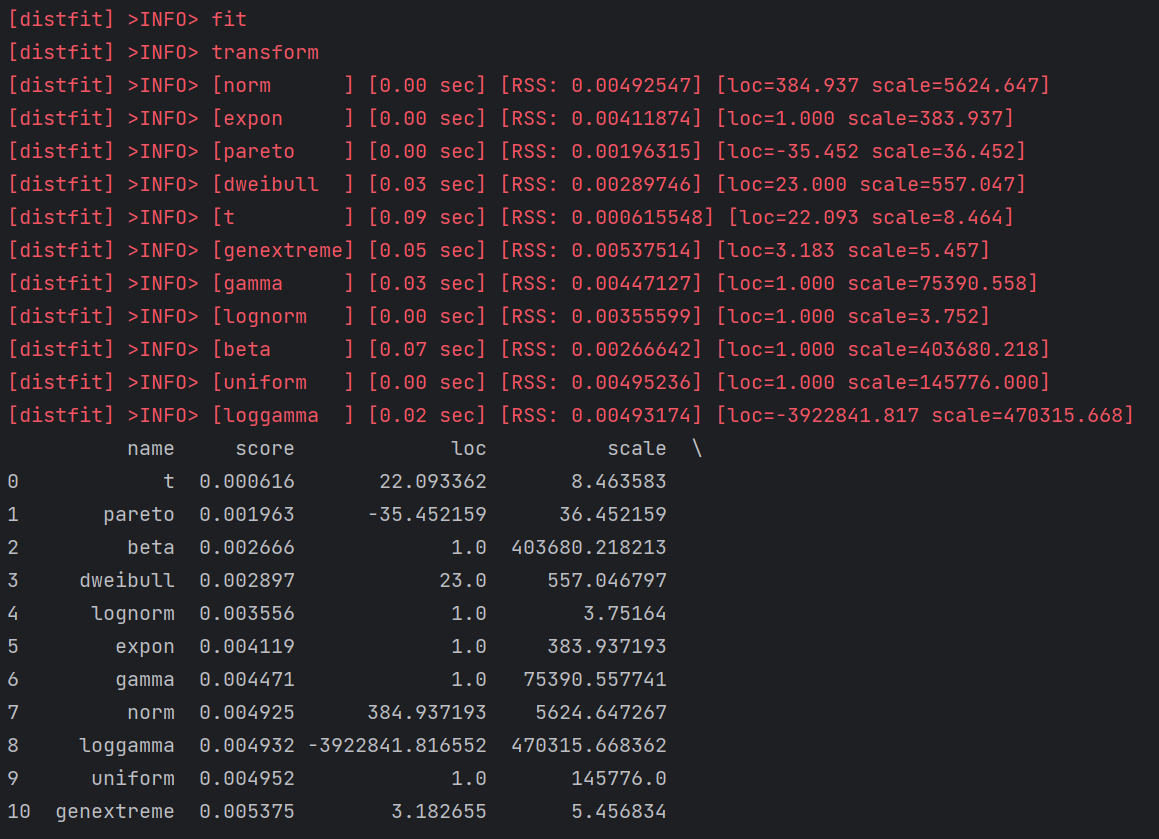
The first graph presents the flow length distribution on a logarithmic scale, showing the frequency of network flows of varying lengths. The distribution is skewed towards shorter flows, indicating that most network flows are relatively short in length. As flow length increases, the frequency significantly decreases, which is typical for many networks where the bulk of flows are small. The second graph depicts the empirical cumulative distribution function (ECDF) for flow lengths. This graph shows that a large proportion of flows are short, as the ECDF rapidly rises to 1 at the lower end of the flow length spectrum.

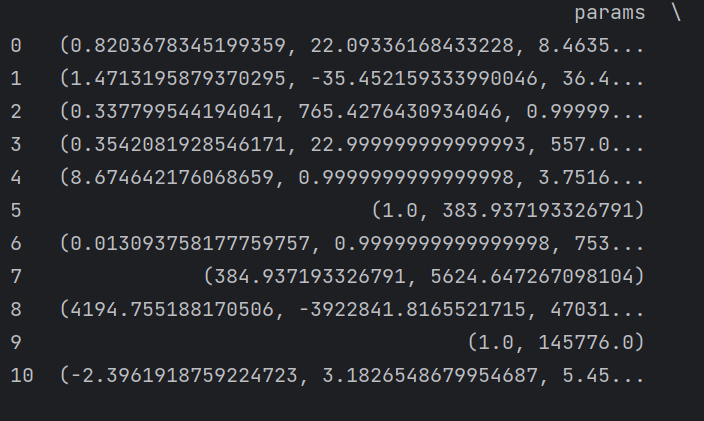
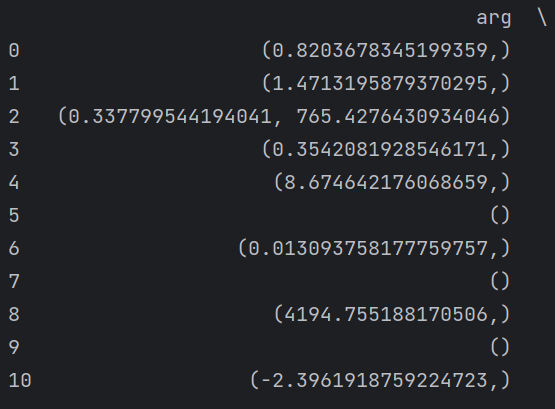
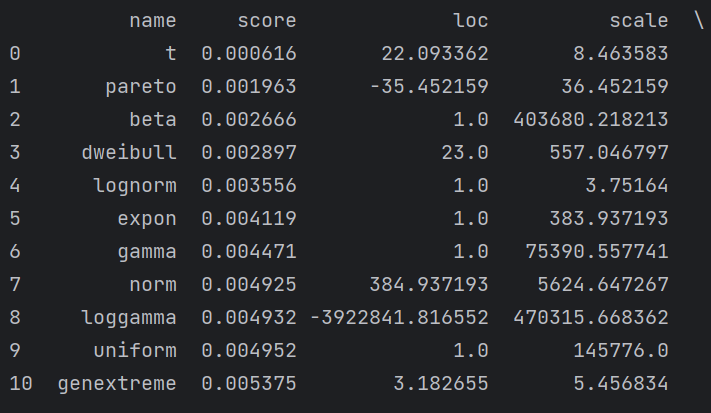


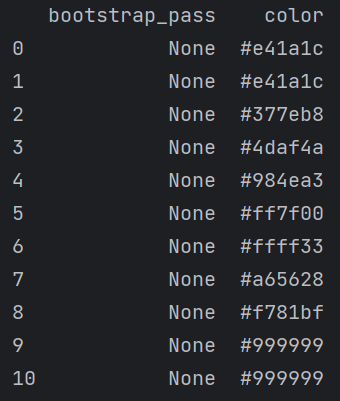
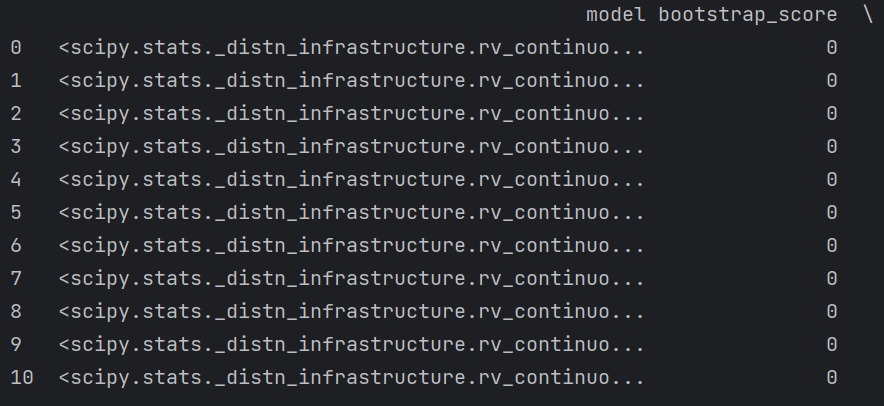
1. Fit a distribution for the flow lengths and validate the model.

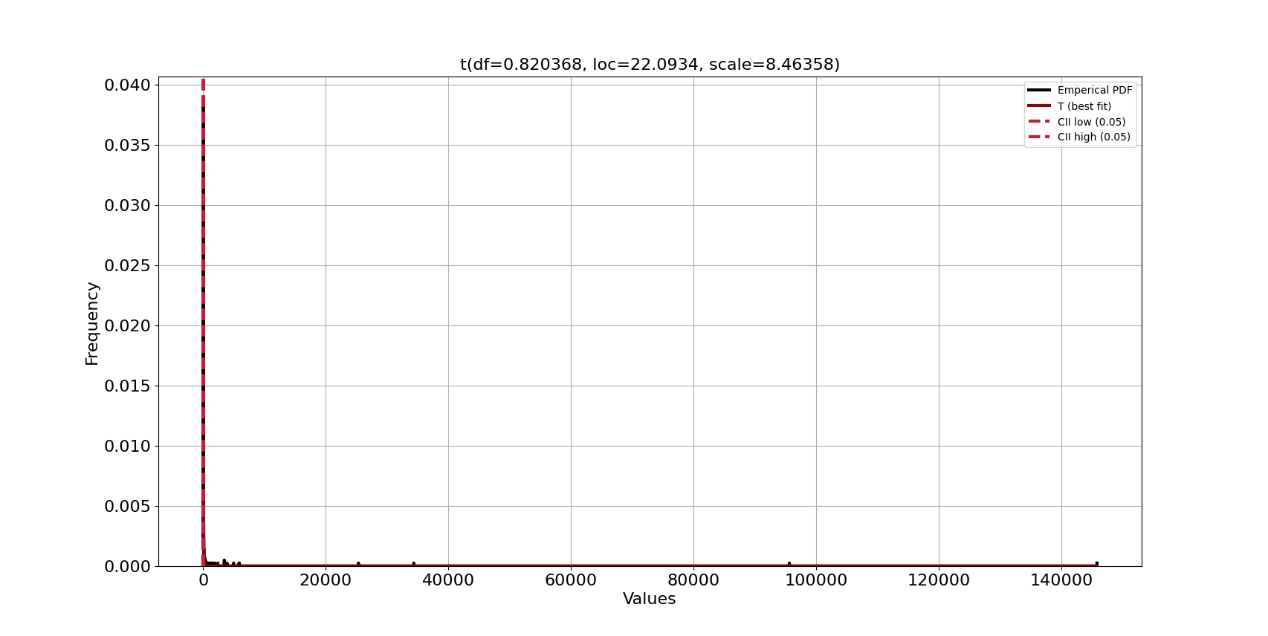
import pandas as pd  
import matplotlib.pyplot as plt  
from distfit import distfit  
  
def convert\_to\_byte(row):  
 units = {'bytes': 1, 'kb': 1024, 'mb': 1024\*\*2}  
  
 try:  
 ld\_bytes\_unit = str(row['ld\_bytes\_unit']).lower()  
 factor = units[ld\_bytes\_unit]  
 ld\_kb = row['ld\_bytes'] \* factor  
  
 rd\_bytes\_unit = str(row['rd\_bytes\_unit']).lower()  
 factor = units[rd\_bytes\_unit]  
 rd\_kb = row['rd\_bytes'] \* factor  
  
 total\_bytes\_unit = str(row['total\_bytes\_unit']).lower()  
 factor = units[total\_bytes\_unit]  
 total\_kb = row['total\_bytes'] \* factor  
  
 return pd.Series({'ld\_bytes': ld\_kb, 'rd\_bytes': rd\_kb, 'total\_bytes': total\_kb, 'server\_ip': row['second\_ip\_interface']})  
 except KeyError as e:  
 print(f"Error processing row {row}: {e}")  
 raise ValueError("Invalid unit. Supported units are 'bytes', 'kb', 'mb.")  
  
df = pd.read\_csv('files/final.txt', sep='\s+', skiprows=5, header=None, skipfooter=1, engine='python')  
  
new\_column\_names = ["first\_ip\_interface", "arrow", "second\_ip\_interface", "ld\_frames", "ld\_bytes", "ld\_bytes\_unit",  
 "rd\_frames", "rd\_bytes", "rd\_bytes\_unit", "total\_frames", "total\_bytes", "total\_bytes\_unit",  
 "start", "duration"]  
  
df.columns = new\_column\_names  
pd.set\_option('display.max\_columns', None)  
df = df.assign(\*\*df.apply(convert\_to\_byte, axis=1))  
data = df['total\_frames']  
dist = distfit()  
dist.fit\_transform(data)  
  
print(dist.summary)  
dist.plot()  
plt.show()

This code is designed to process and analyze network traffic data, with a focus on converting data units and fitting a specific data column to a probability distribution. Initially, it reads network traffic data from a text file, skipping certain lines for formatting purposes, and sets specific column names. Then, it uses a function to convert the data in various columns into a unified byte unit. Next, it employs the distfit library to fit the 'total\_frames' column data to a probability distribution, followed by plotting the best-fit distribution. Finally, a summary is printed, which includes statistical details about the fitted distribution.





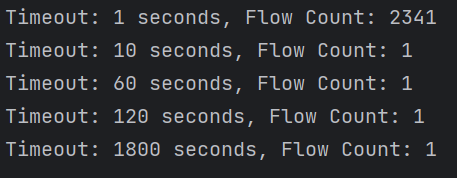




1. Compare the number of flows with 1, 10, 60, 120 and 1800 second timeouts. In this, you need to generate flow data multiple times.

from scapy.all import rdpcap, PacketList  
from datetime import datetime  
  
# Function to analyze TCP flows based on different timeout values  
def analyze\_tcp\_flows(pcap\_file, timeouts):  
 flows\_count = {}  
 for timeout in timeouts:  
 flows = []  
 current\_flow\_packets = []  
 last\_packet\_time = None  
  
 for packet in rdpcap(pcap\_file):  
 if 'IP' in packet and 'TCP' in packet:  
 packet\_time = datetime.fromtimestamp(float(packet.time))  
  
 if last\_packet\_time is None or (packet\_time - last\_packet\_time).seconds > timeout:  
 if current\_flow\_packets:  
 flows.append(PacketList(current\_flow\_packets))  
 current\_flow\_packets = [packet]  
 else:  
 current\_flow\_packets.append(packet)  
  
 last\_packet\_time = packet\_time  
  
 if current\_flow\_packets:  
 flows.append(PacketList(current\_flow\_packets))  
  
 flows\_count[timeout] = len(flows)  
 return flows\_count  
  
# Main code  
pcap\_file\_path = 'files/final.pcap'  
timeout\_seconds = [1, 10, 60, 120, 1800]  
flows\_count\_by\_timeout = analyze\_tcp\_flows(pcap\_file\_path, timeout\_seconds)  
  
for timeout, flow\_count in flows\_count\_by\_timeout.items():  
 print(f"Timeout: {timeout} seconds, Flow Count: {flow\_count}")

This code starts by reading a PCAP file and then iterates through each packet. It uses a series of predefined timeout thresholds (in seconds) to determine whether packets belong to the same flow. If the time difference between two consecutive packets exceeds the current timeout threshold, they are considered to belong to different flows. In this way, the code can count the number of flows for each timeout threshold and print the results.

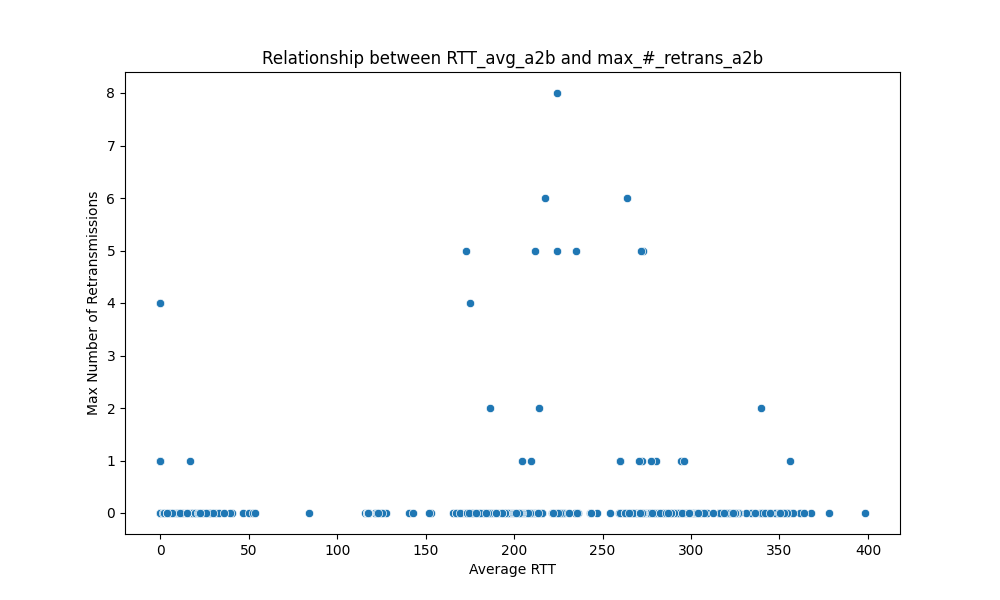


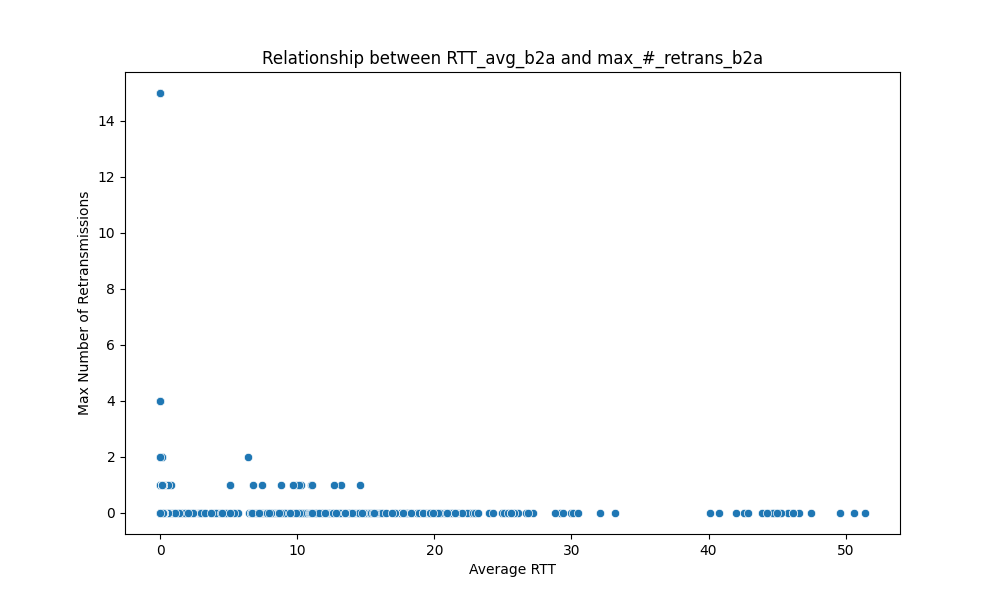
* TCP connection data PS3

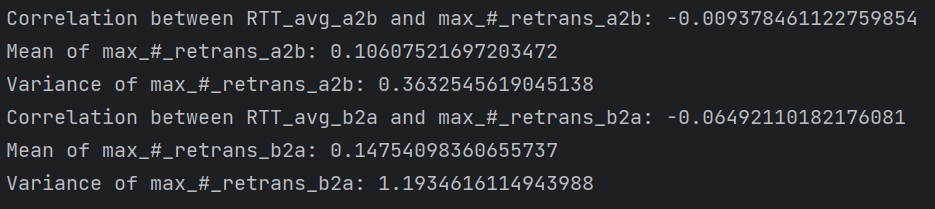
1. Round-trip times and their variance.

import pandas as pd  
import matplotlib.pyplot as plt  
import seaborn as sns  
  
def load\_and\_clean\_data(file\_path, columns\_to\_clean):data = pd.read\_csv(file\_path)  
 data\_cleaned = data.dropna(subset=columns\_to\_clean)  
 return data\_cleaned  
  
def plot\_relationship(data, x\_column, y\_column):plt.figure(figsize=(10, 6))  
 sns.scatterplot(x=data[x\_column], y=data[y\_column])  
 plt.title(f'Relationship between {x\_column} and {y\_column}')  
 plt.xlabel('Average RTT (ms)')  
 plt.ylabel('Max Number of Retransmissions')  
 plt.show()  
  
 # Calculate and print correlation, mean, and variance  
 correlation = data[x\_column].corr(data[y\_column])  
 print(f"Correlation between {x\_column} and {y\_column}: {correlation:.2f}")  
 mean\_value = data[y\_column].mean()  
 variance\_value = data[y\_column].var()  
 print(f"Mean of {y\_column}: {mean\_value:.2f}")  
 print(f"Variance of {y\_column}: {variance\_value:.2f}")  
  
# Main code  
csv\_file\_path = 'files/finaltcp.csv'  
columns\_to\_clean = ['RTT\_avg\_a2b', 'RTT\_avg\_b2a', 'max\_#\_retrans\_a2b', 'max\_#\_retrans\_b2a']  
tcp\_traffic\_data = load\_and\_clean\_data(csv\_file\_path, columns\_to\_clean)  
  
# Column names for average RTT and max number of retransmissions  
rtt\_avg\_columns = ['RTT\_avg\_a2b', 'RTT\_avg\_b2a']  
retrans\_max\_columns = ['max\_#\_retrans\_a2b', 'max\_#\_retrans\_b2a']  
  
# Analyze the relationship between RTT and retransmissions  
for rtt\_col, retrans\_col in zip(rtt\_avg\_columns, retrans\_max\_columns):  
 plot\_relationship(tcp\_traffic\_data, rtt\_col, retrans\_col)

The primary function of this code is to analyze and visualize the relationship between the Average Round-Trip Time (RTT) and the Maximum Number of Retransmissions in network traffic data. It starts by reading data from a CSV file, then conducts analysis on two pairs of columns. For each pair, the code generates a scatter plot to display the relationship between the average RTT and the maximum number of retransmissions and calculates the correlation coefficient between them.







The first two graphs display the relationship between the Average Round-Trip Time (RTT) and the Maximum Number of Retransmissions for two directions, from address A to B (a2b) and from address B to A (b2a), respectively. In both scatter plots, there is no strong and consistent relationship evident. Most data points are clustered at the lower end of the RTT scale, indicating that the majority of communications have a low RTT and few retransmissions.

The statistical output provided in the third graph offers the correlation coefficients between RTT and the maximum number of retransmissions for both directions. The correlations are very weak, suggesting no significant linear relationship between RTT and the maximum number of retransmissions. The average values and variances for the maximum number of retransmissions are quite low, indicating that high retransmissions are not common.

1. Total traffic volume during the connection (you get the volume from PS2).

import pandas as pd  
  
def convert\_units(row):# Define conversion factors for each unit  
 unit\_factors = {'bytes': 1, 'kb': 1024, 'mb': 1024 \*\* 2}  
 converted\_data = {}  
  
 # Process each data type and convert to bytes  
 for data\_type in ['ld', 'rd', 'total']:  
 bytes\_value = row[f'{data\_type}\_bytes']  
 unit = row[f'{data\_type}\_bytes\_unit'].lower()  
  
 # Raise an error if the unit is not recognized  
 if unit not in unit\_factors:  
 raise ValueError(f"Invalid unit '{unit}' for {data\_type}\_bytes. Supported units: 'bytes', 'kb', 'mb'.")  
  
 # Perform conversion and store in the dictionary  
 converted\_data[f'{data\_type}\_bytes\_converted'] = bytes\_value \* unit\_factors[unit]  
  
 # Return a new series including the converted data and server IP  
 return pd.Series({\*\*converted\_data, 'server\_ip': row['second\_ip']})  
  
# Read and preprocess data from a text file  
data\_file = 'files/final.txt'  
column\_names = [  
 "first\_ip", "arrow", "second\_ip", "ld\_frames", "ld\_bytes", "ld\_bytes\_unit",  
 "rd\_frames", "rd\_bytes", "rd\_bytes\_unit", "total\_frames", "total\_bytes", "total\_bytes\_unit",  
 "start\_time", "duration"  
]  
  
# Load the data, skipping the first 5 rows and the last row  
df = pd.read\_csv(  
 data\_file, sep='\s+', skiprows=5, header=None, skipfooter=1, engine='python', names=column\_names  
)  
  
# Apply the conversion function to each row  
converted\_df = df.apply(convert\_units, axis=1)  
  
# Calculate the total traffic volume in bytes  
total\_traffic\_volume\_bytes = converted\_df['total\_bytes\_converted'].sum()  
print(f"The total traffic volume during the connection is: {total\_traffic\_volume\_bytes} bytes")

This code primarily focuses on processing and analyzing network traffic data. It reads traffic data from a text file and then uses a function to convert the byte units in the data into a unified byte format. After the conversion, it calculates the total bytes across all rows in the dataset and prints out this sum.



* Conclusions
* Traffic volume at different time scales. Are there any identifiable patterns or trends that you observed?

The packet traffic volume graphs, calculated by the second and by the minute, show significant spikes in traffic at specific times, indicating that a large amount of data transfer or communication activity may occur at these moments. These peaks are more pronounced and easier to identify on a per-second basis, as traffic can vary greatly within a single second.

Similarly, the flow data graphs also display peaks in traffic volume. However, the peaks in flow data occur less frequently but with greater volume. This suggests that when flows occur, they may involve the transfer of larger amounts of data at once, or multiple packets may be combined into flows.

Overall, the traffic volume measured by the second reveals more detail and variability, as packet transmission can be bursty. On a per-minute scale, the overall trend becomes smoother, potentially averaging out the extreme values seen on a per-second basis, which could be more useful for identifying long-term patterns or trends in network traffic.

* The top 5 most common applications based on their port numbers. Identify the corresponding applications (e.g., HTTPS application) and analyze their characteristics.

Top 5 port numbers: 443, 61784, 52330, 61788, 5259.

Port 443: This is the standard port for HTTPS traffic, which is the secure version of HTTP. The characteristic of this application is that it encrypts communication to ensure secure transactions, typically used for online banking, e-commerce, and any other service where data security is paramount.

Port 61784: Likely used by a custom application or service. The application could be proprietary or specific to a certain vendor, possibly for remote service, gaming, or peer-to-peer applications.

Port 52330: Similar to port 61784, this is likely used by a private service or application. It might be used for internal communication within a private network, for instance, by backend systems of a web service.

Port 61788: As with the other dynamic ports, this port is probably assigned dynamically for a specific session or service. Applications using such ports often include streaming services, virtual private networks (VPN), or other temporary communication sessions.

Port 5259: This port might be employed by a specific application, such as a multimedia streaming service or a networked game that requires real-time data transmission.

* Differences of flow and packet measurements in the example case.

Packet measurement provides a magnified view of network traffic, while flow measurement offers a broader view of extended communication patterns. The fine-grained nature of packet analysis is particularly suitable for detailed troubleshooting and performance analysis. In contrast, flow analysis can help identify trends over longer periods, understand the nature of the traffic, and plan network capacity. High peaks in packet data may reflect transient events or anomalies, whereas the smoother trends in flow data might reflect regular usage patterns or scheduled activities.

* Your findings on retransmissions.

The low average values of retransmissions indicate that most communications in network traffic experience few retransmissions. The weak negative correlation between RTT and retransmissions may suggest that longer RTTs are not necessarily associated with an increased number of retransmissions. This could imply that other factors, such as network congestion, packet loss, or connection quality, might have a more significant impact on retransmissions. The scatter plots show no clear trend, indicating that retransmissions are not solely influenced by RTT and are likely affected by a complex set of factors.

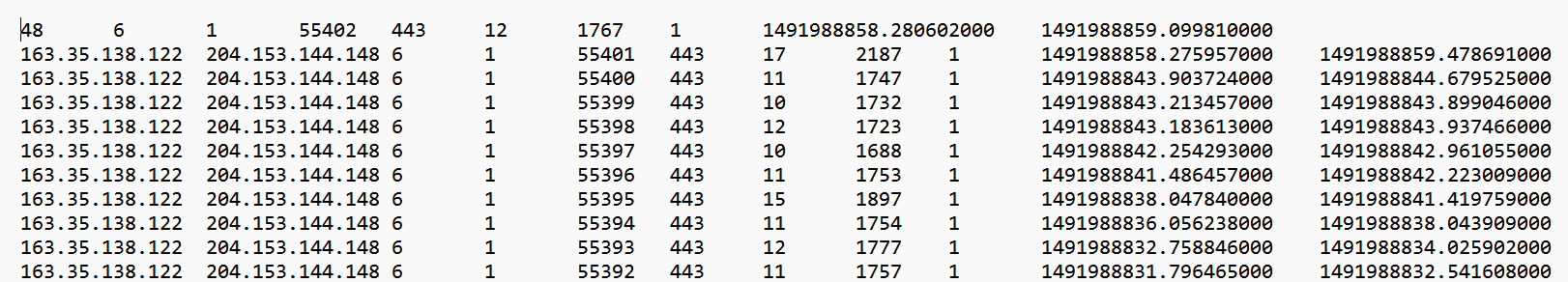
Task 2: Flow data

* Acquiring flow data
* Data pre-processing
* Subnetwork: 163.35.138.0/24
* Commands or code that is used in pre-processing.

gawk ‘$1~/^163\.35\.138\./||$15~/^163\.35\.138\./’ 1200.t2 > ~/my\_1200.t2

The contents of the commands are the same, only the file names are different.

* Short samples (10 lines or so) taken from the distilled data.



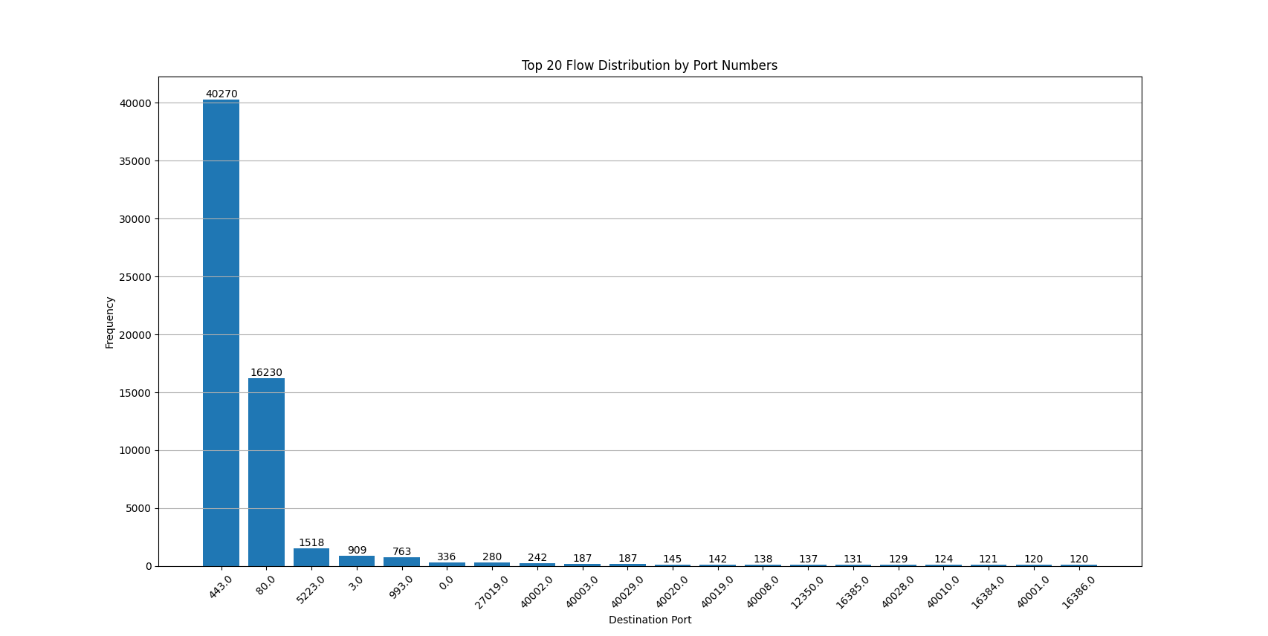
* Data analysis

1. Plot traffic volume

* Visualise flow distribution by port numbers.

import pandas as pd  
import matplotlib.pyplot as plt  
import glob  
  
def read\_and\_combine\_data(directory, file\_pattern, column\_names):  
 data\_frames = []  
 for file\_path in glob.glob(directory + file\_pattern):  
 try:  
 data\_frame = pd.read\_csv(file\_path, sep='\t', header=None, names=column\_names)  
 data\_frames.append(data\_frame)  
 except pd.errors.ParserError as error:  
 print(f"Error reading {file\_path}: {error}")  
 return pd.concat(data\_frames, ignore\_index=True) if data\_frames else pd.DataFrame()  
  
def plot\_top\_destination\_ports(data, top\_n=20):  
 destination\_port\_counts = data['destination\_port'].value\_counts().sort\_values(ascending=False).head(top\_n)  
 plt.figure(figsize=(10, 8))  
 bar\_plot = plt.bar(destination\_port\_counts.index.astype(str), destination\_port\_counts.values)  
  
 for bar in bar\_plot:  
 height = bar.get\_height()  
 plt.text(bar.get\_x() + bar.get\_width() / 2, height, int(height), ha='center', va='bottom')  
  
 plt.grid(axis='y')  
 plt.title('Top 20 Flow Distribution by Port Numbers')  
 plt.xlabel('Destination Port')  
 plt.ylabel('Frequency')  
 plt.xticks(rotation=45)  
 plt.show()  
  
# Path to the directory containing the data files  
data\_dir = 'files/data/'  
  
# Column names for the data  
column\_names = ['source', 'destination', 'protocol', 'status\_ok', 'source\_port',  
 'destination\_port', 'packet\_count', 'byte\_count', 'flow\_count',  
 'first\_seen', 'last\_seen']  
  
# Combine data from all files  
combined\_data = read\_and\_combine\_data(data\_dir, '\*.t2', column\_names)  
  
# Check if DataFrame is not empty before plotting  
if not combined\_data.empty:  
 plot\_top\_destination\_ports(combined\_data)  
else:  
 print("No data available for plotting.")

The purpose of this code is to read and consolidate data from multiple files within a specified directory, where each file contains tab-separated values with specific column names. It then analyzes and visualizes the top destination ports by frequency, displaying the distribution of flow data for these ports in a bar chart.

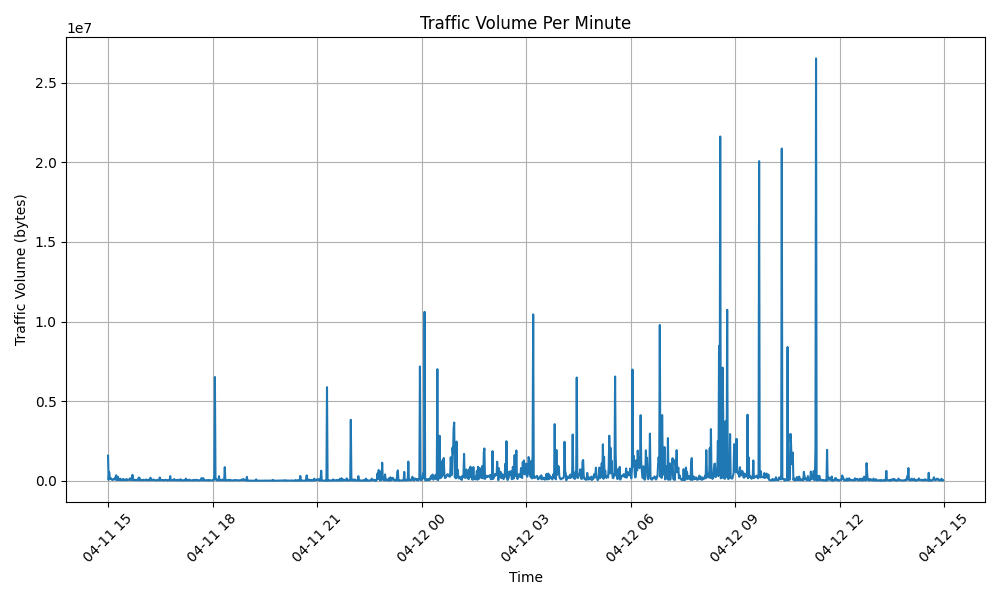


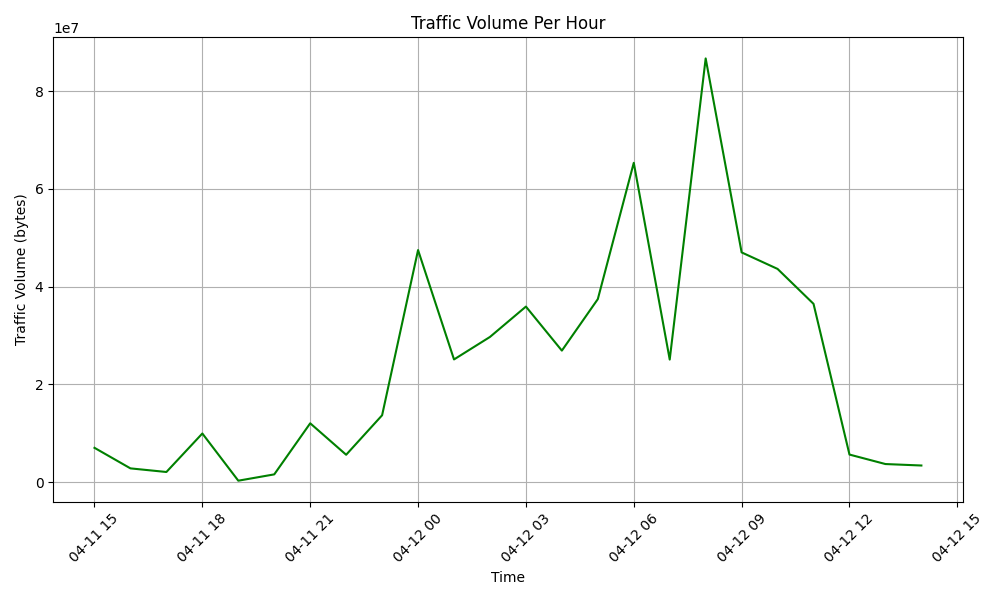
This chart shows the frequency distribution of network or computer system traffic across different destination ports. The frequencies for ports "80" and "443" are significantly higher than those for other ports. This is consistent with standard network traffic, as port 80 is typically used for HTTP traffic and port 443 for HTTPS traffic. The frequencies for the other ports are much lower, indicating less traffic.

* Plot traffic volume as a function of time with at least two sufficiently different time scales.

import pandas as pd  
import matplotlib.pyplot as plt  
import glob  
  
def read\_and\_process\_data(directory, file\_pattern, column\_headers):  
 file\_paths = glob.glob(directory + file\_pattern)  
 dataframes = []  
 for file\_path in file\_paths:  
 try:  
 data = pd.read\_csv(file\_path, sep='\t', header=None, names=column\_headers)  
 if not data.empty:  
 dataframes.append(data)  
 except pd.errors.ParserError as e:  
 print(f"Skipping file {file\_path} due to parsing error: {e}")  
 if not dataframes:  
 raise ValueError("No data files found or all files are empty.")  
 combined\_dataframe = pd.concat(dataframes, ignore\_index=True)  
 return combined\_dataframe  
  
def plot\_traffic\_volume(traffic\_data, title, time\_interval, color='blue'):  
 plt.figure(figsize=(10, 6))  
 plt.plot(traffic\_data, color=color)  
 plt.grid(True)  
 plt.xlabel('Time')  
 plt.ylabel('Traffic Volume (bytes)')  
 plt.title(f'Traffic Volume {title}')  
 plt.xticks(rotation=45)  
 plt.tight\_layout()  
 plt.show()  
  
data\_directory = 'files/data/'  
data\_file\_pattern = '\*.t2'  
column\_headers = ['source', 'destination', 'protocol', 'status\_ok', 'source\_port', 'dest\_port', 'packet\_count', 'byte\_count', 'flow\_count', 'start\_time', 'end\_time']  
  
traffic\_data\_combined = read\_and\_process\_data(data\_directory, data\_file\_pattern, column\_headers)  
  
traffic\_data\_combined['start\_time'] = pd.to\_datetime(traffic\_data\_combined['start\_time'], unit='s')  
  
# Set the 'start\_time' column as the index for resampling  
traffic\_data\_combined.set\_index('start\_time', inplace=True)  
  
# Resample to get traffic volume per minute and per hour  
traffic\_volume\_per\_minute = traffic\_data\_combined.resample('T')['byte\_count'].sum()  
traffic\_volume\_per\_hour = traffic\_data\_combined.resample('H')['byte\_count'].sum()  
  
# Plot the traffic volume per minute and per hour  
plot\_traffic\_volume(traffic\_volume\_per\_minute, 'Per Minute', 'Minute')  
plot\_traffic\_volume(traffic\_volume\_per\_hour, 'Per Hour', 'Hour', color='green')

The purpose of this code is to read and process data from multiple files located in a specified directory. These files contain tab-separated data with specific column headers. The code then conducts time series analysis by resampling the data into minute and hour intervals, calculating the total traffic volume. It subsequently plots two graphs: one displaying traffic volume changes per minute and another showing traffic volume changes per hour.



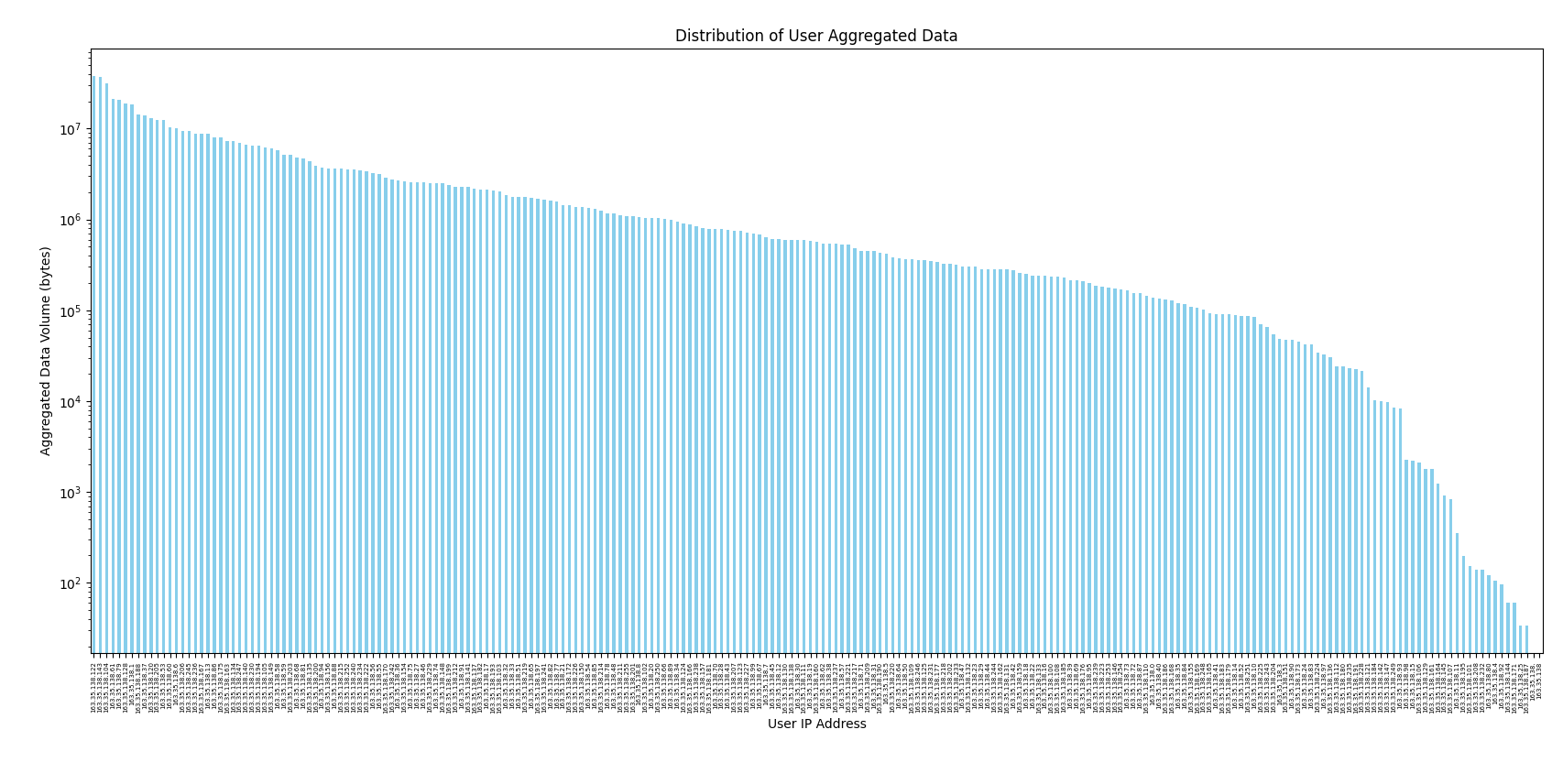


These two charts reflect network traffic variations across different time scales. The first chart, with per-minute granularity, shows the fluctuations in traffic from April 11 to April 15, with many sharp peaks indicating brief periods of high traffic activity. In contrast, the second chart displays the change in traffic on an hourly basis, revealing peaks in traffic volume within certain hours, followed by a gradual decline. Integrating the information from both charts, we can observe that network traffic experiences intense fluctuations at specific moments, while a more macroscopic time scale reveals the overall trends in network usage.

1. Per user data volume

import pandas as pd  
import matplotlib.pyplot as plt  
import glob  
import numpy as np  
  
def load\_and\_combine\_data(directory):  
 # Define column names  
 columns = ['source\_ip', 'destination\_ip', 'protocol', 'status', 'source\_port', 'destination\_port',  
 'packets', 'bytes', 'flows', 'first\_timestamp', 'latest\_timestamp']  
  
 # Use glob to get all file paths in the directory  
 file\_paths = glob.glob(directory + '\*.t2')  
  
 # Read all files and combine into one DataFrame  
 df\_list = [pd.read\_csv(file, sep='\t', header=None, names=columns) for file in file\_paths]  
 return pd.concat(df\_list, ignore\_index=True)  
  
def plot\_user\_data\_volume(data, font\_size=12):  
 # Calculate the aggregated data volume for each user (source IP address)  
 user\_data\_volume = data.groupby('source\_ip')['bytes'].sum().sort\_values(ascending=False)  
  
 # Plot bar chart for the aggregated data volume of all users  
 plt.figure(figsize=(12, 6))  
 user\_data\_volume.plot(kind='bar', color='skyblue')  
 plt.title('Distribution of User Aggregated Data')  
 plt.xlabel('User IP Address')  
 plt.ylabel('Aggregated Data Volume (bytes)')  
 plt.xticks(rotation=90, fontsize=font\_size) # Rotate the x labels and set font size  
 plt.yscale('log') # Use logarithmic scale for better visibility  
 plt.tight\_layout() # Adjust layout to fit IP addresses  
 plt.show()  
  
# Define the directory path for data files  
directory\_path = 'files/data/'  
  
# Load and combine data from files  
df\_combined = load\_and\_combine\_data(directory\_path)  
  
# Plot the user data volume distribution  
plot\_user\_data\_volume(df\_combined, font\_size=5)

The functionality of this code is to load multiple data files with the `.t2` extension from a specified directory, merge them into a single DataFrame, calculate the total data volume for each user (source IP address), and plot a bar chart on a logarithmic scale to visualize the distribution of aggregated data volume among users, facilitating data analysis and visualization.

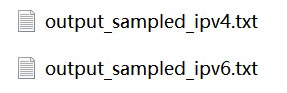


This chart presents the data usage across different users. It reveals significant disparities in data consumption; some users have much higher data usage than others, while the majority of users consume relatively small amounts of data. The chart illustrates a long-tail distribution, where a small number of users account for the majority of the data volume, and the vast majority of users contribute a small fraction of the total usage.

1. Flow sampling

* Shell for sampling:

#!/bin/bash  
  
# Define the input directory and the sample size per file  
input\_directory="/work/courses/unix/T/ELEC/E7130/general/trace/flow-continue"  
sample\_per\_file=7126  
  
# Define the output files for IPv4 and IPv6  
output\_ipv4\_file="./output\_sampled\_ipv4.txt"  
output\_ipv6\_file="./output\_sampled\_ipv6.txt"  
  
# Regular expressions for IPv4 and IPv6  
ipv4\_regex="^([0-9]{1,3}\.){3}[0-9]{1,3}$"  
ipv6\_regex="^(([0-9a-fA-F]{1,4}:){7,7}[0-9a-fA-F]{1,4}|([0-9a-fA-F]{1,4}:){1,7}:|([0-9a-fA-F]{1,4}:){1,6}:[0-9a-fA-F]{1,4}|([0-9a-fA-F]{1,4}:){1,5}(:[0-9a-fA-F]{1,4}){1,2}|([0-9a-fA-F]{1,4}:){1,4}(:[0-9a-fA-F]{1,4}){1,3}|([0-9a-fA-F]{1,4}:){1,3}(:[0-9a-fA-F]{1,4}){1,4}|([0-9a-fA-F]{1,4}:){1,2}(:[0-9a-fA-F]{1,4}){1,5}|[0-9a-fA-F]{1,4}:((:[0-9a-fA-F]{1,4}){1,6})|:((:[0-9a-fA-F]{1,4}){1,7}|:))"  
  
# Remove existing output files  
rm -f "$output\_ipv4\_file" "$output\_ipv6\_file"  
  
# Get the total number of files  
total\_files=$(find "$input\_directory" -type f | wc -l)  
current\_file=0  
  
# Process each file  
for file in "$input\_directory"/\*; do  
 if [ -f "$file" ]; then  
 let current\_file++  
 echo "Processing file ($current\_file / $total\_files): $file"  
  
 # Create temporary files for IPv4 and IPv6  
 temp\_ipv4="./temp\_ipv4"  
 temp\_ipv6="./temp\_ipv6"  
  
 # Clear or initialize these files  
 > "$temp\_ipv4"  
 > "$temp\_ipv6"  
  
 # Split the file into IPv4 and IPv6 parts  
 tail -n +29 "$file" | awk -v ipv4\_regex="$ipv4\_regex" '$1 ~ ipv4\_regex' > "$temp\_ipv4"  
 tail -n +29 "$file" | awk -v ipv6\_regex="$ipv6\_regex" '$1 ~ ipv6\_regex' > "$temp\_ipv6"  
  
 # Sample the IPv4 temporary file  
 total\_lines\_ipv4=$(wc -l < "$temp\_ipv4")  
 if [ $total\_lines\_ipv4 -ge $sample\_per\_file ]; then  
 selected\_lines\_ipv4=($(shuf -i 1-$total\_lines\_ipv4 -n $sample\_per\_file))  
 python3 sample\_script.py "$temp\_ipv4" "${selected\_lines\_ipv4[@]}" >> "$output\_ipv4\_file"  
 else  
 cat "$temp\_ipv4" >> "$output\_ipv4\_file"  
 fi  
  
 # Sample the IPv6 temporary file  
 total\_lines\_ipv6=$(wc -l < "$temp\_ipv6")  
 if [ $total\_lines\_ipv6 -ge $sample\_per\_file ]; then  
 selected\_lines\_ipv6=($(shuf -i 1-$total\_lines\_ipv6 -n $sample\_per\_file))  
 python3 sample\_script.py "$temp\_ipv6" "${selected\_lines\_ipv6[@]}" >> "$output\_ipv6\_file"  
 else  
 cat "$temp\_ipv6" >> "$output\_ipv6\_file"  
 fi  
  
 # No longer need to delete these files  
 # rm -f "$temp\_ipv4" "$temp\_ipv6"  
 fi  
done  
  
echo "Sampling completed, IPv4 results saved in $output\_ipv4\_file, IPv6 results saved in $output\_ipv6\_file"

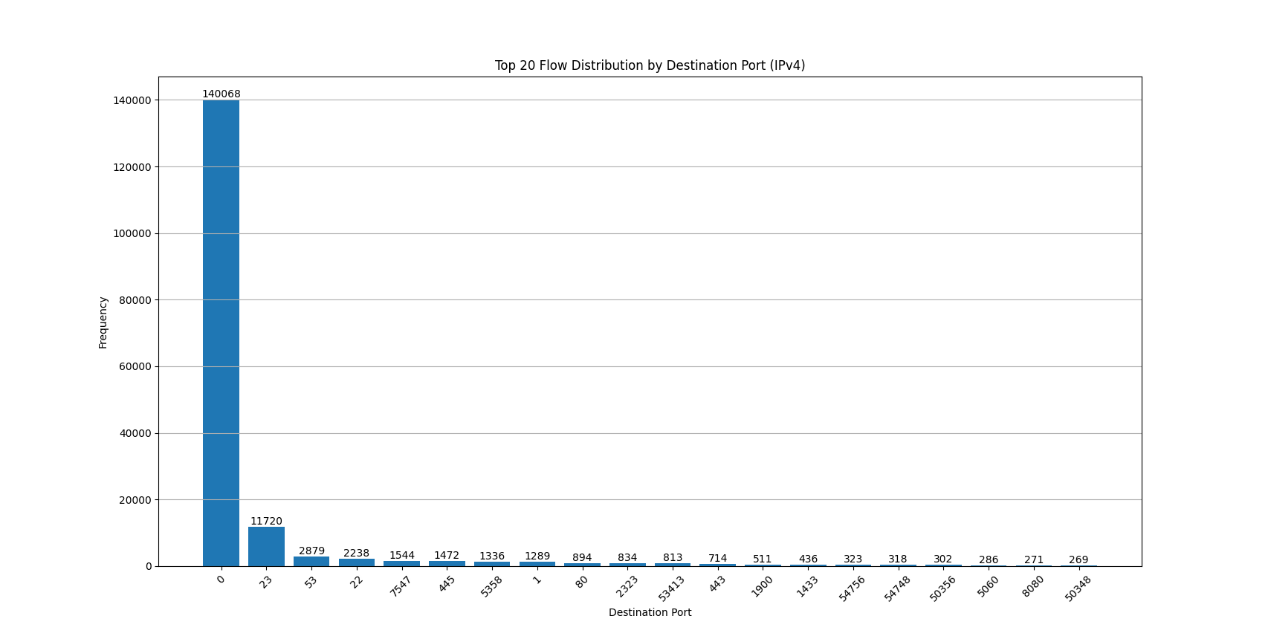


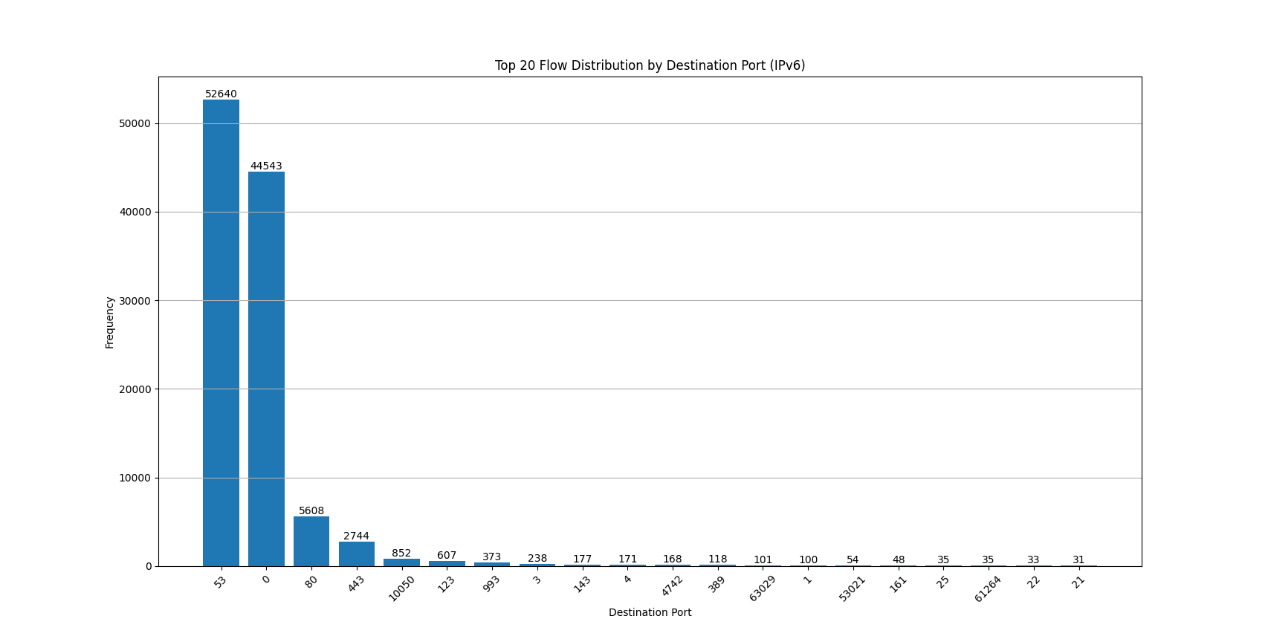
This Bash script processes a collection of data files containing network traffic information, separating IPv4 and IPv6 traffic based on regular expressions, sampling a specified number of lines from each, and saving the sampled data into separate output files.

* Visualise flow distribution by port numbers.

import pandas as pd  
import matplotlib.pyplot as plt  
import glob  
  
# Define the file path for IPv6 traffic data  
ipv6\_traffic\_file\_path = 'files/output\_sampled\_ipv6.txt'  
  
# Define descriptive column names  
column\_names = ['Source\_IP', 'Destination\_IP', 'Protocol', 'Status\_OK', 'Source\_Port', 'Destination\_Port',  
 'Packet\_Count', 'Byte\_Count', 'Flow\_Count', 'Start\_Time', 'End\_Time']  
  
# Read the IPv6 traffic data and create a DataFrame  
ipv6\_traffic\_df = pd.read\_csv(ipv6\_traffic\_file\_path, sep='\t', header=None, names=column\_names)  
  
# Group the data by 'Destination\_Port', calculate counts, and sort in descending order  
top\_destination\_ports = ipv6\_traffic\_df['Destination\_Port'].value\_counts().nlargest(20)  
  
# Create a bar chart  
plt.figure(figsize=(12, 8))  
bars = plt.bar(top\_destination\_ports.index.astype(str), top\_destination\_ports.values)  
  
# Add text annotations above the bars  
for bar in bars:  
 y\_value = bar.get\_height()  
 plt.text(bar.get\_x() + bar.get\_width() / 2, y\_value, int(y\_value), ha='center', va='bottom')  
  
plt.grid(axis='y')  
plt.title('Top 20 Flow Distribution by Destination Port (IPv6)')  
plt.xlabel('Destination Port')  
plt.ylabel('Frequency')  
plt.xticks(rotation=45)  
plt.show()

The functionality of this code is to read IPv4/6 network traffic data files, parse the data within them, and extract destination port information. It then counts and plots the distribution of traffic for the top 20 destination ports, facilitating the analysis and visualization of usage patterns for IPv4/6 traffic data.



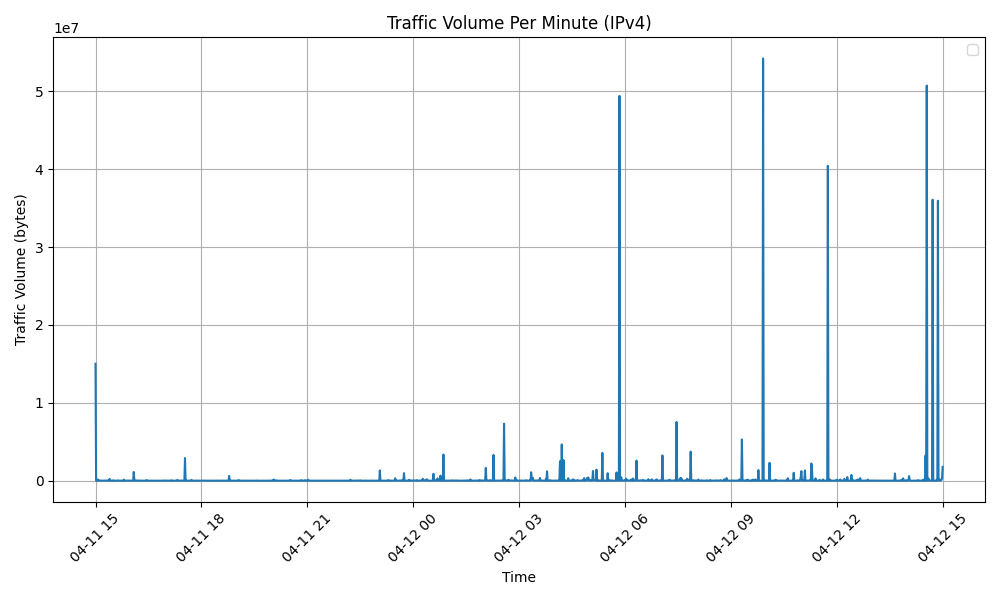


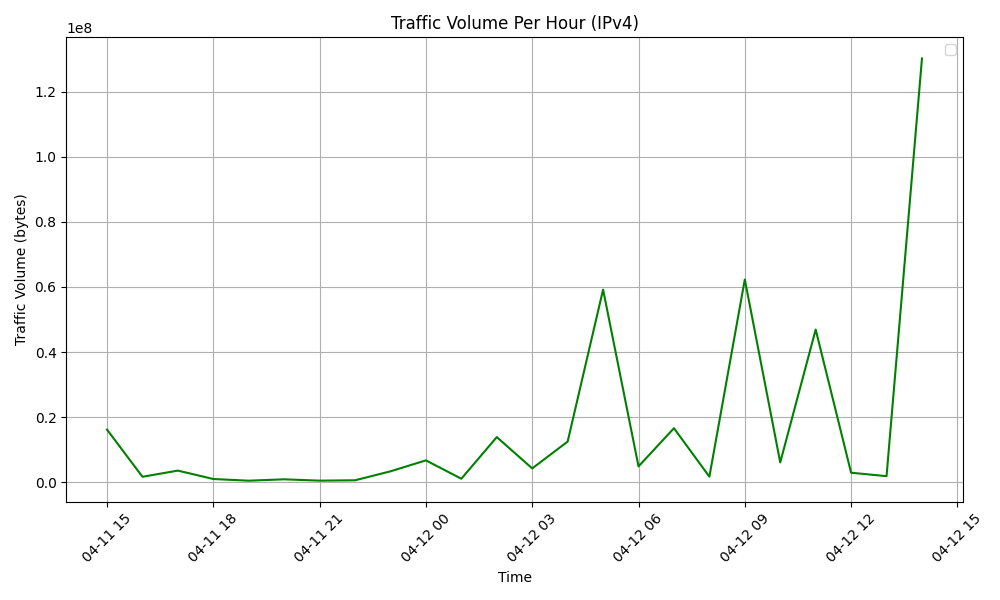
These two charts respectively display the top 20 flow distributions by destination port for IPv4 and Ipv6 traffic. In the Ipv4 chart, the flow peak for the first port exceeds 140,000, significantly higher than the other ports. The second port has about 11,720 flows, followed by a more gradual decline in traffic for the subsequent ports. This indicates that Ipv4 traffic is heavily concentrated on a specific port. In the Ipv6 chart, the flow distribution among the top ports is relatively more uniform.

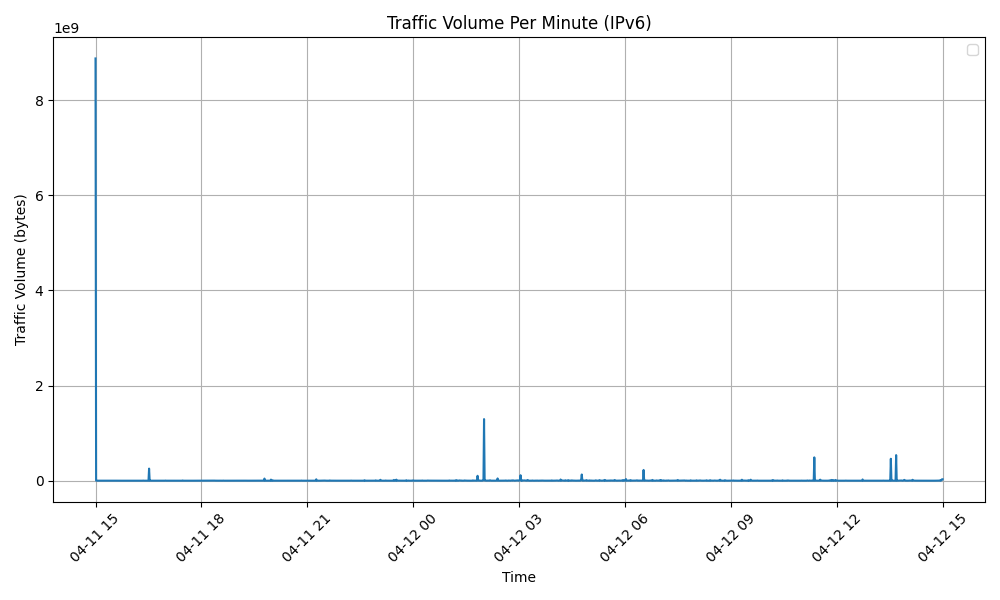
* Plot traffic volume as a function of time with at least two sufficiently different time scales.

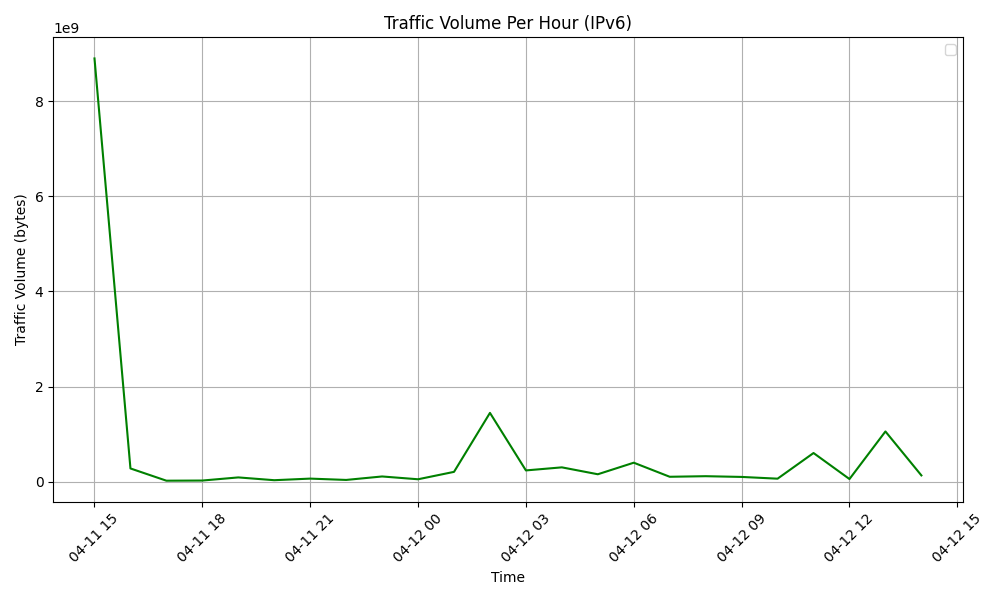
Import pandas as pd  
import matplotlib.pyplot as plt  
  
data\_directory = ‘files/output\_sampled\_ipv6.txt’  
column\_names = [‘Source’, ‘Destination’, ‘Protocol’, ‘Status\_OK’, ‘Source\_Port’, ‘Destination\_Port’,  
 ‘Packet\_Count’, ‘Byte\_Count’, ‘Flow\_Count’, ‘Start\_Time’, ‘End\_Time’]  
  
traffic\_data = pd.read\_csv(data\_directory, sep=’\t’, header=None, names=column\_names)  
  
traffic\_data[‘Start\_Time’] = pd.to\_datetime(traffic\_data[‘Start\_Time’], unit=’s’)  
  
traffic\_data.set\_index(‘Start\_Time’, inplace=True)  
traffic\_volume\_per\_minute = traffic\_data.resample(‘T’)[‘Byte\_Count’].sum()  
traffic\_volume\_per\_hour = traffic\_data.resample(‘H’)[‘Byte\_Count’].sum()  
  
plt.figure(figsize=(10, 6))  
plt.plot(traffic\_volume\_per\_minute)  
plt.grid(True)  
plt.xlabel(‘Time’)  
plt.ylabel(‘Traffic Volume (bytes)’)  
plt.title(‘Traffic Volume Per Minute (Ipv6)’)  
plt.xticks(rotation=45)  
plt.tight\_layout()  
plt.show()  
  
plt.figure(figsize=(10, 6))  
plt.plot(traffic\_volume\_per\_hour, color=’green’)  
plt.grid(True)  
plt.xlabel(‘Time’)  
plt.ylabel(‘Traffic Volume (bytes)’)  
plt.title(‘Traffic Volume Per Hour (Ipv6)’)  
plt.xticks(rotation=45)  
plt.tight\_layout()  
plt.show()

The functionality of this code is to read and preprocess IPv6 network traffic data files and then create two plots, each depicting the traffic trends over time. One plot represents the traffic trends summarized on a per-minute basis, while the other represents traffic trends summarized on an hourly basis.









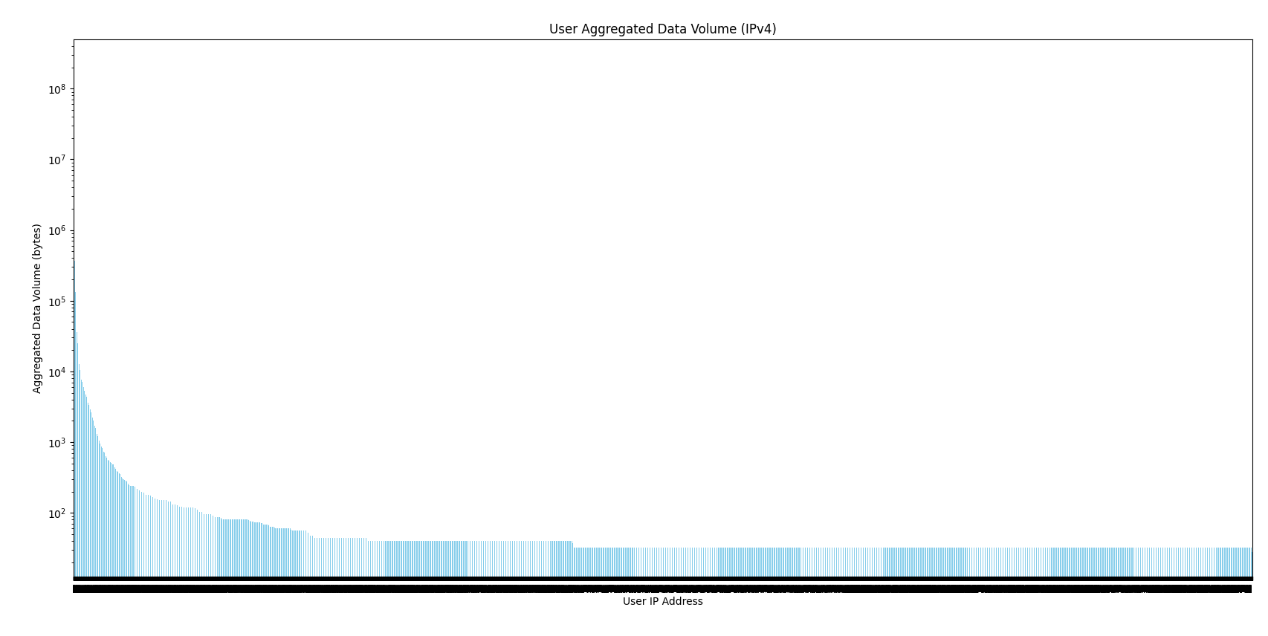
The IPv4 per-minute traffic volume chart shows several peaks, indicating surges in traffic during specific minutes. At other times, the traffic volume remains at a lower level. The per-hour traffic chart shows a smoother trend in volume changes. Although fluctuations are still observable, the difference between peaks and troughs is not as pronounced as in the per-minute chart.

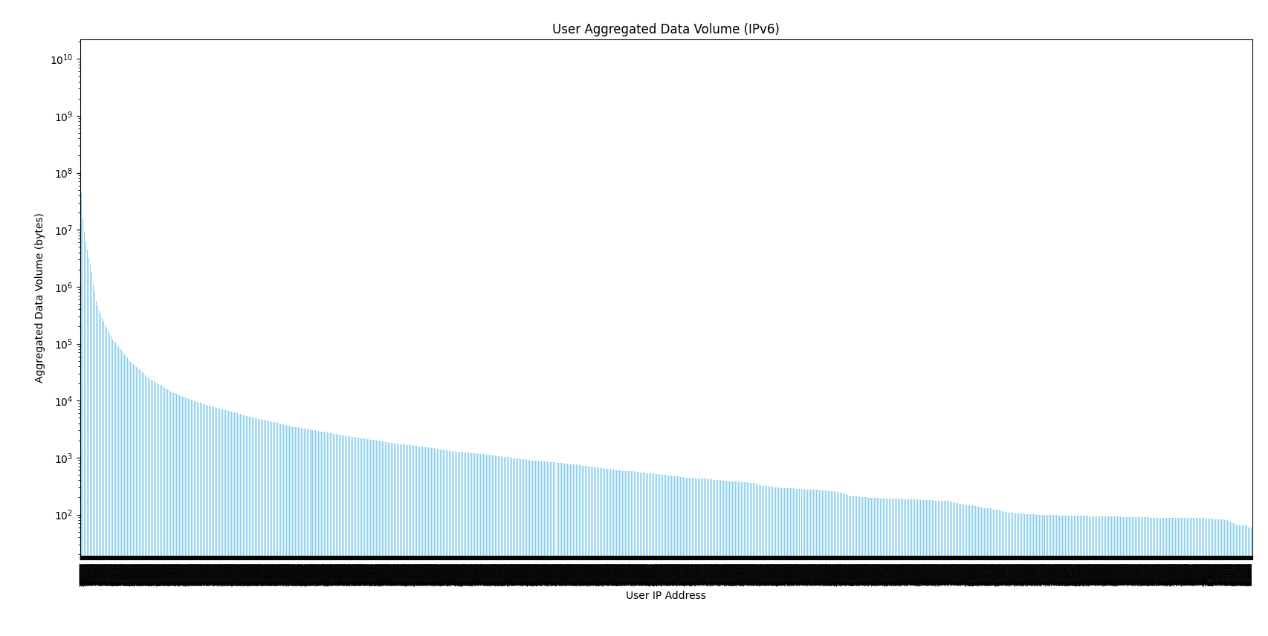
The IPv6 per-minute traffic volume chart exhibits characteristics similar to IPv4, with spikes in traffic volume occurring during certain minutes, though these spikes appear less frequently than in IPv4. The per-hour traffic chart displays a significant initial peak, after which the traffic volume quickly decreases and maintains a lower level, with occasional minor fluctuations. Compared to IPv4, the IPv6 traffic distribution seems more uniform, lacking many of the dramatic fluctuations.

* Per user data volume

import pandas as pd  
import matplotlib.pyplot as plt  
  
# Define the data file path  
file\_path = 'files/output\_sampled\_ipv6.txt'  
  
# Define the column names  
columns = ['src', 'dst', 'pro', 'ok', 'sport', 'dport', 'packets', 'bytes', 'flows', 'first', 'latest']  
  
# Read the file into a DataFrame  
df = pd.read\_csv(file\_path, sep='\t', header=None, names=columns)  
  
# Calculate the aggregated data volume for each user (source IP address)  
user\_data\_volume = df.groupby('src')['bytes'].sum().sort\_values(ascending=False)  
  
plt.figure(figsize=(12, 6))  
user\_data\_volume.plot(kind='bar', color='skyblue', logy=True)  
plt.title('User Aggregated Data Volume (IPv6)')  
plt.xlabel('User IP Address')  
plt.ylabel('Aggregated Data Volume (bytes)')  
plt.xticks(rotation=90, fontsize=0.0001)  
plt.tight\_layout()  
plt.show()

The code's purpose is to process network traffic data stored in a file, specifically focusing on IPv6 traffic. It reads the data into a DataFrame, calculates the aggregated data volume for each user based on their source IP address, and then visualizes this aggregated data volume using a bar chart.





These two graphs respectively illustrate the aggregated data volume per user within IPv4 and IPv6 networks. Both charts exhibit a highly skewed distribution of data volumes among users. A small number of users account for a disproportionately large share of the data traffic, whereas the majority of users consume much less data. The data follows a long-tail distribution, which is typical in network traffic analysis. After an initial steep decline, there is a long tail representing a large number of users with low data volumes. Comparing IPv4 and IPv6, the overall shape of the distribution is similar, indicating that the usage patterns between the two protocols are not significantly different.

* Compare the results to the original task where you used your subnetwork (FS2) only. Can you say the characteristics of your subnetwork is representative? Is there a difference between IPv4 and IPv6?

When comparing the traffic patterns sampled from the subnetwork (FS2) with the overall dataset (FS1), FS2 also reflects FS1's traffic distribution and variability. Therefore, I believe the characteristics of FS2 can represent the larger dataset.

The difference between IPv4 and IPv6 traffic in FS1 suggests they may have different usage patterns or adoption rates. IPv4 shows more significant traffic volume peaks, which could be due to its wider adoption and the nature of the services it supports. Although IPv6 also shows changes in traffic volume, it may not have the same intensity or pattern of usage as IPv4, potentially due to its evolving adoption and different network service characteristics.

* Conclusions
* Traffic volume at different time scales. Are there any identifiable patterns or trends that you observed?

In both the per-minute and per-hour traffic volume charts, identifiable peaks in traffic volume suggest the possibility of periodic high-usage events or scheduled activities. The hourly charts are smoother and show less variability because they aggregate traffic over a longer period, diluting the short-term peaks observed in the per-minute charts. For IPv6, traffic starts with a high volume and then quickly stabilizes, whereas IPv4 traffic displays more variability over time with several distinct peaks, indicating that IPv4 may be handling a mix of baseline traffic and bursty events.

* Identify the top 5 most common applications by studying their port numbers.

Flow data:

1. Port 443: HTTPS (Hypertext Transfer Protocol Secure) - Port 443 is commonly used for secure web browsing, where data is encrypted between the user's web browser and the web server. It ensures secure online transactions and protects sensitive information.
2. Port 80: HTTP (Hypertext Transfer Protocol) - Port 80 is used for standard, non-secure web browsing. It is the default port for web traffic and is used for accessing websites and web services.
3. Port 5223: Apple Push Notification Service (APNS) - Port 5223 is associated with Apple's APNS, used for sending push notifications to iOS devices. It enables applications to deliver real-time updates to users.
4. Port 3: Compression Process - Port 3 is not commonly associated with a specific application, and it's not a well-known port for standard services. It may be used for various purposes depending on the specific configuration.
5. Port 993: IMAPS (Internet Message Access Protocol Secure) - Port 993 is used for secure email communication using IMAPS. It provides encrypted access to email messages stored on an email server, ensuring the privacy and security of email communications.

IPv4:

1. Port 23: Telnet - Telnet is a network protocol used for remote terminal connection and management of devices. It allows users to log in to a remote computer and execute commands.
2. Port 53: Domain Name System (DNS) - DNS is a fundamental network protocol used to translate human-readable domain names into IP addresses and vice versa. It is essential for internet browsing and communication.
3. Port 22: Secure Shell (SSH) - SSH is a secure network protocol used for secure remote access and control of network devices and servers. It provides encrypted communication and authentication.
4. Port 7547: CPE WAN Management Protocol (CWMP) - CWMP is a protocol used for managing customer premises equipment (CPE) in broadband networks, such as DSL modems and routers, by service providers.
5. Port 445: Microsoft-DS (Directory Services) - Port 445 is associated with the Microsoft-DS protocol used for file and printer sharing, as well as other network services, in Windows environments.

IPv6:

1. Port 53: Domain Name System (DNS) - DNS is a fundamental network protocol used for mapping domain names to IP addresses and vice versa. It is widely used on the internet for resolving domain names to access websites and other network resources.
2. Port 80: Hypertext Transfer Protocol (HTTP) - HTTP is a protocol used for transmitting text, images, videos, and other content over the web. It forms the basis for web browsing and accessing web pages.
3. Port 443: Secure Sockets Layer (SSL)/Transport Layer Security (TLS) - Port 443 is commonly used for the HTTPS protocol, which is the secure version of HTTP, ensuring encrypted web communication to maintain data confidentiality and integrity.
4. Port 10050: Zabbix Agent - Port 10050 is typically associated with the Zabbix monitoring system agent, used for monitoring and collecting performance data from servers and network devices.
5. Port 123: Network Time Protocol (NTP) - NTP is used for synchronizing the time of computers and other network devices. It ensures that the clocks of various devices remain accurately synchronized.

* What kind of users there are in the network? Speculate on what kind of network this network could be based on traffic volumes and user profiles. Is your subnetwork different from larger population?

1. Types of Users in the Network:

* Heavy Users: The presence of significant traffic peaks, especially in IPv4, suggests heavy users or power users. These could be servers, large-scale file-sharing systems, or streaming services consuming and distributing large amounts of data.
* General Users: The long tail distribution observed indicates a large number of general users with relatively low data usage. This group likely includes regular consumers using the network for typical activities like browsing, email, and standard-definition video streaming.
* Specialized Users: The use of specific ports (like 443, 80, 23, 22) suggests users engaged in activities like secure web browsing (HTTPS), remote server access (SSH, Telnet), and DNS services.

1. Nature of the Network:

* The mix of heavy and general users, along with the types of services inferred from port usage, suggests a diverse network. This could be a residential or commercial ISP network, a large corporate network, or a university campus network.
* The presence of significant volumes of HTTPS and HTTP traffic points to regular web browsing as a primary activity, common in commercial and educational networks.
* The usage patterns of ports like 22 (SSH) and 23 (Telnet) might indicate a network with administrative or developmental activities, perhaps with a number of servers or development machines.

1. Difference from larger population

* My subnet displays a sharp drop in data volume after the top-ranking users, which may indicate a network with a few heavy users or servers and many light users. In contrast, the larger population should show a more gradual decline, indicating a more uniform distribution of data usage among users.
* In my subnet, data volume noticeably leans toward the top IP addresses, suggesting that this might be a corporate or institutional network with specific nodes handling significant data loads. In a larger network, a smoother gradient might be observed, indicating a network without such pronounced outliers in terms of data usage, such as a residential ISP where individual households have a more uniform data usage.
* The presence of very high-volume IP addresses in my subnet might indicate specialized activities like hosting services, large file transfers, or data processing tasks, which could be characteristic of service providers or large organizations with centralized data operations. In contrast, the larger network's more uniform usage suggests that it might not have as many specialized high-demand nodes and might be structured to provide more consistent service across all nodes, such as a smaller business, educational institution, or community network.
* Comparison of the above results with the result from data set PS2.

1. Traffic Volume and Patterns: Personal networks typically exhibit simpler overall traffic patterns with short-duration traffic peaks and smaller data volumes, primarily reflecting individual internet usage habits, such as evening or weekend browsing, video watching, or social media interaction. In contrast, public networks, like those within companies or schools, due to a larger number of users and devices, display higher usage frequency and larger data volumes. These networks experience more frequent and longer-lasting traffic peaks, reflecting a combination of diverse work and leisure activities.
2. User Behavior Characteristics: In personal networks, user behavior tends to be more uniform, primarily revolving around daily internet activities such as online shopping, news browsing, or using streaming services, which usually don't require substantial data transfer. On the other hand, in public network environments, the variety of user behaviors is more extensive, ranging from basic web browsing to complex data processing and online collaboration. For instance, corporate networks might encompass large-scale file transfers, remote conferencing, complex data analysis, and cloud computing services. This diversity results in more varied and complex data usage volumes and types in public networks compared to personal networks.
3. Service and Application Diversity: Users of private networks typically access a limited set of services that meet basic personal needs, like personal email, social media platforms, and online entertainment content. While these services are widespread, their data usage in terms of volume and type is relatively limited. In contrast, public networks might need to support a broader range of services to accommodate the needs of various users and operations. These might include enterprise resource management systems, cloud storage and computing services, professional online collaboration tools, and access to external websites. The usage of these services not only brings higher data traffic to the network but also imposes higher demands in terms of security and reliability. This diversity makes the maintenance and management of public networks more complex than that of private networks.

Task 3: Analysing active measurements

* Latency measurements (data sets AS1.x), where x includes:

AS1.d1: Name server1 with DNS: dns-st.bahnhof.net.txt

AS1.d2: Name server2 with DNS: ns1.bahnhof.net.txt

AS1.d3: Name server3 with DNS: southeast-2.dns-au.st.txt

AS1.n1: Name server1 with ICMP: dns-st.bahnhof.net.txt

AS1.n2: Name server2 with ICMP: ns1.bahnhof.net.txt

AS1.n3: Name server3 with ICMP: southeast-2.dns-au.st.txt

AS1.r1: Research server1: cbg-uk.ark.caida.org.txt

AS1.r2: Research server2: bjl-gm.ark.caida.org.txt

AS1.r3: Research server3: msy-us.ark.caida.org.txt

AS1.i1: Iperf server1: ok1.iperf.comnet-student.eu.txt

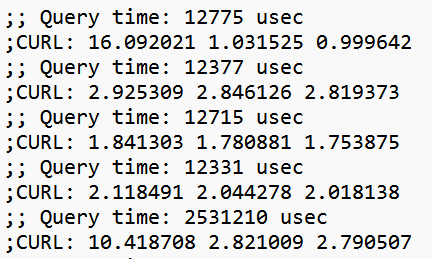
AS1.i2: Iperf server2: blr1.iperf.comnet-student.eu.txt

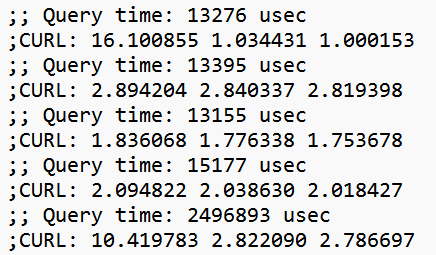
* Throughput measurements (data sets AS2.x), where x includes:

AS2.i1: Iperf server1: ok1.iperf.comnet-student.eu.txt

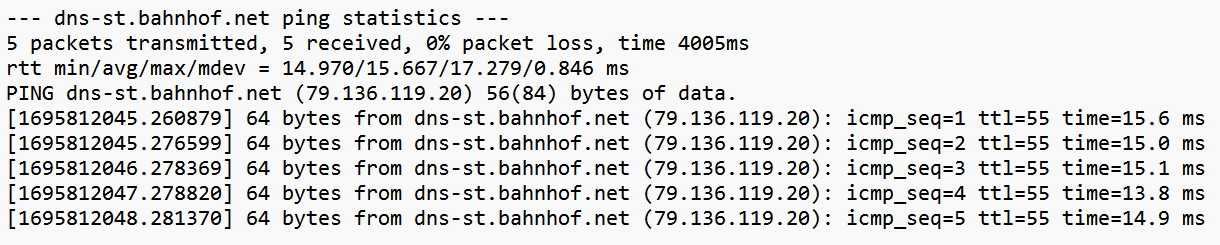
AS2.i2: Iperf server2: blr1.iperf.comnet-student.eu.txt

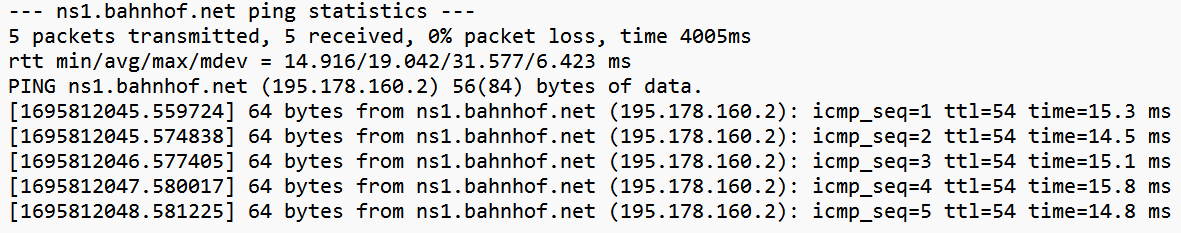
Sample:

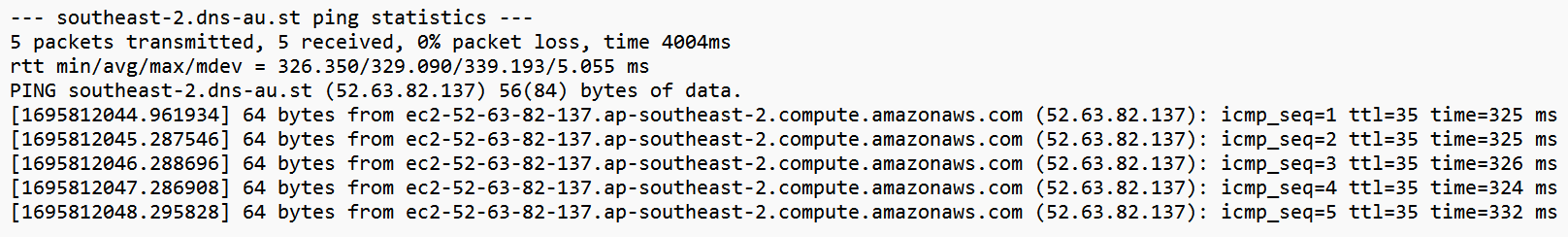


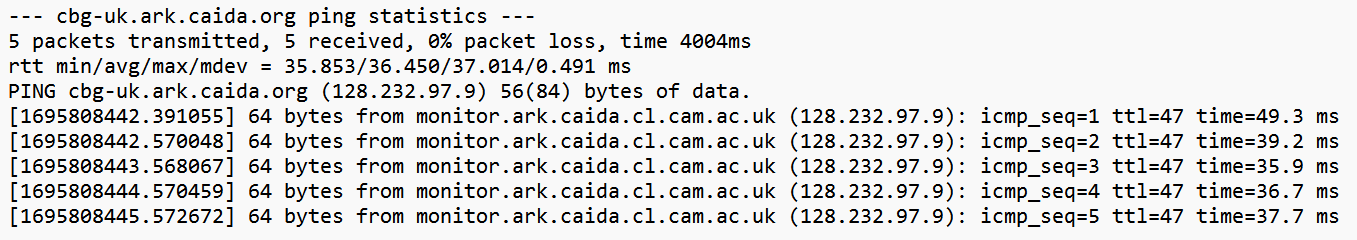


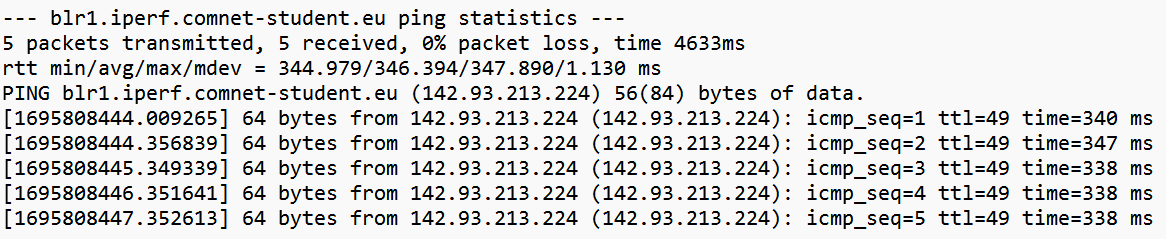


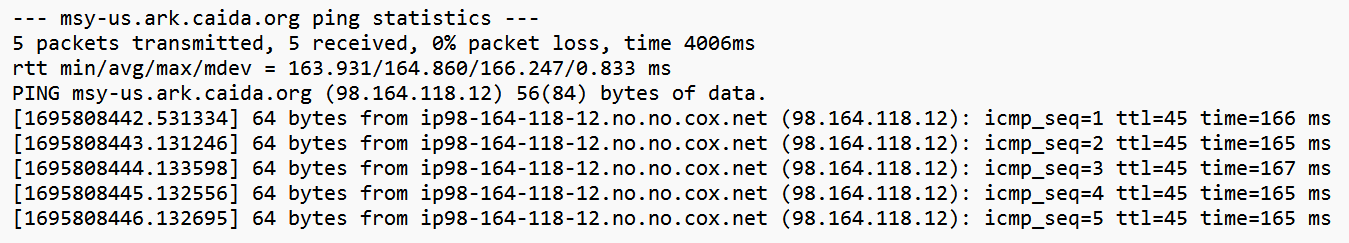


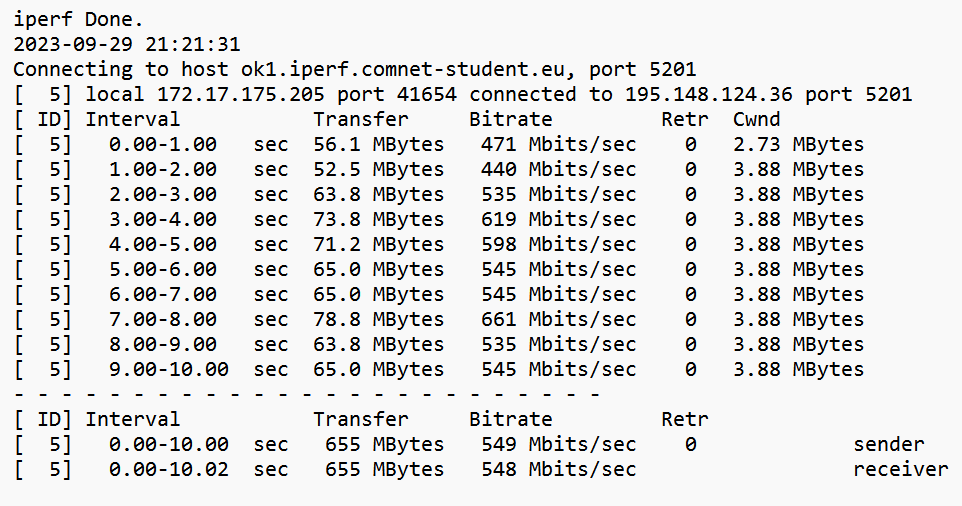


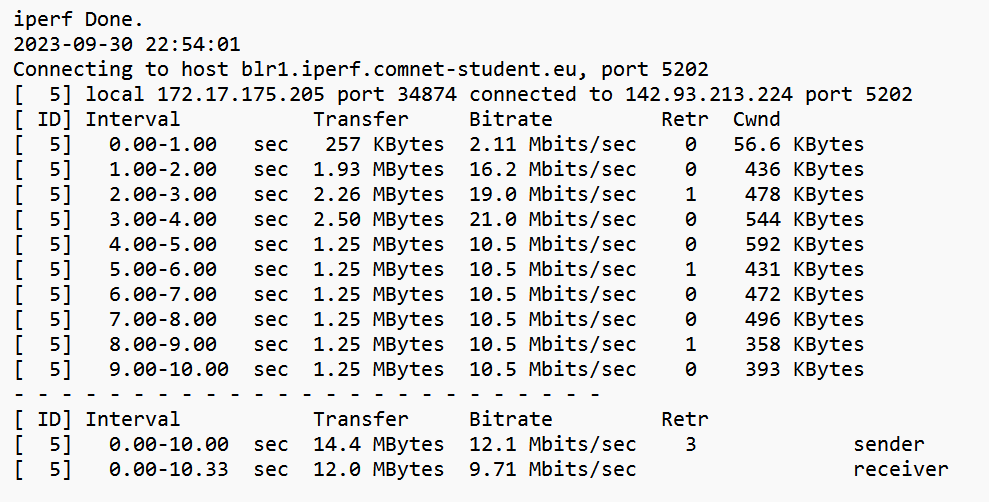












1. Latency data plots (AS1.x)

* Excludes packet loss

# Initialize a list to store extracted latency values  
latency\_values = []  
  
# Open the file and read each line  
file\_path = 'files/ping/southeast-2.dns-au.st.txt'  
with open(file\_path, 'r') as file:  
 for line in file:  
 if 'time=' in line:  
 # Find lines containing 'time=' and split the string  
 parts = line.split()  
 for part in parts:  
 if part.startswith('time='):  
 # Extract the latency value and remove 'ms'  
 latency = part.split('=')[1].rstrip(' ms')  
 try:  
 # Convert the extracted latency value to a floating-point number and store it  
 latency\_values.append(float(latency))  
 except ValueError:  
 # If conversion fails, skip the value  
 continue  
  
latency\_values.sort()  
print(latency\_values)  
  
import matplotlib.pyplot as plt  
  
# Create a boxplot, assuming latency\_values contains latency measurements  
fig, ax = plt.subplots(figsize=(10, 6))  
boxplot\_dict = ax.boxplot(latency\_values, patch\_artist=True)  
  
# Annotate medians  
medians = [median.get\_ydata()[0] for median in boxplot\_dict['medians']]  
for tick, median in zip(ax.get\_xticks(), medians):  
 ax.text(tick, median, f'Median: {median:.2f}', ha='center', va='center', fontdict={'fontsize': 8, 'color': 'white'})  
  
# Annotate quartiles  
boxes = [box.get\_path().vertices for box in boxplot\_dict['boxes']]  
for box in boxes:  
 box\_bottom = box[0, 1]  
 box\_top = box[2, 1]  
 ax.text(box[0, 0], box\_bottom, f'Q1: {box\_bottom:.2f}', ha='center', va='top', fontdict={'fontsize': 8})  
 ax.text(box[2, 0], box\_top, f'Q3: {box\_top:.2f}', ha='center', va='bottom', fontdict={'fontsize': 8})  
  
# Set title and labels  
ax.set\_title(f'Latency Measurements Box Plot {file\_path.split("/")[-1]}')  
ax.set\_ylabel('Latency (ms)')  
ax.set\_xlabel('Measurements')  
plt.grid(True)

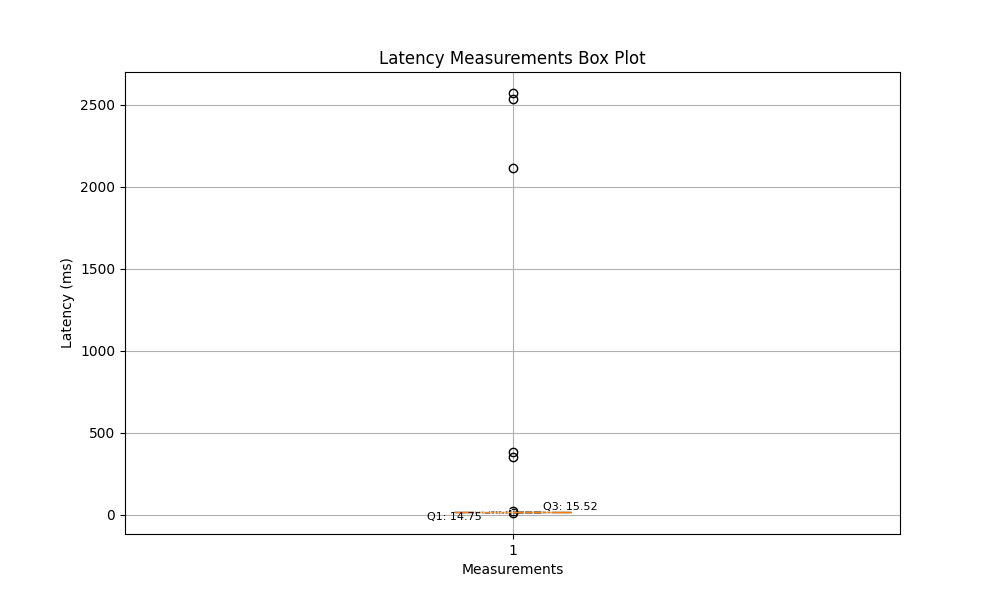
# Show the plot  
plt.show()

This code reads latency measurements from a file, extracts the latency values, and creates a box plot to visualize the distribution of latency measurements. The latency values are sorted and then displayed in a box plot, including median values and quartiles. The resulting plot provides insights into the latency characteristics of the network or system being monitored.

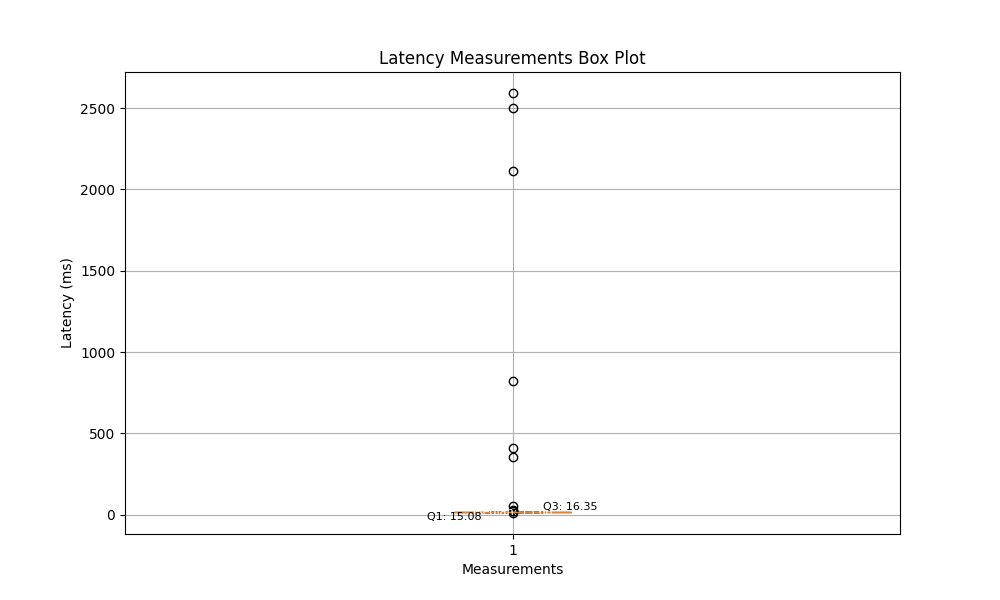
import matplotlib.pyplot as plt  
  
# Initialize lists to store query time lines and CURL lines  
query\_time\_lines = []  
curl\_lines = []  
  
# Open the file and process each line  
with open('files/dig/ns1.bahnhof.net.txt', 'r') as file:  
 skip\_next\_line = False  
 for line in file:  
 stripped\_line = line.strip()  
  
 if skip\_next\_line:  
 skip\_next\_line = False  
 continue  
  
 if "Return" in stripped\_line:  
 skip\_next\_line = True  
 elif "Query" in stripped\_line:  
 query\_time\_lines.append(stripped\_line)  
 elif "CURL" in stripped\_line:  
 curl\_lines.append(stripped\_line)  
  
# Extract query times and connect times  
query\_times\_ms = [float(line.split()[3]) for line in query\_time\_lines]  
connect\_times\_ms = [float(line.split()[3]) for line in curl\_lines]  
  
# Calculate delays and sort them  
delays\_ms = [(query\_time / 1000) + connect\_time for query\_time, connect\_time in zip(query\_times\_ms, connect\_times\_ms)]  
delays\_ms.sort()  
print(delays\_ms)  
  
# Create a boxplot using delays\_ms  
fig, ax = plt.subplots(figsize=(10, 6))  
boxplot\_dict = ax.boxplot(delays\_ms, patch\_artist=True)  
  
# Annotate median values  
medians = [median.get\_ydata()[0] for median in boxplot\_dict['medians']]  
for tick, median in zip(ax.get\_xticks(), medians):  
 ax.text(tick, median, f'Median: {median:.2f}', ha='center', va='center', fontdict={'fontsize': 8, 'color': 'white'})  
  
# Annotate quartiles  
boxes = [box.get\_path().vertices for box in boxplot\_dict['boxes']]  
for box in boxes:  
 box\_bottom = box[0, 1]  
 box\_top = box[2, 1]  
 ax.text(box[0, 0], box\_bottom, f'Q1: {box\_bottom:.2f}', ha='center', va='top', fontdict={'fontsize': 8})  
 ax.text(box[2, 0], box\_top, f'Q3: {box\_top:.2f}', ha='center', va='bottom', fontdict={'fontsize': 8})  
  
# Set title and labels  
ax.set\_title('Latency Measurements Box Plot')  
ax.set\_ylabel('Latency (ms)')  
ax.set\_xlabel('Measurements')  
plt.grid(True)  
# Show the plot  
plt.show()

The functionality of this code is to analyze a network log file containing query times and connect times. It begins by initializing lists to store lines containing query time and CURL time. The code reads each line from the file, identifies and extracts relevant query and CURL lines while skipping unnecessary lines marked by "Return." The extracted query and CURL times are collected and converted into lists. The code then calculates delays by adding query times (converted to seconds) and connect times, sorts these delay values in ascending order, and displays them in a box plot using Matplotlib. The box plot visualizes the distribution of delay data, including median and quartiles, aiding in the analysis of network performance or system response times.

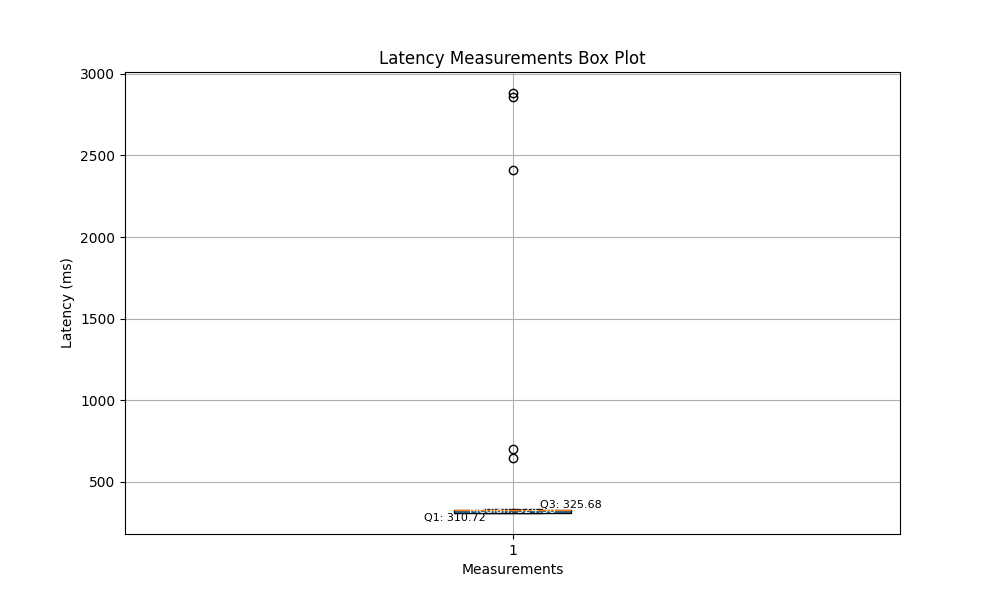
AS1.d1:



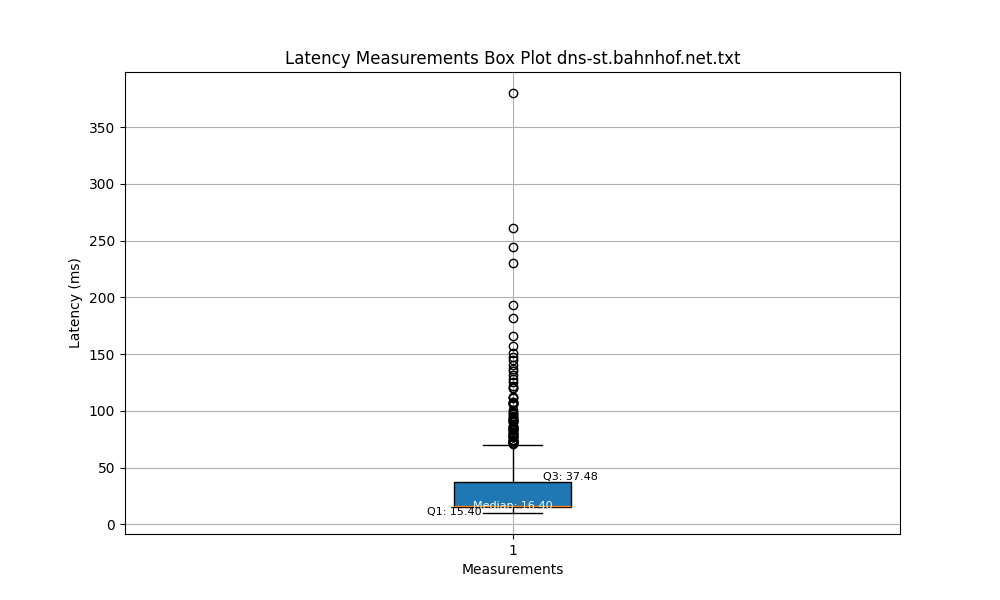
AS1.d2:



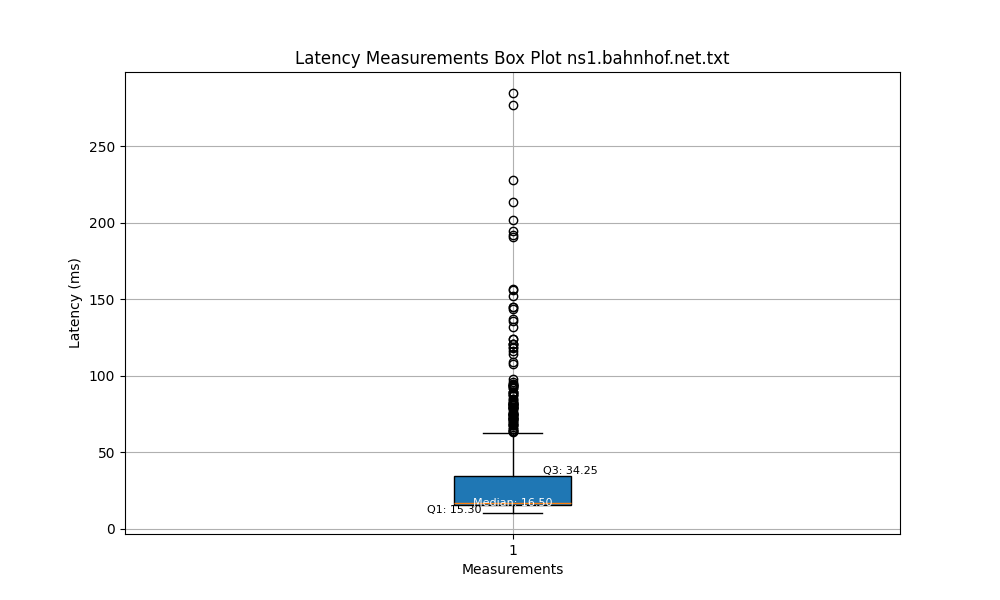
AS1.d3:



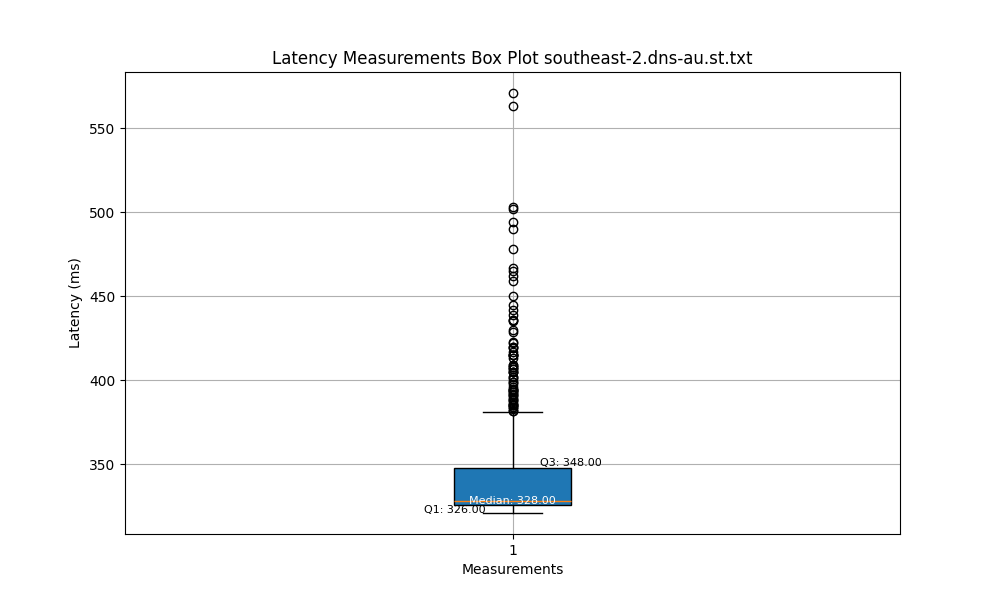
AS1.n1:



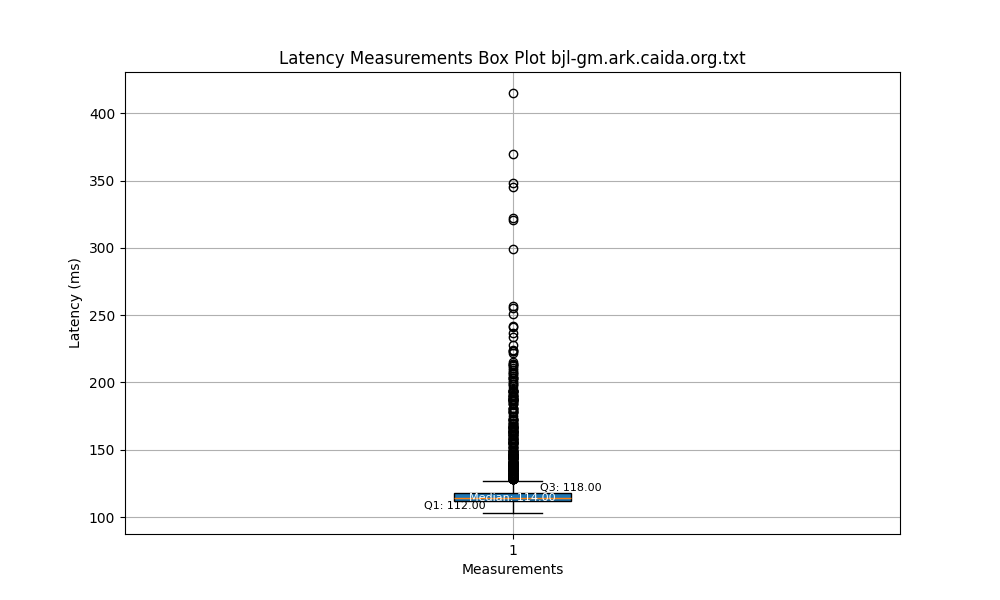
AS1.n2:



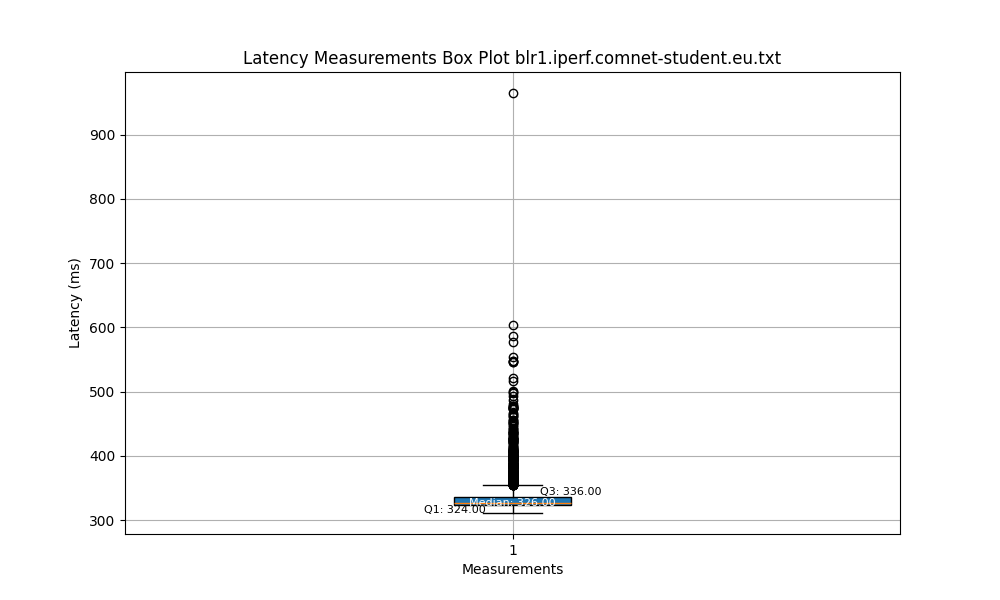
AS1.n3:



AS1.r1:



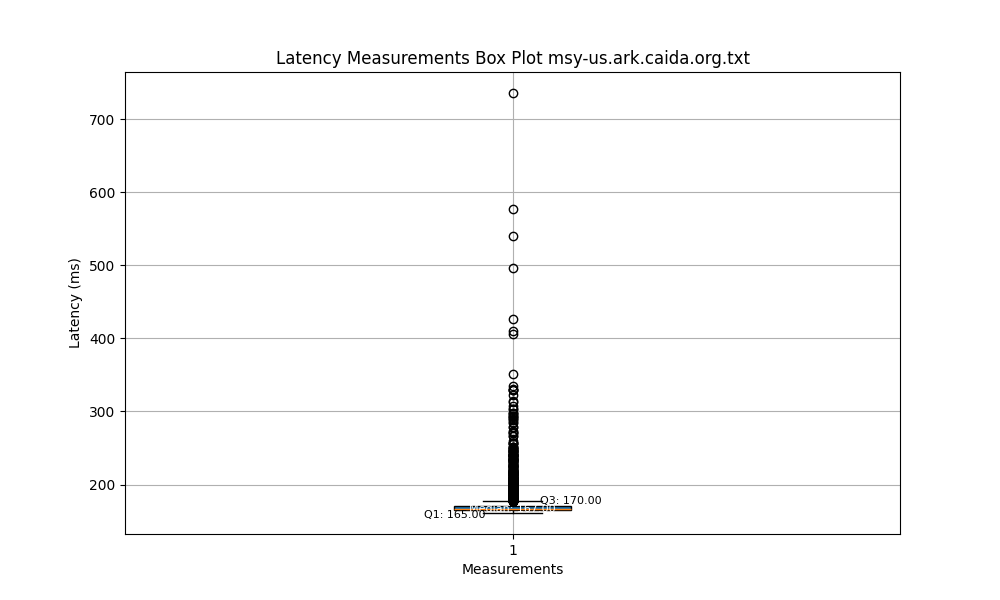
AS1.r2:



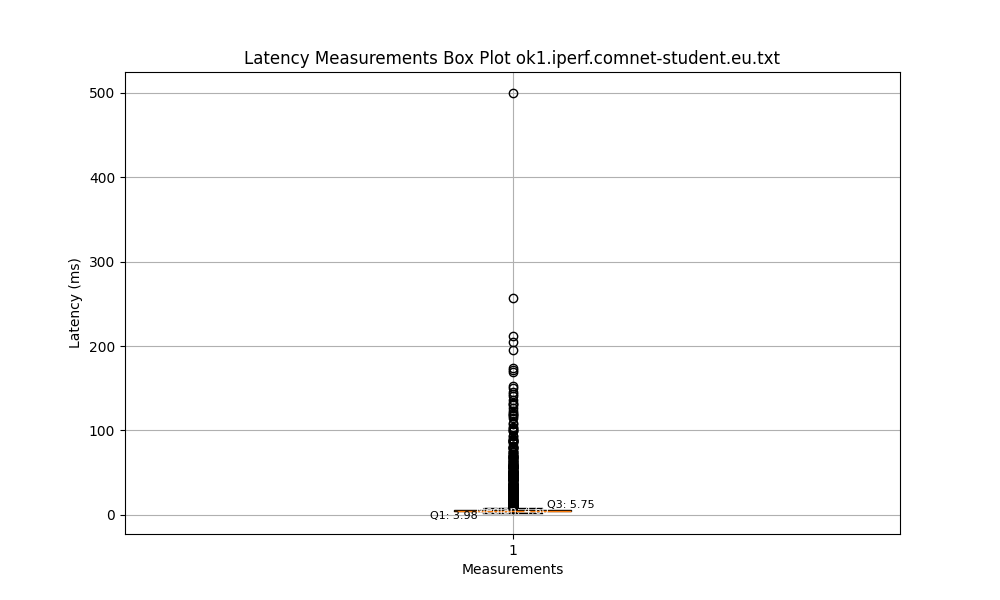
AS1.r3:



AS1.i1:



AS1.i2:



* Notable observations

AS1.d1: The median latency is approximately 15 milliseconds, indicating that most measurement values are relatively low. The interquartile range is very small, suggesting that the majority of the data is concentrated within a narrow latency range. There are some outliers well above Q3, representing individual latency spikes.

AS1.d2: The median latency is around 16 milliseconds. The interquartile range is slightly larger, indicating some dispersion in the data. There are many outliers well above Q3, suggesting frequent latency spikes.

AS1.d3: The median latency is approximately 325 milliseconds, indicating a higher latency level. The interquartile range is quite large, reflecting significant differences among latency measurements. There are several latency outliers, with values much higher than the rest of the measurements.

AS1.n1: The median latency is around 16 milliseconds, indicating that most ICMP response times are fast. The interquartile range is very tight, indicating high consistency among the measurement results. There are a few outliers, indicating occasional latency spikes.

AS1.n2: The median latency is also around 16 milliseconds. The interquartile range is slightly wider than AS1.n1, but the overall measurements are still concentrated. There are some outliers, suggesting occasional latency spikes.

AS1.n3: The median latency is approximately 328 milliseconds, indicating a generally higher latency level. The interquartile range is relatively consistent with AS1.n2, suggesting data consistency. There are few outliers, indicating infrequent occurrences of high latency in ICMP requests.

AS1.r1: The median latency is approximately 38 milliseconds. The interquartile range (IQR), which represents the middle 50% of the data, is very narrow, indicating that most of the latency measurements are closely clustered around the median. There are many outliers extending to about 250 milliseconds, suggesting occasional spikes in latency.

AS1.r2: The median latency is approximately 170 milliseconds. The IQR is wider than that of server 1, indicating greater variability in the latency measurements. There is a significant number of outliers exceeding 400 milliseconds, showing that this server occasionally experiences high latency.

AS1.r3: The median latency is approximately 170 milliseconds. The IQR is quite small, suggesting less variability in the data compared to server 2. There are outliers, with some latency measurements exceeding 600 milliseconds.

AS1.i1: The median latency is very low, about 5.75 milliseconds, which is the lowest among all the servers, including the research servers. The IQR is extremely tight, indicating a highly consistent latency performance. The range of outliers is minimal, with the highest latency measurement just slightly above 100 milliseconds.

AS1.i2: The median latency of this server is the highest among all the servers, at approximately 336 milliseconds. The IQR is relatively large, showing more variability in latency compared to the other iperf server. The range of outliers is wide, extending to nearly 900 milliseconds, suggesting that this server has the most latency spikes.

* Differences in AS1.d\_N and AS1.n\_N

Datasets AS1.d1, d2, d3: These datasets exhibit a very wide range of latency with a very high upper limit, indicating the potential for significant delays. The median values are relatively low compared to the overall range, suggesting that despite instances of extremely high latency, typical (median) performance is much better. The interquartile ranges (IQRs) are small, especially when compared to the range of outliers, indicating that the majority of data points are clustered within a narrow range, but with notable spikes. All three datasets have outliers, with some extreme cases where latency reaches up to 2000-3000 milliseconds, signaling occasional but very high latency peaks.

Datasets AS1.n1, n2, n3: Compared to the first three datasets, these datasets have a much narrower range of latency. This suggests a more stable and consistent latency experience. The medians are positioned towards the lower end of the range, indicating that more than half of the measurements are on the faster side. The IQRs are relatively tight, which shows that the middle 50% of measurements are closely grouped together. There are fewer and less extreme outliers compared to the first three datasets, indicating fewer instances of very high latency.

In summary, the AS1.d datasets indicate the possibility of high latency and more variability, while the AS1.n datasets display a more stable latency performance with fewer extreme outliers.

* Includes packet loss

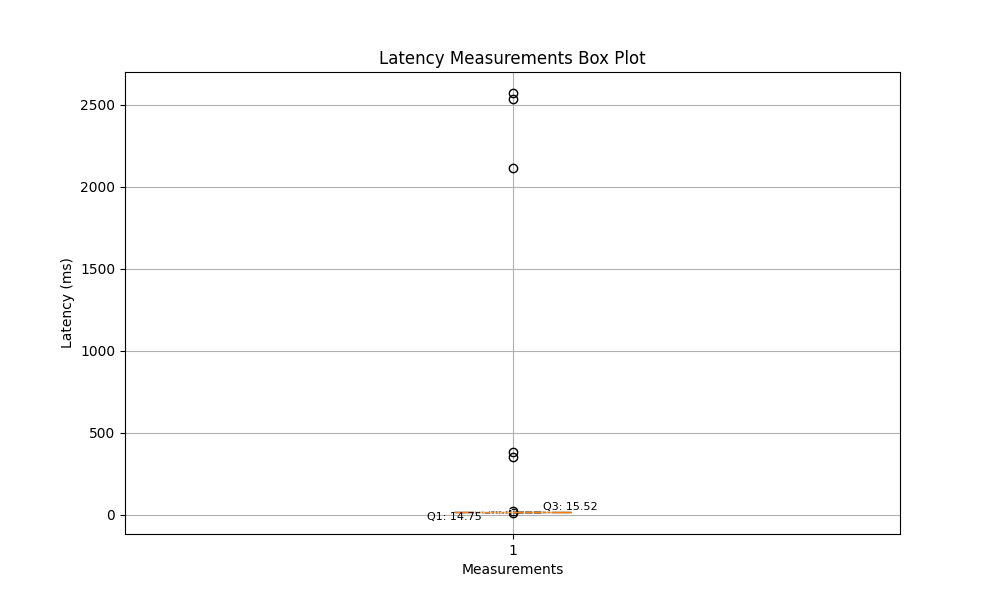
import matplotlib.pyplot as plt  
  
# Initialize a list to store latency measurements  
latency\_measurements = []  
file\_path = 'files/ping/bjl-gm.ark.caida.org.txt'  
  
# Open the file and read each line  
with open(file\_path, 'r') as file:  
 for line in file:  
 if 'time=' in line:  
 # Find lines containing 'time=' and split the string  
 parts = line.split()  
 for part in parts:  
 if part.startswith('time='):  
 # Extract the latency measurement and remove 'ms'  
 latency\_value = part.split('=')[1].rstrip(' ms')  
 try:  
 # Convert the latency measurement to a float and store it  
 if float(latency\_value) > 2000.0:  
 latency\_measurements.append(2000.0)  
 else:  
 latency\_measurements.append(float(latency\_value))  
 except ValueError:  
 # If conversion fails, skip the value  
 continue  
 if 'packet loss' in line:  
 parts = line.split(',')  
 for part in parts:  
 if 'packet loss' in part:  
 # Extract the numeric value before the percentage sign  
 packet\_loss\_percentage = part.split('%')[0].strip()  
 try:  
 # Convert the packet loss percentage to a float and add outliers to the list  
 for i in range(0, int(float(packet\_loss\_percentage) / 20.0)):  
 latency\_measurements.append(2000.0)  
 except ValueError:  
 # If conversion fails, ignore the value  
 continue  
  
# Create a boxplot using latency\_measurements  
fig, ax = plt.subplots(figsize=(10, 6))  
boxplot\_dict = ax.boxplot(latency\_measurements, patch\_artist=True)  
  
# Annotate median values  
medians = [median.get\_ydata()[0] for median in boxplot\_dict['medians']]  
for tick, median in zip(ax.get\_xticks(), medians):  
 ax.text(tick, median, f'Median: {median:.2f}', ha='center', va='center', fontdict={'fontsize': 8, 'color': 'white'})  
  
# Annotate quartile values  
boxes = [box.get\_path().vertices for box in boxplot\_dict['boxes']]  
for box in boxes:  
 box\_bottom = box[0, 1]  
 box\_top = box[2, 1]  
 ax.text(box[0, 0], box\_bottom, f'Q1: {box\_bottom:.2f}', ha='center', va='top', fontdict={'fontsize': 8})  
 ax.text(box[2, 0], box\_top, f'Q3: {box\_top:.2f}', ha='center', va='bottom', fontdict={'fontsize': 8})  
  
# Set title and labels  
ax.set\_title('Latency Measurements Box Plot')  
ax.set\_ylabel('Latency (ms)')  
ax.set\_xlabel('Measurements')  
plt.grid(True)  
plt.show()

This code reads latency measurements and packet loss information from a text file, processes the data to calculate latency values, and then creates a box plot visualization of these latency measurements. The latency measurements are extracted from lines containing 'time=' and 'packet loss' in the input file. Latency values are collected and outliers are added for packet loss percentages. The resulting latency measurements are visualized as a box plot, providing insights into the distribution of latency values in the data. The box plot includes median and quartile annotations to summarize the statistics, and it is displayed using Matplotlib.

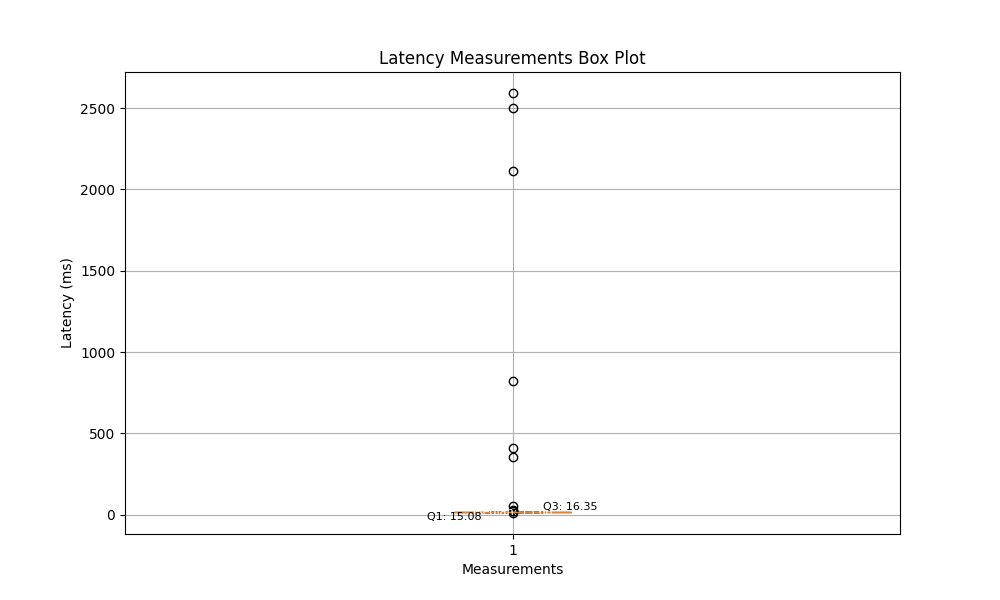
import re  
import matplotlib.pyplot as plt  
  
# Initialize lists with more descriptive names  
query\_time\_lines = []  
curl\_lines = []  
latency\_ms = []  
  
# Open and read the file  
with open('files/dig/dns-st.bahnhof.net.reloaded.txt', 'r') as file:  
 skip\_next\_line = False  
 for line in file:  
 stripped\_line = line.strip()  
 if skip\_next\_line:  
 skip\_next\_line = False  
 continue  
 if re.search("Return", stripped\_line):  
 skip\_next\_line = True  
 latency\_ms.append(2000.0)  
 elif re.search("Query", stripped\_line):  
 query\_time\_lines.append(stripped\_line)  
 elif re.search("CURL", stripped\_line):  
 curl\_lines.append(stripped\_line)  
  
# Initialize lists for query time and connect time  
query\_times\_ms = []  
connect\_times\_ms = []  
  
# Extract query and connect times  
for item in query\_time\_lines:  
 item\_split = item.split(" ")  
 query\_time = float(item\_split[3])  
 query\_times\_ms.append(query\_time)  
  
for item in curl\_lines:  
 item\_split = item.split(" ")  
 connect\_time = float(item\_split[3])  
 connect\_times\_ms.append(connect\_time)  
  
# Calculate total delay  
total\_delays\_ms = [query\_time / 1000 + connect\_time for query\_time, connect\_time in zip(query\_times\_ms, connect\_times\_ms)]  
total\_delays\_ms.sort()  
  
# Create a box plot  
fig, ax = plt.subplots(figsize=(10, 6))  
boxplot\_dict = ax.boxplot(total\_delays\_ms, patch\_artist=True)  
  
# Annotate median  
medians = [median.get\_ydata()[0] for median in boxplot\_dict['medians']]  
for tick, median in zip(ax.get\_xticks(), medians):  
 ax.text(tick, median, f'Median: {median:.2f}', ha='center', va='center', fontdict={'fontsize': 8, 'color': 'white'})  
  
# Annotate quartiles  
boxes = [box.get\_path().vertices for box in boxplot\_dict['boxes']]  
for box in boxes:  
 box\_bottom = box[0, 1]  
 box\_top = box[2, 1]  
 ax.text(box[0, 0], box\_bottom, f'Q1: {box\_bottom:.2f}', ha='center', va='top', fontdict={'fontsize': 8})  
 ax.text(box[2, 0], box\_top, f'Q3: {box\_top:.2f}', ha='center', va='bottom', fontdict={'fontsize': 8})  
  
# Set title and labels  
ax.set\_title('Latency Measurements Box Plot')  
ax.set\_ylabel('Latency (ms)')  
ax.set\_xlabel('Measurements')  
plt.grid(True)  
plt.show()

The functionality of this code is to read data from a text file that contains information about query times and connection times, calculate the total latency, and generate a box plot to visualize the latency data. The code starts by extracting information about query times and connection times from the file and then adding them together to calculate the total latency in milliseconds. It proceeds to sort the total latencies and create a box plot to display the distribution of latency data, with annotations for statistics such as the median and quartiles for better understanding of the data's characteristics. Finally, the generated box plot is displayed using the Matplotlib library.

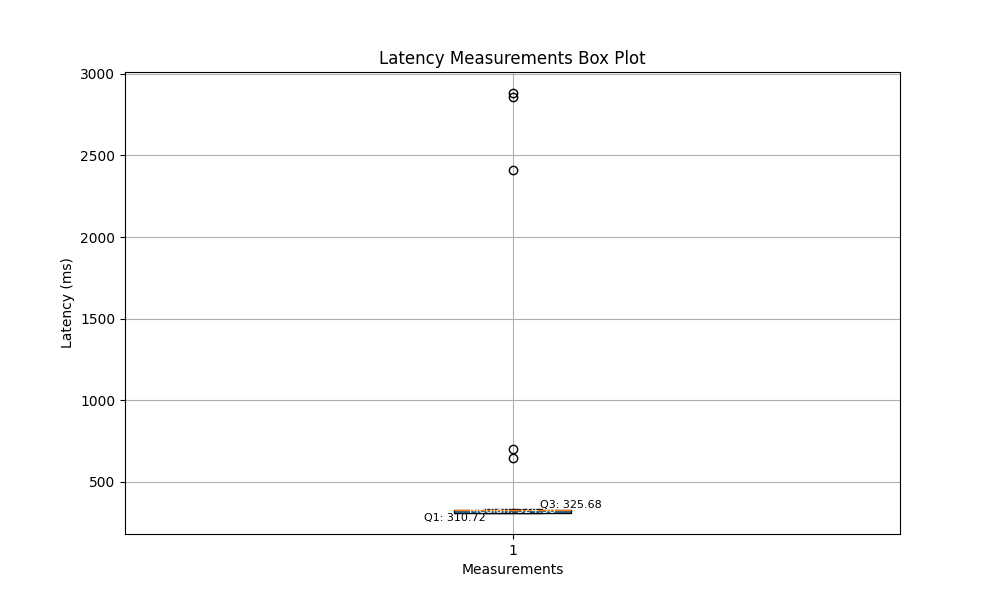
AS1.d1:



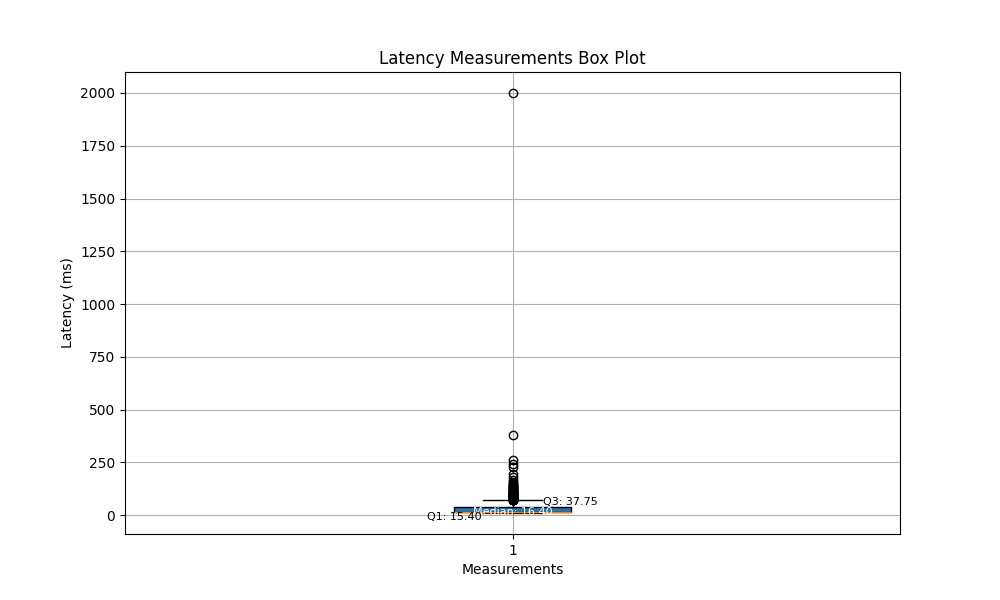
AS1.d2:



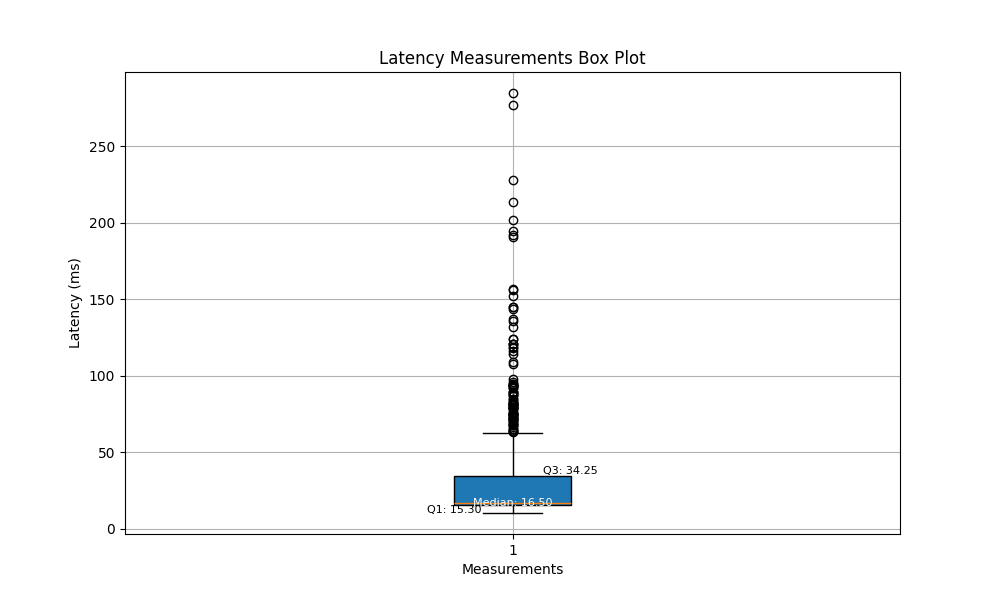
AS1.d3:



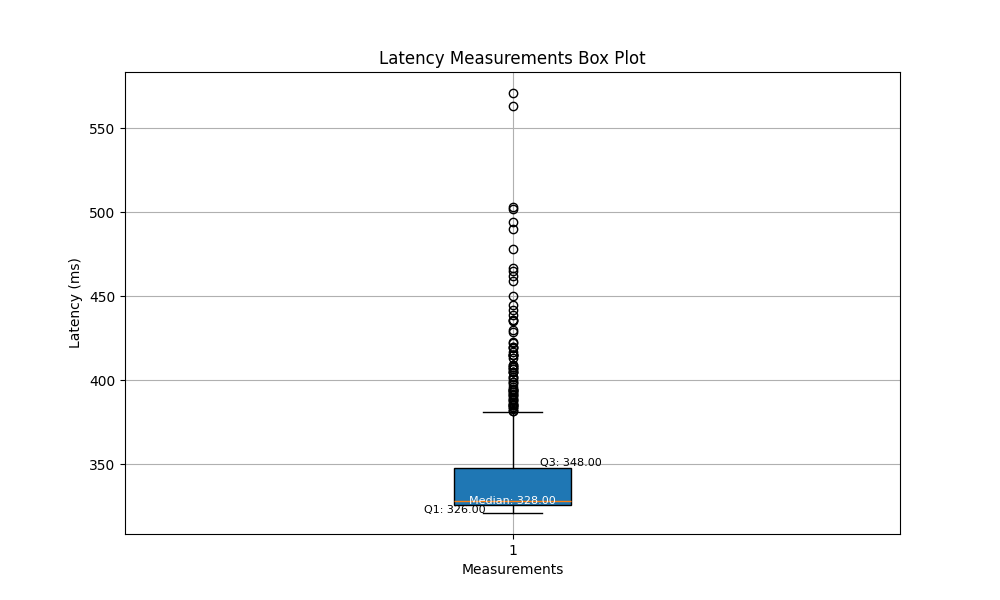
AS1.n1:



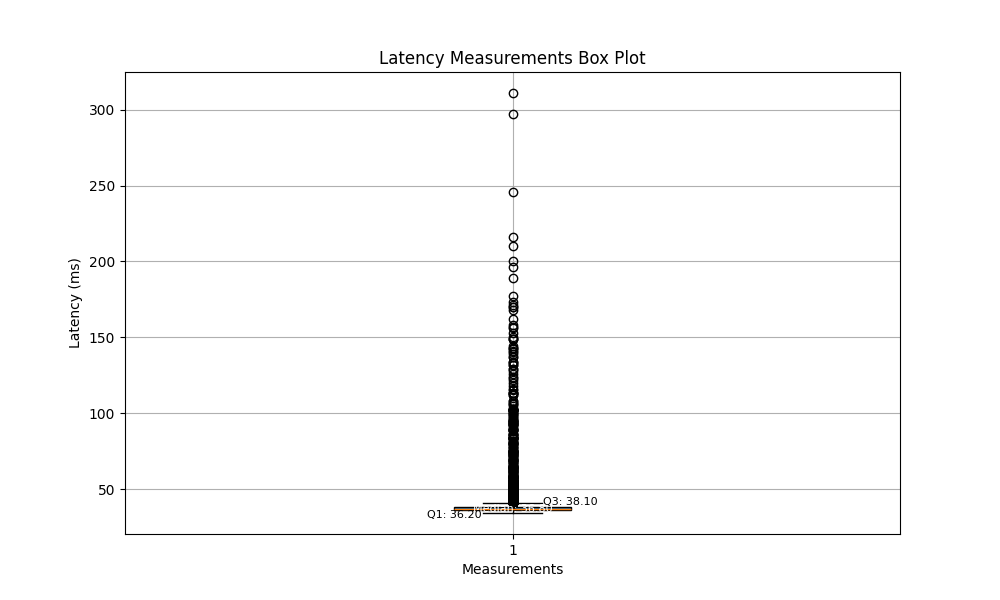
AS1.n2:



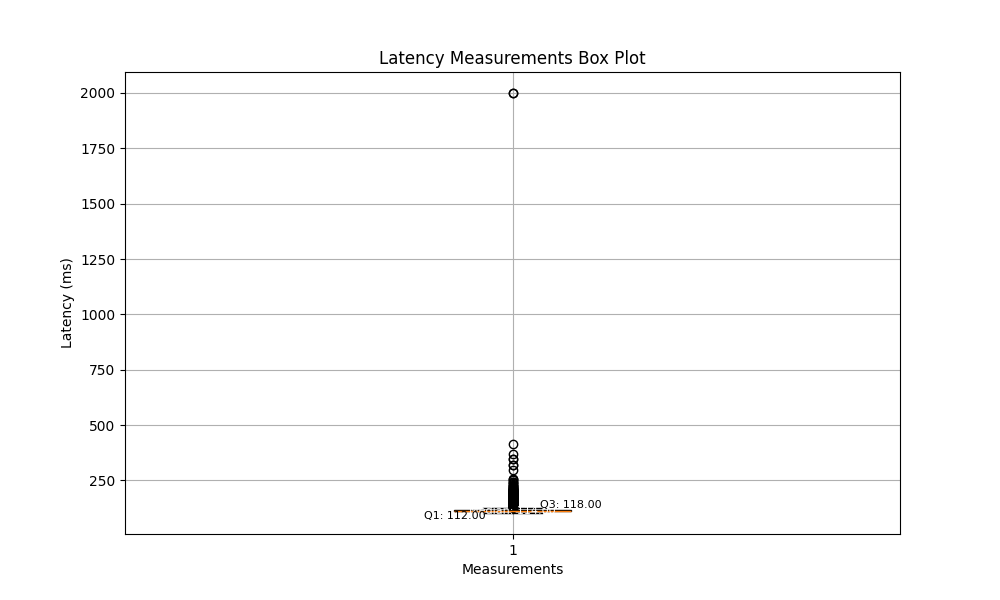
AS1.n3:



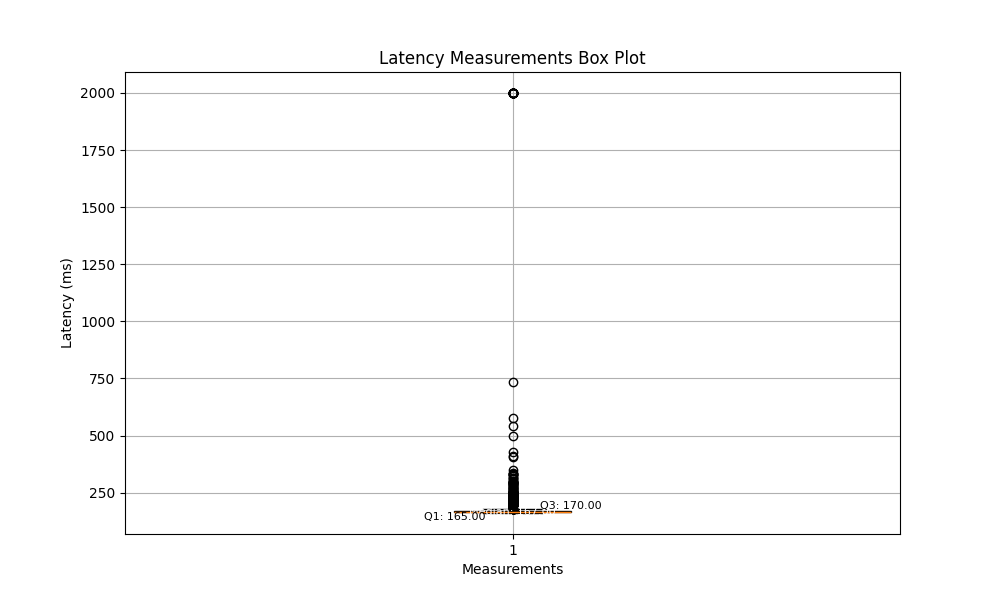
AS1.r1:



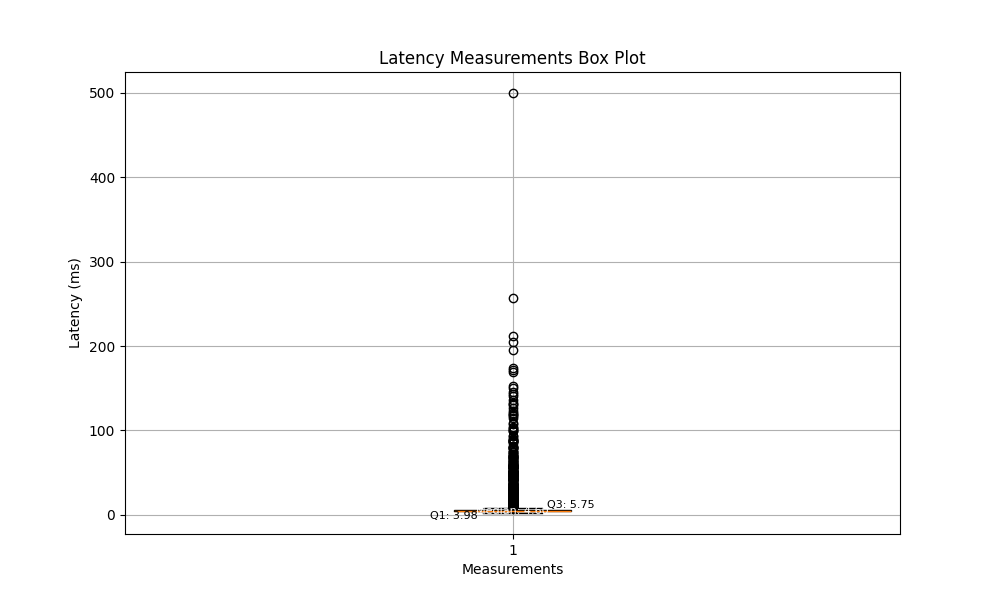
AS1.r2:



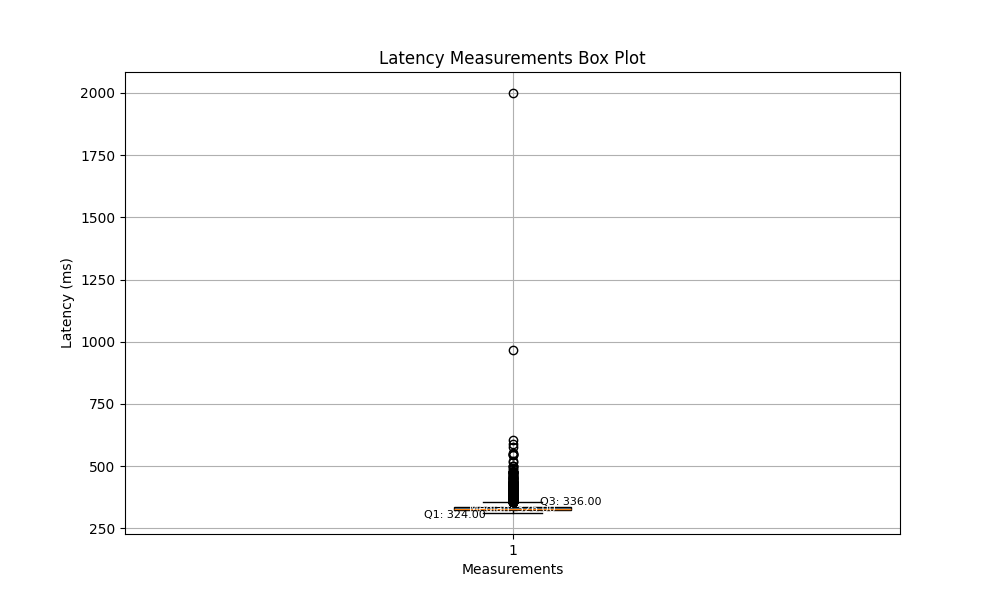
AS1.r3:



AS1.i1:



AS1.i2:



* Notable observations

AS1.d1: The median is relatively low, indicating that most latency measurements are concentrated in a lower range. There are several outliers far from the main cluster of data, suggesting occasional peaks in latency.

AS1.d2: The median is also low, but there is a slightly increased interquartile range, indicating more variation in latency. This data set also contains outliers, suggesting occasional higher latencies.

AS1.d3: Both the median and the interquartile range are higher, indicating greater variability in the measurements. The consistent presence of outliers suggests that latency peaks are not uncommon.

AS1.n1: The latency measurements are very tightly grouped, with a very small interquartile range, indicating very consistent latency performance. There are no outliers in this plot, suggesting that latency is stable with no significant spikes.

AS1.n2: The interquartile range is slightly higher than that of AS1.n1, but the latency measurements still cluster tightly. The presence of some outliers indicates that although the server generally performs consistently, there are occasional spikes in latency.

AS1.n3: The median latency is higher and the interquartile range is also wider, indicating less consistency in latency. There are several outliers, indicating that compared to other ICMP measurements, there are more frequent latency spikes.

AS1.r1: The measurements are clustered around a lower median, indicating a generally low latency. The interquartile range (IQR) is quite compact, suggesting consistent latency performance. There are a number of outliers above the upper quartile, indicating occasional spikes in latency.

AS1.r2: The median is higher than in AS1.r1, with a larger IQR, which implies greater variability in latency. The range of the outliers is extensive, suggesting that there are significant deviations in latency at times.

AS1.r3: This plot shows an even higher median and a wide IQR, reflecting even more variability in latency compared to AS1.r2. There is a similar pattern of outliers, which are spread out, showing the potential for large latency peaks.

AS1.i1: The median latency is low, with a very narrow IQR, indicating a very stable and consistent latency profile. While there are outliers, they are not as pronounced or as far from the IQR as in the research server plots.

AS1.i2: The median is slightly higher than that of AS1.i1, with a comparable IQR, suggesting a good consistency in latency, similar to AS1.i1. The presence of outliers is minimal, which shows that the server maintains a stable latency with occasional variations.

* Differences in AS1.d\_N and AS1.n\_N

The plots for AS1.d1, d2, and d3 exhibit wider interquartile ranges and more outliers, suggesting greater variability in the latency of DNS queries. In contrast, the interquartile ranges for AS1.n1, n2, and n3 are narrower, particularly for AS1.n1 and AS1.n2, indicating more consistent latency for ICMP requests.

The median positions for the DNS servers are higher, especially for AS1.d3, indicating that the average latency is higher compared to ICMP requests. The medians for ICMP are generally lower, suggesting these servers typically respond faster.

The DNS datasets have a greater number of outliers with relatively high latency values, indicating occasional significant delays. There are fewer outliers in the ICMP datasets, and these are closer to the median relative to the interquartile range, suggesting smaller fluctuations in latency for these datasets.

Overall, the latency measurements for DNS queries demonstrate greater variability and instability, while the latency for ICMP requests is relatively lower and more stable. This might reflect the performance characteristics of different types of servers in processing requests.

* PDF and CDF plots

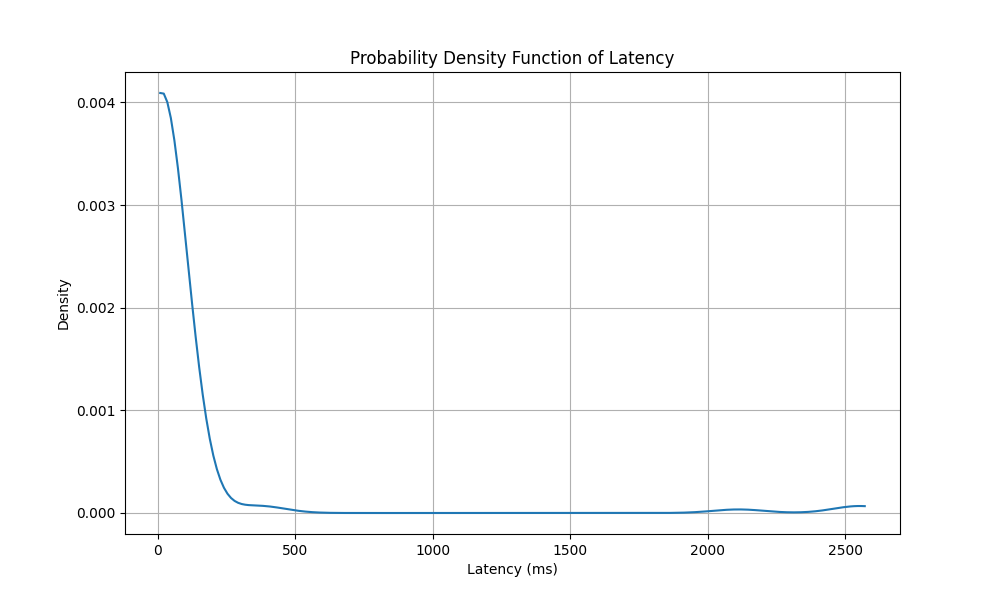
import re  
import numpy as np  
import matplotlib.pyplot as plt  
from scipy.stats import gaussian\_kde  
  
# Initialize lists with more descriptive names  
query\_time\_lines = []  
curl\_lines = []  
query\_times\_ms = []  
connect\_times\_ms = []  
delay\_ms = []  
  
# Read the file and process each line  
with open('files/dig/dns-st.bahnhof.net.reloaded.txt', 'r') as file:  
 skip\_flag = False  
 for line in file:  
 striped\_line = line.strip()  
 if skip\_flag:  
 skip\_flag = False  
 continue  
 if re.search("Return", striped\_line):  
 skip\_flag = True  
 continue  
 if re.search("Query", striped\_line):  
 query\_time\_lines.append(striped\_line)  
 if re.search("CURL", striped\_line):  
 curl\_lines.append(striped\_line)  
  
# Extract query times and connect times  
for item in query\_time\_lines:  
 item\_split = item.split(" ")  
 query\_time = float(item\_split[3])  
 query\_times\_ms.append(query\_time)  
  
for item in curl\_lines:  
 item\_split = item.split(" ")  
 connect\_time = float(item\_split[3])  
 connect\_times\_ms.append(connect\_time)  
  
# Calculate delays  
for i in range(len(query\_time\_lines)):  
 delay = query\_times\_ms[i] / 1000 + connect\_times\_ms[i]  
 delay\_ms.append(delay)  
  
# Calculate probability density function (PDF)  
density = gaussian\_kde(delay\_ms)  
xs = np.linspace(min(delay\_ms), max(delay\_ms), 200)  
density.covariance\_factor = lambda: .25  
density.\_compute\_covariance()  
  
# Plot PDF  
plt.figure(figsize=(10, 6))  
plt.plot(xs, density(xs))  
plt.title('Probability Density Function of Latency')  
plt.xlabel('Latency (ms)')  
plt.ylabel('Density')  
plt.grid(True)  
plt.show()  
  
# Calculate cumulative distribution function (CDF)  
sorted\_data = np.sort(delay\_ms)  
yvals = np.arange(len(sorted\_data)) / float(len(sorted\_data) - 1)  
  
# Plot CDF  
plt.figure(figsize=(10, 6))  
plt.plot(sorted\_data, yvals)  
plt.title('Cumulative Distribution Function of Latency')  
plt.xlabel('Latency (ms)')  
plt.ylabel('Cumulative Probability')  
plt.grid(True)  
plt.show()

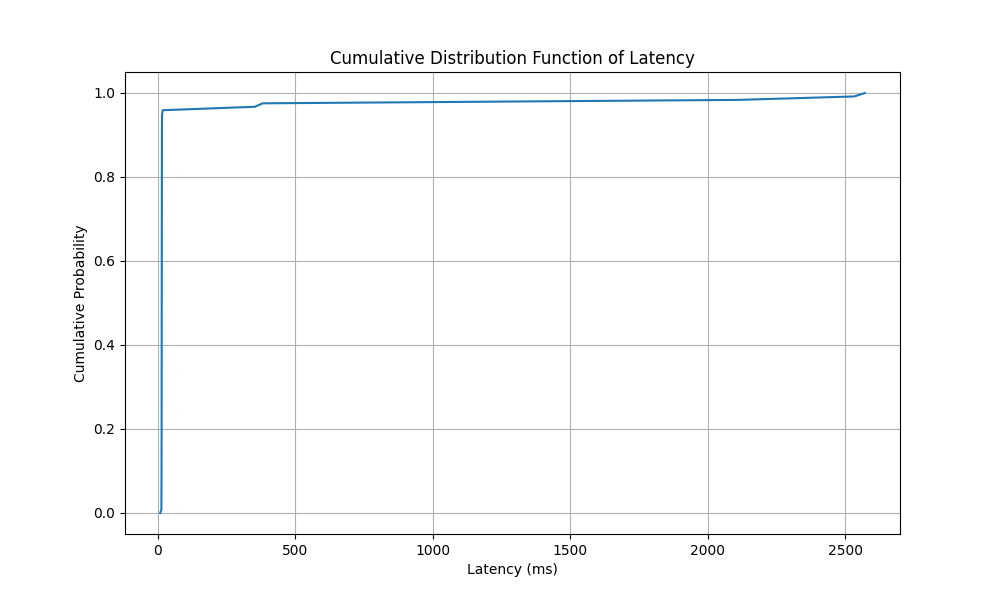
This code reads data from a text file containing latency measurements and calculates both the Probability Density Function (PDF) and Cumulative Distribution Function (CDF) of the latency values. It first processes the input file to extract latency measurements, combines query and connect times to calculate latency delays, and then uses Gaussian kernel density estimation to compute the PDF of the latency values. Finally, it calculates and plots the CDF of the latency values.

import matplotlib.pyplot as plt  
import numpy as np  
from scipy.stats import gaussian\_kde  
  
# Initialize a list to store extracted latency values  
latency\_values = []  
  
# Open the file and read it line by line  
with open('files/ping/southeast-2.dns-au.st.txt', 'r') as file:  
 for line in file:  
 if 'time=' in line:  
 # Find lines containing 'time=' and split the string  
 parts = line.split()  
 for part in parts:  
 if part.startswith('time='):  
 # Extract latency values and remove 'ms'  
 latency = part.split('=')[1].rstrip(' ms')  
 try:  
 # Convert extracted latency values to floating-point and store them  
 latency\_values.append(float(latency))  
 except ValueError:  
 # Skip the value if conversion fails  
 continue  
  
# Calculate the probability density function (PDF)  
density = gaussian\_kde(latency\_values)  
xs = np.linspace(min(latency\_values), max(latency\_values), 200)  
density.covariance\_factor = lambda: .25  
density.\_compute\_covariance()  
  
# Plot the PDF  
plt.figure(figsize=(10, 6))  
plt.plot(xs, density(xs))  
plt.title('Probability Density Function of Latency')  
plt.xlabel('Latency (ms)')  
plt.ylabel('Density')  
plt.grid(True)  
plt.show()  
  
# Calculate the cumulative distribution function (CDF)  
sorted\_data = np.sort(latency\_values)  
yvals = np.arange(len(sorted\_data)) / float(len(sorted\_data) - 1)  
  
# Plot the CDF  
plt.figure(figsize=(10, 6))  
plt.plot(sorted\_data, yvals)  
plt.title('Cumulative Distribution Function of Latency')  
plt.xlabel('Latency (ms)')  
plt.ylabel('Cumulative Probability')  
plt.grid(True)  
plt.show()

This code reads latency measurements from a text file, extracts and processes the latency values, and then calculates and visualizes two statistical functions: the Probability Density Function (PDF) and Cumulative Distribution Function (CDF) of the latency data. It does this by parsing the file to extract latency values, computing a PDF using kernel density estimation, and plotting it to show the probability distribution of latency values. Additionally, it computes a CDF to illustrate the cumulative probability of latency values and displays it using Matplotlib.

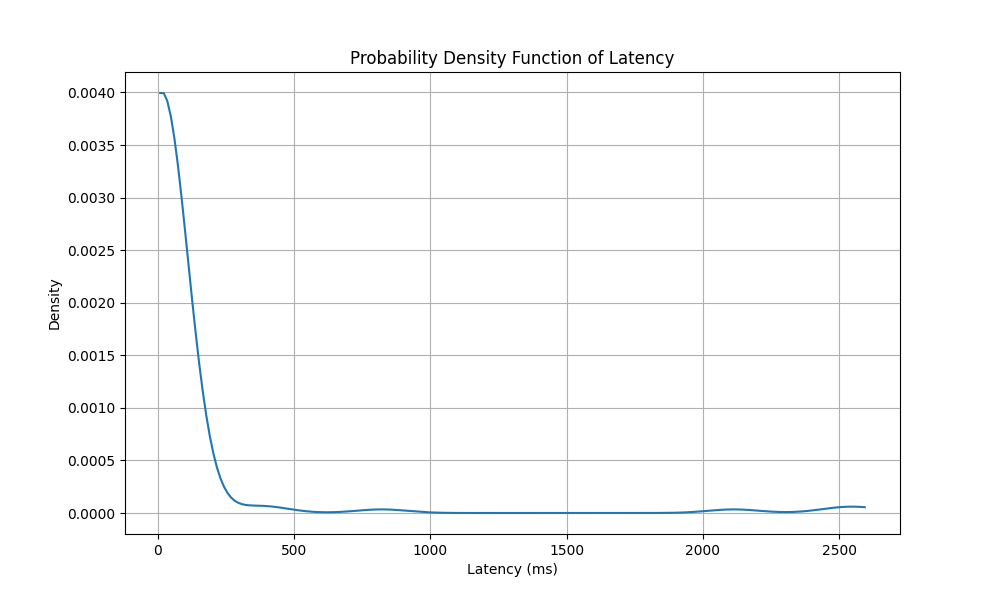
AS1.d1:

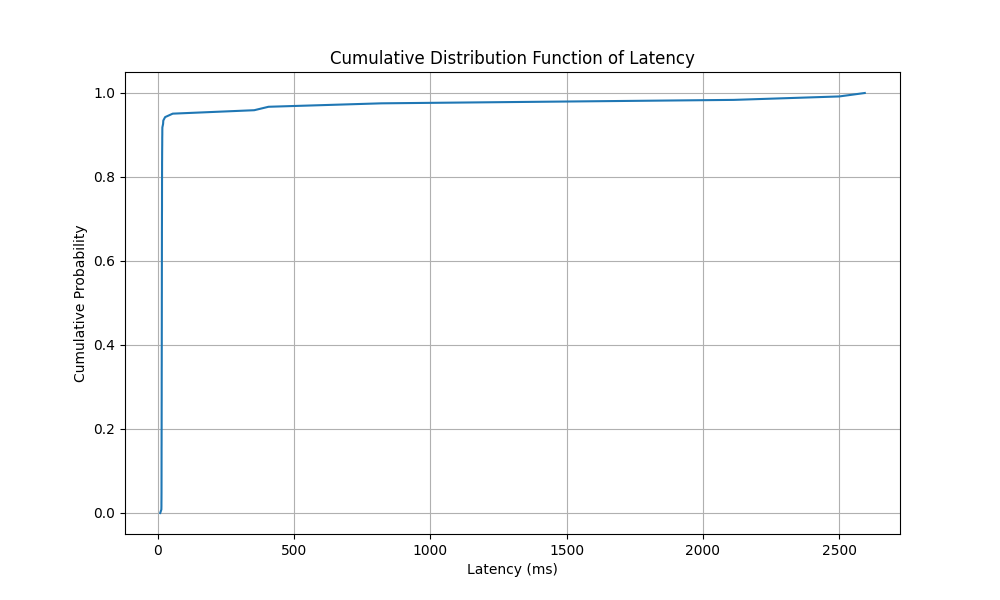




The PDF plot reveals that most latency values are concentrated in the lower latency range, with a peak occurring near zero and then rapidly decreasing, indicating that latency values are primarily concentrated in a small range. The CDF plot shows that almost all latency values are concentrated below 500 milliseconds, with nearly all of the probability mass concentrated in this region, demonstrating the tightness of the latency distribution.

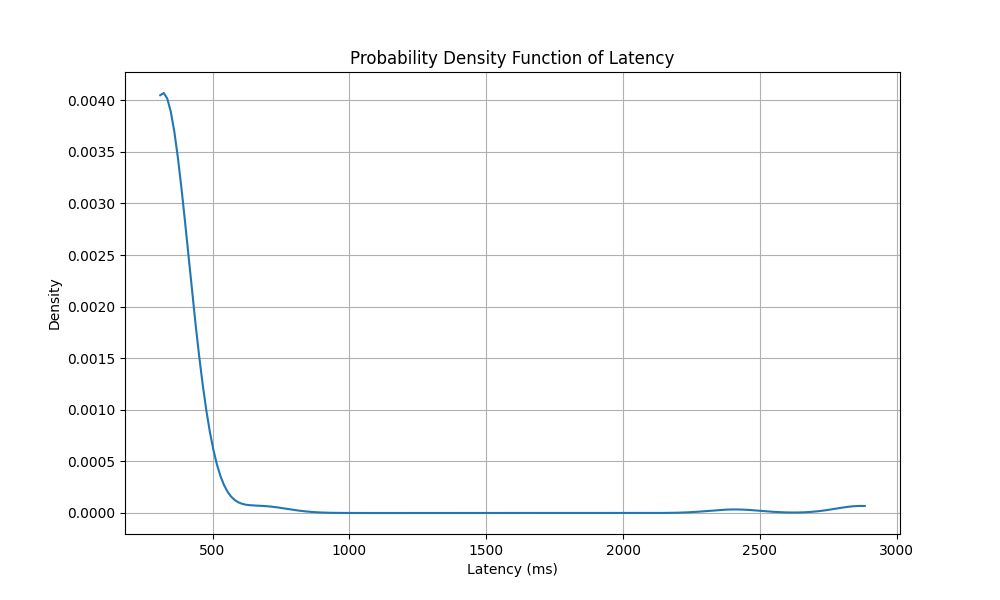
AS1.d2:

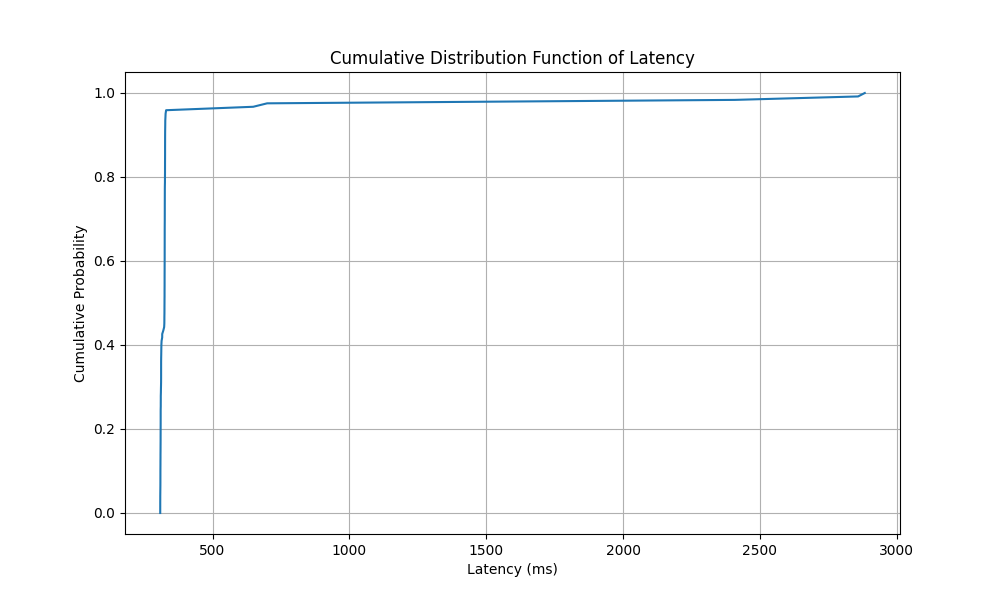




The shape of the PDF is similar to that of AS1.d1, but the tail of the latency value distribution is longer, indicating a wider range of latency values. The shape of the CDF shows that the cumulative probability rises slightly more slowly, indicating that some latency values are distributed in a broader range, but the vast majority of latency values are still concentrated in the lower latency range.

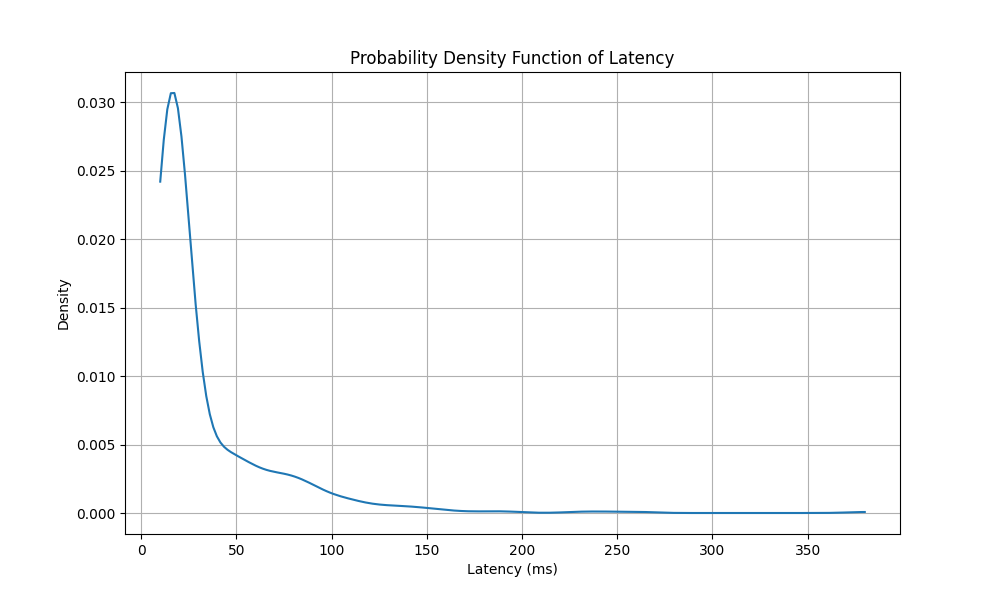
AS1.d3:

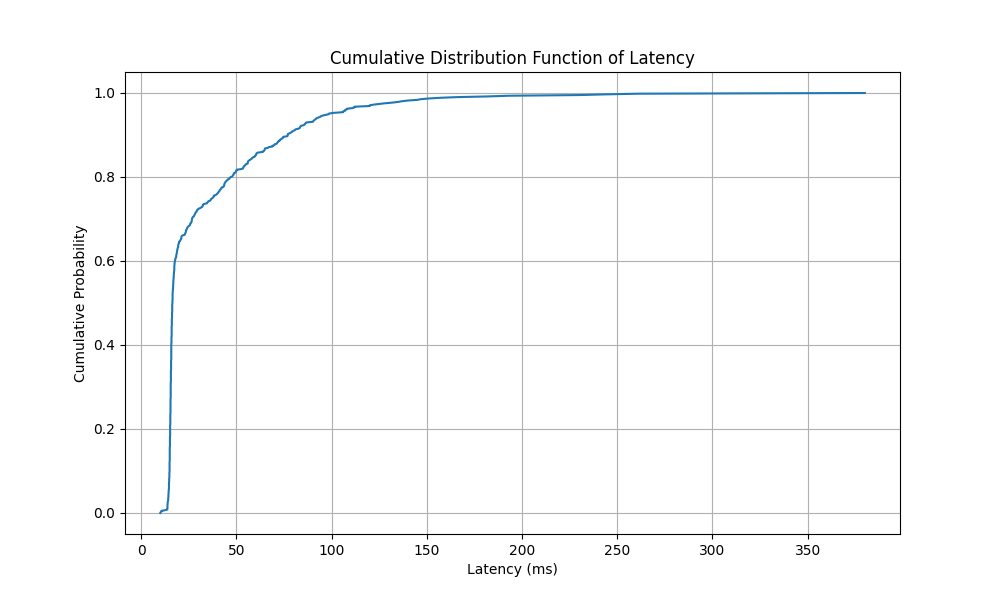




The shape of the PDF is similar to the first two datasets, but the concentration of latency values is not as pronounced as in AS1.d1, and the tail is longer than AS1.d2, indicating greater variability in latency values. The CDF shows a wider range of latency values, with a more uniform distribution of latency values. The cumulative probability rises more gradually, indicating that latency values are dispersed over a larger range.

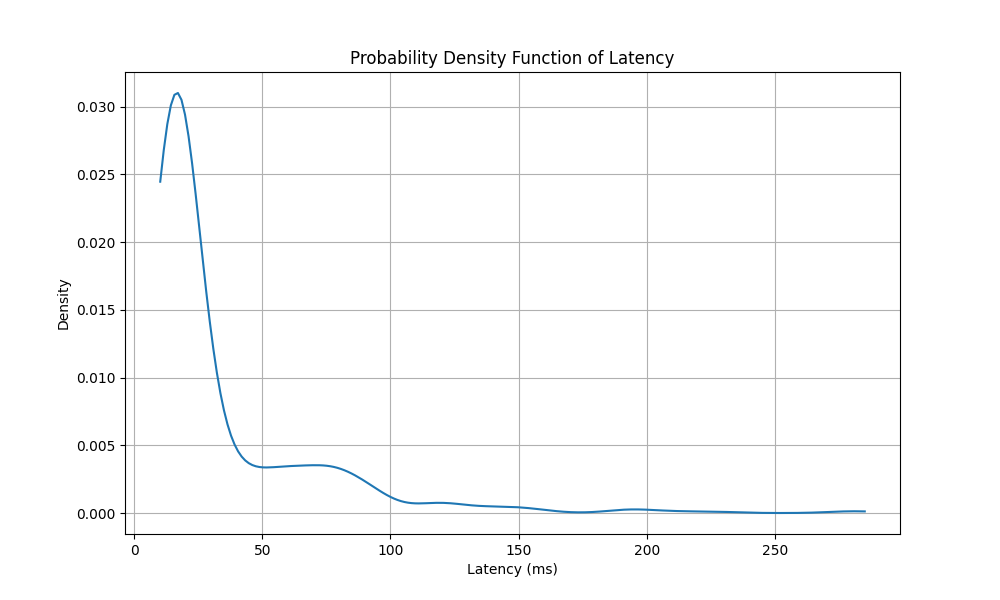
AS1.n1:

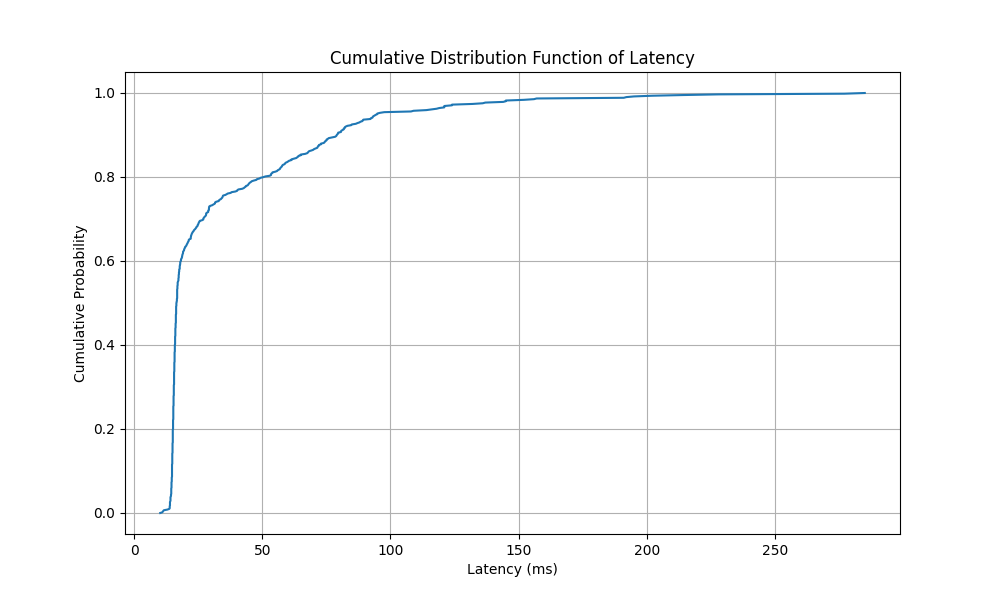




The PDF shows that latency is primarily concentrated in a very low range with a very high peak, indicating that most latency values are very low. The CDF shows rapid accumulation of latency values, with almost all values below 50 milliseconds, indicating that nearly all network requests have very fast response times.

AS1.n2:

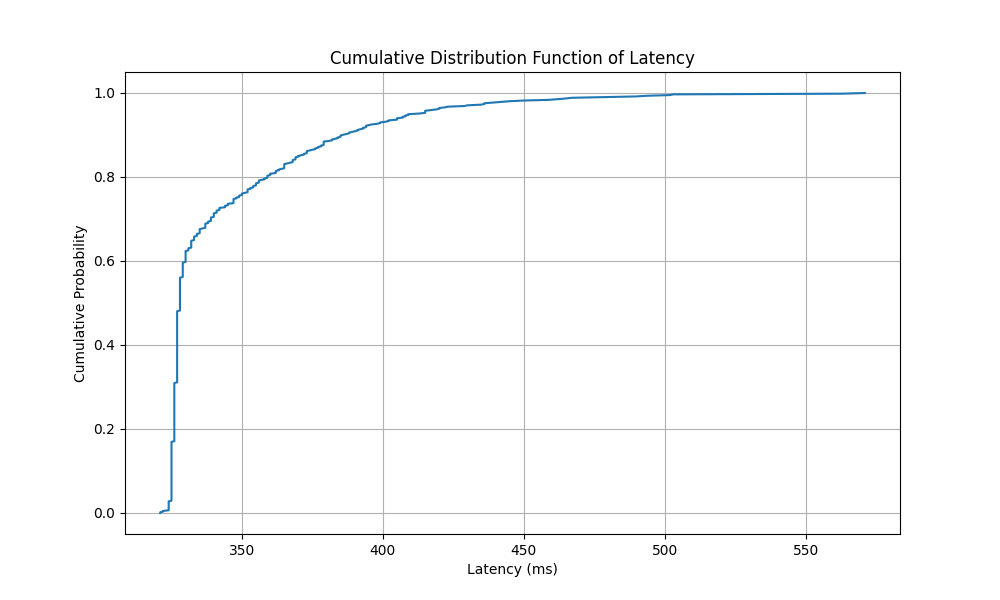




The shape of the PDF is similar to AS1.n1, but with a slightly lower peak, which may suggest a slightly wider distribution of latency values. The shape of the CDF is also similar but rises to high probabilities slightly more slowly, indicating a slightly wider range of latency distribution.

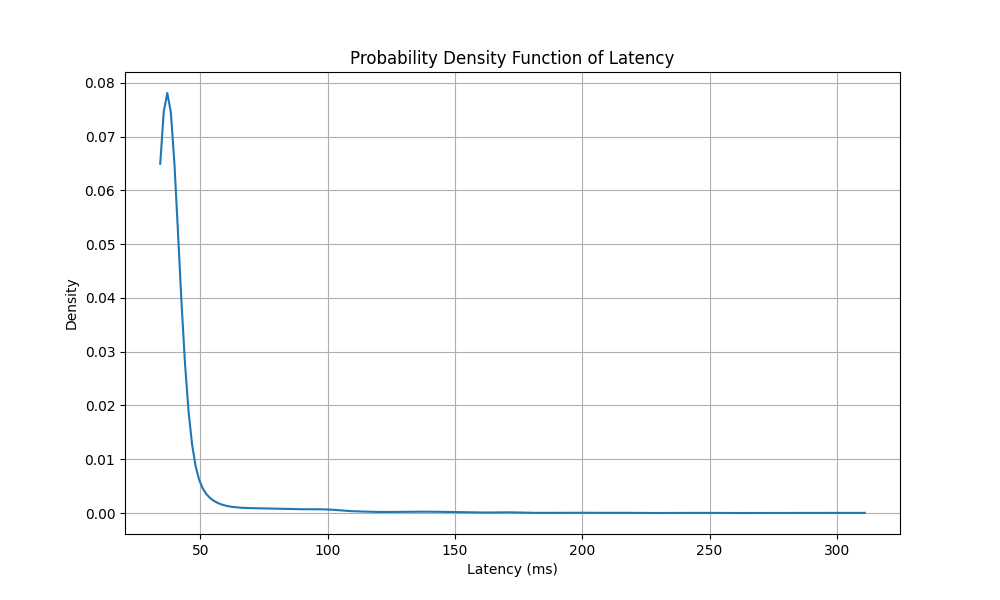
AS1.n3:

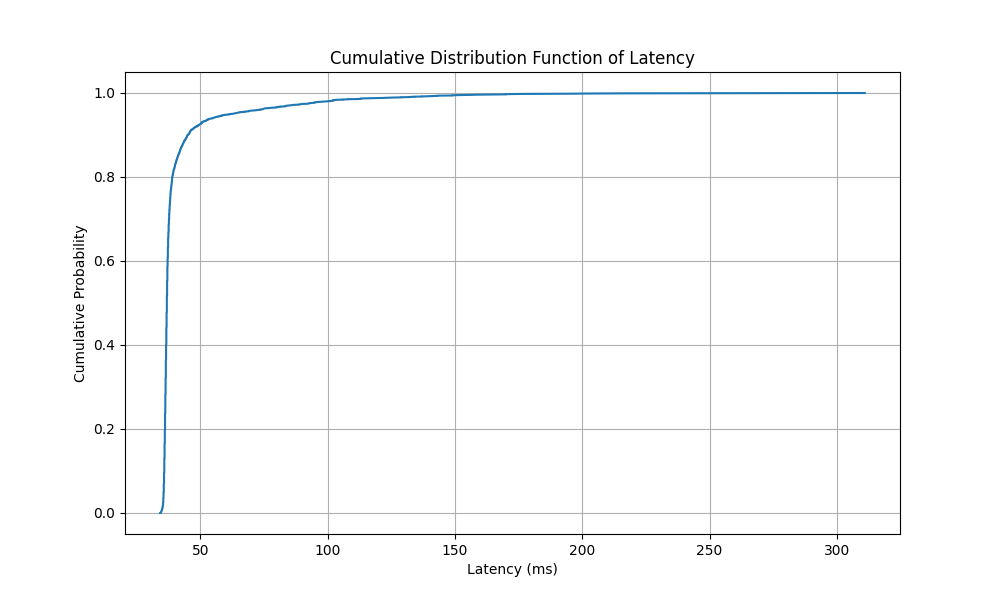




The PDF shows a distribution of latency values similar to the first two datasets but shifted to the right, indicating higher latency values overall. The CDF curve rises more gradually, suggesting that compared to AS1.n1 and AS1.n2, AS1.n3 has more dispersed latency values.

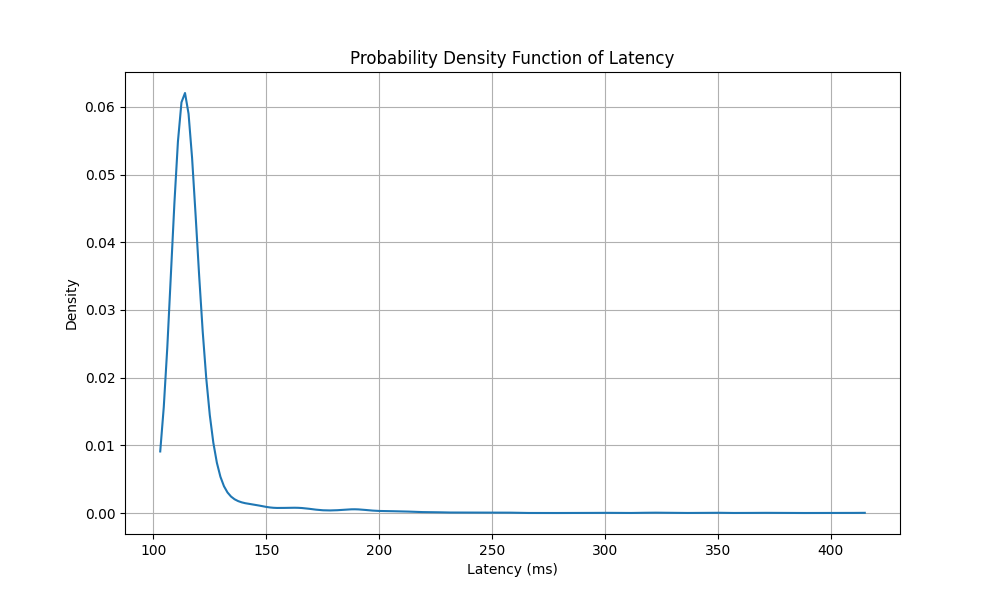
AS1.r1:

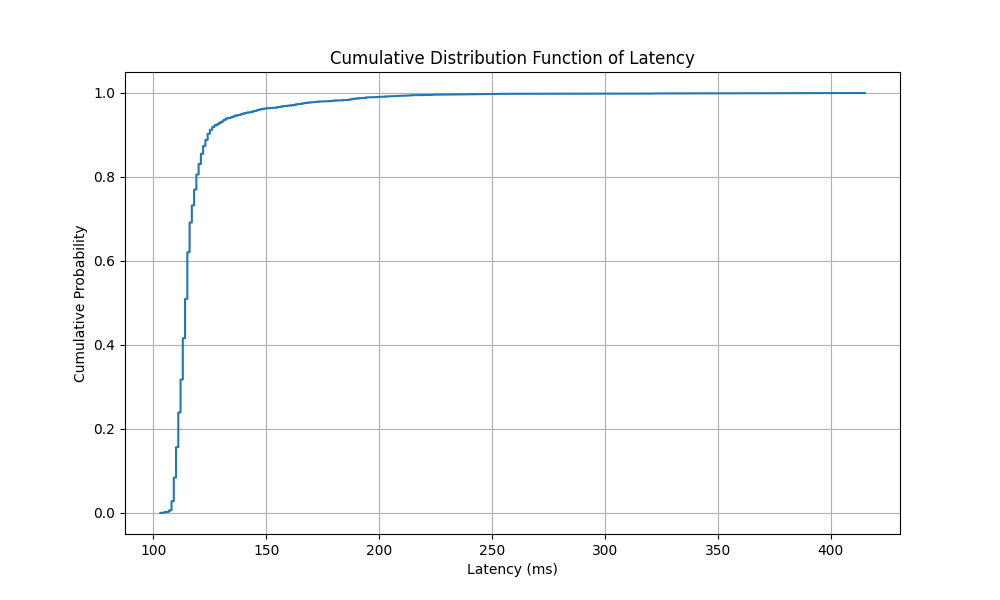




The PDF graph shows that latency values are primarily concentrated in a very low range with a sharp and high peak, indicating consistent and low latency values. The CDF graph indicates that almost all latency values are concentrated below 100 milliseconds, suggesting that the response times for nearly all requests are very short.

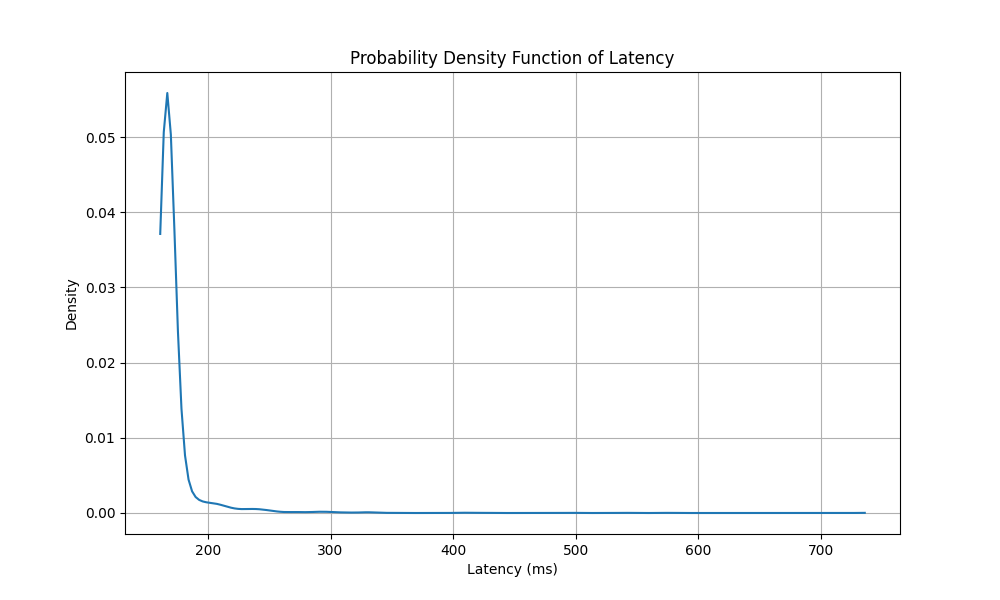
AS1.r2:

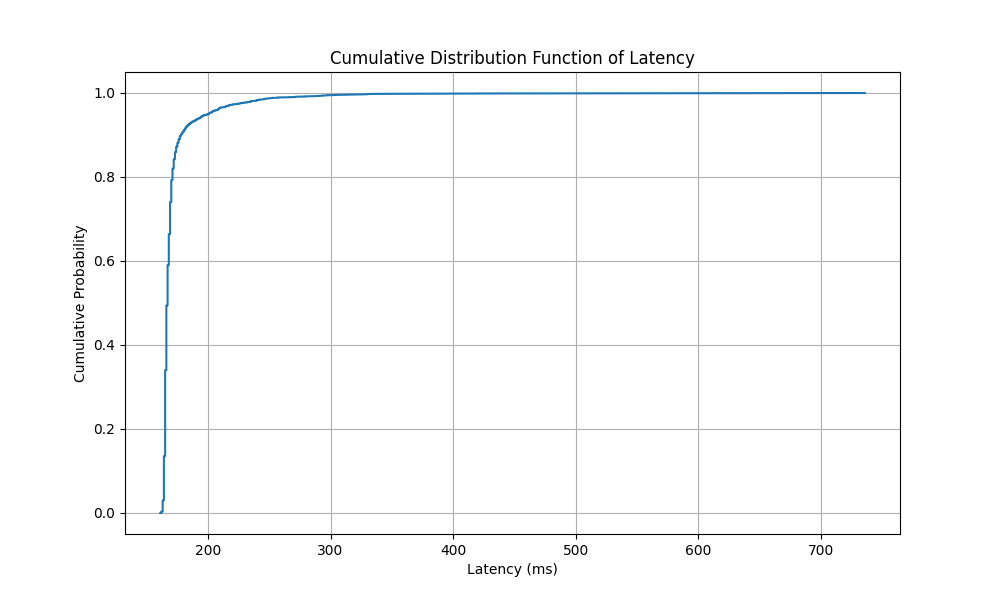




The PDF graph shows a slightly broader distribution of latency values, although there is still a peak, it's not as sharp as AS1.r1. The curve of the CDF graph doesn't rise as steeply as AS1.r1, suggesting that the latency values are distributed over a wider range but still mostly below 100 milliseconds.

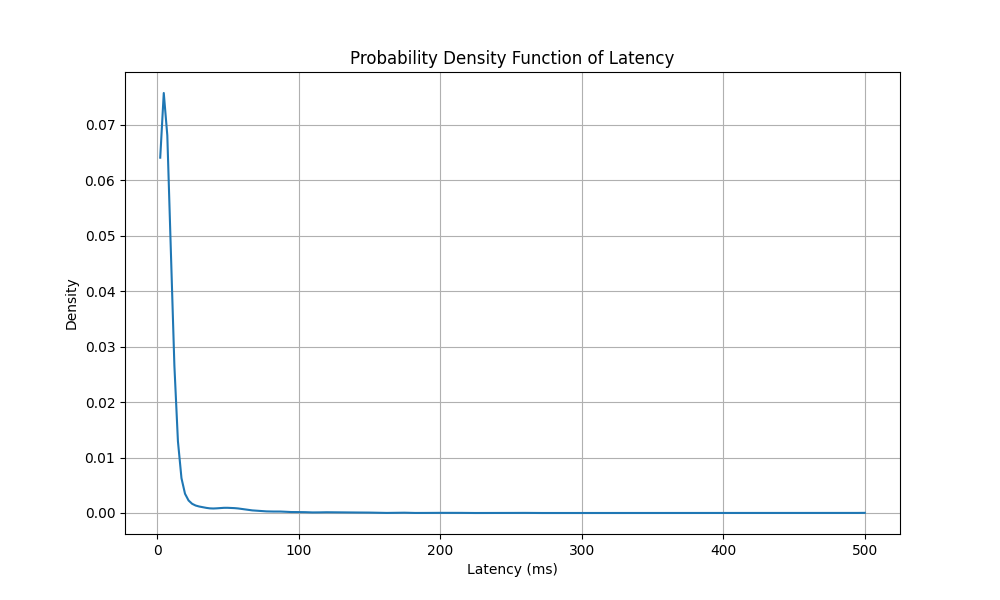
AS1.r3:

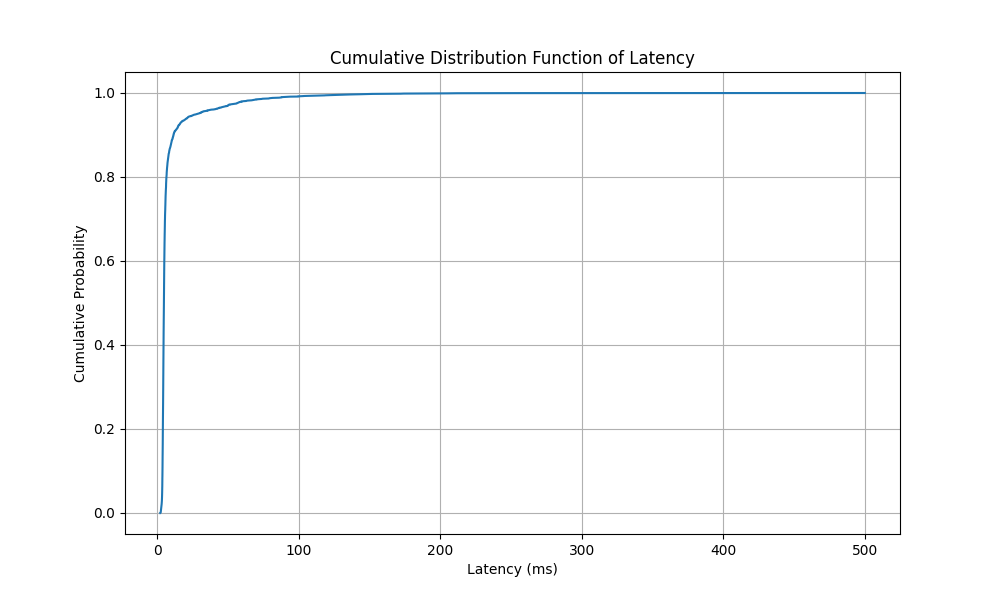




The PDF graph shows a slightly broader distribution of latency values, although there is still a peak, it's not as sharp as AS1.r1. The curve of the CDF graph doesn't rise as steeply as AS1.r1, suggesting that the latency values are distributed over a wider range but still mostly below 100 milliseconds.

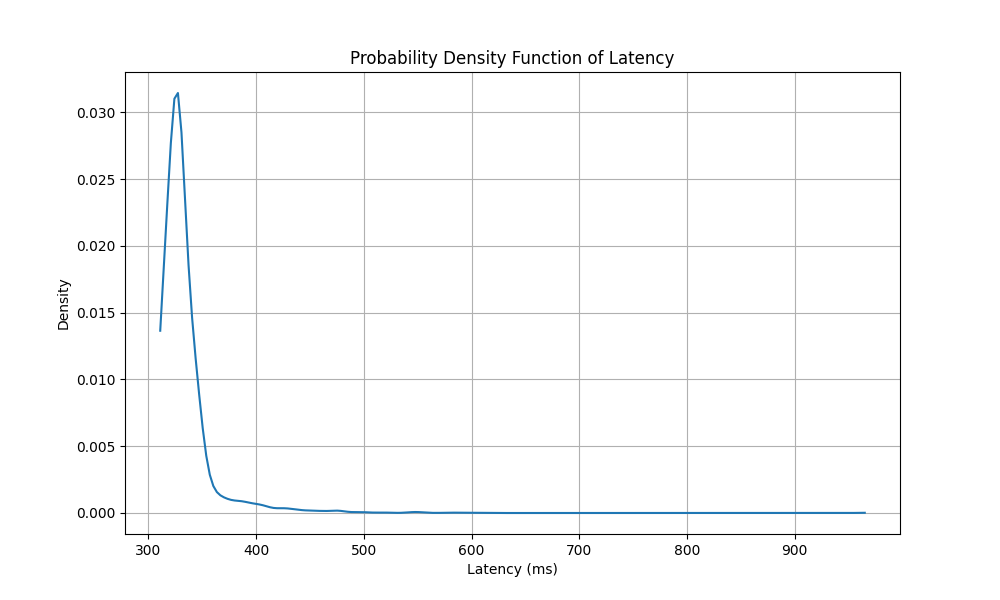
AS1.i1:

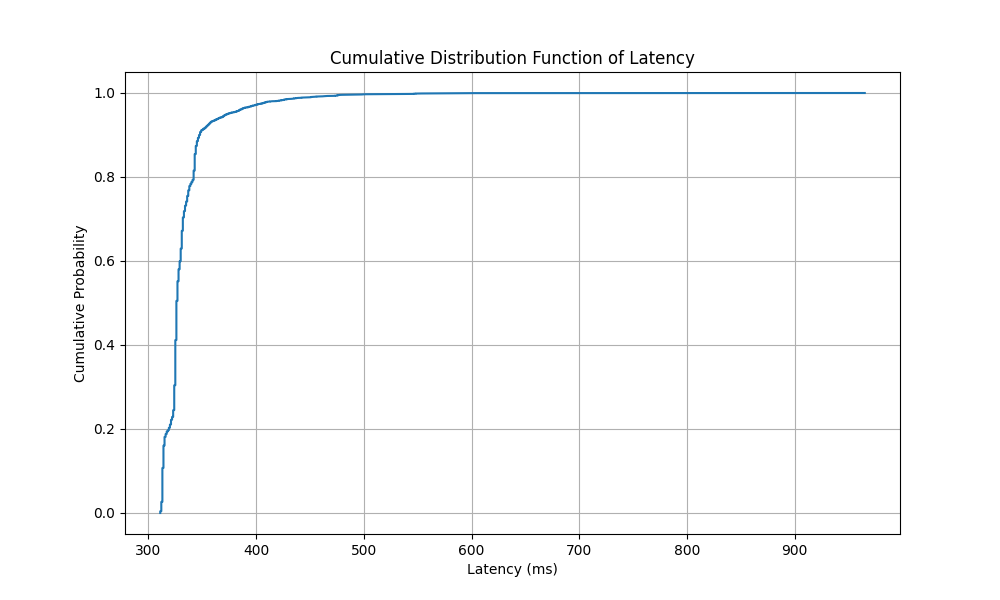




The PDF graph displays an extremely sharp peak, almost at 0 milliseconds, indicating that the majority of latencies are very low, with almost no delay. The CDF graph rapidly rises to values close to 1, further confirming that most latencies are very low, and the cumulative distribution of latencies quickly approaches 100%.

AS1.i2:





The PDF graph shows a wider distribution of latencies compared to AS1.i1, with a lower peak, indicating that latency values are relatively more dispersed, with fewer extremely low latency measurements. The CDF graph doesn't rise as rapidly as AS1.i1, suggesting that the cumulative distribution of latencies increases over a broader range, indicating greater dispersion of latency values.

* A table summarizing delay distributions for all AS1.x data sets

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Data set | First packet delay (ms) | Mean delay (ms) | Proportion of packets outside 1000 ms (%) | Distance between 0.95 and 0.5 (ms) |
| AS1.d1 | 13.77 | 79.11 | 2.44 | 1.50 |
| AS1.d2 | 14.28 | 86.79 | 2.44 | 37.19 |
| AS1.d3 | 323.28 | 383.27 | 2.44 | 3.93 |
| AS1.n1 | 15.60 | 33.02 | 0.00 | 82.07 |
| AS1.n2 | 31.60 | 33.51 | 0.00 | 78.33 |
| AS1.n3 | 339.00 | 343.63 | 0.00 | 82.20 |
| AS1.r1 | 36.00 | 41.15 | 0.00 | 25.03 |
| AS1.r2 | 119.00 | 118.17 | 0.00 | 25.00 |
| AS1.r3 | 165.00 | 171.94 | 0.00 | 33.00 |
| AS1.i1 | 5.15 | 8.65 | 0.00 | 23.52 |
| AS1.i2 | 348.00 | 333.01 | 0.00 | 48.00 |

import re  
import numpy as np  
  
# Initialize a list to store latency differences  
diff = []  
  
# Open the file and read it line by line  
with open('files/ping/blr1.iperf.comnet-student.eu.txt', 'r') as file:  
 for line in file:  
 if 'time=' in line:  
 # Find lines containing 'time=' and split the string  
 parts = line.split()  
 for part in parts:  
 if part.startswith('time='):  
 # Extract latency differences and remove 'ms'  
 latency\_diff = part.split('=')[1].rstrip(' ms')  
 try:  
 # Convert extracted latency differences to floating-point and store them  
 diff.append(float(latency\_diff))  
 except ValueError:  
 # Skip the value if conversion fails  
 continue  
  
# Calculate First Packet Delay  
first\_packet\_delay = diff[0]  
  
# Calculate Mean Delay  
mean\_delay = np.mean(diff)  
  
# Calculate the proportion of delays exceeding 1000 ms  
proportion\_outside = np.sum(np.array(diff) > 1000.0) / len(diff)  
  
# Calculate the distance between quantiles  
quantile\_95 = np.quantile(diff, 0.95)  
quantile\_50 = np.quantile(diff, 0.5)  
distance\_between\_quantiles = quantile\_95 - quantile\_50  
  
# Print results  
print(f"First Packet Delay: {first\_packet\_delay} ms")  
print(f"Mean Delay: {mean\_delay} ms")  
print(f"Proportion of Delays Over 1000ms: {proportion\_outside \* 100:.2f}%")  
print(f"Distance Between 0.95 and 0.50 Quantiles: {distance\_between\_quantiles} ms")

This code reads latency differences from a log file, calculates various statistics on these differences, and prints the results. It specifically extracts and analyzes latency measurements in the form of differences, calculates the first packet delay, mean delay, proportion of delays exceeding 1000 ms, and the distance between the 0.95 and 0.50 quantiles.

import re  
import numpy as np  
  
# Initialize lists with more descriptive names  
query\_time\_lines = []  
curl\_lines = []  
query\_times\_ms = []  
connect\_times\_ms = []  
diff = []  
  
# Open and read the file  
with open('files/dig/southeast-2.dns-au.st.reloaded.txt', 'r') as file:  
 skip\_flag = False  
 for line in file:  
 striped\_line = line.strip()  
 if skip\_flag:  
 skip\_flag = False  
 continue  
 if re.search("Return", striped\_line):  
 skip\_flag = True  
 continue  
 if re.search("Query", striped\_line):  
 query\_time\_lines.append(striped\_line)  
 if re.search("CURL", striped\_line):  
 curl\_lines.append(striped\_line)  
  
# Extract query times and connect times  
for item in query\_time\_lines:  
 item\_split = item.split(" ")  
 query\_time = float(item\_split[3])  
 query\_times\_ms.append(query\_time)  
  
for item in curl\_lines:  
 item\_split = item.split(" ")  
 connect\_time = float(item\_split[3])  
 connect\_times\_ms.append(connect\_time)  
  
# Calculate differences (formerly known as delays) in milliseconds  
for i in range(len(query\_time\_lines)):  
 difference = query\_times\_ms[i] / 1000 + connect\_times\_ms[i]  
 diff.append(difference)  
  
# Calculate First Packet Delay  
first\_packet\_delay = diff[0]  
  
# Calculate Mean Delay  
mean\_delay = np.mean(diff)  
  
# Calculate the proportion of packets exceeding 1000 ms  
proportion\_outside = np.sum(np.array(diff) > 1000.0) / len(diff)  
  
# Calculate the distance between quantiles  
quantile\_95 = np.quantile(diff, 0.95)  
quantile\_50 = np.quantile(diff, 0.5)  
distance\_between\_quantiles = quantile\_95 - quantile\_50  
  
# Print results  
print(f"First Packet Delay: {first\_packet\_delay} ms")  
print(f"Mean Delay: {mean\_delay} ms")  
print(f"Proportion of Packets Over 1000ms: {proportion\_outside \* 100:.2f}%")  
print(f"Distance Between 0.95 and 0.50 Quantiles: {distance\_between\_quantiles} ms")

The purpose of this code is to extract and analyze query and connection time data from a log file, calculate the differences (latencies) between these time data, compute the latency of the first packet, the mean latency, the proportion of latencies exceeding 1000 milliseconds, and the distance between the 0.95th and 0.50th percentiles. Finally, it prints the calculated results.

1. Latency data time series

* Plot time series of each data set AS1.x

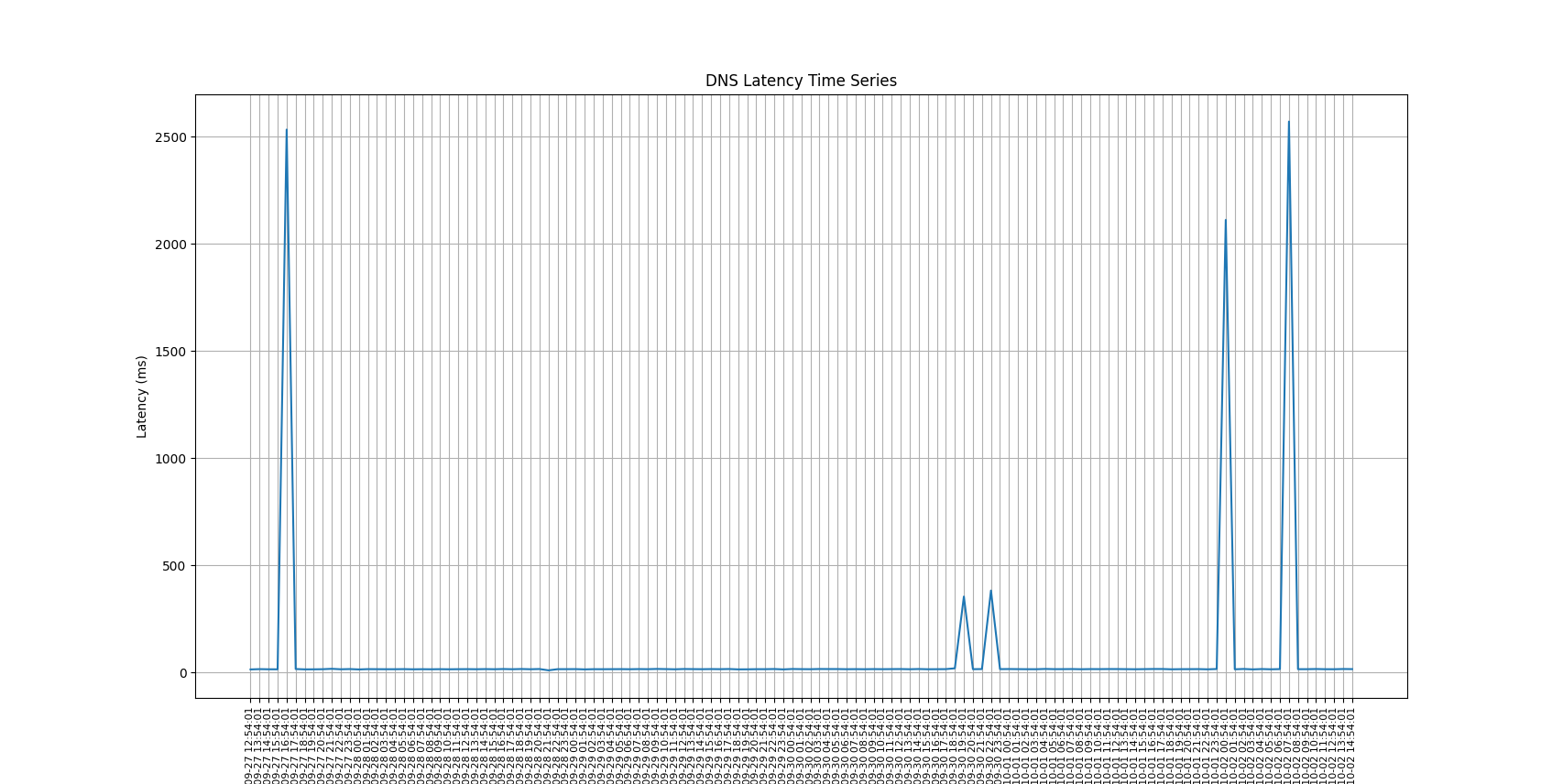
import re  
import numpy as np  
import matplotlib.pyplot as plt  
  
query\_time\_lines = []  
curl\_lines = []  
timestamps = []  
  
with open('files/dig/dns-st.bahnhof.net.txt', 'r') as file:  
 skip\_flag = False  
 for line in file:  
 striped\_line = line.strip()  
 if skip\_flag:  
 skip\_flag = False  
 continue  
 if re.search("Return", striped\_line):  
 skip\_flag = True  
 timestamps.pop(-1)  
 continue  
 if re.search("Query", striped\_line):  
 query\_time\_lines.append(striped\_line)  
 if re.search("CURL", striped\_line):  
 curl\_lines.append(striped\_line)  
 if re.search("2023-", striped\_line):  
 timestamps.append(striped\_line)  
  
query\_time\_ms = []  
connect\_time\_ms = []  
  
for item in query\_time\_lines:  
 item\_split = item.split(" ")  
 query\_time = float(item\_split[3])  
 query\_time\_ms.append(query\_time)  
  
for item in curl\_lines:  
 item\_split = item.split(" ")  
 connect\_time = float(item\_split[3])  
 connect\_time\_ms.append(connect\_time)  
  
latency\_ms = []  
for i in range(0, len(query\_time\_lines)):  
 latency = query\_time\_ms[i] / 1000 + connect\_time\_ms[i]  
 latency\_ms.append(latency)  
  
# Create a time series plot  
plt.figure(figsize=(20, 8)) # Make the plot wider  
plt.plot(timestamps, latency\_ms, linestyle='-')  
plt.xlabel('Date and Time')  
plt.ylabel('Latency (ms)')  
plt.title('DNS Latency Time Series')  
plt.grid(True)  
  
# Format x-axis to display both date and time, every 10th timestamp  
plt.xticks(rotation=90, fontsize=8)  
  
# Show the plot  
plt.show()

This code reads a log file containing DNS query and CURL request information, extracts relevant data, and calculates the latency for each event. It then creates a time series plot to visualize how DNS latency varies over time.

import re  
import matplotlib.pyplot as plt  
from datetime import datetime  
  
# Initialize lists to store timestamps, TTLs, and time values  
timestamps = []  
ttls = []  
time\_values = []  
  
# Open the file for reading  
with open('files/ping/bjl-gm.ark.caida.org.txt', 'r') as file:  
 # Read each line in the file  
 for line in file:  
 # Use regular expressions to extract timestamp, TTL, and time value  
 match = re.search(r'\[(\d+\.\d+)\].\*ttl=(\d+).\*time=(\d+(\.\d+)?)', line)  
 if match:  
 timestamp = float(match.group(1))  
 ttl = int(match.group(2))  
 time = float(match.group(3))  
 timestamps.append(timestamp)  
 ttls.append(ttl)  
 time\_values.append(time)  
  
# Convert timestamps to datetime objects with both date and time  
datetime\_timestamps = [datetime.fromtimestamp(ts).strftime('%Y-%m-%d %H:%M:%S') for ts in timestamps]  
  
# Create a time series plot for latency with transparent background  
plt.figure(figsize=(12, 6))  
plt.plot(datetime\_timestamps, time\_values, linestyle='-')  
plt.xlabel('Date and Time')  
plt.ylabel('Time (ms)')  
plt.title('Latency Time Series')  
plt.grid(axis="y")  
plt.xticks(rotation=45, fontsize=0.1)  
plt.show()

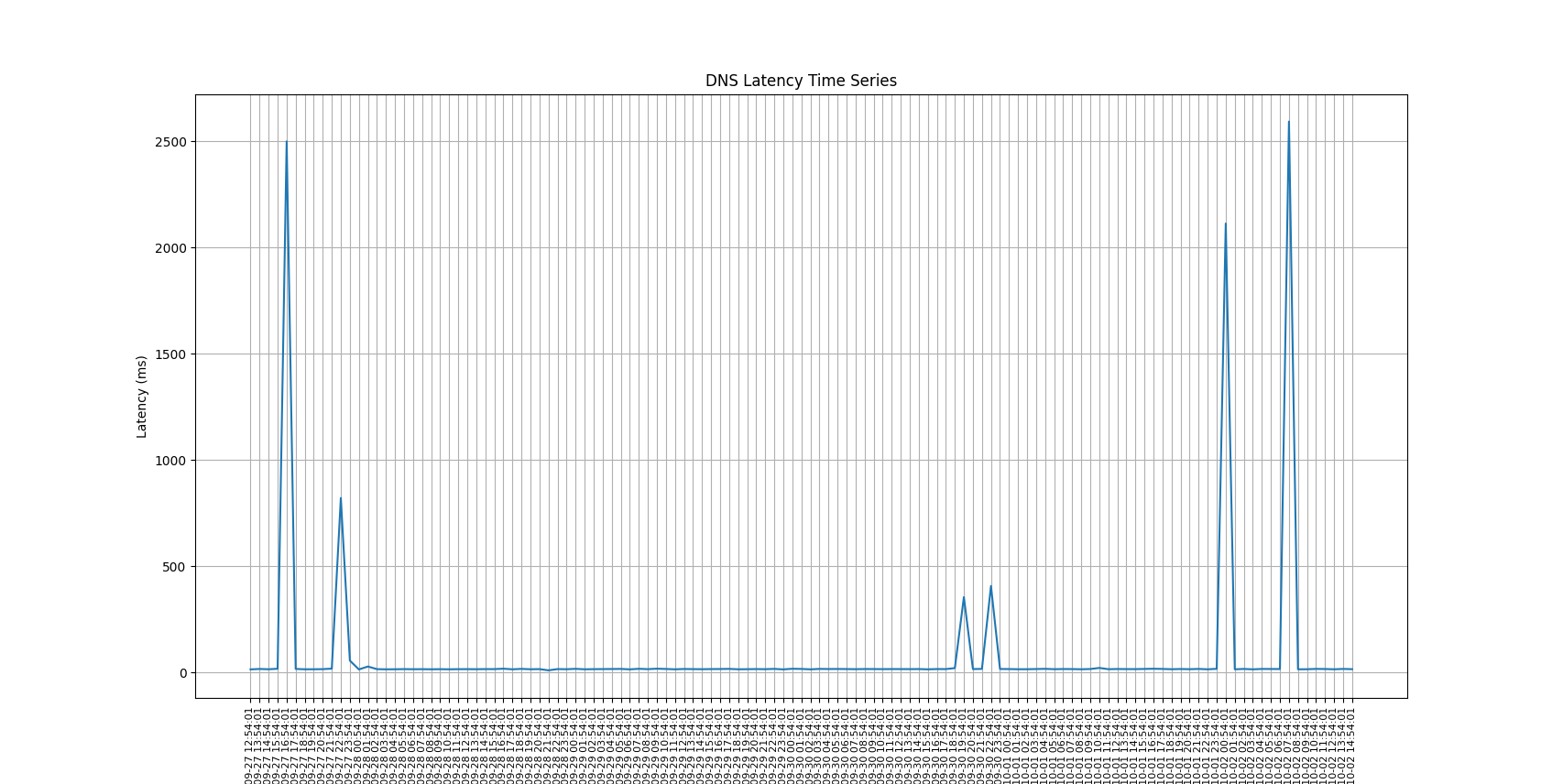
The purpose of this code is to read network latency data from a text file, where each line contains timestamp, TTL (Time to Live), and time value (latency) information. It uses regular expressions to parse this information and converts timestamps into date and time format. Then, it creates a time series plot that displays the trend of latency changing over time.

AS1.d1:



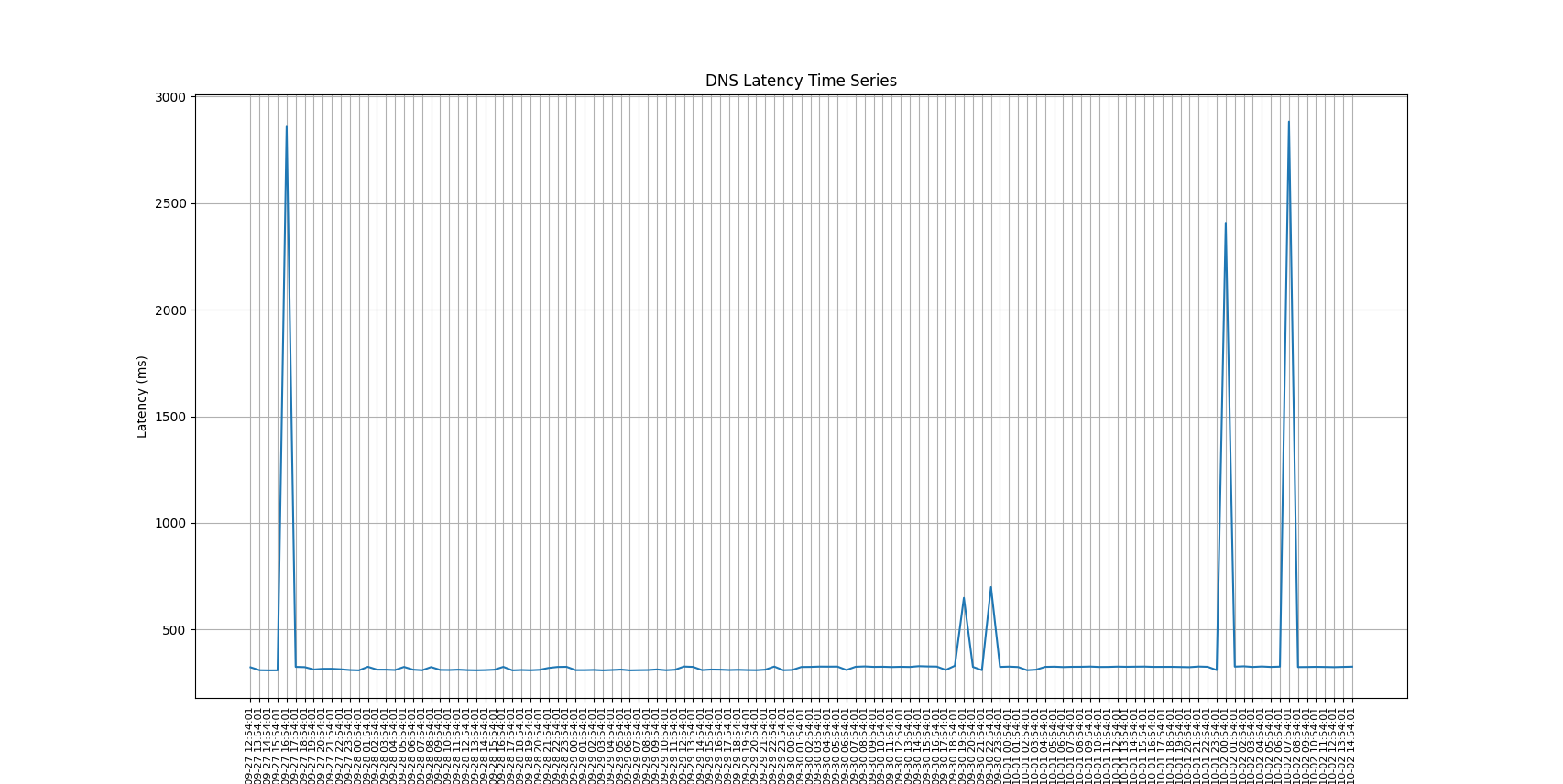
This chart displays the DNS query latency time series for Name Server 1. The graph shows that for the most part, latency is very low, close to 0 milliseconds, but there are several abnormal peaks, indicating moments of sharp increases in latency. These peaks could be due to network congestion, the server reaching its processing capacity limit, or network failures, among other reasons. The highest peaks in latency are close to 2500 milliseconds.

AS1.d2:



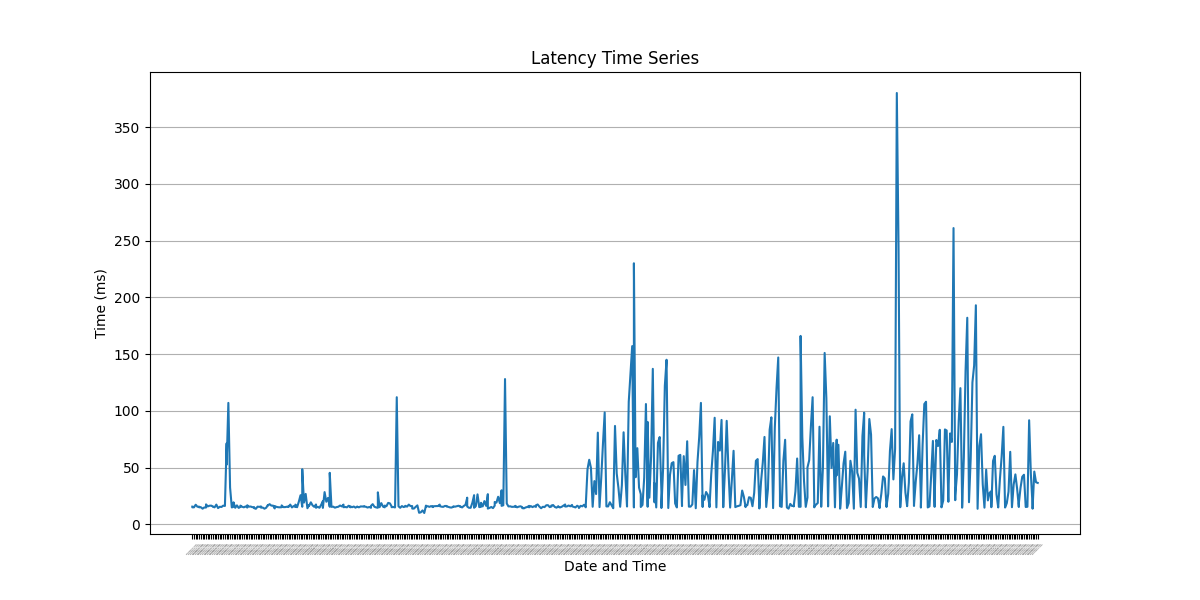
This chart displays the DNS query latency time series for Name Server 1. The graph shows that for the most part, latency is very low, close to 0 milliseconds, but there are several abnormal peaks, indicating moments of sharp increases in latency. These peaks could be due to network congestion, the server reaching its processing capacity limit, or network failures, among other reasons. The highest peaks in latency are close to 2500 milliseconds.

AS1.d3:



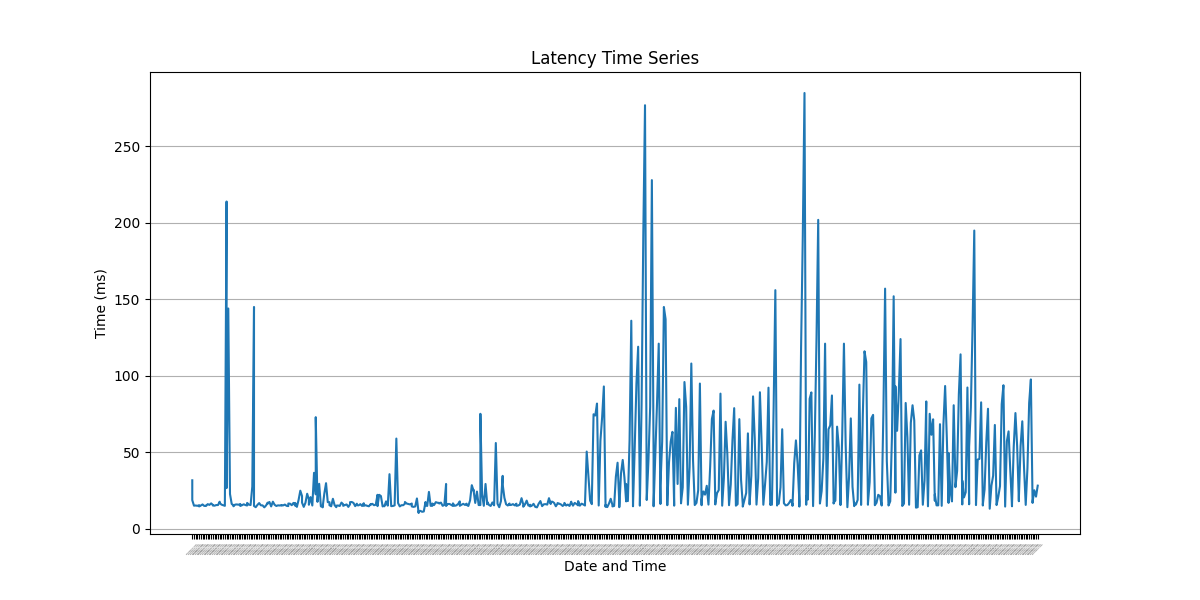
The peaks in the DNS query latency chart for Name Server 3 are more frequent, and the peak heights exceed those of the previous two servers, with the highest even surpassing 2500 milliseconds. Frequent high latency may indicate more serious stability issues with this server, possibly due to the server's location or longer network paths.

AS1.n1:



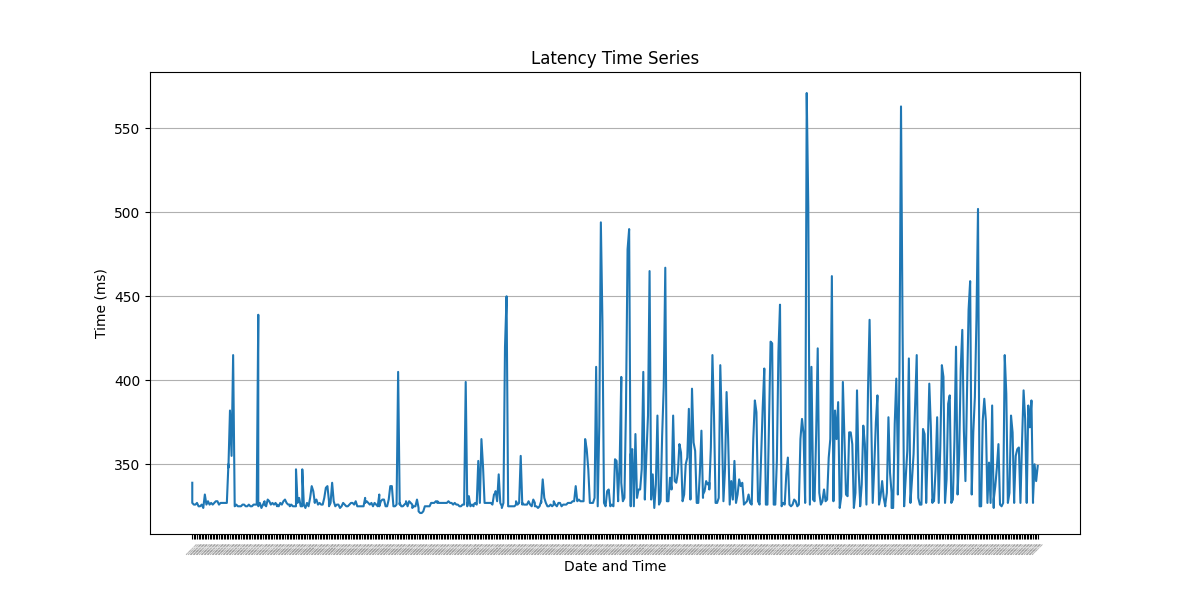
For ICMP queries to Name Server 1, the overall latency is lower, which may imply that the network connection itself is relatively stable. Nevertheless, there are several significant increases in latency, which could be the result of transient network issues or ICMP traffic being treated with a lower priority.

AS1.n2:



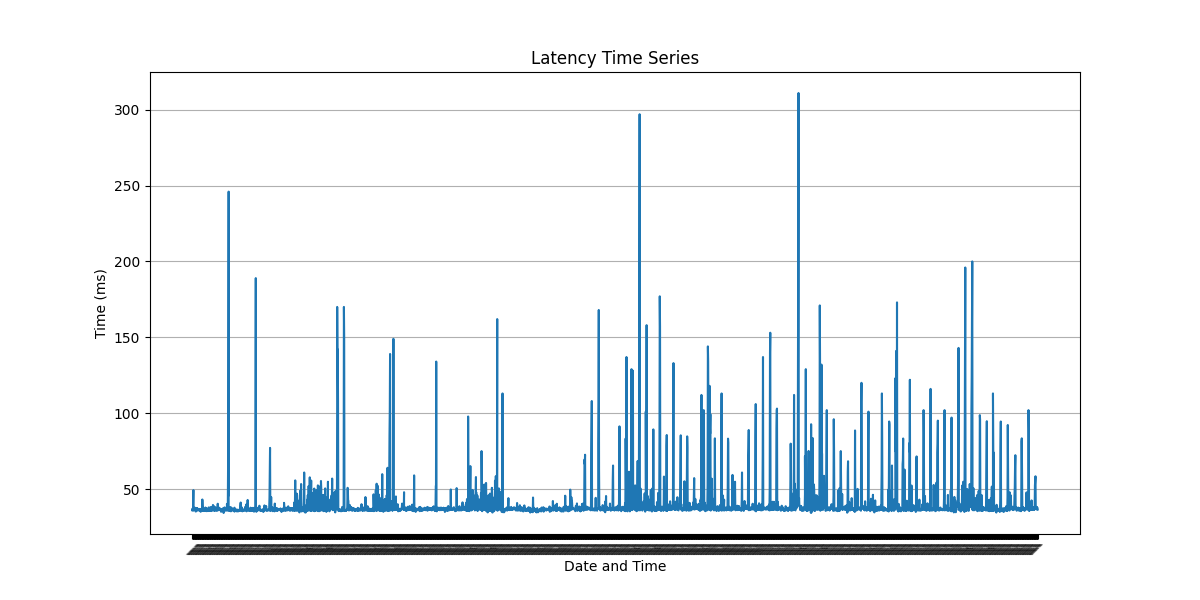
The ICMP latency chart for Name Server 2 shows a lower baseline latency, similar to Server 1. Despite a few sudden increases in latency, the network generally displays good stability. However, any peak in latency could affect applications that require real-time responses.

AS1.n3:



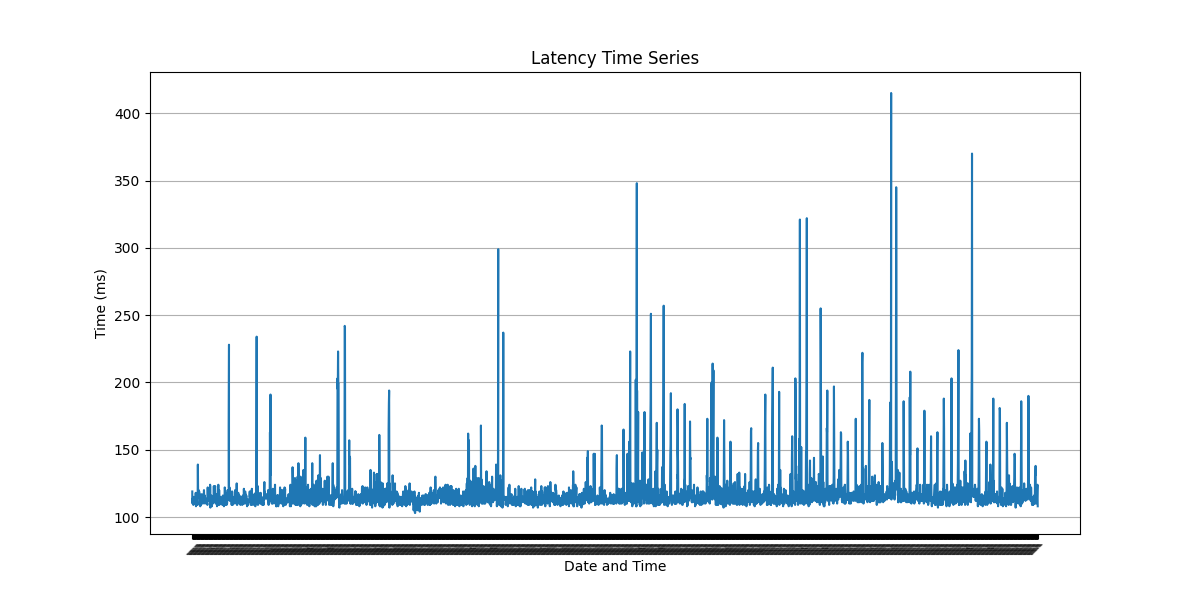
The ICMP latency chart for Name Server 3 shows more frequent high peaks, similar to its DNS query latency graph, suggesting that this server might have more stability issues. The peak latency often exceeds 500 milliseconds, which is unacceptable for any network service that requires a quick response.

AS1.r1:



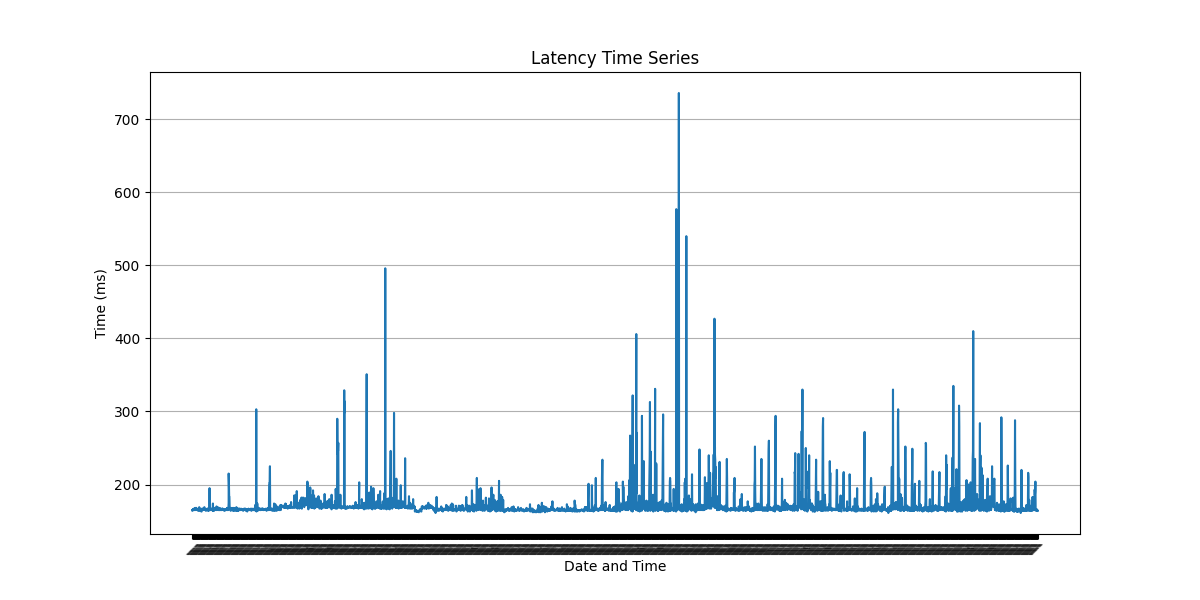
The latency baseline is low, most of the time maintaining around 50 milliseconds, but there are several spikes indicating brief increases in latency, with the highest peaks nearing 300 milliseconds. These peaks could be caused by temporary network fluctuations, instantaneous server load increases, or short-term issues with the network infrastructure.

AS1.r2:



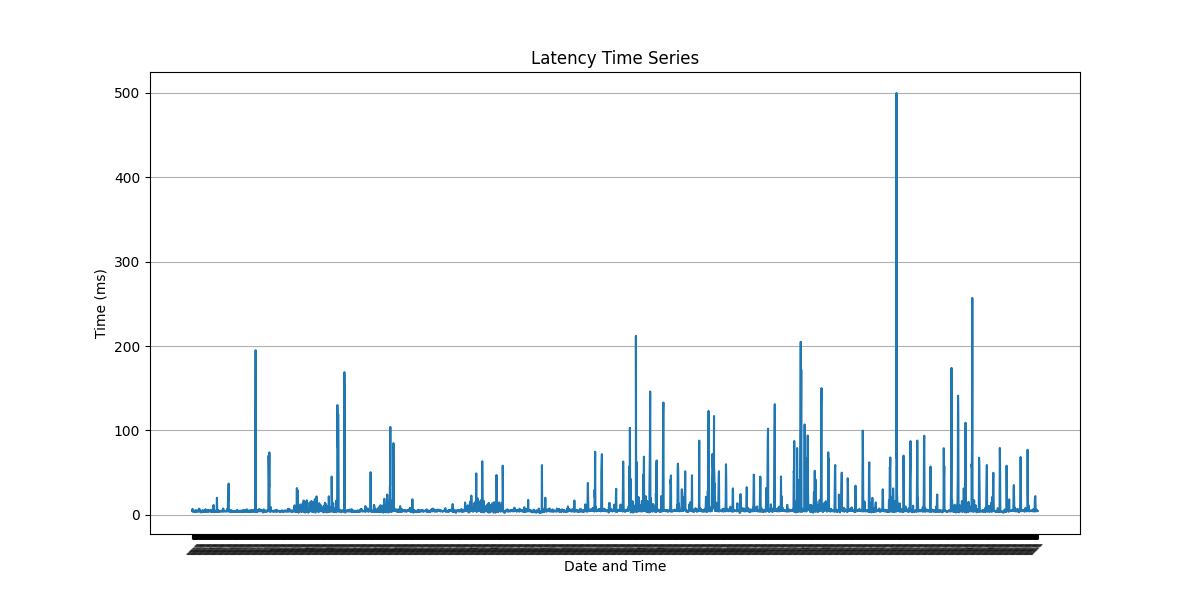
The chart for the second research server shows more frequent peaks, and these peaks are typically higher, reaching above 400 milliseconds. This suggests that the server may face more network stability challenges than the first server. The frequency of high peaks could indicate more common network connection issues.

AS1.r3:



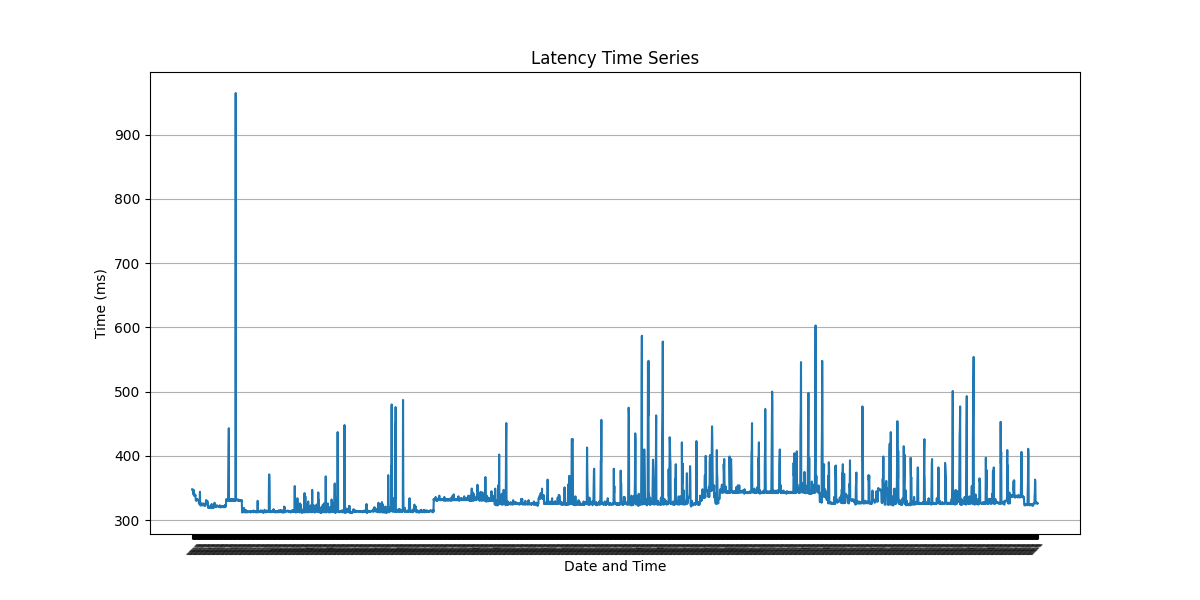
The chart shows very pronounced peaks, not only frequent but also very high, with some peaks even exceeding 700 milliseconds. Such frequent and high latency peaks suggest the server may have more serious network connection issues or limited processing capabilities.

AS1.i1:



The chart for shows that while latency remains at a lower level most of the time, there are occasional larger spikes, nearing 500 milliseconds at their highest. This indicates that the server maintains good performance most of the time but can occasionally be affected by network fluctuations or other external factors.

AS1.i2:



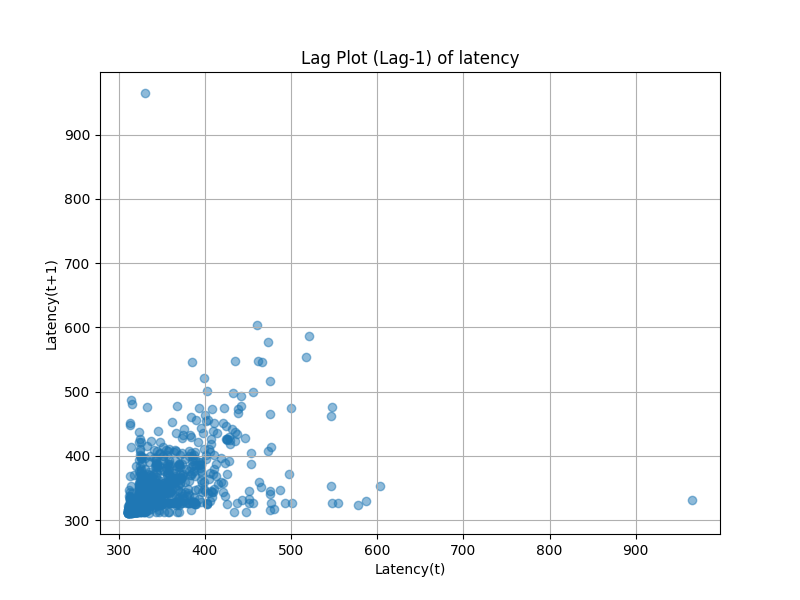
The chart displays even more extreme latency peaks, with these peaks nearing 900 milliseconds. Such high latency peaks are significant and may point to more severe network problems or stability issues with the server.

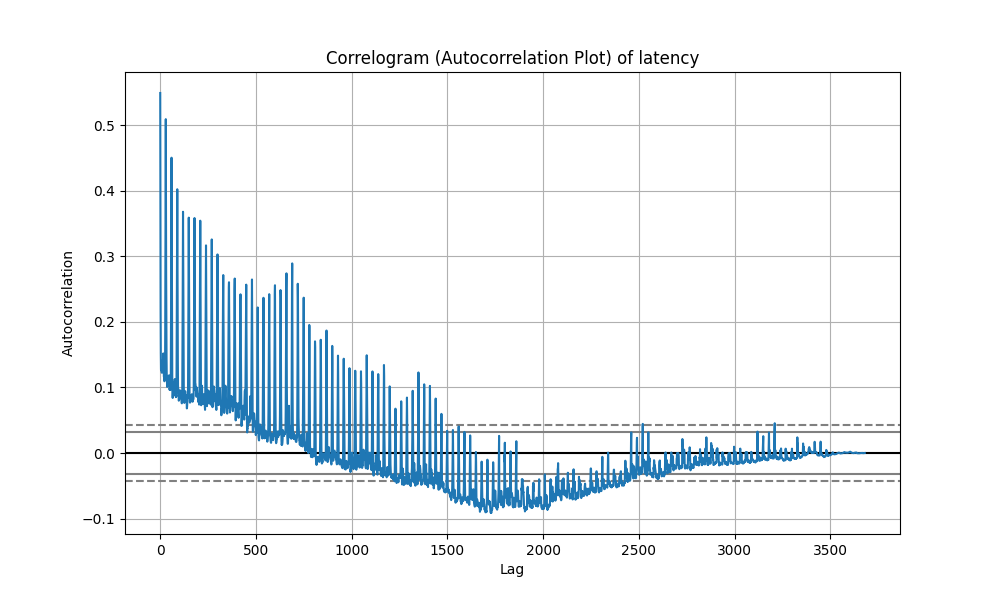
* Autocorrelation plot

import matplotlib.pyplot as plt  
import numpy as np  
from scipy.stats import gaussian\_kde  
import pandas as pd  
  
  
def extract\_latencies(filename):  
 # Initialize a list to store extracted latency values  
 latency\_values = []  
  
 # Open the file and read each line  
 with open(filename, 'r') as file:  
 for line in file:  
 if 'time=' in line:  
 # Find lines containing 'time=' and split the string  
 parts = line.split()  
 for part in parts:  
 if part.startswith('time='):  
 # Extract the latency value and remove 'ms'  
 latency = part.split('=')[1].rstrip(' ms')  
 try:  
 # Convert the extracted latency value to a float and store it  
 latency\_values.append(float(latency))  
 except ValueError:  
 # Skip the value if conversion fails  
 continue  
  
 return latency\_values  
  
  
def create\_lag\_plot(latency\_values):  
 # Create a lag plot  
 plt.figure(figsize=(8, 6))  
 plt.scatter(latency\_values[:-1], latency\_values[1:], alpha=0.5)  
 plt.title('Lag Plot (Lag-1) of latency')  
 plt.xlabel('Latency(t)')  
 plt.ylabel('Latency(t+1)')  
 plt.grid(True)  
 plt.show()  
  
  
def create\_autocorrelation\_plot(latency\_values):  
 # Create an autocorrelation plot  
 plt.figure(figsize=(10, 6))  
 pd.plotting.autocorrelation\_plot(latency\_values, ax=plt.gca())  
 plt.title('Correlogram (Autocorrelation Plot) of latency')  
 plt.xlabel('Lag')  
 plt.grid(True)  
 plt.show()  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 filename = 'files/ping/ok1.iperf.comnet-student.eu.txt'  
  
 # Extract latency values from the file  
 latency\_values = extract\_latencies(filename)  
  
 # Create and display the lag plot  
 create\_lag\_plot(latency\_values)  
  
 # Create and display the autocorrelation plot  
 create\_autocorrelation\_plot(latency\_values)

This code's functionality is to extract network latency data from a specified file and then use Matplotlib and Pandas to create two different plots for analyzing this latency data. First, it extracts the latency values from the file using the extract\_latencies function and stores them in a list. Then, it creates a lag plot using the create\_lag\_plot function to visualize the relationship between latency data points. Next, it generates an autocorrelation plot using the create\_autocorrelation\_plot function to analyze the autocorrelation of the latency data.

AS1.i2:

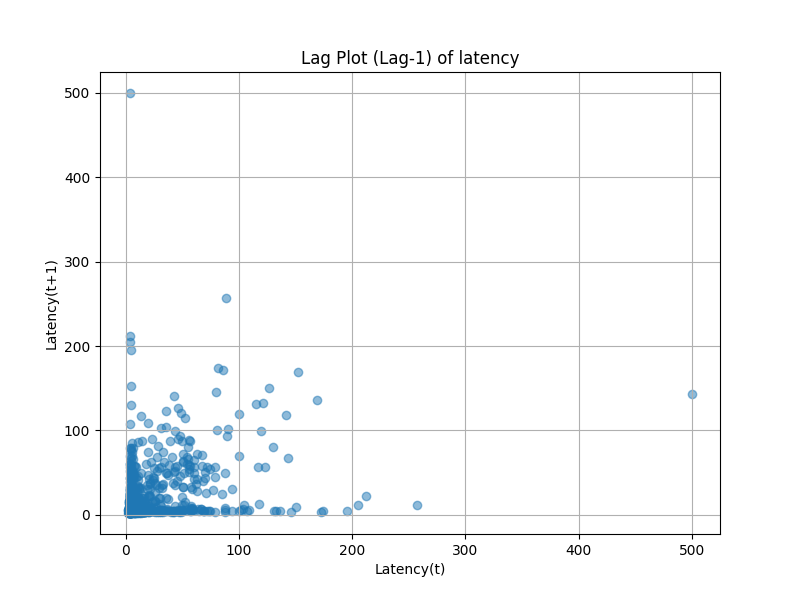


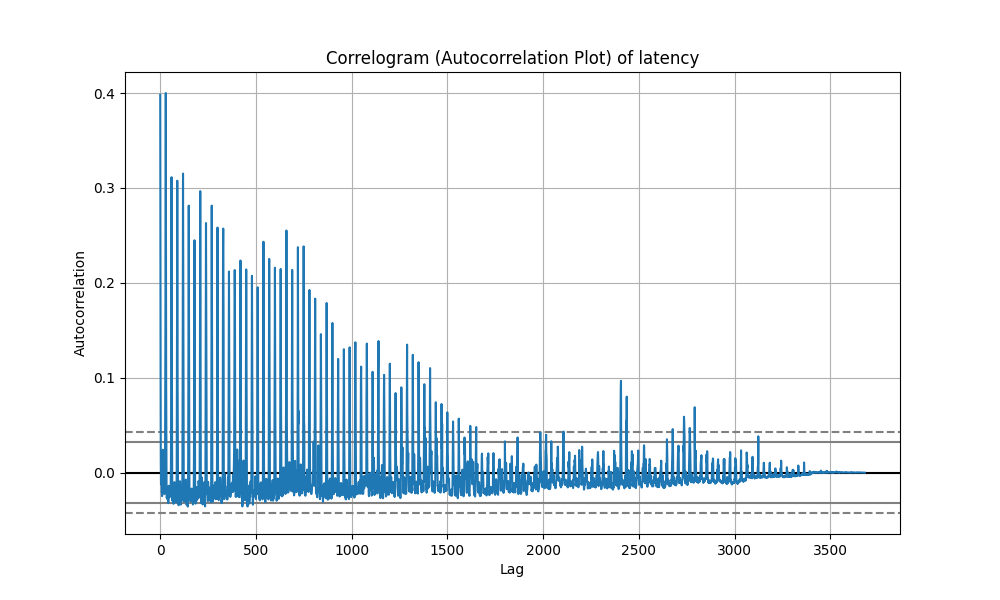


The lag plot shows the relationship between each observation and its previous one. The data points are concentrated in the lower left corner, indicating that at lower lag values, many observations have latency close to that of their previous observation. The absence of a clear structure or pattern suggests a random distribution of data points, which may imply that the values in the dataset are independent, with no obvious time series trend or periodicity.

The autocorrelation plot indicates the degree of autocorrelation of the data points, that is, the correlation of data points with their previous points in the time series. At initial lags (low lag values), the autocorrelation is higher, suggesting that observations in the time series are related to their immediate predecessors. As the lag increases, the autocorrelation gradually decreases, indicating that the association between observations diminishes over longer time intervals. The autocorrelation values drop below the confidence interval at certain points, suggesting that the correlations at these specific lags may not be coincidental.

AS1.i1:

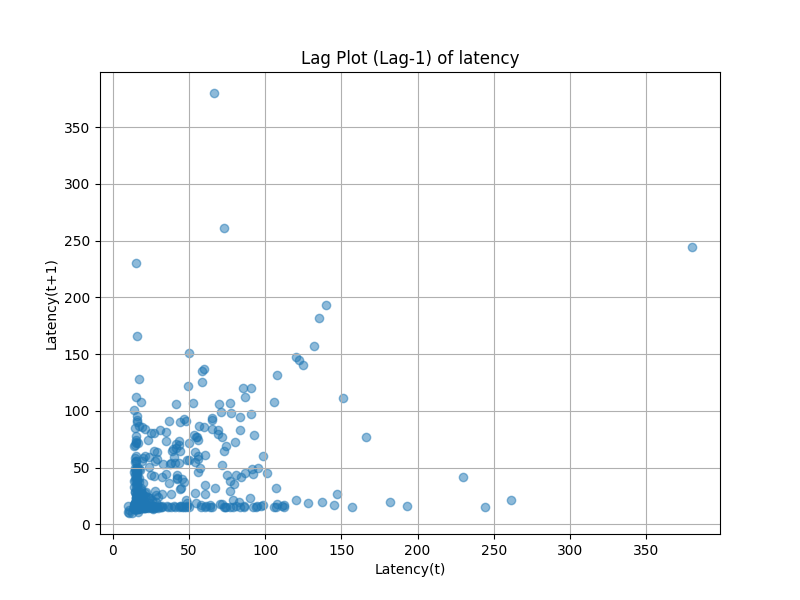


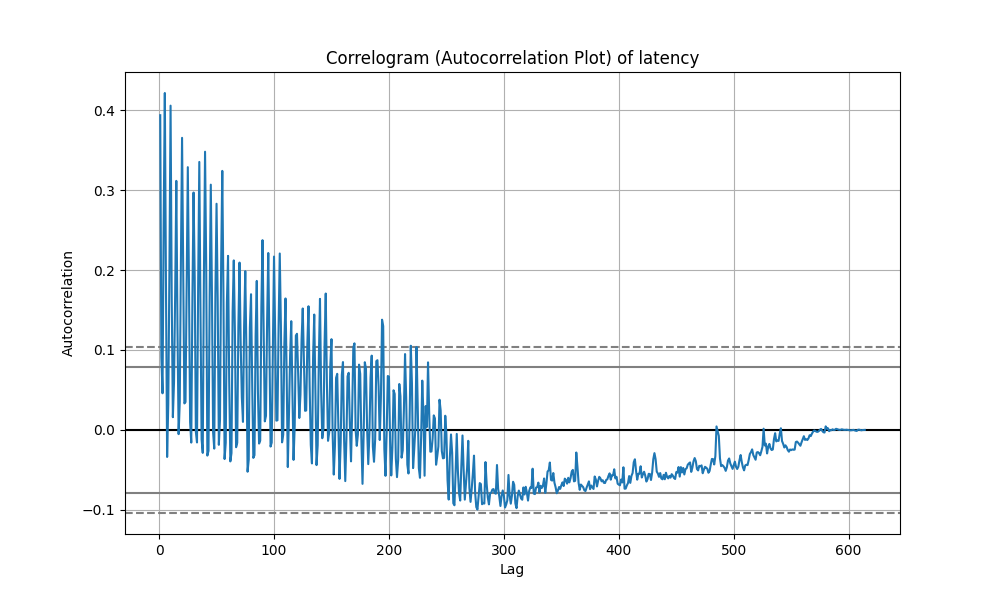


In the lag plot, data points are primarily concentrated near the origin, indicating that for most observations, the lag compared to the previous observation is quite small. As the lag values increase, the distribution of observations also broadens, but there is still no apparent pattern or trend, which may suggest that the values in the dataset are independent, without obvious trends or periodicity in the time series.

The correlogram displays the autocorrelation of the data, which decreases rapidly with increasing lag. Initially, the autocorrelation is quite high, meaning that observations in the series are highly correlated with their preceding values. The autocorrelation drops off quickly as lag increases, tending towards zero after nearly 50 lag units, indicating that the values in the time series become rapidly uncorrelated as the lag increases. The peaks in the autocorrelation plot decrease swiftly and approach near zero, suggesting any periodic features in the data are very short-lived.

AS1.n1:





The lag plot shows the relationship at lag-1 between each observation and its preceding value. Data points are densely packed near the origin, indicating that many observations do not change much compared to their previous value. The absence of a clear pattern or trend in the plot typically suggests that the observations in the series may be randomly distributed, without showing strong autocorrelation or time dependency.

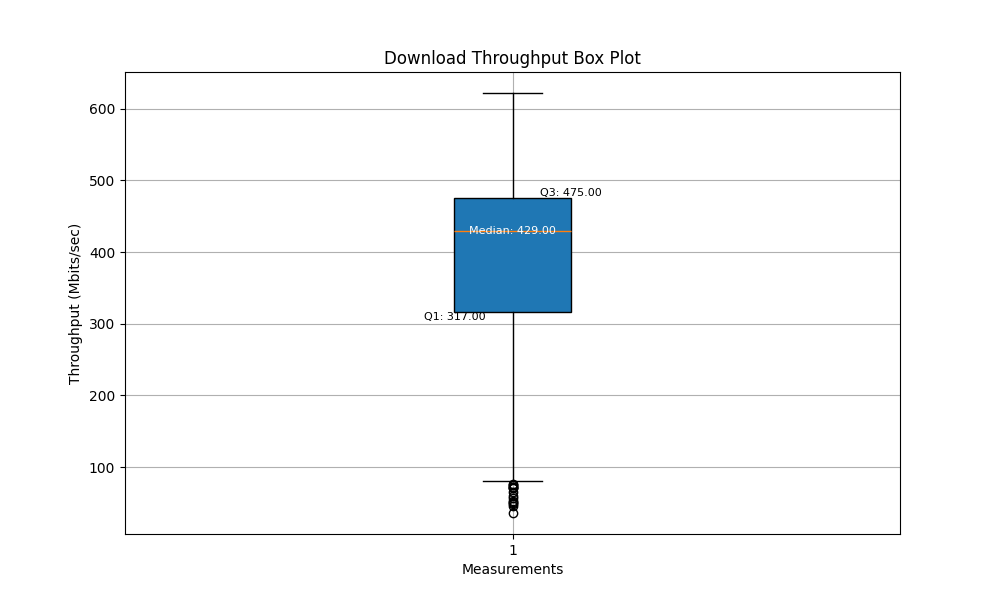
The correlogram displays the change in autocorrelation of the data with the lag. Autocorrelation is relatively high at initial lags, but as the lag increases, this correlation quickly diminishes. Autocorrelation stabilizes around 100 lag units and remains close to zero or slightly fluctuates in subsequent lags. Most of the autocorrelation values are within the confidence interval without any significant breaches, indicating that these correlations might be due to randomness rather than actual associations in the data.

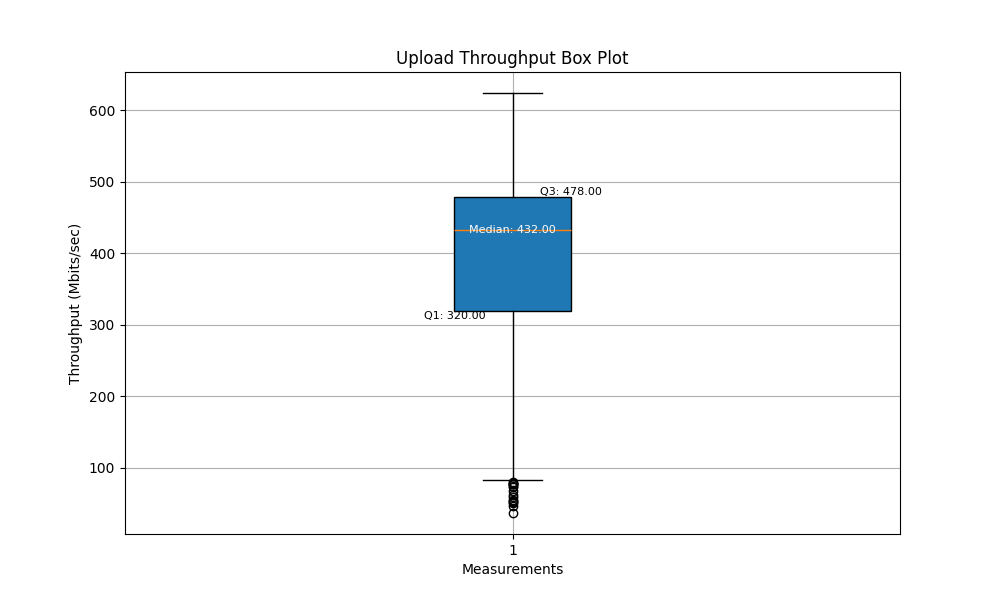
1. Throughput

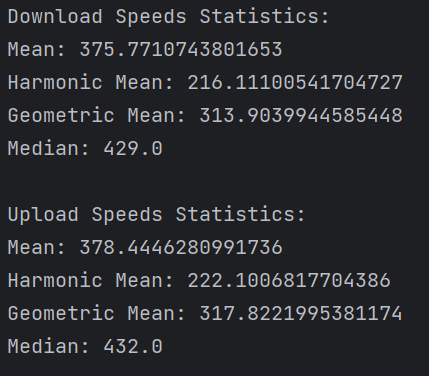
import re  
import matplotlib.pyplot as plt  
import numpy as np  
  
# Rename variables for clarity  
download\_speeds = []  
upload\_speeds = []  
  
# Process the file and collect download and upload speeds  
with open('files/iperf/ok1.iperf.comnet-student.eu.txt', 'r') as file:  
 for line in file:  
 stripped\_line = line.strip()  
 if re.search("receiver", stripped\_line):  
 items = stripped\_line.split()  
 download\_speeds.append(float(items[6]))  
 print(stripped\_line)  
 elif re.search("sender", stripped\_line):  
 items = stripped\_line.split()  
 upload\_speeds.append(float(items[6]))  
 print(stripped\_line)  
  
# Display download and upload speeds  
print("Download Speeds:", download\_speeds)  
print("Upload Speeds:", upload\_speeds)  
  
# Create box plots for download speeds  
fig, ax = plt.subplots(figsize=(10, 6))  
boxplot\_dict = ax.boxplot(download\_speeds, patch\_artist=True)  
  
# Annotate median, quartiles, and labels  
medians = [median.get\_ydata()[0] for median in boxplot\_dict['medians']]  
for tick, median in zip(ax.get\_xticks(), medians):  
 ax.text(tick, median, f'Median: {median:.2f}', ha='center', va='center', fontdict={'fontsize': 8, 'color': 'white'})  
  
boxes = [box.get\_path().vertices for box in boxplot\_dict['boxes']]  
for box in boxes:  
 box\_bottom = box[0, 1]  
 box\_top = box[2, 1]  
 ax.text(box[0, 0], box\_bottom, f'Q1: {box\_bottom:.2f}', ha='center', va='top', fontdict={'fontsize': 8})  
 ax.text(box[2, 0], box\_top, f'Q3: {box\_top:.2f}', ha='center', va='bottom', fontdict={'fontsize': 8})  
  
ax.set\_title('Download Throughput Box Plot')  
ax.set\_ylabel('Throughput (Mbits/sec)')  
ax.set\_xlabel('Measurements')  
plt.grid(True)  
plt.show()  
  
# Create box plots for upload speeds  
fig, ax = plt.subplots(figsize=(10, 6))  
boxplot\_dict = ax.boxplot(upload\_speeds, patch\_artist=True)  
  
medians = [median.get\_ydata()[0] for median in boxplot\_dict['medians']]  
for tick, median in zip(ax.get\_xticks(), medians):  
 ax.text(tick, median, f'Median: {median:.2f}', ha='center', va='center', fontdict={'fontsize': 8, 'color': 'white'})  
  
boxes = [box.get\_path().vertices for box in boxplot\_dict['boxes']]  
for box in boxes:  
 box\_bottom = box[0, 1]  
 box\_top = box[2, 1]  
 ax.text(box[0, 0], box\_bottom, f'Q1: {box\_bottom:.2f}', ha='center', va='top', fontdict={'fontsize': 8})  
 ax.text(box[2, 0], box\_top, f'Q3: {box\_top:.2f}', ha='center', va='bottom', fontdict={'fontsize': 8})  
  
ax.set\_title('Upload Throughput Box Plot')  
ax.set\_ylabel('Throughput (Mbits/sec)')  
ax.set\_xlabel('Measurements')  
plt.grid(True)  
plt.show()  
  
# Calculate statistics for download speeds  
mean\_download = np.mean(download\_speeds)  
harmonic\_mean\_download = 1 / np.mean(1 / np.array(download\_speeds))  
geometric\_mean\_download = np.exp(np.mean(np.log(download\_speeds)))  
median\_download = np.median(download\_speeds)  
  
# Calculate statistics for upload speeds  
mean\_upload = np.mean(upload\_speeds)  
harmonic\_mean\_upload = 1 / np.mean(1 / np.array(upload\_speeds))  
geometric\_mean\_upload = np.exp(np.mean(np.log(upload\_speeds)))  
median\_upload = np.median(upload\_speeds)  
  
# Display the statistics  
print("Download Speeds Statistics:")  
print("Mean:", mean\_download)  
print("Harmonic Mean:", harmonic\_mean\_download)  
print("Geometric Mean:", geometric\_mean\_download)  
print("Median:", median\_download)  
  
print("\nUpload Speeds Statistics:")  
print("Mean:", mean\_upload)  
print("Harmonic Mean:", harmonic\_mean\_upload)  
print("Geometric Mean:", geometric\_mean\_upload)  
print("Median:", median\_upload)

This code extracts network speed data from a text file, stores download and upload speeds in separate lists, then creates box plots to visualize the distribution of these speed data. Next, it calculates statistics for each speed list, including mean, harmonic mean, geometric mean, and median, and prints these statistical results for analysis and comparison of network performance.

AS1.i1:





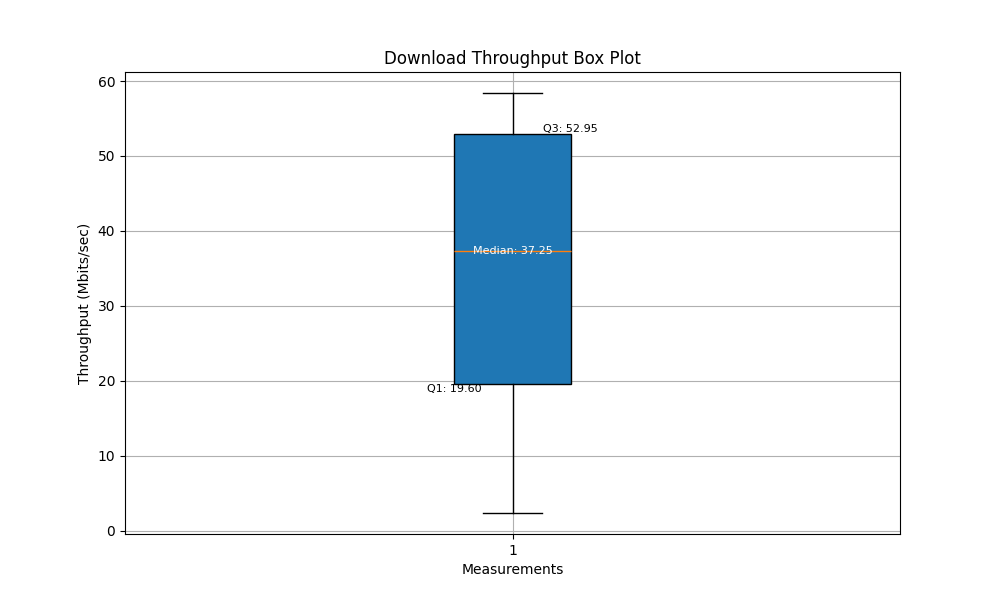


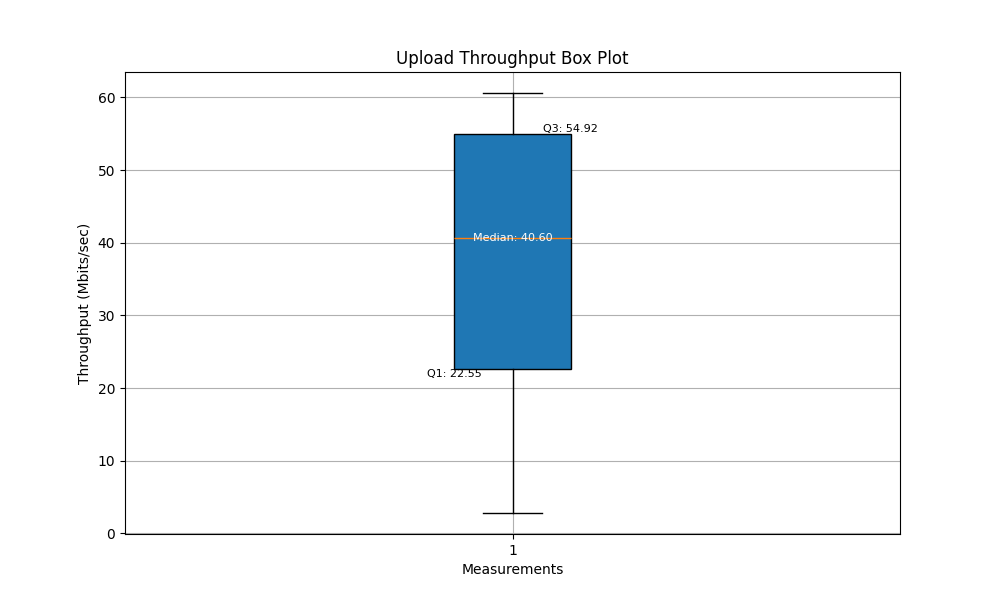
The download throughput box plot indicates that the median download speed is 429 Mbps. The first quartile (Q1) is 317 Mbps, showing that 25% of the download speed measurements are below this value. The third quartile (Q3) is 475 Mbps, meaning that 75% of download speeds are below this value. The range between Q1 and Q3, known as the interquartile range (IQR), represents the middle 50% of the data distribution. There are some outliers below the lower edge, indicating that some download speeds are significantly lower than the bulk of the data.

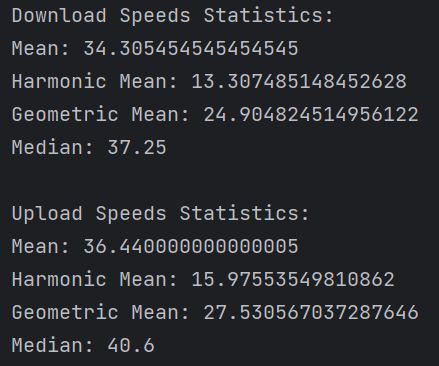
The upload throughput box plot shows a median upload speed of 432 Mbps. The first quartile (Q1) is 320 Mbps, so 25% of upload speed measurements are below this value. The third quartile (Q3) is 478 Mbps, indicating that 75% of upload speeds are below this value. The IQR for upload speeds is slightly larger than that for download speeds, suggesting a slightly greater variability in the middle 50% of the upload data. Similar to download speeds, there are outliers below the lower edge, pointing out instances of significantly lower than normal upload speeds.

Overall, both download and upload speeds exhibit a right-skewed distribution, which is evident from the lower harmonic and geometric means compared to the arithmetic means. The median values are relatively high, indicating that more than half of the measurements are at the higher end of the throughput range. The presence of outliers, especially on the lower end, impacts the mean values, pulling them down, which is why the harmonic and geometric means are notably lower. The consistency of the median values in the box plots and the statistical data indicates an accurate representation of the visual data.

AS1.i2:







Similarly, for AS2.i2, we can draw the following conclusions. The harmonic and geometric means being lower than the arithmetic means indicate that both download and upload speeds display a right-skewed distribution. The median values are higher than the arithmetic means, which is typical for a skewed distribution and confirms that the majority of speeds are above the average. The absence of outliers in both box plots suggests that there are no extreme values far from the bulk of the measurements. The median values in the statistical data closely align with those presented in the box plots, confirming the accuracy of the visual representation.

1. Throughput time series

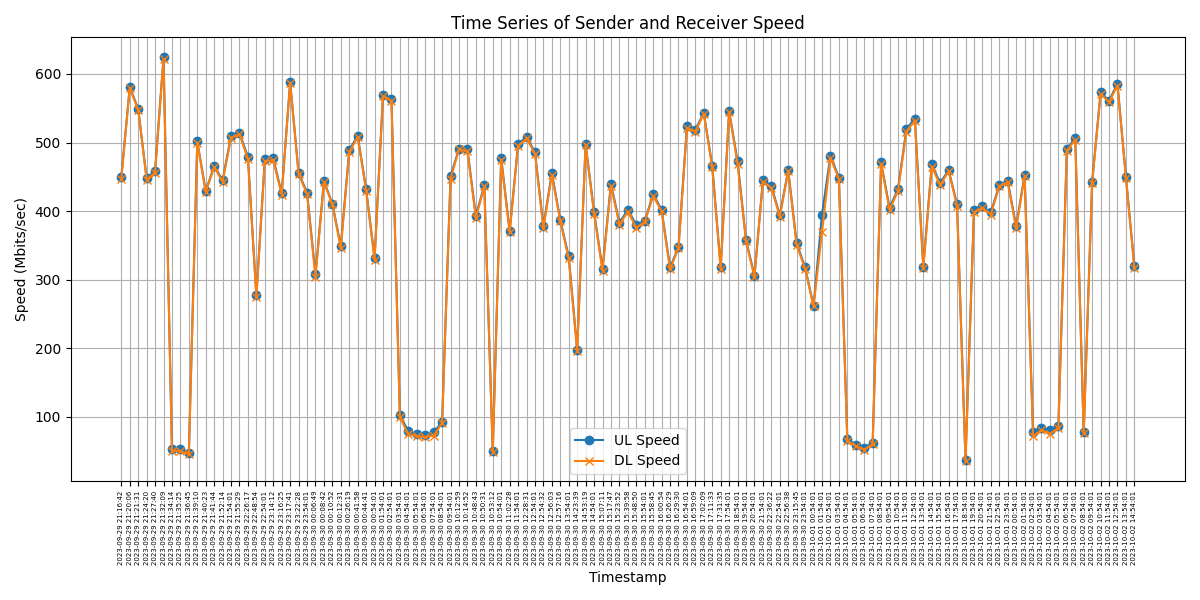
import matplotlib.pyplot as plt  
import re  
  
def read\_iperf\_data(file\_path):  
 with open(file\_path, 'r') as file:  
 data = file.read()  
 return data  
  
def parse\_iperf\_data(data):  
 timestamps = []  
 speeds\_dl = []  
 speeds\_ul = []  
 last\_is\_date = False  
  
 lines = data.split('\n')  
  
 for line in lines:  
 if re.search(r'\d{4}-\d{2}-\d{2} \d{2}:\d{2}:\d{2}', line):  
 if not last\_is\_date:  
 timestamps.append(line.strip())  
 last\_is\_date = True  
 else:  
 timestamps.pop()  
 timestamps.append(line.strip())  
 else:  
 striped\_line = line.strip()  
 if re.search("receiver", striped\_line):  
 items = striped\_line.split()  
 speeds\_dl.append(float(items[6]))  
 last\_is\_date = False  
 elif re.search("sender", striped\_line):  
 items = striped\_line.split()  
 speeds\_ul.append(float(items[6]))  
 last\_is\_date = False  
  
 return timestamps, speeds\_ul, speeds\_dl  
  
def plot\_iperf\_data(timestamps, speeds\_ul, speeds\_dl):  
 plt.figure(figsize=(12, 6))  
 plt.plot(timestamps, speeds\_ul, label='UL Speed', marker='o')  
 plt.plot(timestamps, speeds\_dl, label='DL Speed', marker='x')  
 plt.title('Time Series of Sender and Receiver Speed')  
 plt.xlabel('Timestamp')  
 plt.ylabel('Speed (Mbits/sec)')  
 plt.legend()  
 plt.grid(True)  
 plt.xticks(rotation=90, fontsize=5)  
 plt.tight\_layout()  
 plt.show()  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 file\_path = 'files/iperf/ok1.iperf.comnet-student.eu.txt'  
 iperf\_data = read\_iperf\_data(file\_path)  
 timestamps, speeds\_ul, speeds\_dl = parse\_iperf\_data(iperf\_data)  
 plot\_iperf\_data(timestamps, speeds\_ul, speeds\_dl)

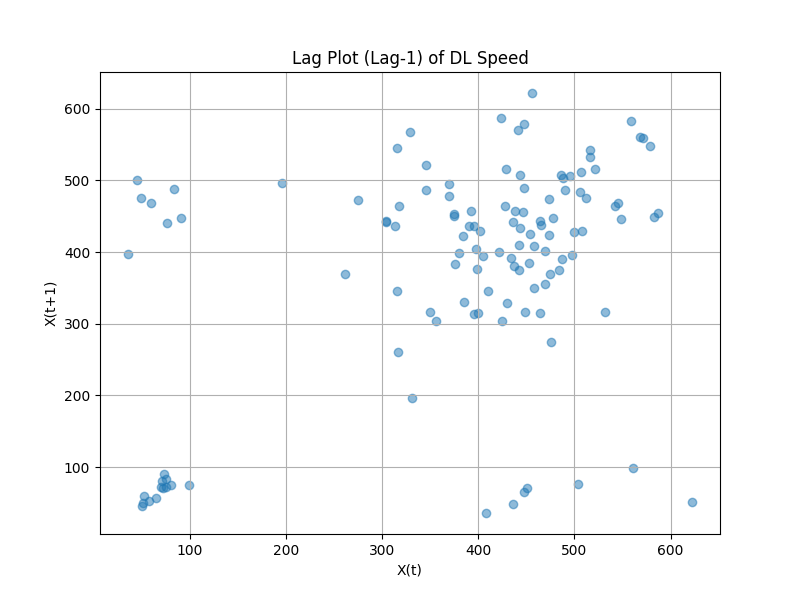
This code first reads network performance test data from a specified file using iperf. It then parses the data to extract timestamps, upload speeds, and download speeds. Next, it utilizes the Matplotlib library to create a time series chart, where the x-axis represents timestamps and the y-axis represents speeds (both upload and download). Upload and download speeds are labeled on the chart.

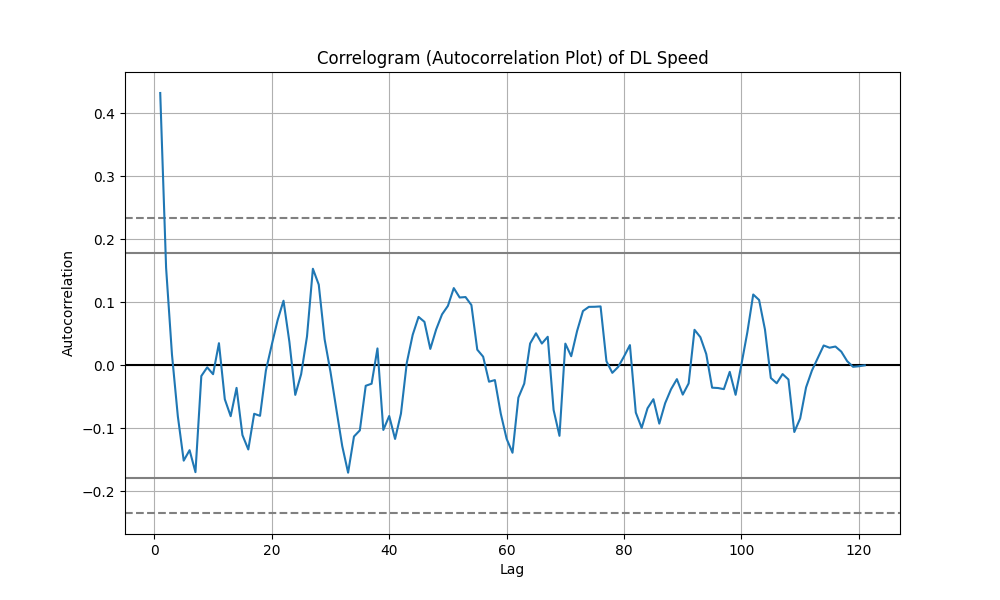
import matplotlib.pyplot as plt  
import re  
import pandas as pd  
  
# Read data from the file  
with open('files/iperf/ok1.iperf.comnet-student.eu.txt', 'r') as file:  
 data = file.read()  
  
# Extract time and speed data  
timestamps = []  
speeds\_dl = []  
speeds\_ul = []  
  
# Initialize a variable to track the last line type (0: none, 1: date, 2: receiver, 3: sender)  
last\_line\_type = 0  
  
lines = data.split('\n')  
  
for line in lines:  
 if re.search(r'\d{4}-\d{2}-\d{2} \d{2}:\d{2}:\d{2}', line):  
 if last\_line\_type == 1:  
 timestamps.pop()  
 timestamps.append(line.strip())  
 last\_line\_type = 1  
 else:  
 striped\_line = line.strip()  
 if re.search("receiver", striped\_line):  
 items = striped\_line.split()  
 speeds\_dl.append(float(items[6]))  
 last\_line\_type = 2  
 elif re.search("sender", striped\_line):  
 items = striped\_line.split()  
 speeds\_ul.append(float(items[6]))  
 last\_line\_type = 3  
  
# Create lag plots and autocorrelation plots for download speed (DL)  
plt.figure(figsize=(8, 6))  
plt.scatter(speeds\_dl[:-1], speeds\_dl[1:], alpha=0.5)  
plt.title('Lag Plot (Lag-1) of DL Speed')  
plt.xlabel('X(t)')  
plt.ylabel('X(t+1)')  
plt.grid(True)  
plt.show()  
  
plt.figure(figsize=(10, 6))  
pd.plotting.autocorrelation\_plot(speeds\_dl, ax=plt.gca())  
plt.title('Correlogram (Autocorrelation Plot) of DL Speed')  
plt.xlabel('Lag')  
plt.grid(True)  
plt.show()  
  
# Create lag plots and autocorrelation plots for upload speed (UL)  
plt.figure(figsize=(8, 6))  
plt.scatter(speeds\_ul[:-1], speeds\_ul[1:], alpha=0.5)  
plt.title('Lag Plot (Lag-1) of UL Speed')  
plt.xlabel('X(t)')  
plt.ylabel('X(t+1)')  
plt.grid(True)  
plt.show()  
  
plt.figure(figsize=(10, 6))  
pd.plotting.autocorrelation\_plot(speeds\_ul, ax=plt.gca())  
plt.title('Correlogram (Autocorrelation Plot) of UL Speed')  
plt.xlabel('Lag')  
plt.grid(True)  
plt.show()

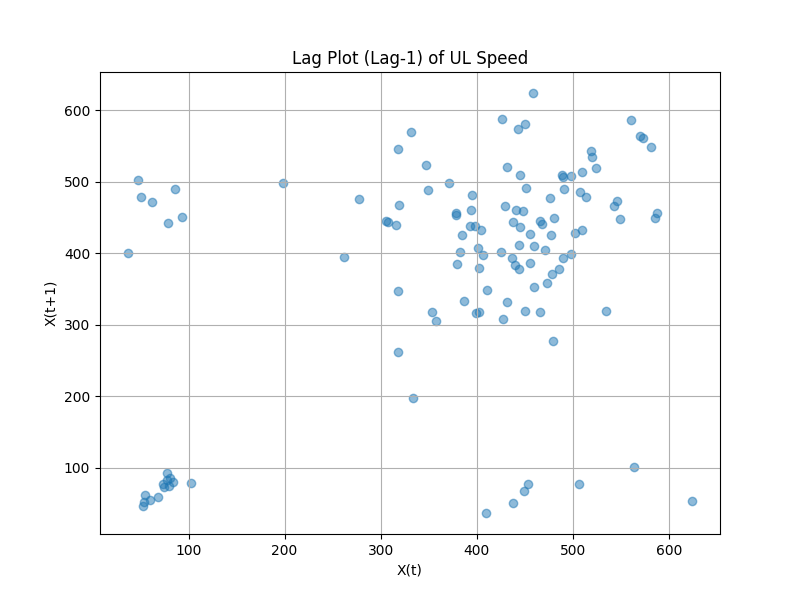
The purpose of this code is to read network performance test data from a specified file using iperf, extract timestamps, download speeds, and upload speeds from the data, and create lag plots and autocorrelation plots. These plots help analyze the relationship between download speeds and upload speeds as well as their autocorrelation over time.

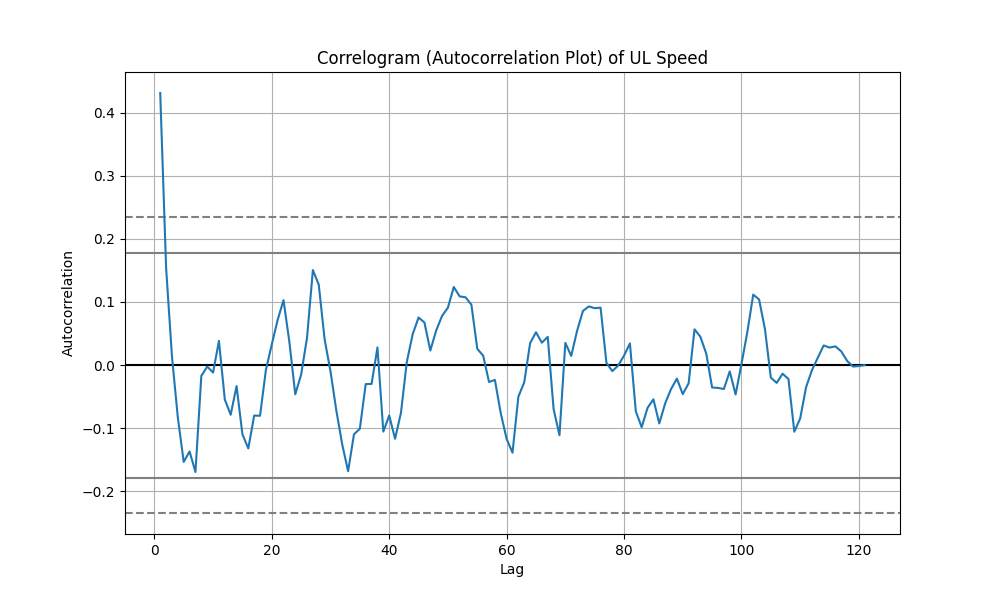
AS2.i1:









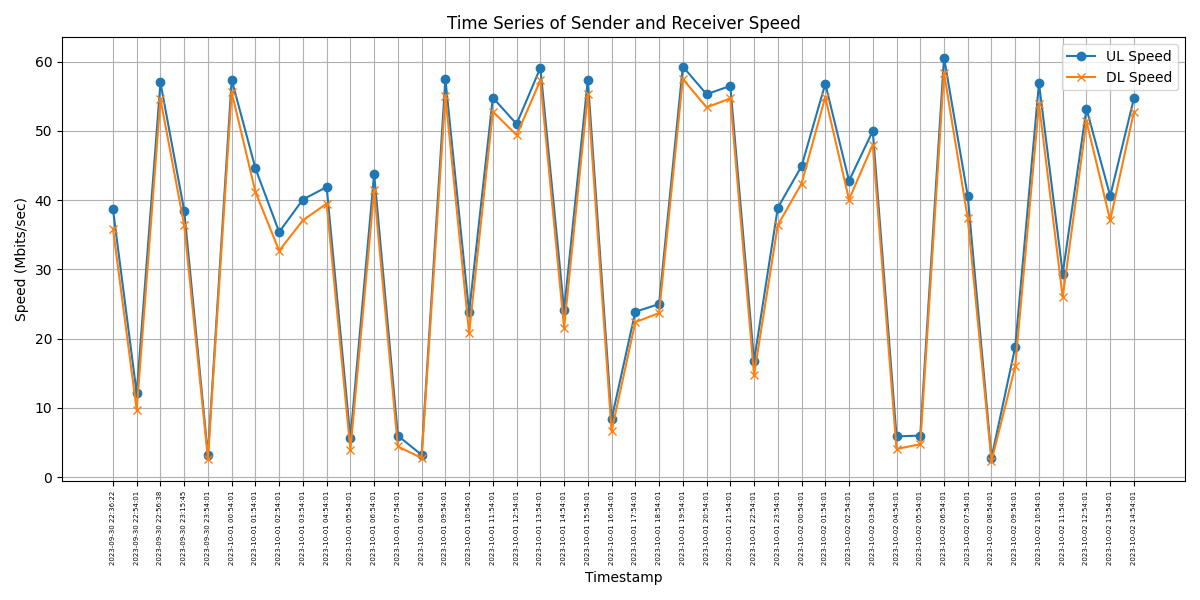


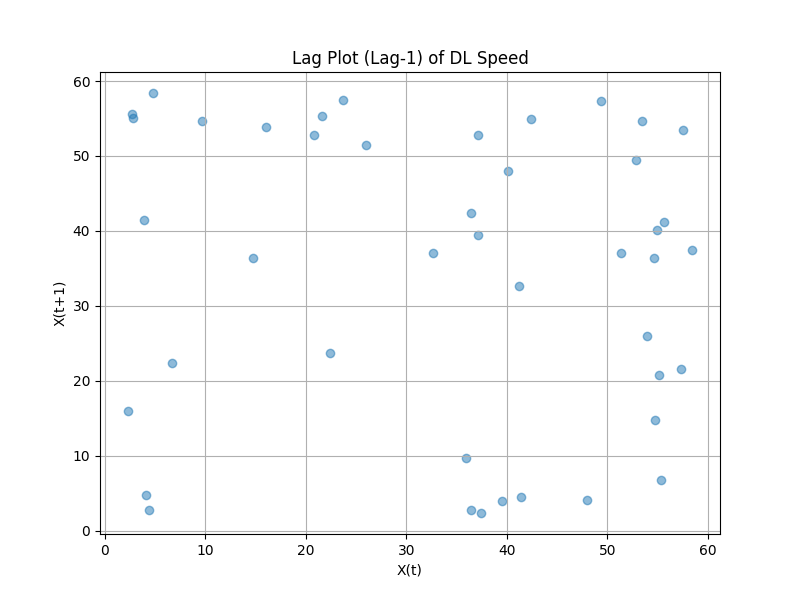
The first graph displays the time series data, where both upload speed (UL Speed) and download speed (DL Speed) vary over time. We can observe that both demonstrate significant fluctuations, indicating that at different points in time, there are notable peaks and troughs in speed. There seems to be a pattern of periodicity in both upload and download speeds, suggesting the possibility of periodic patterns in network usage, such as peak and off-peak periods.

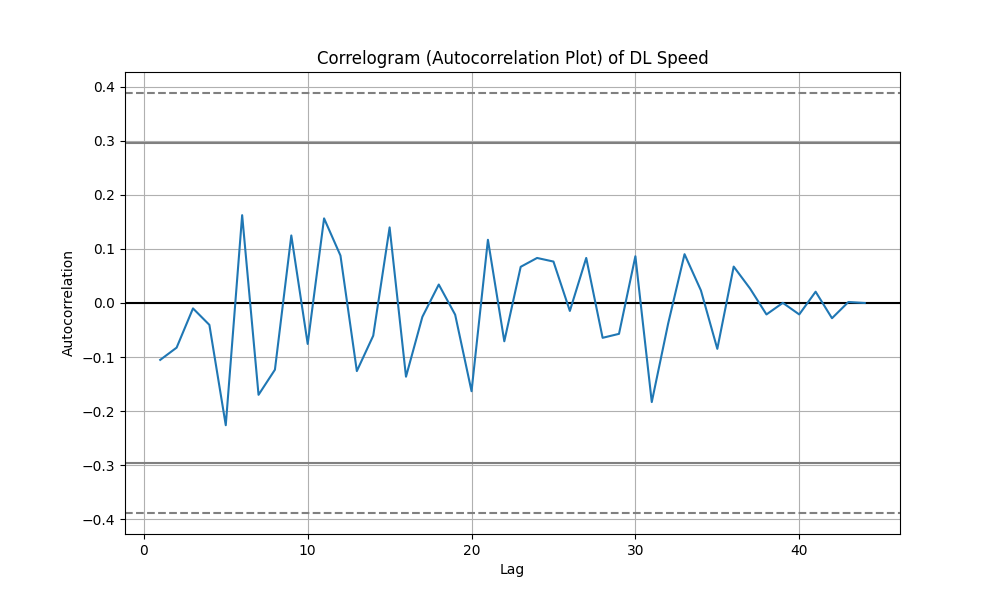
The second and fourth graphs are lag plots for upload and download speeds. These plots show the relationship of data points with their previous time point (lag-1). For the download speed (DL Speed), the lag plot exhibits a degree of positive linear relationship, indicating that the value at one time point is related to its previous value, which may imply autocorrelation. For the upload speed (UL Speed), this relationship seems to be less pronounced, but a degree of positive linear relationship can still be observed.

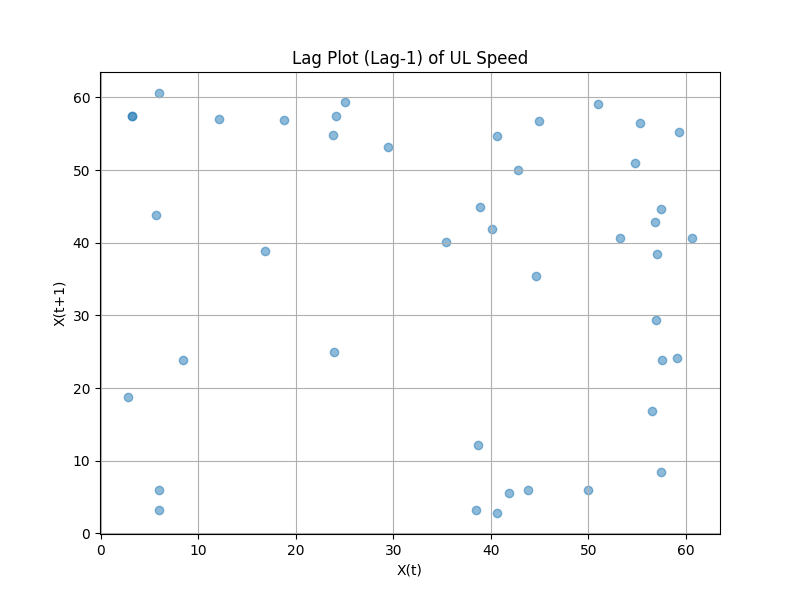
The third and fifth graphs are autocorrelation plots of upload and download speeds, showing the degree of autocorrelation at different lags. For download speed, the autocorrelation plot reveals a high initial autocorrelation at the first lag, but this correlation quickly decreases and fluctuates at different lags. This indicates the possibility of short-term dependencies. For upload speed, the autocorrelation plot also shows a high initial autocorrelation, but this correlation weakens as lag increases and exhibits greater volatility, which might indicate the presence of noise in the data.

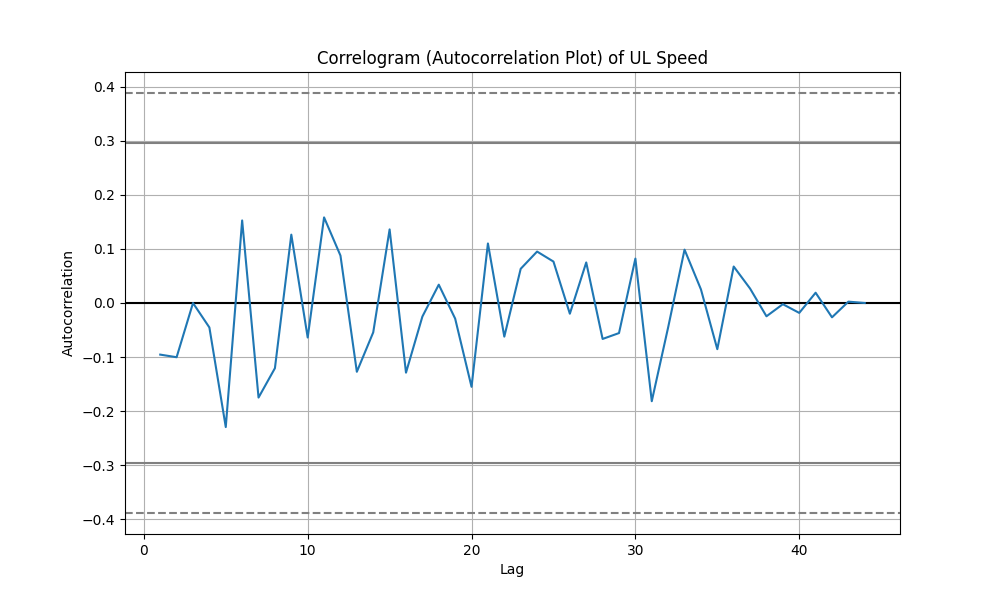
AS2.i2:











The first graph displays the time series of upload speed (UL Speed) and download speed (DL Speed). There is a certain level of variability over time, but the amplitude of fluctuations is much smaller compared to the AS2.i1 dataset. This suggests more stable network performance, with relatively consistent network conditions during the test period. There is a certain synchronicity between upload and download speeds, with both increasing or decreasing simultaneously at most time points, potentially indicating that they are affected by similar network conditions.

The second graph is a lag plot for download speed (DL Speed). Compared to AS2.i1, this dataset's lag plot shows a more dispersed pattern, indicating a less pronounced association between a speed measurement and its preceding value. The fourth graph, the lag plot for upload speed (UL Speed), also displays dispersed points, suggesting a similarly weak temporal association for upload speeds.

The third graph is an autocorrelation plot for download speed (DL Speed). It shows the autocorrelation coefficients fluctuating at different lags but generally trending near zero, which indicates a lack of strong autocorrelation between observations of download speed. The fifth graph is an autocorrelation plot for upload speed (UL Speed). The fluctuation pattern is similar to that of download speed, with autocorrelation coefficients primarily oscillating around the zero line, indicating that the autocorrelation between observations of upload speed is also not strong.

Compare to 3.2:

The AS1 series exhibits stronger autocorrelation during the initial lags and maintains a higher level of significance over a larger number of lags, suggesting that the AS1 series might have more predictable patterns or trends.

The AS2 series shows less persistence in both download and upload speeds, with autocorrelation quickly dropping to near-zero levels after a few initial lags. This may imply that the AS2 series is more influenced by random variations or external factors that disrupt the signal's persistence. In both AS2.i1 and AS2.i2, download and upload speeds exhibit similar behaviors, suggesting that the underlying factors affecting download and upload speeds are likely similar within each dataset.

Overall, the AS1 datasets seem to represent processes with longer memories and more predictable patterns, while the AS2 datasets are characterized by lower predictability and greater randomness over time.

* Conclusion
* Describe the system from which you obtained measurements and any challenges you encountered during the process.

I encountered the following challenges. First, collecting data with sufficient granularity to capture fluctuations without creating an overwhelming volume can be challenging. Balancing granularity, storage capacity, and processing power is often a key issue. Network performance can vary due to a multitude of factors such as bandwidth, network traffic, hardware limitations, and service provider issues. This variability makes it difficult to establish a baseline for performance. Ensuring the accuracy and consistency of the collected data can be difficult, especially if the measurements are affected by transient issues that do not reflect the typical performance of the network. External factors, such as server response times, routing paths, and end-user device performance, can also affect the measurements.

* Was there any correlation between the path length (number of routers, which can be checked using traceroute and/or the TTL value of ICMP Echo Responses) and the stability of measurements? If you also recorded the TTL value, did it change over time?

Longer network paths, indicated by a greater number of routers or hops, can potentially lead to decreased stability in measurements. Each additional router introduces a new potential point for delays, packet loss, or variations. However, this relationship is not always direct. Network stability also depends on the performance and reliability of each router in the path, overall network traffic, and the quality of the network links.

TTL values can indicate the length of a network path, as each router along the path typically decrements the TTL by one. However, the starting TTL value set by the sending host can vary, so the absolute TTL value may not always accurately reflect the exact number of hops. Changes in TTL values over time can indicate changes in the network path. For example, if the network dynamically reroutes traffic due to congestion or link failure, this might be reflected in varying TTL values.

If TTL values change frequently, it could indicate a less stable network path, as route changes could affect latency and packet loss. Conversely, consistent TTL values over time might suggest a stable route without frequent path changes, which could correlate with more stable network performance measurements.

* Was there any observable correlation between the throughput and latency?

In general, there is an inverse relationship between throughput and latency. High latency typically leads to lower throughput, and vice versa. This is because high latency may indicate delays in the transmission of data packets, which can consequently reduce the rate at which data is successfully transferred (throughput). In networks with high latency, even with high bandwidth, throughput may not be fully utilized due to the delay in acknowledging received packets. During network congestion, both latency and throughput can be adversely affected. Latency increases because packets take longer to be processed and delivered, and throughput decreases due to packet loss and the need for retransmissions.

* Final 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?

In my own traffic (Task 1), it can be observed that the destination ports are more concentrated on commonly used ports, such as HTTPS (port 443). The port distribution is relatively stable because I only have access to a limited variety of services. Additionally, the overall traffic is low, but there may be sudden spikes. Traffic patterns may fluctuate significantly over time, reflecting individual usage habits.

On the other hand, the provided data (Task 2) includes a wider range of destination ports, encompassing various enterprise software, applications, mail servers, databases, internal web applications, and various remote access services. Traffic is higher during working hours and may be lower during non-working hours. Overall traffic is higher and exhibits a more even distribution, reflecting day-to-day business activities.

* Comparing RTT latency about TCP connections (3.1), were active latency measurements around the same magnitude or was another much larger than the other?

The active latency measurements vary across different orders of magnitude. Some servers exhibit very low latency, consistently staying below 20 milliseconds, while others have latencies around 50 milliseconds. There are also servers with latencies ranging from 200 milliseconds to 300 milliseconds. Therefore, the active latency measurements span different orders of magnitude and are not clustered around the same magnitude.

* 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).

If I were an employee of a network provider conducting similar measurements in a network provider network where traffic is generated by both individuals and companies, data protection would be a significant consideration.

1. Privacy Concerns: Whether it's individuals or companies, customers using the services of a network provider have a reasonable expectation of privacy. Any data collected during measurements should be anonymized and aggregated to ensure that individual users or businesses cannot be identified from the collected data.
2. Data Minimization: Data collection should be limited to only what is strictly necessary for the specific measurements or analysis. Avoid collecting any unnecessary personal or sensitive information.
3. Informed Consent: Ensure that customers are informed about the nature and purpose of the measurements being conducted. If feasible, customers should have the option to choose to participate in or opt out of data collection.
4. Data Security: Implement robust security measures to prevent unauthorized access, breaches, or data leaks. This includes the use of encryption, access controls, and other security protocols to protect the data.
5. Data Retention: Define clear policies regarding data retention and deletion. Data should not be retained longer than necessary for the intended purpose, and there should be a secure process for disposing of data when it's no longer needed.
6. Compliance with Regulations: Comply with relevant data protection regulations and laws, such as the European GDPR (General Data Protection Regulation).
7. Transparency: Maintain transparency about data collection and processing practices. Customers should have access to information about what data is being collected, how it's used, and who can access it.
8. Data Use Limitation: Ensure that the collected data is used only for the specific purpose of network performance measurements and analysis. It should not be repurposed for other uses without explicit consent.
9. Data Access Requests: Be prepared to respond to customer requests for access to their own data or requests for data deletion in accordance with applicable data protection laws.
10. Regular Audits: Conduct regular internal audits and assessments of data protection practices to identify and address potential risks and compliance issues.

In summary, when conducting measurements in a network provider network with data generated by customers, data protection and privacy considerations should be a top priority. Adhering to privacy principles, regulations, and best practices is essential for maintaining customer trust and complying with legal requirements.

* Discuss how data protection needs to be taken into account if you as a company ICT support group employee were doing similar measurements as in this assignment in a company network (traffic generated by employees and customers).
  + - 1. Ownership of Data and Consent: Determine who owns the data being collected, whether it's the company, its employees, or its customers. Obtain explicit data collection consent from employees and customers in accordance with data ownership and local regulations, especially if data collection extends beyond regular network monitoring.
      2. Anonymization and Pseudonymization: Ensure that any personal or sensitive data collected is subjected to anonymization to prevent the identification of individual employees or customers. If complete anonymization is not feasible, pseudonymization can be employed to replace personally identifiable information (PII) with artificial identifiers.
      3. Data Minimization: Collect only the data that is strictly necessary for network performance measurement and analysis. Avoid collecting any unnecessary personal or sensitive information that is not directly related to the measurement objectives.
      4. Security Measures: Utilize strong encryption to safeguard data during transit and storage, preventing unauthorized access or breaches. Implement access controls to ensure that only authorized personnel can access the collected data. Conduct regular security audits to identify vulnerabilities and ensure compliance with data protection policies.
      5. Data Retention and Deletion: Define explicit data retention policies specifying how long data will be stored. Establish secure data disposal procedures to prevent data leaks or violations when data is no longer needed.
      6. Transparency and Communication: Clearly communicate to employees and customers the information pertaining to the data being collected, the reasons for its collection, and how it will be used. Provide easily accessible privacy policies outlining data handling practices.
      7. Compliance with Regulations: Ensure that data collection and processing practices adhere to applicable data protection regulations and laws, such as GDPR, CCPA, or other regional laws.
* 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?

For me, both Task 2 and Task 3 are considered complex. In Task 2, I need to delve into a lot of advanced computer knowledge and write a significant amount of scripts. In Task 3, I have to switch between many different files, which also results in a substantial workload. In contrast, Task 1 is relatively straightforward as it involves simply observing and describing the data. Considering data volume, while dealing with larger datasets may require more time, it typically doesn't pose issues as long as memory and computing resources are managed effectively. Therefore, data volume might extend the time required for the tasks but doesn't necessarily make them more complex. Overall, I find the tasks in the final assignment to be quite demanding and time-consuming.