Name: Yutong Du

Date: 2023-11-25

Email: Yutong.du@aalto.fi

Student ID: 101481573

# Task 1: Capturing data

## Acquiring packet capture data

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

**Wireshark**

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

**25/11/2023: From 17:31 to 19:31. Measurement setting: Default measurement setting of Wireshark. File name: final\_a.pcap.**

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

**Wireshark) or file name if you are using the some public traces.**

|  |
| --- |
|  |

## Packet data PS1

### 1.1: Visualising packet distribution by port numbers.

**Code:**

|  |
| --- |
| import pyshark import matplotlib.pyplot as plt from collections import Counter from operator import itemgetter  tshark\_path = 'D:\\0x00\_Softwares\\Wireshark\\tshark.exe' file\_path = 'files/final\_a.pcap'  cap = pyshark.FileCapture(file\_path, tshark\_path=tshark\_path, keep\_packets=True)  port\_counts = Counter() for packet in cap:  if 'TCP' in packet or 'UDP' in packet:  layer = packet.tcp if 'TCP' in packet else packet.udp  port\_counts[layer.dstport] += 1  cap.close()  sorted\_port\_counts = dict(sorted(port\_counts.items(), key=itemgetter(1), reverse=True)[:20])  plt.figure(figsize=(10, 10)) plt.pie(sorted\_port\_counts.values(), labels=sorted\_port\_counts.keys(), autopct='%1.1f%%', startangle=140) plt.title('Top 20 Packet Distribution by Port Numbers') plt.show() |

**Packet distribution:**

|  |
| --- |
|  |

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

**Exported the part1.pcap as part1.csv.**

**Code:**

|  |
| --- |
| import pandas as pd import matplotlib.pyplot as plt  csv\_file\_path = 'files/final\_a.csv'  df = pd.read\_csv(csv\_file\_path)  df['Time'] = pd.to\_datetime(df['Time'], unit='s')  traffic\_volume = df.groupby('Time')['Length'].sum().reset\_index()  traffic\_per\_second = traffic\_volume.set\_index('Time').resample('S').sum().fillna(0) traffic\_per\_minute = traffic\_volume.set\_index('Time').resample('T').sum().fillna(0)  plt.figure(figsize=(14, 7))  plt.subplot(1, 2, 1) plt.plot(traffic\_per\_second.index, traffic\_per\_second['Length'], linestyle='-') plt.title('Traffic Volume per Second') plt.xlabel('Time (second)') plt.ylabel('Traffic Volume (bytes)') plt.xticks(rotation=45)  plt.subplot(1, 2, 2) plt.plot(traffic\_per\_minute.index, traffic\_per\_minute['Length'], marker='o', linestyle='-', color='orange') plt.title('Traffic Volume per Minute') plt.xlabel('Time (minute)') plt.ylabel('Traffic Volume (bytes)') plt.xticks(rotation=45)  plt.tight\_layout() plt.show() |

**Result:**

|  |
| --- |
|  |

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

**Code:**

|  |
| --- |
| import pandas as pd import matplotlib.pyplot as plt import numpy as np  csv\_file\_path = 'files/final\_a.csv'  df = pd.read\_csv(csv\_file\_path)  plt.figure(figsize=(14, 7))  plt.subplot(1, 2, 1) bin\_width = 100 bins = range(min(df['Length']), max(df['Length']) + bin\_width, bin\_width) plt.hist(df['Length'], bins=bins, color='blue', alpha=0.7, log=True) plt.title('Packet Length Distribution (Log Scale)') plt.xlabel('Packet Length (bytes)') plt.ylabel('Frequency (Log Scale)')  plt.subplot(1, 2, 2) sorted\_length = np.sort(df['Length']) yvals = np.arange(1, len(sorted\_length) + 1) / len(sorted\_length) plt.plot(sorted\_length, yvals, marker='.', linestyle='none') plt.title('Empirical Cumulative Distribution Function (ECDF)') plt.xlabel('Packet Length (bytes)') plt.ylabel('ECDF')  plt.tight\_layout() plt.show()  print("Summary statistics for packet lengths:") print(df['Length'].describe()) |

**Result:**

|  |
| --- |
|  |

**Key summary statistics:**

|  |
| --- |
| **Summary statistics for packet lengths:**  count 606080.000000  mean 1021.099314  std 1893.895590  min 42.000000  25% 93.000000  50% 1292.000000  75% 1292.000000  max 65226.000000 |

## Flow data PS2

### 1.4: Visualise flow distribution by port numbers.

**Use the command below to convert part2.cap to flow data:**

|  |
| --- |
| **tshark -r final\_a.pcap -q -z conv,tcp > final\_a.txt** |

**Code:**

|  |
| --- |
| import pandas as pd import numpy as np import matplotlib.pyplot as plt from matplotlib.lines import Line2D  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\_a.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))  df['port'] = df['second\_ip\_interface'].str.split(':').str[1].astype(str)  port\_flow\_count = df.groupby('port').size().reset\_index(name='count') port\_flow\_count = port\_flow\_count.sort\_values(by='count', ascending=False)  plt.figure(figsize=(10, 6)) bar\_plot = plt.bar(port\_flow\_count['port'], port\_flow\_count['count'])  for bar in bar\_plot:  yval = bar.get\_height()  plt.text(bar.get\_x() + bar.get\_width() / 2, yval, int(yval), va='bottom', ha='center')  plt.xlabel('Port Number') plt.ylabel('Number of Flows') plt.title('Flow Count Distribution by Port Numbers (Descending)') plt.xticks(rotation=45) plt.show() |

**Flow distribution:**

|  |
| --- |
|  |

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

**Code:**

|  |
| --- |
| import pandas as pd import matplotlib.pyplot as plt import re import datetime  # Function to parse each line of the data 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\_a.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 different time scales # 1. Resampling to seconds df\_seconds = full\_df.resample('1S', on='Start').sum()  # 2. Resampling to minutes df\_minutes = full\_df.resample('1T', on='Start').sum()  # Plotting the data fig, axs = plt.subplots(2, 1, figsize=(12, 10))  # Plot for second-wise data axs[0].plot(df\_seconds.index, df\_seconds['Total\_Bytes']) axs[0].set\_title('Traffic Volume per Second') axs[0].set\_xlabel('Time') axs[0].set\_ylabel('Total Bytes') axs[0].grid(True)  # Plot for minute-wise data axs[1].plot(df\_minutes.index, df\_minutes['Total\_Bytes'], marker='x',color='orange') axs[1].set\_title('Traffic Volume per Minute') axs[1].set\_xlabel('Time') axs[1].set\_ylabel('Total Bytes') axs[1].grid(True)  # Adjust layout and display the plot plt.tight\_layout() plt.show() |

**Result:**

|  |
| --- |
|  |

### 1.6: Visualise flow distribution by country.

**Code:**

|  |
| --- |
| import pandas as pd import matplotlib.pyplot as plt from geoip2.database import Reader  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\_a.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))  geoip\_reader = Reader('others/GeoLite2-Country.mmdb')  def get\_country(ip):  try:  response = geoip\_reader.country(ip)  return response.country.name  except:  return "Unknown"  df['country'] = df['second\_ip\_interface'].str.split(':').str[0].apply(get\_country)  country\_traffic = df.groupby('country').size()  plt.figure(figsize=(12, 8)) country\_traffic.plot(kind='bar') plt.xlabel('Country') plt.ylabel('Total Traffic (Bytes)') plt.title('Flow Distribution by Country') plt.xticks(rotation=45)  # 添加数字标签 for i, v in enumerate(country\_traffic):  plt.text(i, v, str(v), ha='center', va='bottom', fontsize=8)  plt.show() |

**Result:**

|  |
| --- |
|  |

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

**Code:**

|  |
| --- |
| import pandas as pd import matplotlib.pyplot as plt from geoip2.database import Reader  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\_a.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))  df['src\_dst\_pair'] = df['first\_ip\_interface'].str.split(':').str[0] + ' - ' + df['second\_ip\_interface'].str.split(':').str[0]  traffic\_data = df.groupby('src\_dst\_pair')['total\_bytes'].sum() flow\_counts = df.groupby('src\_dst\_pair').size()  sorted\_traffic\_data = traffic\_data.sort\_values(ascending=False) sorted\_flow\_counts = flow\_counts.sort\_values(ascending=False)  plt.figure(figsize=(10, 6)) plt.plot(sorted\_traffic\_data.values) plt.xlabel('Source-Destination Pairs') plt.ylabel('Total Data Volume (Bytes)') plt.title('Zipf Plot of Data Volume by Source-Destination Pairs') plt.yscale('log') plt.xscale('log') plt.show()  plt.figure(figsize=(10, 6)) plt.plot(sorted\_flow\_counts.values) plt.xlabel('Source-Destination Pairs') plt.ylabel('Number of Flows') plt.title('Zipf Plot of Flows by Source-Destination Pairs') plt.yscale('log') plt.xscale('log') plt.show() |

**Result:**

|  |
| --- |
|  |
|  |

### 1.8: Plot flow length distribution, its empirical cumulative distribution function and key summary statistics.

**Code:**

|  |
| --- |
| df = df.assign(\*\*df.apply(convert\_to\_byte, axis=1))  # Plot flow length distribution (Histogram) 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.show()  # Plot the ECDF plt.figure(figsize=(10, 6)) sns.ecdfplot(df['total\_frames']) plt.title('Empirical Cumulative Distribution Function (ECDF) of Flow Length') plt.xlabel('Flow Length (Frames)') plt.ylabel('ECDF') plt.grid(True) # Adding a grid for better readability plt.show()  # Display key summary statistics print("Key Summary Statistics (Total Frames):") print(df['total\_frames'].describe()) |

**Result:**

|  |
| --- |
|  |
| **Zoom in the part of 0-2000 flow length:** |
|  |

**Key Summary Statistics (Total Frames):**

|  |
| --- |
| **Key Summary Statistics (Total Frames):**  **count 2944.000000**  **mean 52.886209**  **std 920.973239**  **min 1.000000**  **25% 11.000000**  **50% 11.000000**  **75% 24.000000**  **max 47943.000000** |

### 1.9: Fit a distribution for the flow lengths and validate the model.

**Code:**

|  |
| --- |
| 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/part2.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))  # Assuming df is your DataFrame and 'total\_frames' is the column with flow lengths data = df['total\_frames']   # Create a distfit object and fit it dist = distfit() dist.fit\_transform(data)  # Print the summary to see the results print(dist.summary) # Plot the fitted distribution dist.plot() plt.show() |

**Output:**

|  |
| --- |
| **name score loc scale**  **0 dweibull 0.047392 11.0 1011.249797**  **1 genextreme 0.053895 11.653233 4.957702**  **2 t 0.059903 11.0 0.0**  **3 lognorm 0.063001 0.281611 16.59488**  **4 pareto 0.071559 -57.382389 58.382389**  **5 beta 0.071931 1.0 117008.479671**  **6 expon 0.072035 1.0 51.886209**  **7 norm 0.081486 52.886209 920.81681**  **8 loggamma 0.081623 -325912.86128 46506.518536**  **9 uniform 0.08186 1.0 47942.0**  **10 gamma 0.081868 1.0 4.007575** |

**Validation with plots:**

|  |
| --- |
|  |

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

**Code:**

|  |
| --- |
| from scapy.all import rdpcap, PacketList from datetime import datetime, timedelta  # 读取 PCAP 文件 packets = rdpcap('files/final\_a.pcap')  # 定义超时设置 timeouts = [1, 10, 60, 120, 1800] # 单位为秒  # 创建一个字典来存储不同超时设置下的流数量 flows\_for\_timeouts = {}  for timeout in timeouts:  # 初始化流列表和当前流的数据包列表  flows = []  current\_flow\_packets = []  last\_packet\_time = None   for packet in packets:  if 'IP' in packet and 'TCP' in packet:  # 获取当前数据包的时间戳  current\_packet\_time = datetime.fromtimestamp(float(packet.time))   # 检查是否是流的第一个数据包或者当前数据包与上一个数据包的时间差是否超过了超时设置  if last\_packet\_time is None or (current\_packet\_time - last\_packet\_time).total\_seconds() > timeout:  # 如果是新的流，先保存当前流，然后开始一个新的流  if current\_flow\_packets:  flows.append(PacketList(current\_flow\_packets))  current\_flow\_packets = [packet]  else:  # 否则，将数据包添加到当前流中  current\_flow\_packets.append(packet)   # 更新最后一个数据包的时间  last\_packet\_time = current\_packet\_time   # 保存最后一个流  if current\_flow\_packets:  flows.append(PacketList(current\_flow\_packets))   # 记录当前超时设置下的流数量  flows\_for\_timeouts[timeout] = len(flows)  for timeout, count in flows\_for\_timeouts.items():  print(f"Timeout: {timeout} seconds, Flow Count: {count}") |

**Result:**

|  |
| --- |
| Timeout: 1 seconds, Flow Count: 1821  Timeout: 10 seconds, Flow Count: 22  Timeout: 60 seconds, Flow Count: 1  Timeout: 120 seconds, Flow Count: 1  Timeout: 1800 seconds, Flow Count: 1 |

## TCP connection data PS3

### 1.11: Round-trip times and their variance.

**Use the command for converting .pcap to .csv and delete rows that are not relative:**

|  |
| --- |
| **tcptrace -l -r -n --csv final\_a.pcap > final\_a\_tcptrace.csv** |

**Code:**

|  |
| --- |
| import pandas as pd import matplotlib.pyplot as plt import seaborn as sns  # 加载数据 file\_path = 'files/final\_a\_tcptrace.csv' df = pd.read\_csv(file\_path)  # 查找相关的列 rtt\_avg\_columns = ['RTT\_avg\_a2b', 'RTT\_avg\_b2a'] retrans\_max\_columns = ['max\_#\_retrans\_a2b', 'max\_#\_retrans\_b2a']  # 分析 RTT 平均值与最大重传次数的关系 for rtt\_col, retrans\_col in zip(rtt\_avg\_columns, retrans\_max\_columns):  # 绘制散点图来查看这两个变量之间的关系  plt.figure(figsize=(10, 6))  sns.scatterplot(x=df[rtt\_col], y=df[retrans\_col])  plt.title(f'Relationship between {rtt\_col} and {retrans\_col}')  plt.xlabel('Average RTT')  plt.ylabel('Max Number of Retransmissions')  plt.show()   # 计算相关系数  correlation = df[rtt\_col].corr(df[retrans\_col])  print(f"Correlation between {rtt\_col} and {retrans\_col}: {correlation}") |

**Result:**

|  |
| --- |
|  |
|  |

**Output:**

|  |
| --- |
| **Correlation between RTT\_avg\_a2b and max\_#\_retrans\_a2b: 0.18100518093596465**  **Correlation between RTT\_avg\_b2a and max\_#\_retrans\_b2a: -0.013199099179595365** |

I have analyzed the relationship between the average Round-Trip Time (RTT) and the maximum number of retransmissions in both directions (a2b and b2a). Here are the results of the analysis:

For RTT\_avg\_a2b and max\_#\_retrans\_a2b:

The scatter plot displays the relationship between these two variables.

The correlation coefficient is 0.181, indicating a slight positive correlation.

For RTT\_avg\_b2a and max\_#\_retrans\_b2a:

The scatter plot shows the relationship between these two variables.

The correlation coefficient is -0.013, indicating almost no correlation.

These results suggest that the relationship between the average RTT and the number of retransmissions may not be very strong, especially in the b2a direction.

### 1.12: Total traffic volume during the connection (you get the volume from PS2).

**Code:**

|  |
| --- |
| import pandas as pd import matplotlib.pyplot as plt from geoip2.database import Reader  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\_a.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))  print(df["total\_bytes"].sum()) |

**Result total bytes:**

|  |
| --- |
| **165982587** |

## Conclusions

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

Spikes in Traffic Volume:

Both time scale charts show significant spikes in traffic volume intermittently. These spikes are indicative of bursty traffic patterns, which may suggest periods of high activity or data transfer events.

Time Scale Differences:

The chart with the traffic volume per second shows more granularity and allows for the observation of specific moments when the traffic volume sharply increases.

On the other hand, the traffic volume per minute aggregates these spikes, providing a view of prolonged periods of high traffic volume. The peaks are still noticeable, but they are fewer and more spread out.

Patterns:

There doesn't seem to be a regular pattern of the spikes in traffic volume; they appear to be sporadic.

Trends:

There is no clear increasing or decreasing trend in overall traffic volume; instead, the traffic seems to fluctuate significantly over time.

Downtime:

There are periods where the traffic volume drops to a low level, which could indicate times of low activity or network inactivity.

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

Port 53: This port is typically used by the Domain Name System (DNS), which is essential for resolving human-readable hostnames into IP addresses that are used in network routing.

Port 443: This port is associated with HTTPS, indicating secure web traffic through SSL/TLS encryption. This is widely used for secure communication over the internet, particularly for transactions and data transfer in web applications.

Port 80: It is used for HTTP, which is the foundational protocol for data communication on the World Wide Web. Traffic on this port indicates non-encrypted web traffic.

Port 123: This port is used by the Network Time Protocol (NTP), which is designed to synchronize clocks of networked devices.

Analyzing their characteristics:

DNS (Port 53): Fundamental for internet browsing as it translates domain names to IP addresses. High flow counts indicate frequent domain name resolutions, possibly due to web browsing, email services, or any internet-based application requiring domain lookups.

HTTPS (Port 443): Signifies secure communications and is critical for ensuring confidentiality and integrity of data. The high flow count reflects its widespread use in secure web browsing, online shopping, banking, and other services requiring secure data transmission.

HTTP (Port 80): Despite the shift towards secure communication channels, many websites and services still operate on HTTP, especially those not dealing with sensitive information. Its high flow count may also include traffic from devices or applications that have not migrated to HTTPS.

NTP (Port 123): Essential for time-sensitive applications and network security protocols which require accurate timekeeping. Its presence indicates network devices and applications are frequently synchronizing their clocks.

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

The number of packets is very high for certain flows, indicating a lot of small packets, which is a sign of a chatty protocol or an application sending many small updates.

Conversely, a flow with a large number of bytes but a smaller packet count indicates the transfer of large files or data streams.

Flow duration metrics reveal long-standing connections, such as those used by streaming services, versus short, bursty flows, like a web page request.

Packet analysis shows a diverse set of destination ports, which implies multiple types of services being accessed, whereas flow analysis shows that most data is exchanged with only a few IP addresses, suggesting centralized services or servers.

### Your findings on retransmissions.

I have analyzed the relationship between the average Round-Trip Time (RTT) and the maximum number of retransmissions in both directions (a2b and b2a). Here are the results of the analysis:

For RTT\_avg\_a2b and max\_#\_retrans\_a2b:

The scatter plot displays the relationship between these two variables.

The correlation coefficient is 0.181, indicating a slight positive correlation.

For RTT\_avg\_b2a and max\_#\_retrans\_b2a:

The scatter plot shows the relationship between these two variables.

The correlation coefficient is -0.013, indicating almost no correlation.

These results suggest that the relationship between the average RTT and the number of retransmissions may not be very strong, especially in the b2a direction.

# Task 2: Flow data

My student ID is 101481573, so my subnetwork is 163.35.139.0/24

So I used the commands(record by history command) below to extract my data:

|  |
| --- |
| 243 nohup gawk '$1~/^163\.35\.139\./||$15~/^163\.35\.139\./' 15-2-1800.t2 > ~/my\_15-2-1800.t2 &  244 nohup gawk '$1~/^163\.35\.139\./||$15~/^163\.35\.139\./' 15-2-1900.t2 > ~/my\_15-2-1900.t2 &  245 nohup gawk '$1~/^163\.35\.139\./||$15~/^163\.35\.139\./' 15-2-2000.t2 > ~/my\_15-2-2000.t2 &  246 nohup gawk '$1~/^163\.35\.139\./||$15~/^163\.35\.139\./' 15-2-2100.t2 > ~/my\_15-2-2100.t2 &  247 nohup gawk '$1~/^163\.35\.139\./||$15~/^163\.35\.139\./' 15-2-2200.t2 > ~/my\_15-2-2200.t2 &  248 nohup gawk '$1~/^163\.35\.139\./||$15~/^163\.35\.139\./' 15-2-2300.t2 > ~/my\_15-2-2300.t2 &  249 nohup gawk '$1~/^163\.35\.139\./||$15~/^163\.35\.139\./' 15-3-0000.t2 > ~/my\_15-3-0000.t2 &  250 nohup gawk '$1~/^163\.35\.139\./||$15~/^163\.35\.139\./' 15-3-0100.t2 > ~/my\_15-3-0100.t2 &  251 nohup gawk '$1~/^163\.35\.139\./||$15~/^163\.35\.139\./' 15-3-0200.t2 > ~/my\_15-3-0200.t2 &  252 nohup gawk '$1~/^163\.35\.139\./||$15~/^163\.35\.139\./' 15-3-0300.t2 > ~/my\_15-3-0300.t2 &  253 nohup gawk '$1~/^163\.35\.139\./||$15~/^163\.35\.139\./' 15-3-0400.t2 > ~/my\_15-3-0400.t2 &  254 nohup gawk '$1~/^163\.35\.139\./||$15~/^163\.35\.139\./' 15-3-0500.t2 > ~/my\_15-3-0500.t2 &  255 nohup gawk '$1~/^163\.35\.139\./||$15~/^163\.35\.139\./' 15-3-0600.t2 > ~/my\_15-3-0600.t2 &  256 nohup gawk '$1~/^163\.35\.139\./||$15~/^163\.35\.139\./' 15-3-0700.t2 > ~/my\_15-3-0700.t2 &  257 nohup gawk '$1~/^163\.35\.139\./||$15~/^163\.35\.139\./' 15-3-0800.t2 > ~/my\_15-3-0800.t2 &  258 nohup gawk '$1~/^163\.35\.139\./||$15~/^163\.35\.139\./' 15-3-0900.t2 > ~/my\_15-3-0900.t2 &  259 ls  260 nohup gawk '$1~/^163\.35\.139\./||$15~/^163\.35\.139\./' 15-3-1000.t2 > ~/my\_15-3-1000.t2 &  261 nohup gawk '$1~/^163\.35\.139\./||$15~/^163\.35\.139\./' 15-3-1100.t2 > ~/my\_15-3-1100.t2 &  262 nohup gawk '$1~/^163\.35\.139\./||$15~/^163\.35\.139\./' 15-3-1200.t2 > ~/my\_15-3-1200.t2 &  263 nohup gawk '$1~/^163\.35\.139\./||$15~/^163\.35\.139\./' 15-3-1300.t2 > ~/my\_15-3-1300.t2 &  264 nohup gawk '$1~/^163\.35\.139\./||$15~/^163\.35\.139\./' 15-3-1400.t2 > ~/my\_15-3-1400.t2 &  265 nohup gawk '$1~/^163\.35\.139\./||$15~/^163\.35\.139\./' 15-3-1500.t2 > ~/my\_15-3-1500.t2 &  266 nohup gawk '$1~/^163\.35\.139\./||$15~/^163\.35\.139\./' 15-3-1600.t2 > ~/my\_15-3-1600.t2 &  267 nohup gawk '$1~/^163\.35\.139\./||$15~/^163\.35\.139\./' 15-3-1700.t2 > ~/my\_15-3-1700.t2 &  268 nohup gawk '$1~/^163\.35\.139\./||$15~/^163\.35\.139\./' 15-3-1800.t2 > ~/my\_15-3-1800.t2 &  269 nohup gawk '$1~/^163\.35\.139\./||$15~/^163\.35\.139\./' 15-3-1800.t2.25 > ~/my\_15-3-1800.t2.25 & |

Short samples:

|  |
| --- |
|  |

## 2.1: Plot traffic volume

I chose 1.4 and 1.5:

### Visualising flow distribution by port numbers.

Here is my code:

|  |
| --- |
| import pandas as pd import matplotlib.pyplot as plt import glob  # 定义数据文件的目录 directory\_path = 'files/my\_files/'  # 列名定义 columns = ['src', 'dst', 'pro', 'ok', 'sport', 'dport', 'packets', 'bytes', 'flows', 'first', 'latest']  # 使用 glob 来获取目录下的所有文件路径 file\_paths = glob.glob(directory\_path + '\*.t2')  # 读取所有文件并合并为一个 DataFrame df\_list = [pd.read\_csv(file, sep='\t', header=None, names=columns) for file in file\_paths] df\_combined = pd.concat(df\_list, ignore\_index=True)  # 按 'dport' 聚合数据，获取计数并按降序排列 dport\_counts = df\_combined['dport'].value\_counts().sort\_values(ascending=False).head(20)  # 绘制条形图 plt.figure(figsize=(12, 8)) bars = plt.bar(dport\_counts.index.astype(str), dport\_counts.values, color='skyblue')  # 在条形上方添加文本注释 for bar in bars:  yval = bar.get\_height()  plt.text(bar.get\_x() + bar.get\_width() / 2, yval, int(yval), ha='center', va='bottom')  plt.title('Top 20 Flow Distribution by Destination Port (dport) Across All Files') plt.xlabel('Destination Port') plt.ylabel('Frequency') plt.xticks(rotation=45) plt.show() |

Result:

|  |
| --- |
|  |

### Plotting traffic volume as a function of time with at least two sufficiently different time scales.

Here is my code:

|  |
| --- |
| import pandas as pd import matplotlib.pyplot as plt import glob  # 定义数据文件的目录 directory\_path = 'files/my\_files/'  # 列名定义 columns = ['src', 'dst', 'pro', 'ok', 'sport', 'dport', 'packets', 'bytes', 'flows', 'first', 'latest']  # 使用 glob 来获取目录下的所有文件路径 file\_paths = glob.glob(directory\_path + '\*.t2')  # 读取所有文件并合并为一个 DataFrame df\_list = [pd.read\_csv(file, sep='\t', header=None, names=columns) for file in file\_paths] df\_combined = pd.concat([df for df in df\_list if not df.empty], ignore\_index=True)  # 确保 'first' 列是 datetime 类型 df\_combined['first'] = pd.to\_datetime(df\_combined['first'], unit='s')  # 按分钟和小时汇总流量 df\_combined.set\_index('first', inplace=True) traffic\_per\_minute = df\_combined.resample('T')['bytes'].sum() traffic\_per\_hour = df\_combined.resample('H')['bytes'].sum()   # 绘制流量随时间变化的图表 - 按分钟 plt.figure(figsize=(12, 6)) plt.plot(traffic\_per\_minute, label='Per Minute', color='blue') plt.xlabel('Time') plt.ylabel('Traffic Volume (bytes)') plt.title('Traffic Volume Per Minute') plt.xticks(rotation=45) plt.legend() plt.tight\_layout() plt.show()  # 绘制流量随时间变化的图表 - 按小时 plt.figure(figsize=(12, 6)) plt.plot(traffic\_per\_hour, label='Per Hour', color='green') plt.xlabel('Time') plt.ylabel('Traffic Volume (bytes)') plt.title('Traffic Volume Per Hour') plt.xticks(rotation=45) plt.legend() plt.tight\_layout() plt.show() |

Result:

|  |
| --- |
|  |
|  |

## 2.2: Per user data volume

Code:

|  |
| --- |
| import pandas as pd import matplotlib.pyplot as plt import glob import numpy as np  # 定义数据文件的目录 directory\_path = 'files/my\_files/'  # 列名定义 columns = ['src', 'dst', 'pro', 'ok', 'sport', 'dport', 'packets', 'bytes', 'flows', 'first', 'latest']  # 使用 glob 来获取目录下的所有文件路径 file\_paths = glob.glob(directory\_path + '\*.t2')  # 读取所有文件并合并为一个 DataFrame df\_list = [pd.read\_csv(file, sep='\t', header=None, names=columns) for file in file\_paths] df\_combined = pd.concat(df\_list, ignore\_index=True)  # 计算每个用户（源IP地址）的聚合数据量 user\_data\_volume = df\_combined.groupby('src')['bytes'].sum().sort\_values(ascending=False)  # 将用户分成三个等分 chunk\_size = int(np.ceil(len(user\_data\_volume) / 3)) user\_chunks = [user\_data\_volume[i:i + chunk\_size] for i in range(0, len(user\_data\_volume), chunk\_size)]  # 为每个部分绘制条形图 for i, chunk in enumerate(user\_chunks, start=1):  plt.figure(figsize=(12, 6))  chunk.plot(kind='bar', color='skyblue')  plt.title(f'User Aggregated Data Volume - Part {i}')  plt.xlabel('User IP Address')  plt.ylabel('Aggregated Data Volume (bytes)')  plt.xticks(rotation=90) # Rotate the x labels for better readability  plt.tight\_layout() # Adjust layout to fit IP addresses  plt.yscale('log')  plt.show() |

Result:

|  |
| --- |
|  |
|  |
|  |

Finally, here is a chart without all of the user IP addresses (axis Y is linear):

|  |
| --- |
|  |

Here is the code for the chart:

|  |
| --- |
| plt.figure(figsize=(12, 6)) user\_data\_volume.plot(kind='line', color='skyblue') plt.title(f'User Aggregated Data Volume') plt.ylabel('Aggregated Data Volume (bytes)') plt.show() |

## 2.3: Flow sampling

Firstly, I detected the number of files and the number of lines of my subnetwork sample with the code below:

|  |
| --- |
| import glob  # 定义数据文件的目录 directory\_path = 'files/my\_files/'  # 使用 glob 来获取目录下的所有文件路径 file\_paths = glob.glob(directory\_path + '\*.t2')  # 统计文件数量 file\_count = len(file\_paths)  # 统计所有文件的总行数 total\_lines = sum(1 for file in file\_paths for \_ in open(file))  print(f"files number：{file\_count}") print(f"lines number：{total\_lines}") |

Output:

|  |
| --- |
| files number：26  lines number：185279 |

So I should sample 185279/26=7126 lines from each flow data file (for both IPV4 and IPV6)

Then I programmed the shell script for sampling the data:

|  |
| --- |
| **#!/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): $file"   # Directly use specified file paths  temp\_ipv4="./temp\_ipv4"  temp\_ipv6="./temp\_ipv6"   # Clear or initialize these files  > "$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" |

It can be seen that I implemented a Python program to sample the data from each file. Here is the code sample\_script.py:

|  |
| --- |
| import sys  def sample\_lines(file\_path, line\_numbers):  with open(file\_path, 'r') as file:  for i, line in enumerate(file, start=1):  if i in line\_numbers:  yield line  def main():  # 第一个参数是文件路径，后续参数是行号  file\_path = sys.argv[1]  line\_numbers = set(map(int, sys.argv[2:]))   for line in sample\_lines(file\_path, line\_numbers):  print(line, end='')  if \_\_name\_\_ == "\_\_main\_\_":  main() |

With the shell script and the Python program, it is obvious that I avoid massive data being stored in memory and Heavy Disk Random Read/Write.

By implementing the shell script, I have two sampling data whose names are:

|  |
| --- |
| output\_sampled\_ipv4.txt  output\_sampled\_ipv6.txt |

### Visualizing port distribution of IPV4 and IPV6.

my code is:

|  |
| --- |
| import pandas as pd import matplotlib.pyplot as plt import glob  # 定义数据文件的目录 directory\_path = 'files/2\_3\_sample\_files/output\_sampled\_ipv4.txt'  # 列名定义 columns = ['src', 'dst', 'pro', 'ok', 'sport', 'dport', 'packets', 'bytes', 'flows', 'first', 'latest']   # 读取所有文件并合并为一个 DataFrame df = pd.read\_csv(directory\_path, sep='\t', header=None, names=columns)  # 按 'dport' 聚合数据，获取计数并按降序排列 dport\_counts = df['dport'].value\_counts().sort\_values(ascending=False).head(20)  # 绘制条形图 plt.figure(figsize=(12, 8)) bars = plt.bar(dport\_counts.index.astype(str), dport\_counts.values, color='skyblue')  # 在条形上方添加文本注释 for bar in bars:  yval = bar.get\_height()  plt.text(bar.get\_x() + bar.get\_width() / 2, yval, int(yval), ha='center', va='bottom')  plt.title('Top 20 Flow Distribution by Destination Port (dport) IPV4') plt.xlabel('Destination Port') plt.ylabel('Frequency') plt.xticks(rotation=45) plt.show() |

Result:

|  |
| --- |
|  |
|  |

For getting per-user data volume, here is my code:

|  |
| --- |
| import pandas as pd import matplotlib.pyplot as plt import numpy as np  # 定义数据文件的目录 directory\_path = 'files/2\_3\_sample\_files/output\_sampled\_ipv4.txt'  # 列名定义 columns = ['src', 'dst', 'pro', 'ok', 'sport', 'dport', 'packets', 'bytes', 'flows', 'first', 'latest']  # 读取文件并转换为 DataFrame df = pd.read\_csv(directory\_path, sep='\t', header=None, names=columns)  # 计算每个用户（源IP地址）的聚合数据量 user\_data\_volume = df.groupby('src')['bytes'].sum().sort\_values(ascending=False)  # 取前180名用户的数据量 top\_users = user\_data\_volume.head(180)  # 将其他用户的数据量汇总到 'Other Users' other\_users\_volume = user\_data\_volume.iloc[180:].sum() other\_users\_series = pd.Series([other\_users\_volume], index=['Other Users'])  # 合并 top\_users 和 other\_users\_series top\_users\_with\_others = pd.concat([top\_users, other\_users\_series])  # 计算每个图表的大小 chunk\_size = int(np.ceil(len(top\_users\_with\_others) / 4)) user\_chunks = [top\_users\_with\_others[i:i + chunk\_size] for i in range(0, len(top\_users\_with\_others), chunk\_size)]  # 为每个部分绘制条形图 for i, chunk in enumerate(user\_chunks, start=1):  plt.figure(figsize=(12, 6))  chunk.plot(kind='bar', color='skyblue')  plt.title(f'User Aggregated Data Volume - Part {i}')  plt.xlabel('User IP Address')  plt.ylabel('Aggregated Data Volume (bytes)')  plt.xticks(rotation=90)  plt.tight\_layout()  plt.yscale('log')  plt.show() |

There are so many different user ips, so I just put the top 180 ips into the plot.

Results for IPV4:

|  |
| --- |
|  |
|  |
|  |
|  |

Result of all the users:

|  |
| --- |
|  |

Results for IPV6:

|  |
| --- |
|  |
|  |
|  |
|  |

Result of all the users:

|  |
| --- |
|  |

### Plotting traffic volume as a function of time with at least two sufficiently different time scales.

Here is my code:

|  |
| --- |
| import pandas as pd import matplotlib.pyplot as plt import glob  # 定义数据文件的目录 directory\_path = 'files/2\_3\_sample\_files/output\_sampled\_ipv4.txt'  # 列名定义 columns = ['src', 'dst', 'pro', 'ok', 'sport', 'dport', 'packets', 'bytes', 'flows', 'first', 'latest']  df = pd.read\_csv(directory\_path, sep='\t', header=None, names=columns)  # 确保 'first' 列是 datetime 类型 df['first'] = pd.to\_datetime(df['first'], unit='s')  # 按分钟和小时汇总流量 df.set\_index('first', inplace=True) traffic\_per\_minute = df.resample('T')['bytes'].sum() traffic\_per\_hour = df.resample('H')['bytes'].sum()   # 绘制流量随时间变化的图表 - 按分钟 plt.figure(figsize=(12, 6)) plt.plot(traffic\_per\_minute, label='Per Minute', color='blue') plt.xlabel('Time') plt.ylabel('Traffic Volume (bytes)') plt.title('Traffic Volume Per Minute') plt.xticks(rotation=45) plt.legend() plt.tight\_layout() plt.show()  # 绘制流量随时间变化的图表 - 按小时 plt.figure(figsize=(12, 6)) plt.plot(traffic\_per\_hour, label='Per Hour', color='green') plt.xlabel('Time') plt.ylabel('Traffic Volume (bytes)') plt.title('Traffic Volume Per Hour') plt.xticks(rotation=45) plt.legend() plt.tight\_layout() plt.show() |

Result for IPV4:

|  |
| --- |
|  |
|  |

Result for IPV6:

|  |
| --- |
|  |
|  |

## Conclusion

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

#### For my subnetwork flow data:

|  |
| --- |
|  |
|  |

**Traffic Volume Over Time (Per Minute):**

The first chart displays traffic volume by the minute. It shows a fluctuating pattern with distinct peaks and troughs. These peaks might indicate periods of heavy data transfer or events causing sudden spikes in traffic, such as network attacks, backup processes, or streaming events. The troughs could represent times of low traffic, like late-night hours or off-peak times.

**Traffic Volume Over Time (Per Hour):**

The second chart, which displays traffic volume by the hour, offers a more macroscopic view. At this time scale, the fluctuations in traffic are smoother, but there are still noticeable increases in traffic volume. This could indicate daily usage patterns, like the start and end of work hours, or traffic changes due to specific events.

#### For the IPv4 sampling flow data:

**Traffic Volume Over Time (Per Minute):**

The traffic volume chart on a per-minute basis shows a number of significant spikes, which suggest moments of high activity or bursts of data transfer. These could be indicative of specific events or activities that cause a temporary increase in traffic, such as the start of a streaming event, large file transfers, or perhaps even automated backups.

Between the spikes, there are periods of relatively low traffic, which could correspond to the less active times of day or normal operational traffic.

**Traffic Volume Over Time (Per Hour):**

On an hourly basis, the traffic volume appears to be less volatile and more smoothed out compared to the per-minute chart. This is expected as short-term fluctuations are averaged out over each hour.

There is a clear and significant rise in traffic towards the end of the observed period. This could indicate a scheduled event that causes a consistent increase in traffic volume at certain times, or it might suggest a cumulative buildup of traffic leading to a peak.

The overall trend seems to show increasing traffic volume as time progresses, which could be part of a daily pattern or could indicate a specific trend or change in network usage.

From these observations, we can speculate that:

The network experiences variable traffic throughout the day, with periods of both high and low activity.

The spikes in the per-minute data might be missed when looking at the per-hour data due to the aggregation over a longer period, highlighting the importance of analyzing traffic at multiple time scales for a comprehensive understanding.

The end-of-period peak on the hourly chart suggests that there may be a predictable pattern of increased activity during specific hours.

#### For the IPv6 sampling flow data:

**Traffic Volume Over Time (Per Minute):**

At this finer granularity of time scale, the traffic exhibits significant fluctuations. There are several pronounced spikes, indicating substantial increases in traffic volume during those specific minutes. These peaks could be caused by large-scale data transfers such as file downloads, video streaming, or system backup operations.

Between these peaks, the traffic falls back to lower levels, demonstrating the baseline of regular traffic.

**Traffic Volume Over Time (Per Hour):**

When expanding the time scale to per hour, the volatility of traffic decreases, showing a smoother trend. Nonetheless, we can still observe some spikes which may represent periodic high-traffic events, such as the start and end of working hours.

The latter part of the chart shows a very significant peak, which could indicate the occurrence of a major event or a periodic activity, warranting further investigation to confirm the specifics.

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

#### For my subnetwork flow data:

**Port 80:** This port is commonly used by HTTP, which is the foundation of data communication for the World Wide Web. The high frequency suggests a lot of web traffic.

**Port 443:** This port is used by HTTPS, which is HTTP over TLS/SSL, providing secure web traffic. The prevalence of port 443 indicates a significant amount of secure web browsing or data transfer.

**Port 23:** Traditionally, this port is used for Telnet, which is a protocol used on the Internet or local area networks to provide a bidirectional interactive text-oriented communication facility using a virtual terminal connection. However, Telnet is insecure and has often been replaced by SSH on Port 22.

**Port 21:** This is the default port for FTP (File Transfer Protocol), used for the transfer of computer files between a client and server on a computer network.

**Port 25:** This port is primarily used for SMTP (Simple Mail Transfer Protocol), which is used for sending emails.

#### For the IPv4 sampling flow data:

**Port 53:** This port is typically used by the Domain Name System (DNS) for translating domain names into IP addresses, which is critical for internet browsing and accessing network resources.

**Port 80:** Investigated previously.

**Port 443:** Investigated previously.

**Port 123:** This port is used by the Network Time Protocol (NTP), which is used to synchronize the clocks of computers over a network.

**Port 25:** Investigated previously.

#### For the IPv6 sampling flow data:

**Port 53:** Investigated previously.

**Port 80:** Investigated previously.

**Port 443:** Investigated previously.

**Port 10053:** This port is often used by the Zabbix Agent. Zabbix is an enterprise-level software designed for real-time monitoring of millions of metrics collected from various servers, virtual machines, and network devices.

**Port 123:** This port is used by the Network Time Protocol (NTP), which is utilized to synchronize the clocks of computers to some time reference. NTP is essential for systems that rely on synchronized time settings, such as distributed systems, various security mechanisms, and logging services.

### 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?

#### For my subnetwork flow data:

**Variety of Users:**

The charts show a wide range of data usage among users. A few IP addresses have significantly higher data volumes compared to others, suggesting that these could be power users or servers that handle large amounts of data. This is typical for users who might be streaming high-definition content, participating in large file transfers, or operating as servers that provide content or services to other users.

**Potential Network Types:**

The presence of a few high-volume users alongside many lower-volume users might suggest a mixed-use network, possibly serving both residential and commercial purposes.

The "long-tail" distribution, where the majority of users consume a smaller amount of data, is typical of residential networks where most users engage in day-to-day internet activities like browsing, emailing, and streaming at a moderate resolution.

On the other hand, the presence of several high-volume users could indicate a business or educational network where certain nodes are designated for high-data tasks like hosting databases, large-scale computations, or providing media content.

**Network Speculation:**

A network with such a distribution could be an enterprise network with designated servers and workstations that vary in their data usage.

It could also be a university campus network where certain departments have higher data requirements for research and educational purposes, alongside dormitories and administrative offices with lower data usage.

Another possibility is a data center or cloud service provider, with some machines dedicated to intense computational tasks or serving as storage servers, while others handle lighter tasks.

**User Profiles:**

The top users are likely to be either servers or users engaged in data-intensive activities. They could be hosting web services, databases, or involved in scientific research that requires substantial data transfer.

The majority of users likely represent the typical internet consumer, engaging in web browsing, streaming, and general online activities.

#### For the IPv4 sampling flow data:

**User Types:**

The charts show a skewed distribution of data usage, with a small number of IP addresses consuming a large amount of data and many IP addresses consuming much less. This pattern is typical of networks with a mix of heavy users (like servers or power users who stream a lot of media, download large files, or engage in other data-intensive activities) and light users (like casual web browsers or small-scale consumers of data).

**Network Type:**

The presence of a few very high-volume users and many low-volume users suggests this could be a corporate or educational network, where certain nodes (like servers or research computers) need to handle large amounts of data.

The network might also be a residential ISP, with the variation in volume reflecting different household usage patterns.

Another possibility is a data center or cloud provider, where some machines are heavily used for tasks like hosting, computation, or storage, while others serve lighter roles.

**Network Speculation:**

A corporate network might have a few servers with very high traffic (for hosting company services, databases, etc.) and many workstations with moderate traffic.

An educational network might show high usage from research departments and lower usage from student accommodations and administrative offices.

A residential ISP network might show high usage from users who work from home, stream a lot of media, or have multiple users on the same network, with other users displaying more typical residential patterns.

#### For the IPv6 sampling flow data:

Same as the former IPv4 sampling flow data.

#### Is your subnetwork different from the larger population?

**User Types:**

My subnetwork: Shows a steep drop-off in data volume after the top few users, which could indicate a network with a few heavy users or servers and many light users.

Larger population: Seems to display a more gradual decline, which suggests a more uniform distribution of data usage among users, indicating a potentially more homogeneous user base.

**Network Type:**

My subnetwork: The pronounced skew in data volume towards the top IP addresses in my subnetwork might suggest a corporate or institutional network with specific nodes (like servers) handling significant data loads.

Larger population: The smoother gradient in the larger network might suggest 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.

**Network Speculation:**

My subnetwork: The presence of very high-volume IP addresses in my subnetwork 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.

Larger population: The larger network's more uniform usage suggests that it might not have as many specialized high-demand nodes, and might be structured around providing more consistent service across all nodes, like a smaller business, educational institution, or community network.

### Comparison of the above results with the result from data set PS2.

**Complexity and traffic magnitude:**

Private networks (PCs and routers) may exhibit very simple traffic patterns and low traffic volume because there is only a single user.

Public networks (internal company networks) will have more complex traffic patterns and higher traffic volumes because there are multiple users and services.

**Traffic consistency:**

Private networks may show consistent traffic patterns based on individual usage habits.

The shared network displays different traffic patterns throughout the day, reflecting the collective activity of all employees and company operations.

**Service diversity:**

Private networks may have access to a limited set of services, based on user needs.

Public networks may show access to a wide range of services, including enterprise resources, cloud services, and external websites.

# Task 3: Flow data

* Latency measurements:

AS1.d1: Name server1 with DNS => **anycastdns2.nic.td**

AS1.d2: Name server2 with DNS => **ns-td.afrinic.net**

AS1.d3: Name server3 with DNS => **pch.nic.td**

AS1.n1: Name server1 with ICMP => **anycastdns2.nic.td**

AS1.n2: Name server2 with ICMP => **ns-td.afrinic.net**

AS1.n3: Name server3 with ICMP => **pch.nic.td**

AS1.r1: Research server1 => **cbg-uk.ark.caida.org**

AS1.r2: Research server2 => **cjj-kr.ark.caida.org**

AS1.r3: Research server3=> **san2-us.ark.caida.org**

AS1.i1: Iperf server1 => **ok1.iperf.comnet-student.eu**

AS1.i2: Iperf server2 => **sgp1.iperf.comnet-student.eu**

* Throughput measurement:

AS2.i1: Iperf server1 => **ok1.iperf.comnet-student.eu**

AS2.i2: Iperf server2 => **sgp1.iperf.comnet-student.eu**

## 3.1 Latency data plots (AS1.x)

*The unit of all of the numbers in this chapter is* ***ms****.*

### Ping method(ICMP):

* AS1.n1: Name server1 with ICMP => **anycastdns2.nic.td**

**Excluding packet loss:**

|  |
| --- |
|  |

**Including packet loss:**

|  |
| --- |
|  |

* AS1.n2: Name server2 with ICMP => **ns-td.afrinic.net**

**Excluding packet loss:**

|  |
| --- |
|  |

**Including packet loss:**

|  |
| --- |
|  |

* AS1.n3: Name server3 with ICMP => **pch.nic.td**

**Excluding packet loss:**

|  |
| --- |
|  |

**Including packet loss:**

|  |
| --- |
|  |

* AS1.r1: Research server1 => **cbg-uk.ark.caida.org**

**Excluding packet loss:**

|  |
| --- |
|  |

**Including packet loss:**

|  |
| --- |
|  |

* AS1.r2: Research server2 => **cjj-kr.ark.caida.org**

**Excluding packet loss:**

|  |
| --- |
|  |

**Including packet loss:**

|  |
| --- |
|  |

* AS1.r3: Research server3=> **san2-us.ark.caida.org**

**Excluding packet loss:**

|  |
| --- |
|  |

**Including packet loss:**

|  |
| --- |
|  |

* AS1.i1: Iperf server1 => **ok1.iperf.comnet-student.eu**

**Excluding packet loss:**

|  |
| --- |
|  |

**Including packet loss:**

|  |
| --- |
|  |

* AS1.i2: Iperf server2 => **sgp1.iperf.comnet-student.eu**

**Excluding packet loss:**

|  |
| --- |
|  |

**Including packet loss:**

|  |
| --- |
|  |

* **Codes:**

**Code for Excluding packet loss:**

|  |
| --- |
| # 初始化一个列表来存储提取的时间值 time\_values = []  # 打开文件并读取每一行 with open('files/ping/sgp1.iperf.comnet-student.eu.txt', 'r') as file: # 请替换 'your\_file.txt' 为您的文件名  for line in file:  if 'time=' in line:  # 找到包含 'time=' 的行，并分割字符串  parts = line.split()  for part in parts:  if part.startswith('time='):  # 提取时间值，并移除 'ms'  time = part.split('=')[1].rstrip(' ms')  try:  # 将提取的时间值转换为浮点数并存储  time\_values.append(float(time))  except ValueError:  # 如果转换失败，则跳过该值  continue time\_values.sort() print(time\_values)  import matplotlib.pyplot as plt  # Assuming time\_values contains the latency measurements fig, ax = plt.subplots(figsize=(10, 6)) boxplot\_dict = ax.boxplot(time\_values, 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')  # Show the plot plt.show() |

**Code for Including packet loss:**

|  |
| --- |
| # 初始化一个列表来存储提取的时间值 time\_values = []  # 打开文件并读取每一行 with open('files/ping/sgp1.iperf.comnet-student.eu.txt', 'r') as file: # 请替换 'your\_file.txt' 为您的文件名  for line in file:  if 'time=' in line:  # 找到包含 'time=' 的行，并分割字符串  parts = line.split()  for part in parts:  if part.startswith('time='):  # 提取时间值，并移除 'ms'  time = part.split('=')[1].rstrip(' ms')  try:  if float(time) > 2000.0:  time\_values.append(float(2000.0))  else:  # 将提取的时间值转换为浮点数并存储  time\_values.append(float(time))  except ValueError:  # 如果转换失败，则跳过该值  continue  if 'packet loss' in line:  # 找到包含 'packet loss' 的行，并分割字符串来提取百分数  parts = line.split(',')  for part in parts:  if 'packet loss' in part:  # 提取百分比前的数字  percent\_loss = part.split('%')[0].strip()  try:  # 将提取的百分比转换为浮点数并累加到总和  for i in range(0,int(float(percent\_loss)/20.0)):  time\_values.append(float(2000.0))  # print(f"{i} outlier detected")  except ValueError:  # 如果转换失败，则忽略该值  continue   import matplotlib.pyplot as plt  # Assuming time\_values contains the latency measurements fig, ax = plt.subplots(figsize=(10, 6)) boxplot\_dict = ax.boxplot(time\_values, 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')  # Show the plot plt.show() |

### Dig method(DNS):

*Treating 'no servers could be reached' as packet loss.*

**Pre-processed by the command below:**

|  |
| --- |
| # awk '/Query time|Return|CURL/' anycastdns2.nic.td.txt > anycastdns2.nic.td.reloaded.txt # awk '/Query time|Return|CURL/' ns-td.afrinic.net.txt > ns-td.afrinic.net.reloaded.txt # awk '/Query time|Return|CURL/' pch.nic.td.txt > pch.nic.td.reloaded.txt |

* AS1.d1: Name server1 with DNS => **anycastdns2.nic.td**

**Excluding packet loss:**

|  |
| --- |
|  |

**Including packet loss:**

|  |
| --- |
|  |

* AS1.d2: Name server2 with DNS => **ns-td.afrinic.net**

**Excluding packet loss:**

|  |
| --- |
|  |

**Including packet loss(No packet loss):**

|  |
| --- |
|  |

* AS1.d3: Name server3 with DNS => **pch.nic.td**

**Excluding packet loss:**

|  |
| --- |
|  |

**Including packet loss(No packet loss):**

|  |
| --- |
|  |

* **Codes:**

**Code for Excluding packet loss:**

|  |
| --- |
| # awk '/Query time|Return|CURL/' anycastdns2.nic.td.txt > anycastdns2.nic.td.reloaded.txt # awk '/Query time|Return|CURL/' ns-td.afrinic.net.txt > ns-td.afrinic.net.reloaded.txt # awk '/Query time|Return|CURL/' pch.nic.td.txt > pch.nic.td.reloaded.txt  import re import numpy as np  query\_time\_lines = [] curl\_lines = []  with open('files/dns\_query/pch.nic.td.reloaded.txt', 'r') as file:  skip\_flag = 0  for line in file:  # 这里可以处理每一行的内容  striped\_line = line.strip()  # print(striped\_line)  if skip\_flag == 1:  skip\_flag = 0  continue  if re.search("Return", striped\_line):  skip\_flag = 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)  query\_time\_s = [] connect\_time\_s = []  for item in query\_time\_lines:  item\_split = item.split(" ")  query\_time = float(item\_split[3])  query\_time\_s.append(query\_time)  for item in curl\_lines:  item\_split = item.split(" ")  connect\_time = float(item\_split[3])  connect\_time\_s.append(connect\_time)  delay\_s = [] for i in range(0,len(query\_time\_lines)):  delay = query\_time\_s[i]/1000 + connect\_time\_s[i]  delay\_s.append(delay) delay\_s.sort() print(delay\_s)  import matplotlib.pyplot as plt  # Assuming time\_values contains the latency measurements fig, ax = plt.subplots(figsize=(10, 6)) boxplot\_dict = ax.boxplot(delay\_s, 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')  # Show the plot plt.show() |

**Code for Including packet loss:**

|  |
| --- |
| # awk '/Query time|Return|CURL/' anycastdns2.nic.td.txt > anycastdns2.nic.td.reloaded.txt # awk '/Query time|Return|CURL/' ns-td.afrinic.net.txt > ns-td.afrinic.net.reloaded.txt # awk '/Query time|Return|CURL/' pch.nic.td.txt > pch.nic.td.reloaded.txt  import re import numpy as np  query\_time\_lines = [] curl\_lines = [] delay\_s = []  with open('files/dns\_query/pch.nic.td.reloaded.txt', 'r') as file:  skip\_flag = 0  for line in file:  # 这里可以处理每一行的内容  striped\_line = line.strip()  # print(striped\_line)  if skip\_flag == 1:  skip\_flag = 0  continue  if re.search("Return", striped\_line):  skip\_flag = 1  delay\_s.append(2000.0)  continue  if re.search("Query", striped\_line):  query\_time\_lines.append(striped\_line)  if re.search("CURL", striped\_line):  curl\_lines.append(striped\_line)  query\_time\_s = [] connect\_time\_s = []  for item in query\_time\_lines:  item\_split = item.split(" ")  query\_time = float(item\_split[3])  query\_time\_s.append(query\_time)  for item in curl\_lines:  item\_split = item.split(" ")  connect\_time = float(item\_split[3])  connect\_time\_s.append(connect\_time)   for i in range(0,len(query\_time\_lines)):  delay = query\_time\_s[i]/1000 + connect\_time\_s[i]  delay\_s.append(delay) delay\_s.sort() print(delay\_s)  import matplotlib.pyplot as plt  # Assuming time\_values contains the latency measurements fig, ax = plt.subplots(figsize=(10, 6)) boxplot\_dict = ax.boxplot(delay\_s, 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')  # Show the plot plt.show() |

### TCP connect latency(IPERF3):

* AS1.i1: Iperf server1 => **ok1.iperf.comnet-student.eu**

|  |
| --- |
|  |

* AS1.i2: Iperf server2 => **sgp1.iperf.comnet-student.eu**

|  |
| --- |
|  |

### Provide PDF and CDF plots including all AS1.x delay distributions.

* AS1.d1: Name server1 with DNS => **anycastdns2.nic.td**

|  |
| --- |
|  |
|  |

* AS1.d2: Name server2 with DNS => **ns-td.afrinic.net**

|  |
| --- |
|  |
|  |

* AS1.d3: Name server3 with DNS => **pch.nic.td**

|  |
| --- |
|  |
|  |

* AS1.n1: Name server1 with ICMP => **anycastdns2.nic.td**

|  |
| --- |
| A graph of a line  Description automatically generated |
| A graph with a line  Description automatically generated |

* AS1.n2: Name server2 with ICMP => **ns-td.afrinic.net**

|  |
| --- |
| A graph of a line  Description automatically generated |
|  |

* AS1.n3: Name server3 with ICMP => **pch.nic.td**

|  |
| --- |
|  |
|  |

* AS1.r1: Research server1 => **cbg-uk.ark.caida.org**

|  |
| --- |
|  |
|  |

* AS1.r2: Research server2 => **cjj-kr.ark.caida.org**

|  |
| --- |
|  |
|  |

* AS1.r3: Research server3=> **san2-us.ark.caida.org**

|  |
| --- |
|  |
|  |

* AS1.i1: Iperf server1 => **ok1.iperf.comnet-student.eu**

**Ping:**

|  |
| --- |
|  |
|  |

**IPERF3:**

|  |
| --- |
|  |
|  |

* AS1.i2: Iperf server2 => **sgp1.iperf.comnet-student.eu**

**Ping:**

|  |
| --- |
|  |
|  |

**IPERF3:**

|  |
| --- |
|  |
|  |

* **Code**

**(Replace XXX with the delay data list)**

|  |
| --- |
| import matplotlib.pyplot as plt import numpy as np from scipy.stats import gaussian\_kde  # 计算概率密度函数 XXX = gaussian\_kde(delay\_s) xs = np.linspace(min(XXX), max(XXX), 200) density.covariance\_factor = lambda: .25 density.\_compute\_covariance()  # 绘制PDF plt.figure(figsize=(10, 6)) plt.plot(xs, density(xs), label='PDF') plt.title('Probability Density Function of Latency') plt.xlabel('Latency (ms)') plt.ylabel('Density') plt.legend() plt.show()  # 计算CDF sorted\_data = np.sort(delay\_s) yvals = np.arange(len(sorted\_data)) / float(len(sorted\_data) - 1)  # 绘制CDF plt.figure(figsize=(10, 6)) plt.plot(sorted\_data, yvals, label='CDF') plt.title('Cumulative Distribution Function of Latency') plt.xlabel('Latency (ms)') plt.ylabel('Cumulative Probability') plt.legend() plt.show() |