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: Visualise 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 443:

Application: HTTPS (Hypertext Transfer Protocol Secure)

Characteristics: Encrypts data transmissions to ensure secure communication over the internet, used for secure transactions like online banking and shopping.

The following ports, 50060, 53863, 59054, and 64844, are not standard ports and could be used for a variety of custom services. In typical network configurations, ports above 49152 are considered dynamic (ephemeral) ports used by client applications to communicate with servers. Without specific context or network traffic analysis, it is not possible to accurately determine which applications these ports correspond to. They could be used for internal services, private applications, or dynamic allocation by various protocols.

Port 50060:

Possible Application: Custom or proprietary service

Characteristics: Often high-numbered ports are used for internal network applications, could be related to a specific software's communication protocol.

Port 53863:

Possible Application: Custom or proprietary service

Characteristics: This port might be used by specific server applications, perhaps for a database, a gaming server, or another service that isn't widely recognized.

Port 59054:

Possible Application: Custom or proprietary service

Characteristics: It could be used for services such as remote access tools, specialized communication applications, or other network services that require a specific port allocation.

Port 64844:

Possible Application: Custom or proprietary service

Characteristics: This port may be dedicated to an application with a specific function within a private network or used for a temporary service endpoint.

**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 Visualise 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:

|  |
| --- |
|  |

* **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:

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

* **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 |

For 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:

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

Results for IPV66:

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