ELEC-E7130 Internet Traffic Measurements and Analysis

Assignment 1. Basic programming and processing data

Name: Xingji Chen

Student ID: 101659554

E-mail: xingji.chen@aalto.fi

Aalto University
School of Electrical
Engineering

Task 1: Programming tools

In the first task, answer the following questions:

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

   1) The awk command is a versatile text processing tool, mainly used for line-by-line scanning and processing of text or data files. It is particularly useful for extracting and manipulating data for specific fields or columns.
   2) The principle of the Awk command is to read the text line by line, separated by default by a space or tab key, save the separated fields into built-in variables, and execute editing commands according to patterns or conditions. The result can be printed with the print function. In the process of using awk commands, we can use the logical operators "&&" for "with", "||" for "or ", "!" means "not"; we can also perform simple mathematical operations, such as +, -, *, /, %, ^, respectively, add, subtract, multiply, divide, take the balance and multiply.
   3) Examples:
      - Print the first line:

        ```
        awk '{print $3, $7, $10, $17, $21, $24}' log_tcp_complete | head -n 1
        ```

      - Calculate the average of the columns 3:

        ```
        awk 'NR > 1 { sum += $7; count++ } END { if (count > 0) print "Average of Column 3:", sum / count }' log_tcp_complete
        ```

      - Calculate the maximum of the column 3:

        ```
        awk 'NR > 1 {
            if ($3 > max3) max3 = $3;
        }
        END {
            printf("Maximum of Column 3: %s\n", max3);
        }' log_tcp_complete
        ```

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

   HINT: Consider in terms of programming experience, applications, plotting, or more.

1) Similarities:
   - Both Python and R have a rich ecosystem of data analysis, statistical and machine learning libraries and packages. For example, Python has libraries such as NumPy, pandas and Matplotlib, while R has packages such as dplyr, ggplot2 and caret.
   - Python and R have powerful data visualization capabilities.
   - Python and R have strong data statistics and analysis capabilities and can perform most statistical tasks efficiently.
2) Differences:
   - Syntax: Python has a generalized and more readable syntax, making it more accessible to beginners and more versatile for a variety of programming tasks. However, R's syntax is very specialized and geared towards statistical analysis, which may be less intuitive for those unfamiliar with it.
   - Programming experience: Python is generally more applicable to a wider range of programming scenarios such as web development and scripting. R, while versatile within the data science domain, is less applicable to non-data science tasks.
   - Community and packages: Python has a larger and more diverse community, which means a wider range of available packages and resources. r's community is more specialized.
3) Situations Python is more suitable:
   - Web development: Python is better suited for web development using frameworks such as Django or Flask.
   - Automation and scripting: Python is the language of choice for automating tasks such as repetitive processes, managing files or creating system scripts.
   - Putting Machine Learning Models into Production: Python is often chosen when putting machine learning models into production because of its strong support for model deployment and integration with web services.
4) Situations Python is more suitable:
   - Data analysis: R is a natural choice for in-depth statistical analysis, hypothesis testing and data exploration. Examples include detailed statistical analysis of clinical trial data.
   - Data Visualization: When highly customizable and publication-quality data visualization is required, R's ggplot2 package provides fine control over plotting aesthetics.
   - Statistical data research: Researchers and academics in the field of statistics often prefer to use R because of its statistical rigor and rich statistical package.

Aalto University
School of Electrical
Engineering

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

1) Python
   - pandas:
     Pandas provides data structures like DataFrame and Series to make working with structured data easy. It also provides powerful data filtering, grouping and merging features.
   - NumPy:
     NumPy stands for "NumericalPython" and is the basic library for numerical computation in Python. It supports large, multi-dimensional arrays and matrices, as well as a set of mathematical functions for manipulating these arrays.
   - scikit-learn:
     Scikit-learn is a comprehensive machine learning library for Python. It provides a wide range of machine learning algorithms for a variety of tasks such as classification, regression and clustering. Scikit-learn also provides tools for model selection, evaluation, and preprocessing.

2) R
   - dplyr:
     Dplyr is a commonly used data manipulation library in R. It provides a set of intuitive functions for data manipulation tasks such as filtering, sorting, selecting specific columns, grouping data, and creating summary statistics.
   - ggplot2:
     Ggplot2 is a data visualization package for R. It follows the philosophy of "graphical syntax" and allows users to create complex and custom data visualizations using a relatively simple and consistent syntax.
   - caret:
     Caret (classification and regression training) is a R package designed to simplify the process of training and evaluating machine learning models. It provides a unified interface to a variety of modeling techniques, including cross-validation, hyperparameter tuning, and feature selection.

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

1) Latency
   - Latency is the amount of time or delay required to transmit a single unit of data (e.g., a packet or request) from source to destination. It is a measure of the responsiveness or speed of a system.
   - Measurement:
     - Ping (Round Trip Latency): Ping is a network utility that sends a small packet of data to a destination host and measures the time it takes for the packet to travel from the source to the destination host and back again. It provides round-trip latency measurements, lower ping times indicate lower latency and better responsiveness.
     - Application-specific monitoring: Many applications and services have built-in mechanisms for measuring latency. For example, a web application may log the time it takes to process a request and send a response. Monitoring tools, such as NewRelic or Prometheus, can capture and display application-specific latency metrics.

2) Throughput
   - Throughput measures the rate at which data can be successfully transferred between two points in a system or network at a given time. It quantifies the ability or bandwidth of a system to process data.
   - Measurement
     - Network Speed Test: Network speed tests, such as Ookla's Speedtest.net, measure the throughput of an Internet connection by downloading and uploading data between a device and a remote server.
     - Iperf (Network Throughput): Iperf is a command line tool that measures network throughput between two systems. It generates network traffic and measures the data transfer rate in bits per second (bps) or bytes per second (Bps) at specific time intervals.

Aalto University
School of Electrical
Engineering

Task 2: Processing CSV data using awk

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

   1) Use head or tail to Preview Rows

      Before using awk, you can use the head or tail command to preview the beginning or end of the file to get an idea of its structure and content.

      For example:

      To view the first 10 rows of a CSV file:

      ```
      head -n 10 file.csv
      ```

      To view the last 10 rows of a CSV file:

      ```
      tail -n 10 file.csv
      ```

   2) Use awk for Data Sampling

      For example, to extract a random sample of 1000 rows from the file:

      ```
      awk 'BEGIN {srand()} {if (rand() <= 0.01) print}' file.csv > sample.csv
      ```

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

   Command:

   - chenx16@cavalieri:/work/courses/unix/T/ELEC/E7130/general/trace/tstat/2017_04_11_18_00.out$ awk 'NR==1 {print $3, $7, $10, $17, $21, $24}' log_tcp_complete

   Output:

   - c_pkts_all:3 c_bytes_uniq:7 c_pkts_retx:10 s_pkts_all:17 s_bytes_uniq:21 s_pkts_retx:24

   Description:

   - NR==1:This is a condition that indicates that only the first line in the file is processed.
   - {print$3,$7,$10,$17,$21,$24}:It prints the contents of columns 3, 7, 10, 17, 21, and 24 in line 1.
   - log_tcp_complete: This is the name of the input file to be processed, which contains information about the data to be extracted.

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

Aalto University
School of Electrical
Engineering

Command:

- chenx16@cavalieri:/work/courses/unix/T/ELEC/E7130/general/trace/tstat/2017_04_11_18_00.out$ awk 'NR > 1 { sum += $3; count++ } END { if (count > 0) print "Average of Column 3:", sum / count }' log_tcp_complete
- chenx16@cavalieri:/work/courses/unix/T/ELEC/E7130/general/trace/tstat/2017_04_11_18_00.out$ awk 'NR > 1 { sum += $7; count++ } END { if (count > 0) print "Average of Column 7:", sum / count }' log_tcp_complete
- chenx16@cavalieri:/work/courses/unix/T/ELEC/E7130/general/trace/tstat/2017_04_11_18_00.out$ awk 'NR > 1 { sum += $10; count++ } END { if (count > 0) print "Average of Column 10:", sum / count }' log_tcp_complete
- chenx16@cavalieri:/work/courses/unix/T/ELEC/E7130/general/trace/tstat/2017_04_11_18_00.out$ awk 'NR > 1 { sum += $17; count++ } END { if (count > 0) print "Average of Column 17:", sum / count }' log_tcp_complete
- chenx16@cavalieri:/work/courses/unix/T/ELEC/E7130/general/trace/tstat/2017_04_11_18_00.out$ awk 'NR > 1 { sum += $21; count++ } END { if (count > 0) print "Average of Column 21:", sum / count }' log_tcp_complete
- chenx16@cavalieri:/work/courses/unix/T/ELEC/E7130/general/trace/tstat/2017_04_11_18_00.out$ awk 'NR > 1 { sum += $24; count++ } END { if (count > 0) print "Average of Column 24:", sum / count }' log_tcp_complete

Output:

- Average of Column 3: 93.5731
- Average of Column 7: 36741.2
- Average of Column 10: 0.491827
- Average of Column 17: 178.716
- Average of Column 21: 214088
- Average of Column 24: 2.2859

Description:

- NR>1: This is a condition indicating that only lines in the file with a line number greater than 1 will be processed, the first line (the header line) will be skipped.
- {sum+=$3;count++}:This part is executed when the condition is satisfied. It accumulates the value of column 3 (sum+=$3) and adds a counter (count++) to keep track of the number of rows processed.
- END: This is a special awk block that is executed after processing the file.

- if(count>0): This is a condition that checks to see if there is at least one row that meets the condition (with a row number greater than 1).
- print" AverageofColumn3:", sum/count: If there is a row that matches the condition, it prints the average of column 3. The average is calculated as sum/count.

4. Calculate the percentage of records where column10/column7 exceeds a)0.01, b) 0.10, c) 0.20 (in other words, the value in column 10 is divided by the value in column 7 for each line and the result must exceed the values indicated)

Command:

```
chenx16@cavalieri:/work/courses/unix/T/ELEC/E7130/general/trace/tstat/2017_04_11_
18_00.out$ awk 'NR > 1 {
    if ($7 != 0) {
        ratio = $10 / $7;
        if (ratio > 0.01) count01++;
        if (ratio > 0.10) count10++;
        if (ratio > 0.20) count20++;
    }
    total++;
}
END {
    if (total > 0) {
        percent01 = (count01 / total) * 100;
        percent10 = (count10 / total) * 100;
        percent20 = (count20 / total) * 100;
        printf("Percentage exceeding 0.01: %.2f%%\n", percent01);
        printf("Percentage exceeding 0.10: %.2f%%\n", percent10);
        printf("Percentage exceeding 0.20: %.2f%%\n", percent20);
    }
}' log_tcp_complete
```

Output:

```
Percentage exceeding 0.01: 1.71%
Percentage exceeding 0.10: 0.50%
Percentage exceeding 0.20: 0.30%
```

Description:

- 'NR > 1 { ... }': This is the script part of awk, containing the following content, which will be executed on lines with line numbers greater than 1.
- if ($7 != 0) { ... }: This condition checks whether the 7th column is not equal to 0 to avoid dividing by 0.
- ratio = $10 / $7;: Calculates the ratio of column 10 divided by column 7.
- if (ratio > 0.01) count01++; : If the ratio is greater than 0.01, increase the count01 counter.

  if (ratio > 0.10) count10++;: If the ratio is greater than 0.10, increase the count10 counter.

  if (ratio > 0.20) count20++;: If the ratio is greater than 0.20, increase the count20 counter.
- total++;: Increase the total counter to count the number of rows processed.
- END { ... }': This is a special block of awk, which is executed after processing the entire file and contains the following content:
- if (total > 0) { ... }: This condition checks whether at least one row meets the condition (row number is greater than 1).
- percent01 = (count01 / total) * 100;: Calculate the number of rows with a ratio greater than 0.01 as a percentage of the total number of rows.

  percent10 = (count10 / total) * 100;: Calculate the number of rows with a ratio greater than 0.10 as a percentage of the total number of rows.

  percent20 = (count20 / total) * 100;: Calculate the number of rows with a ratio greater than 0.20 as a percentage of the total number of rows.
- printf("Percentage exceeding 0.01: %.2f%%\n", percent01);: Use printf to print the percentage in a formatted manner, retaining two decimal places.

  printf("Percentage exceeding 0.10: %.2f%%\n", percent10);: Print the percentage with a ratio greater than 0.10.

  printf("Percentage exceeding 0.20: %.2f%%\n", percent20);: Print the percentage with a ratio greater than 0.20.

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

Command:

chenx16@cavalieri:/work/courses/unix/T/ELEC/E7130/general/trace/tstat/2017_04_11_18_00.out$ awk 'NR > 1 {

```
        if ($3 > max3) max3 = $3;
        if ($9 > max9) max9 = $9;
        if ($17 > max17) max17 = $17;
        if ($23 > max23) max23 = $23;
        if ($31 > max31) max31 = $31;
    }
    END {
        printf("Maximum of Column 3: %s\n", max3);
        printf("Maximum of Column 9: %s\n", max9);
        printf("Maximum of Column 17: %s\n", max17);
        printf("Maximum of Column 23: %s\n", max23);
        printf("Maximum of Column 31: %s\n", max31);
    }' log_tcp_complete
```

Output:

```
Maximum of Column 3: 1700014
Maximum of Column 9: 313220528
Maximum of Column 17: 2791706
Maximum of Column 23: 3835783508
Maximum of Column 31: 72901107.834000
```

Description:

- 'NR > 1 { ... }': This is the script part of awk, containing the following content, which will be executed on lines with line numbers greater than 1.
- if ($3 > max3) max3 = $3;: If the value of column 3 is greater than the current value of max3, update max3 to the value of column 3.
  if ($9 > max9) max9 = $9;: If the value of column 9 is greater than the current value of max9, update max9 to the value of column 9.
  if ($17 > max17) max17 = $17;: If the value of column 17 is greater than the current value of max17, update max17 to the value of column 17.
  if ($23 > max23) max23 = $23;: If the value of column 23 is greater than the current value of max23, update max23 to the value of column 23.
  if ($31 > max31) max31 = $31;: If the value of column 31 is greater than the current value of max31, update max31 to the value of column 31.
- END { ... }': This is a special block of awk, which is executed after processing the

entire file and contains the following content:

- printf("Maximum of Column 3: %s\n", max3);: Use printf to print the maximum value of column 3 in a formatted manner.

  printf("Maximum of Column 9: %s\n", max9);: Same as above, print the maximum value of column 9.

  printf("Maximum of Column 17: %s\n", max17);: Same as above, print the maximum value of column 17.

  printf("Maximum of Column 23: %s\n", max23);: Same as above, print the maximum value of column 23.

  printf("Maximum of Column 31: %s\n", max31);: Same as above, print the maximum value of column 31.

Aalto University
School of Electrical
Engineering

Task 3: Processing throughput and latency data

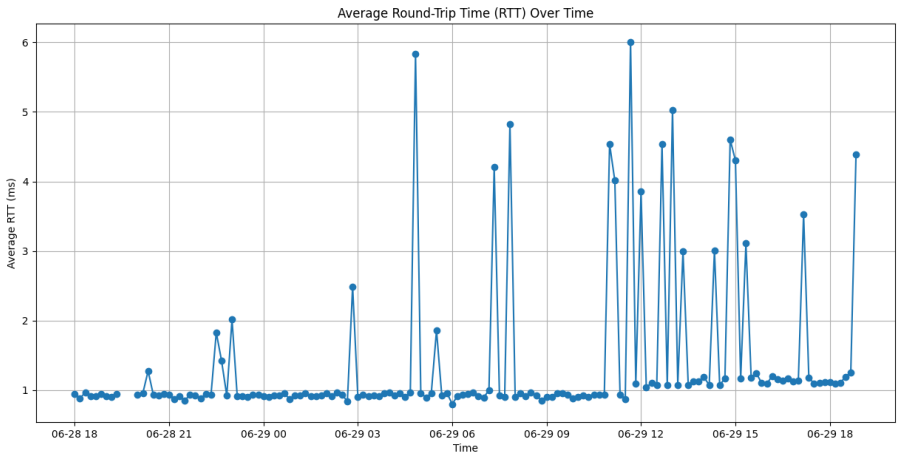
3.1 Latency data using ping

1. Load the CSV into a DataFrame and change the timestamp to date format.

```
# Load ping data from CSV and convert timestamps
ping = pd.read_csv('ping_data.csv')
ping['Timestamp'] = pd.to_datetime(ping['Timestamp'], unit='s')
```

2. Plot the average RTT over a time series from the provided CSV file as a first approach to the data and analysis of this.

```
# Create a figure for plotting
plt.figure(figsize=(15, 7))

# Plot Average RTT Over Time
plt.plot(ping['Timestamp'], ping['Avg RTT (ms)'], marker='o')
plt.title('Average Round-Trip Time (RTT) Over Time')
plt.xlabel('Time')
plt.ylabel('Average RTT (ms)')
plt.grid(True)

# Display the plot
plt.show()
```

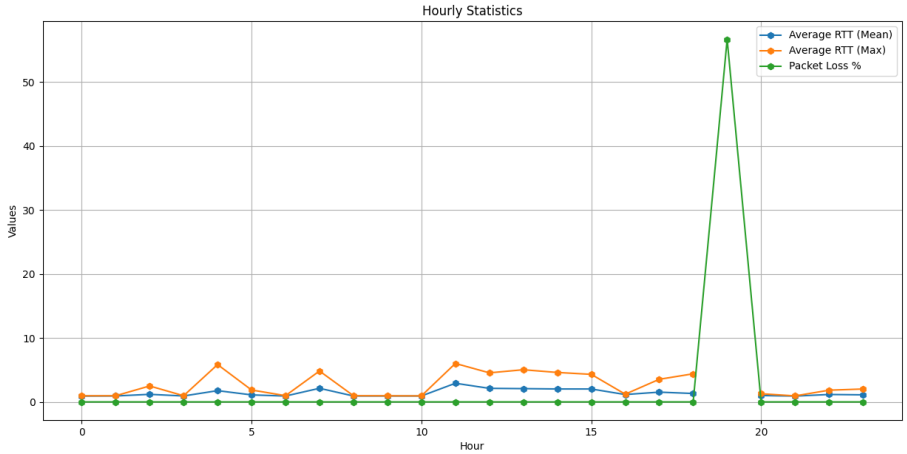


3. Generate another CSV file or dataframe with the values calculated of the average of successful RTTs, the maximum of RTTs, and the percentage of packet loss every hour.

```
# Create a new column 'Hour' by extracting the hour part
ping['Hour'] = ping['Timestamp'].dt.hour
# Group the data by hour and calculate statistics
hourly_stats = ping.groupby('Hour').agg({
    'Avg RTT (ms)': ['mean', 'max'],  # Mean and maximum of
average RTT
    'Transmitted packets': 'sum',    # Total transmitted packets
    'Successful packets': 'sum'      # Total successful packets
})
# Calculate the packet loss percentage for each hour
hourly_stats['Packet Loss %'] = ((hourly_stats['Transmitted
packets'] - hourly_stats['Successful packets']) /
hourly_stats['Transmitted packets']) * 100
# Rename column headers for clarity
hourly_stats.columns = ['Avg RTT (ms) Mean', 'Avg RTT (ms) Max',
'Transmitted packets', 'Successful packets', 'Packet Loss %']
# Reset the index to convert 'Hour' back from an index to a
DataFrame column
hourly_stats.reset_index(inplace=True)
```

4. Plot another time series to observe the behavior of the RTTs (average and maximum) according to the measurements calculated in each hour.

```
plt.figure(figsize=(15, 7))
plt.plot(hourly_stats['Hour'], hourly_stats['Avg RTT (ms) Mean'],
marker='h', label='Average RTT (Mean)')
plt.plot(hourly_stats['Hour'], hourly_stats['Avg RTT (ms) Max'],
marker='h', label='Average RTT (Max)')
plt.plot(hourly_stats['Hour'], hourly_stats['Packet Loss %'],
marker='h', label='Packet Loss %')
plt.title('Hourly Statistics')
plt.xlabel('Hour')
plt.ylabel('Values')
plt.legend()
plt.grid(True)
plt.show()
```



5. Can you make any conclusions of stability and latency based on data?

- From the figure average RTT, it can be seen that from 06-28 18 to the early morning of 06-29, the average RTT stays at a very low level of about 1ms, which indicates that the network is relatively stable during this period of time. When close to noon, the average RTT has a significant increase and accompanied by great fluctuations, which indicates that the quality of the network decreases in this time period.
- These data can also be represented in the second figure, the average and maximum RTT is basically stable in the first 10 hours. And there is a significant increase in the RTT from 11 to 18 hours. During these hours, the packet loss rate stays within a very low level, while in the 19th hour, the network latency suddenly increases and accompanied by fluctuations, and the packet loss rate instantly rises to a very high value. After that it returned to stability.
- So, it can be concluded that there is a strong relationship between the stability of the network and the RTT, which reflects the latency of the network. When the latency is low and stable, the network remains stable. However, when the delay increases and is accompanied by drastic changes, the stability of the network also decreases and the packet loss rate increases.

Aalto University
School of Electrical
Engineering

3.2 Throughput data using iperf3

1. Load the CSV into a DataFrame and change the timestamp to date format.

```
iperf = pd.read_csv('iperf_data.csv')
```

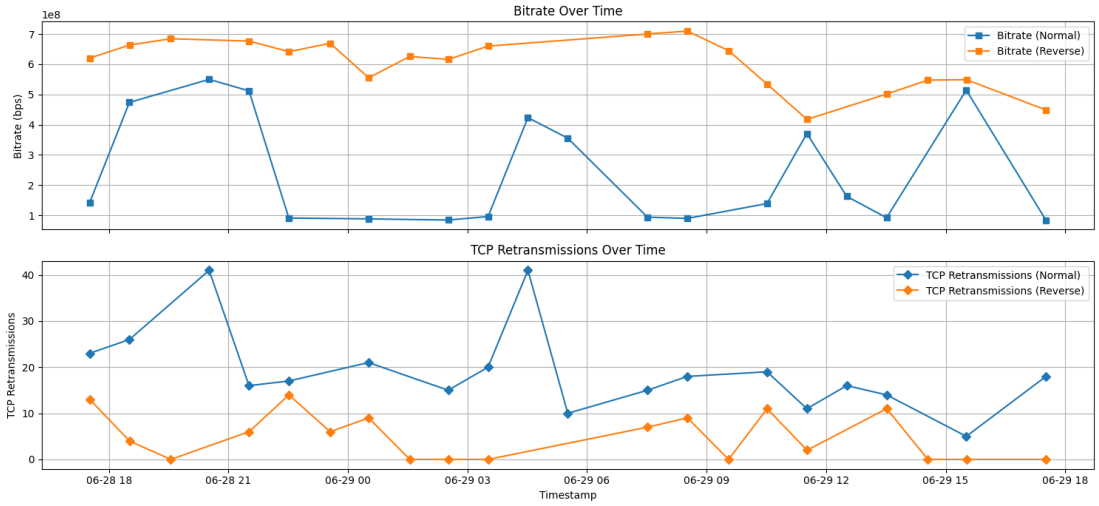
2. Remove the rows with values “-1” and classify the mode sent based on the column ‘Mode’, where ‘0’ is normal (client-server) and ‘1’ refers to reverse (server-client)

```
iperf = iperf[iperf['Sent bitrate (bps)'] != -1]  # Remove rows
with -1 bitrate
iperf['Timestamp'] = pd.to_datetime(iperf['Timestamp'], unit='s')

# Classify the mode based on the 'Mode' column:
iperf['Direction'] = iperf['Mode'].apply(lambda x: 'Normal' if x
== 0 else 'Reverse')
```

3. Plot comparing bitrate and TCP retransmissions over a time series (one for normal direction and one for reverse)

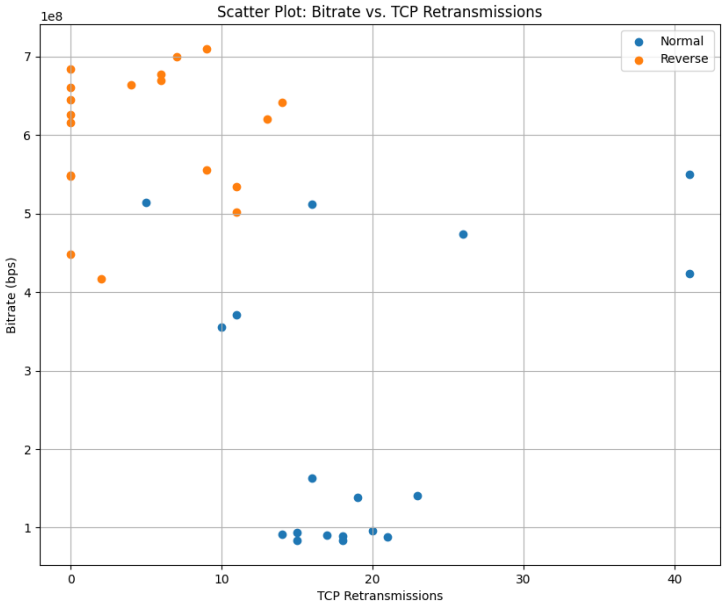
```
# Create two subplots for bitrate and TCP retransmissions over
time.
fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(15, 7),
sharex=True)

for direction in ['Normal', 'Reverse']:
    direction_df = iperf[iperf['Direction'] == direction]
    ax1.plot(direction_df['Timestamp'], direction_df['Sent bitrate
(bps)'], label=f'Bitrate ({direction})', marker='s')
    ax2.plot(direction_df['Timestamp'],
direction_df['Retransmissions'], label=f'TCP Retransmissions
({direction})', marker='D')

# Set titles, labels, and legends for the subplots.
ax1.set_title('Bitrate Over Time')
ax1.set_ylabel('Bitrate (bps)')
ax1.grid(True)
ax1.legend()
```

```
ax2.set_title('TCP Retransmissions Over Time')
ax2.set_xlabel('Timestamp')
ax2.set_ylabel('TCP Retransmissions')
ax2.grid(True)
ax2.legend()
# Show the subplots.
plt.tight_layout()
plt.show()
```



4. Create a scatter plot to observe the relationship between TCP retransmissions and bitrate (one for normal direction and one for reverse)

```
# Create a scatter plot to observe the relationship between TCP
retransmissions and bitrate for both directions.
plt.figure(figsize=(10, 8))
for direction in ['Normal', 'Reverse']:
    direction_df = iperf[iperf['Direction'] == direction]
    plt.scatter(direction_df['Retransmissions'],
direction_df['Sent bitrate (bps)'], label=direction)
plt.title('Scatter Plot: Bitrate vs. TCP Retransmissions')
plt.xlabel('TCP Retransmissions')
plt.ylabel('Bitrate (bps)')
plt.legend()
plt.grid(True)
plt.show()
```



5. Can you make any conclusions of stability based on data and relationship between bitrate and TCP retransmissions?

- Observe the bitrate and TCP retransmissions over a time series figure, especially for data in the forward and reverse directions. If the TCP retransmissions remain low, with no noticeable spikes or constant growth trends, then it can be concluded that the network is stable most of the time. If there are frequent spikes in TCP retransmissions or there is a trend of sustained growth, this may indicate that there are problems with the network. These problems may include network congestion, packet loss, and latency, all of which can affect stability.
- If the scatter plot shows a positive correlation between bit rate and TCP retransmissions, for example, as bit rate increases, TCP retransmissions also increase, this may indicate that the network has insufficient capacity, resulting in congestion and retransmissions. Conversely, if there is no significant correlation between bit rate and TCP retransmissions, or if there is a negative correlation, then the network is more stable.