# PART 1:

1) Factors that could continue to the slow transfer would be:

**-** Congestion Control: If the network is congested, TCP will slow down the transmission rate to avoid further packet loss.

-Packet Loss & Retransmissions: If packets are being lost, TCP will retransmit them, which slows down the transfer.

-Window Size Limitations: TCP uses a sliding window to control how much data can be sent before receiving an acknowledgment. If the window size is too small, the sender will have to wait frequently.

-Bandwidth and Latency: If the network has high latency, the round-trip time (RTT) for acknowledgments will be high, leading to slower throughput.

-Receiver's Processing Power: If the receiver cannot process packets quickly enough, it will advertise a smaller window size, slowing down the sender.

and to troubleshoot it we could:

-Check if there is high packet loss using tools like Wireshark.

-Analyze congestion window behavior to see if TCP is slowing down due to network congestion.

-Check the advertised window size to see if the receiver is limiting the speed.

-Compare performance over different networks (Wi-Fi vs. Ethernet) to rule out network-specific issues.

2) If the sender has significantly higher processing power than the receiver then:

-The sender might be ready to send more data, but if the receiver's buffer is filling up, it will advertise a smaller window size.

-This slows down the transmission rate since the sender must wait for the receiver's acknowledgment and buffer availability.

-If the receiver's buffer is always full, it could lead to delays and even dropped packets if the sender doesn't react quickly.

-In extreme cases, it may cause underutilization of network resources since the sender isn't transmitting at full capacity.

3) Path choice can affect network performance in multiple ways such as:
-Latency: Shorter paths generally reduce delay, improving real-time applications.
-Throughput: Paths with higher bandwidth allow more data to be transmitted efficiently.
-Congestion: A heavily used path can experience packet loss and retransmissions, reducing overall performance.
- Reliability: Unstable paths with frequent outages can lead to disrupted connections and higher error rates.
- Jitter**:** Inconsistent delays across different paths can negatively impact time-sensitive applications.

Factors to Consider in Routing Decisions would be:
- Network Congestion: Choosing less congested routes can improve stability and reduce packet drops.
- Available Bandwidth: High-bandwidth routes should be prioritized for large data transfers.
- Routing Protocols: Protocols like OSPF and BGP dynamically adjust paths based on real-time conditions.
- Redundancy & Failover: Multipath routing can ensure backup routes in case of failures.
- Quality of Service (QoS): Some paths may be prioritized based on application requirements.
- Cost & Policies: Some networks might prioritize cheaper or preferred routes based on agreements or operational costs.

4) MPTCP Improves Network Performance by:
- Increased Throughput: Instead of being limited to one path, MPTCP can split traffic across different network interfaces (for example Wi-Fi and 4G).
- Better Reliability: If one path fails, the connection remains active through another.

- Load Balancing: It distributes traffic across multiple links, reducing congestion on any single route.

5) Potential causes of packet loss at the Network layer are:
- Congestion: If a router’s buffer is full, excess packets are dropped (tail drop).
- Routing Issues: Suboptimal routing, loops, or frequent route changes can cause delays and packet loss.
- Hardware Failures**:** Faulty network interfaces, overheating, or damaged cables can lead to dropped packets.
- MTU (Maximum Transmission Unit) Issues: If packet sizes exceed the MTU and fragmentation isn't handled correctly, packets may be lost.
- Interference (Wireless Networks): Signal degradation, interference from other devices, or weak Wi-Fi signals can cause packet loss.

Potential causes of packet loss at the Transport layer are:
- Retransmissions Due to Loss: If packets are lost, TCP will retransmit them, reducing efficiency.
- Misconfigured TCP Window Size: A small receive window can cause unnecessary delays or dropped packets.
- TCP Congestion Control: If TCP detects packet loss, it reduces the transmission rate, which can affect performance.
- Packet Reordering: Out-of-order packets may be discarded, leading to retransmissions and additional delays.

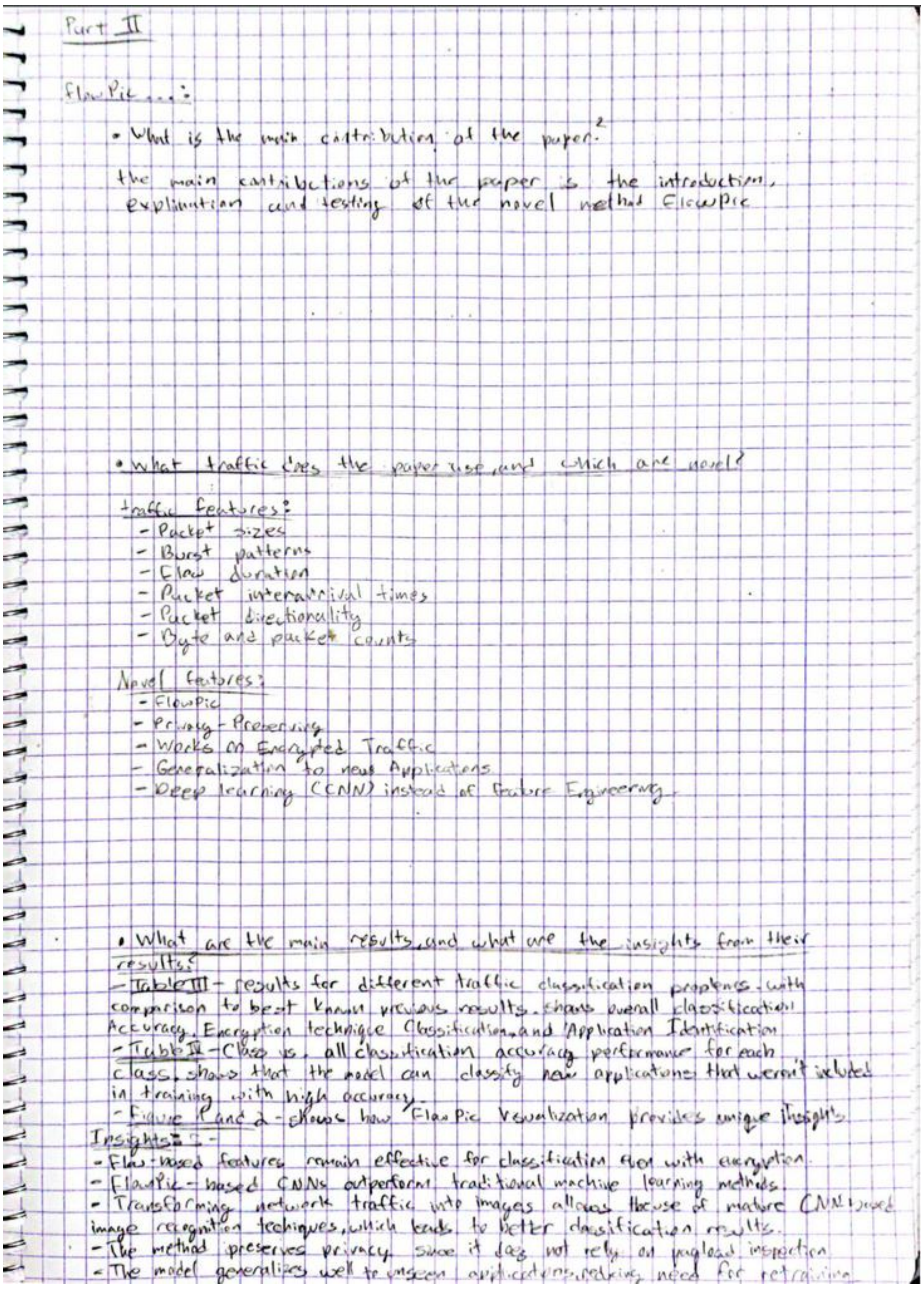
Steps to resolve the issue:
- Monitor Router Performance: Check CPU and memory usage to ensure routers are not overloaded.
- Analyze Congestion: Use tools like Wireshark or traceroute to identify where packets are being dropped.
- Optimize Routing: Ensure routing tables are updated and avoid loops or unstable paths.
- Check Hardware: Replace faulty network cables, update firmware, and inspect

router logs for errors.

- Adjust MTU Settings: Ensure proper MTU values to prevent fragmentation issues.
- Enable QoS: Prioritize critical traffic to reduce packet loss for important applications.
- Tune TCP Parameters: Adjust TCP window size and consider using congestion control algorithms to optimize performance.

# Part 2 :

FlowPic_Encrypted_Internet_Traffic_Classification_is_as_Easy_as_Image Recognition:



Part II

FlowPic...:

- What is the main contribution of the paper?

the main contributions of the paper is the introduction, explanation and testing of the novel method FlowPic

- what traffic does the paper use, and which are novel?

traffic features:
- Packet sizes
- Burst patterns
- Flow duration
- Packet interarrival times
- Packet directionality
- Byte and packet counts

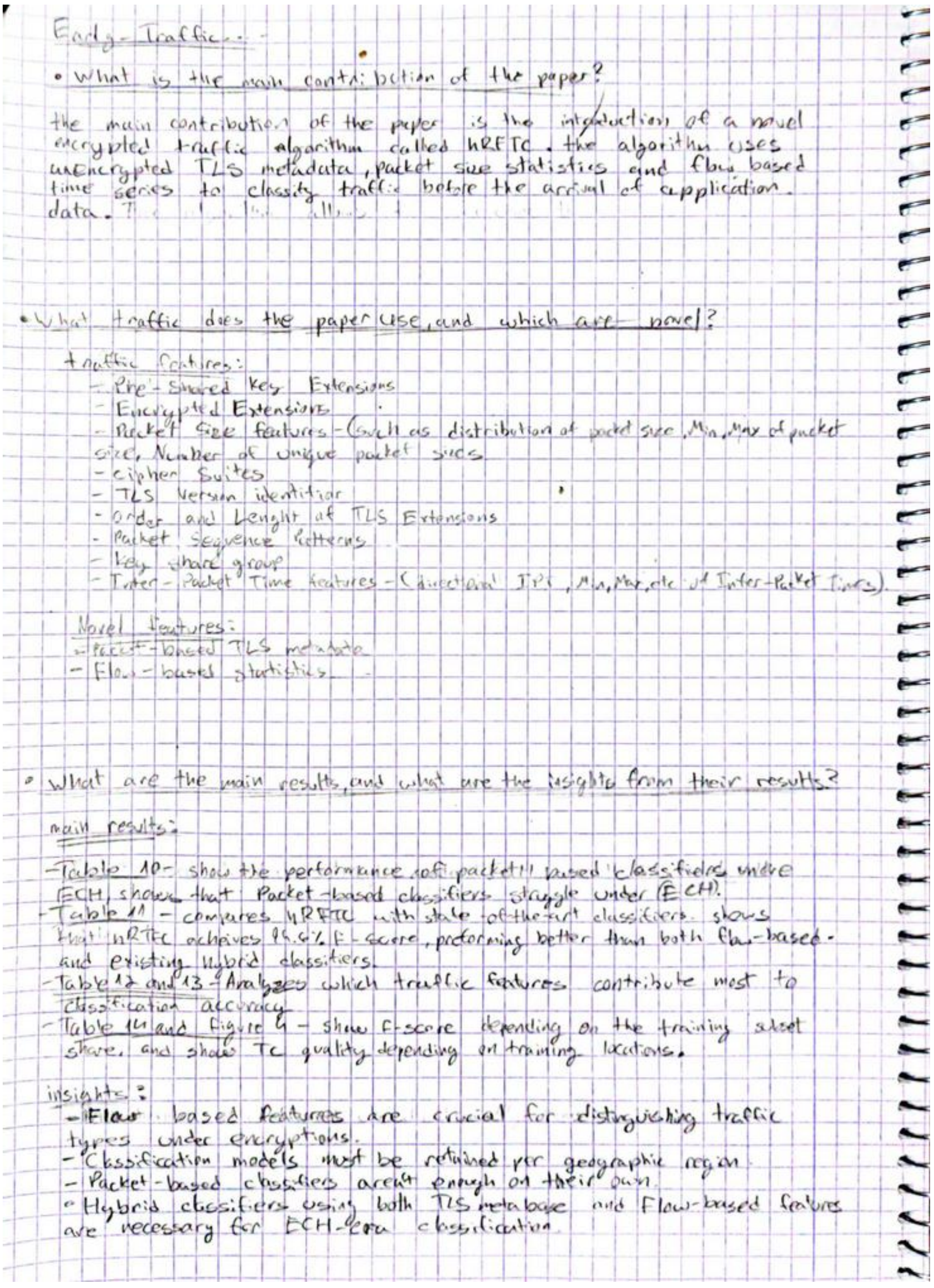
Novel features:
- FlowPic
- Privacy-Preserving
- Works on Encrypted Traffic
- Generalization to new Applications
- Deep learning (CNN) instead of Feature Engineering.

- What are the main results, and what are the insights from their results?

- Table III - results for different traffic classification problems - with comparison to best known previous results. shows overall classification Accuracy, Encryption technique Classification, and Application Identification
- Table IV - Class vs. all classification accuracy performance for each class, shows that the model can classify new applications that weren't included in training with high accuracy.
- Figure 1 and 2 - shows how FlowPic visualization provides unique insights

Insights: -
- Flow-based features remain effective for classification even with encryption.
- FlowPic-based CNNs outperform traditional machine learning methods.
- Transforming network traffic into images allows the use of mature CNN-based image recognition techniques, which leads to better classification results.
- The method preserves privacy since it does not rely on payload inspection
- The model generalizes well to unseen applications, reducing need for retraining

Early_Traffic_Classification_With_Encrypted_ClientHello_A_Multi-Country Study:



Early-Traffic..

• What is the main contribution of the paper?

the main contribution of the paper is the introduction of a novel encrypted traffic algorithm called hRFTC. the algorithm uses unencrypted TLS metadata, packet size statistics and flow based time series to classify traffic before the arrival of application data. [illegible]

• What traffic does the paper use, and which are novel?

traffic features:
- Pre-Shared Key Extensions
- Encrypted Extensions
- Packet size features - (such as distribution of packet size, Min, Max of packet size, Number of unique packet sizes
- cipher Suites
- TLS Version identifier
- order and Lenght of TLS Extensions
- Packet Sequence Patterns
- Key share group
- Inter - Packet Time features - (directional IPT, Min, Max, etc of Inter-Packet Times).

Novel features:
- Packet-based TLS metadata
- Flow-based statistics

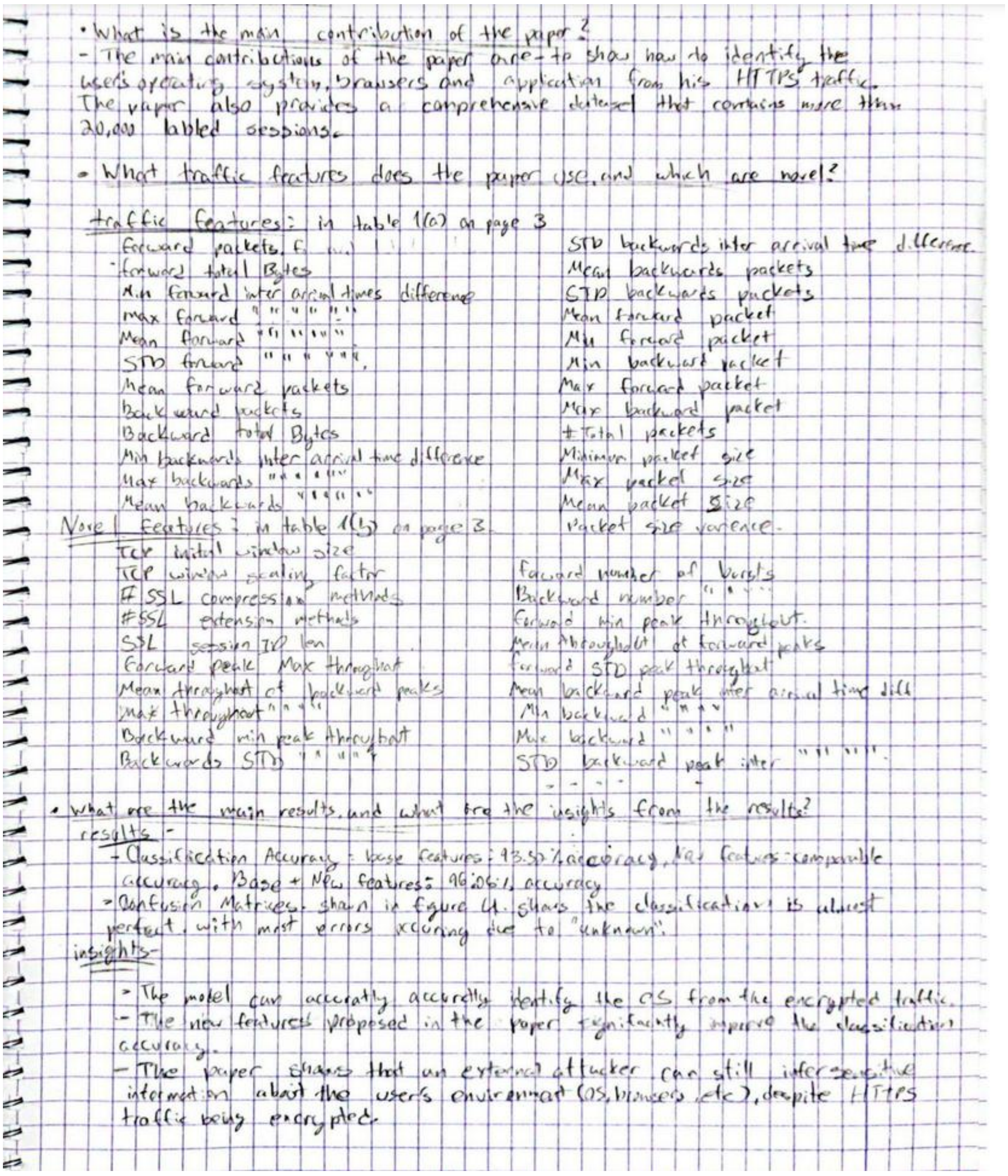
• What are the main results, and what are the insights from their results?

main results:

- Table 10 - show the performance of packet based classifiers under ECH, shows that Packet-based classifiers struggle under (ECH).
- Table 11 - compares hRFTC with state-of-the-art classifiers. shows that hRFTC achieves 94.6% F-score, preforming better than both flow-based and existing hybrid classifiers.
- Table 12 and 13 - Analyzes which traffic features contribute most to classification accuracy
- Table 14 and figure 4 - show F-score depending on the training subset share, and shows TC quality depending on training locations.

insights:
- Flow based features are crucial for distinguishing traffic types under encryptions.
- Classification models must be retrained per geographic region
- Packet-based classifiers aren't enough on their own.
- Hybrid classifiers using both TLS metabase and Flow-based features are necessary for ECH-era classification

## Analyzing HTTPS Encrypted Traffic to Identify User's Operating System, Browser and Application:



- What is the main contribution of the paper?

- The main contributions of the paper are - to show how to identify the user's operating system, browsers and application from his HTTPS traffic. The paper also provides a comprehensive dataset that contains more than 20,000 labled sessions.

- What traffic features does the paper use, and which are novel?

traffic features: in table 1(a) on page 3

Forward packets, f ...
forward total Bytes
Min forward inter arrival times difference
max forward " " " " "
Mean forward " " " " "
STD forward " " " " ".
Mean forward packets
Backward packets
Backward total Bytes
Min backwards inter arrival time difference
Max backwards " " " "
Mean backwards " " " "
STD backwards inter arrival time difference.
Mean backwards packets
STD backwards packets
Mean forward packet
Min forward packet
Min backward packet
Max forward packet
Max backward packet
# Total packets
Minimum packet size
Max packet size
Mean packet size
Packet size varience.

Novel features: in table 1(b) on page 3.

TCP initial window size
TCP window scaling factor
# SSL compression methods
#SSL extension methods
SSL session ID len
Forward peak Max throughput
Mean throughput of backward peaks
Max throughput " " "
Backward min peak throughput
Backwards STD " " " "
Forward number of bursts
Backward number " " "
Forward min peak throughput.
Mean throughput of forward peaks
forward STD peak throughput
Mean backward peak inter arrival time diff
Min backward " " "
Max backward " " "
STD backward peak inter " " " ".

- what are the main results, and what are the insights from the results?

results -

- Classification Accuracy: base features: 93.52% accuracy, New features: comparable accuracy. Base + New features: 96.06% accuracy
- Confusion Matrices: shown in figure 4. shows the classification is almost perfect, with most errors occuring due to "unknown".

insights -

- The model can accuratly accurretly identify the OS from the encrypted traffic.
- The new features proposed in the paper significantly improve the classification accuracy.
- The paper shows that an external attacker can still infer sensitive information about the user's environment (OS, browsers, etc), despite HTTPS traffic being encrypted.

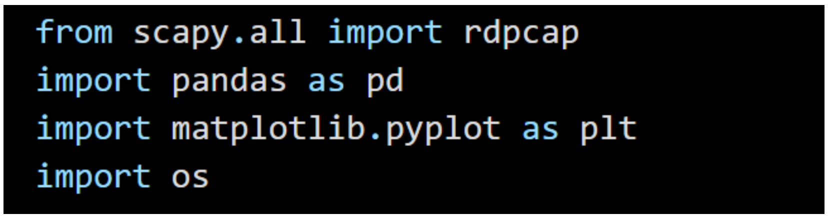
# Part 3

## Sections 1 and 2:

### Required installations in order to run code on fresh python installation:

pip3 install scapy pandas matplotlib

## Code explanation:



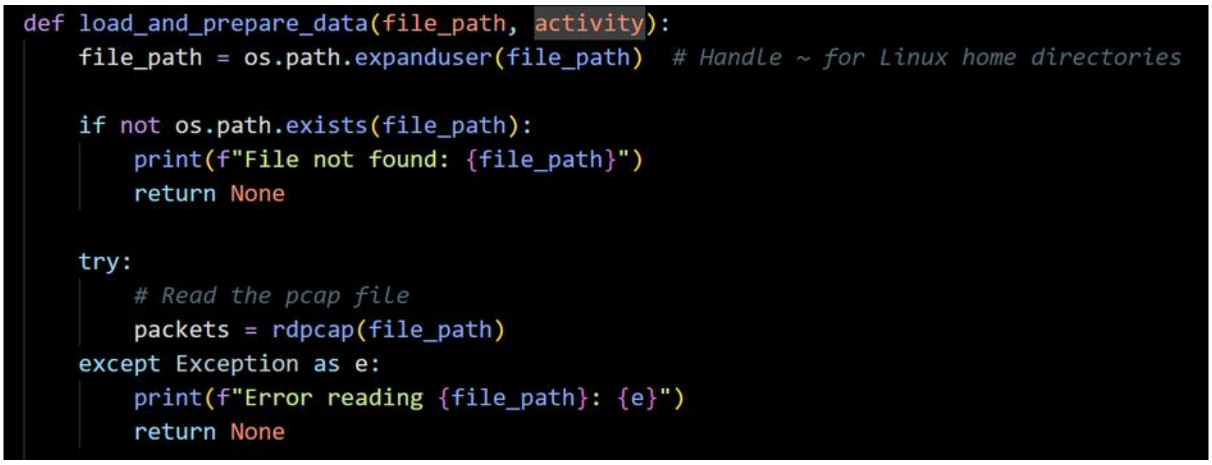
```python
from scapy.all import rdpcap
import pandas as pd
import matplotlib.pyplot as plt
import os
```

We imported “scapy” which will help us analyze network traffic from the pcap files

We imported “pandas” to store extracted packet information.

We imported “matplotlib” to create the plots required for the assignment.

We imported “os” for basic file handling operations.



```python
def load_and_prepare_data(file_path, activity):
    file_path = os.path.expanduser(file_path)  # Handle ~ for Linux home directories

    if not os.path.exists(file_path):
        print(f"File not found: {file_path}")
        return None

    try:
        # Read the pcap file
        packets = rdpcap(file_path)
    except Exception as e:
        print(f"Error reading {file_path}: {e}")
        return None
```

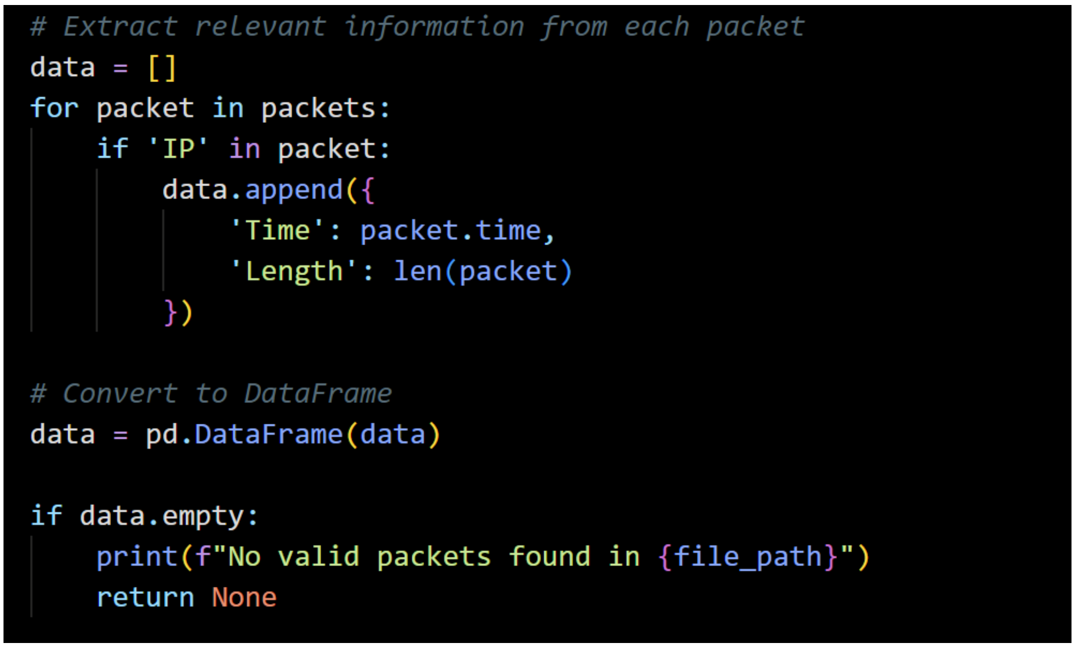
We created an array containing the names of the activities we recorded, and we will append each activity’s data to the files_data array in order.

Next, we created the load_and_prepare_data method which takes a file path and an activity, and reads the pcap file, extracts packet information from IP packets, processes the data into a structured pandas DataFrame.

file_path = os.path.expanduser(file_path): This line insures files are loaded even when users use the “~” to type the file destination faster.

next, if the file was not found, we print an error message to the user.

else, we load the packets of the file into a new array named “packets” and print an error message if that fails and return.



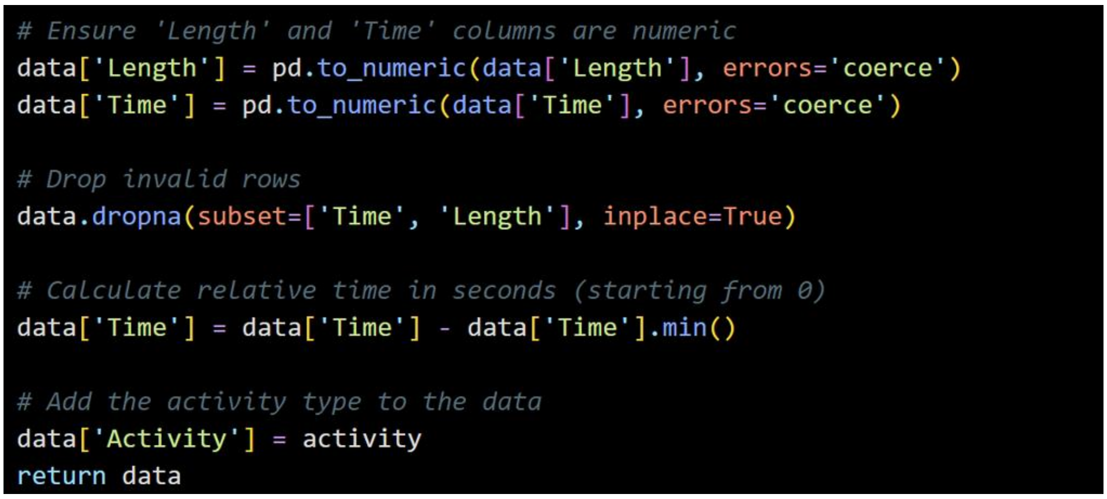
```python
# Extract relevant information from each packet
data = []
for packet in packets:
    if 'IP' in packet:
        data.append({
            'Time': packet.time,
            'Length': len(packet)
        })

# Convert to DataFrame
data = pd.DataFrame(data)

if data.empty:
    print(f"No valid packets found in {file_path}")
    return None
```

Next, for each IP packet in “packets”, we store the packet’s time capture and it’s size in bytes in the data list.

Next, we convert the list into a pandas DataFrame to simplify data processing and operations, and as always, we print an error message if the list is empty.



```python
# Ensure 'Length' and 'Time' columns are numeric
data['Length'] = pd.to_numeric(data['Length'], errors='coerce')
data['Time'] = pd.to_numeric(data['Time'], errors='coerce')

# Drop invalid rows
data.dropna(subset=['Time', 'Length'], inplace=True)

# Calculate relative time in seconds (starting from 0)
data['Time'] = data['Time'] - data['Time'].min()

# Add the activity type to the data
data['Activity'] = activity
return data
```

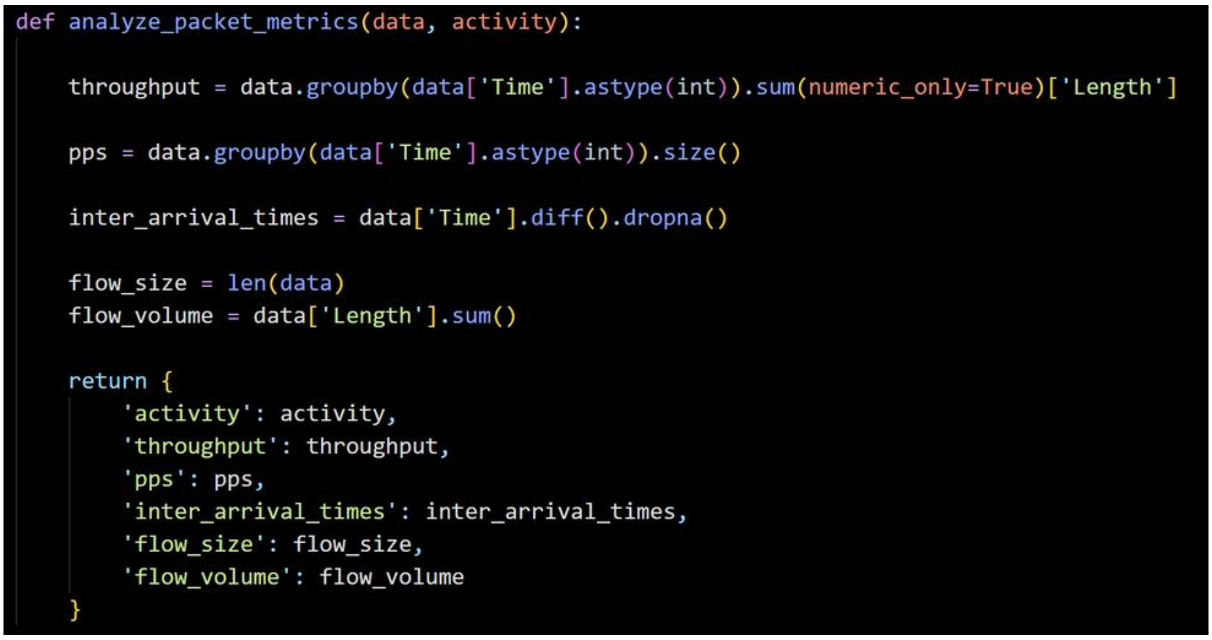
Next, we convert the length and time of each packet into a numerical value, if there are non-numeric values for certain packets, it converts them into NaN.

Next, we remove any packet where the time or length is NaN to ensure the dataset only contains valid packets.

Next, we change the packets time values so that the first packet's start time is 0 and the rest of the packets times become relative to that first timestamp which is 0.

Next, we add a new column called "Activity" which contains the name of the activity whether it's web surfing or audio streaming etc.

Next we return data as a fully processed dataframe ready to be analyzed.



```python
def analyze_packet_metrics(data, activity):

    throughput = data.groupby(data['Time'].astype(int)).sum(numeric_only=True)['Length']

    pps = data.groupby(data['Time'].astype(int)).size()

    inter_arrival_times = data['Time'].diff().dropna()

    flow_size = len(data)
    flow_volume = data['Length'].sum()

    return {
        'activity': activity,
        'throughput': throughput,
        'pps': pps,
        'inter_arrival_times': inter_arrival_times,
        'flow_size': flow_size,
        'flow_volume': flow_volume
    }
```

Next, we created the analyze_packet_metrics function, which processes the data extracted from a pcap file and computes several important network metrics, such as how much data was transferred, how many packets were sent, and the timing between them.

**Throughput**: This is the total amount of data transmitted per second, we calculate it by grouping the data based on the second of each packet's timestamp ('Time'.astype(int)) and then summing the 'Length' field for each group. This gives us the total data transferred in bytes for each second.

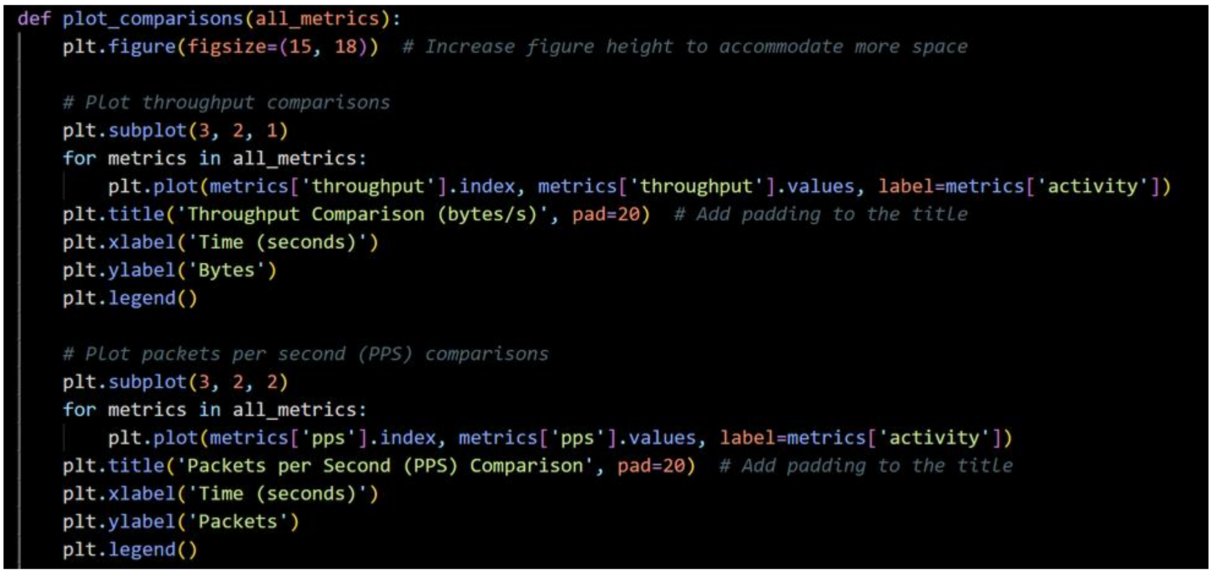
**Packets per Second (PPS)**: This metric represents how many packets are sent per second, we calculate this by grouping the data by the second ('Time'.astype(int)) and then counting the number of packets in each group using size().

**Packet Inter-Arrival Times**: This measures the time between consecutive packets, we calculate this by subtracting the timestamp of each packet from the timestamp of the previous one using diff(), and we drop any null values that result from the first packet (since it doesn't have a previous packet).

**Flow Size**: This tells us how many packets were captured in total, it's simply the length of the data (len(data)).

**Flow Volume**: This represents the total amount of data transmitted, calculated by summing up the 'Length' field of the data.

Finally, we return a dictionary with all the calculated metrics along with the activity name to help identify which set of metrics belong to which activity.



```
def plot_comparisons(all_metrics):
    plt.figure(figsize=(15, 18))  # Increase figure height to accommodate more space

    # Plot throughput comparisons
    plt.subplot(3, 2, 1)
    for metrics in all_metrics:
        plt.plot(metrics['throughput'].index, metrics['throughput'].values, label=metrics['activity'])
    plt.title('Throughput Comparison (bytes/s)', pad=20)  # Add padding to the title
    plt.xlabel('Time (seconds)')
    plt.ylabel('Bytes')
    plt.legend()

    # Plot packets per second (PPS) comparisons
    plt.subplot(3, 2, 2)
    for metrics in all_metrics:
        plt.plot(metrics['pps'].index, metrics['pps'].values, label=metrics['activity'])
    plt.title('Packets per Second (PPS) Comparison', pad=20)  # Add padding to the title
    plt.xlabel('Time (seconds)')
    plt.ylabel('Packets')
    plt.legend()
```

Next, we created the plot_comparisons function to visually compare the network metrics (throughput, packets per second, inter-arrival times, flow size, and flow volume) for multiple activities.

We setup the figure by setting it's size to (15, 18) to provide enough space for all the subplots, especially when we have multiple comparisons to display.

**Plot 1:**

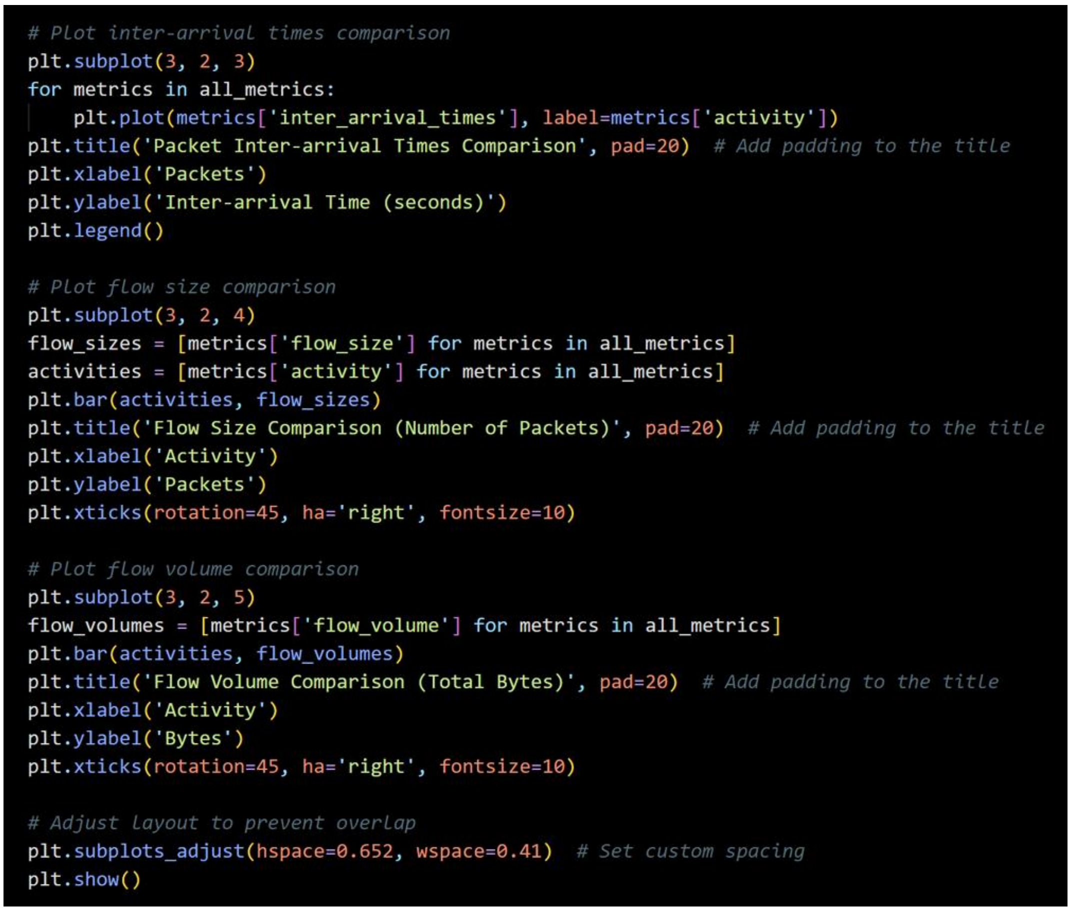
We create the first subplot (top-left corner) to compare throughput (in bytes per second). The for loop iterates over all the metrics for different activities and plots each activity's throughput over time, we use plt.plot to create the line plot.

The title, x-label, and y-label are added to indicate that this graph shows throughput over time, and the legend helps differentiate between the different activities.

**Plot 2:**

The second subplot (top-right) compares the number of packets per second for each activity. Like the throughput comparison, we iterate over the metrics and plot the PPS values.

The title and labels are set to reflect the comparison of packets per second.



```
# Plot inter-arrival times comparison
plt.subplot(3, 2, 3)
for metrics in all_metrics:
    plt.plot(metrics['inter_arrival_times'], label=metrics['activity'])
plt.title('Packet Inter-arrival Times Comparison', pad=20)  # Add padding to the title
plt.xlabel('Packets')
plt.ylabel('Inter-arrival Time (seconds)')
plt.legend()

# Plot flow size comparison
plt.subplot(3, 2, 4)
flow_sizes = [metrics['flow_size'] for metrics in all_metrics]
activities = [metrics['activity'] for metrics in all_metrics]
plt.bar(activities, flow_sizes)
plt.title('Flow Size Comparison (Number of Packets)', pad=20)  # Add padding to the title
plt.xlabel('Activity')
plt.ylabel('Packets')
plt.xticks(rotation=45, ha='right', fontsize=10)

# Plot flow volume comparison
plt.subplot(3, 2, 5)
flow_volumes = [metrics['flow_volume'] for metrics in all_metrics]
plt.bar(activities, flow_volumes)
plt.title('Flow Volume Comparison (Total Bytes)', pad=20)  # Add padding to the title
plt.xlabel('Activity')
plt.ylabel('Bytes')
plt.xticks(rotation=45, ha='right', fontsize=10)

# Adjust layout to prevent overlap
plt.subplots_adjust(hspace=0.652, wspace=0.41)  # Set custom spacing
plt.show()
```

**Plot 3:**

The third subplot (middle-left) compares the inter-arrival times (the time difference between packets) for each activity. Since inter-arrival times are plotted against packet indices, we use plt.plot again to visualize the timing between packets for each activity.

This comparison shows the packet intervals, and the title and labels describe the data.

**Plot 4:**

The fourth subplot (middle-right) compares the flow sizes, which are the number of packets captured for each activity. Here, we use a bar chart (plt.bar) to represent the flow sizes for each activity. We extract the flow sizes and the corresponding activity names, then plot the bar chart, we also made sure that the labels on the x axis are rotated and smaller so that they don't collapse into one another.

The title and labels reflect that we're comparing the number of packets per activity.

**Plot 5:**

The fifth subplot (bottom-left) compares the flow volume, which is the total amount of data (in bytes) transmitted for each activity. Similar to the flow size plot, we use a bar chart to represent the total bytes for each activity, we also made sure that the labels on the x axis are rotated and smaller so that they don't collapse into one another.

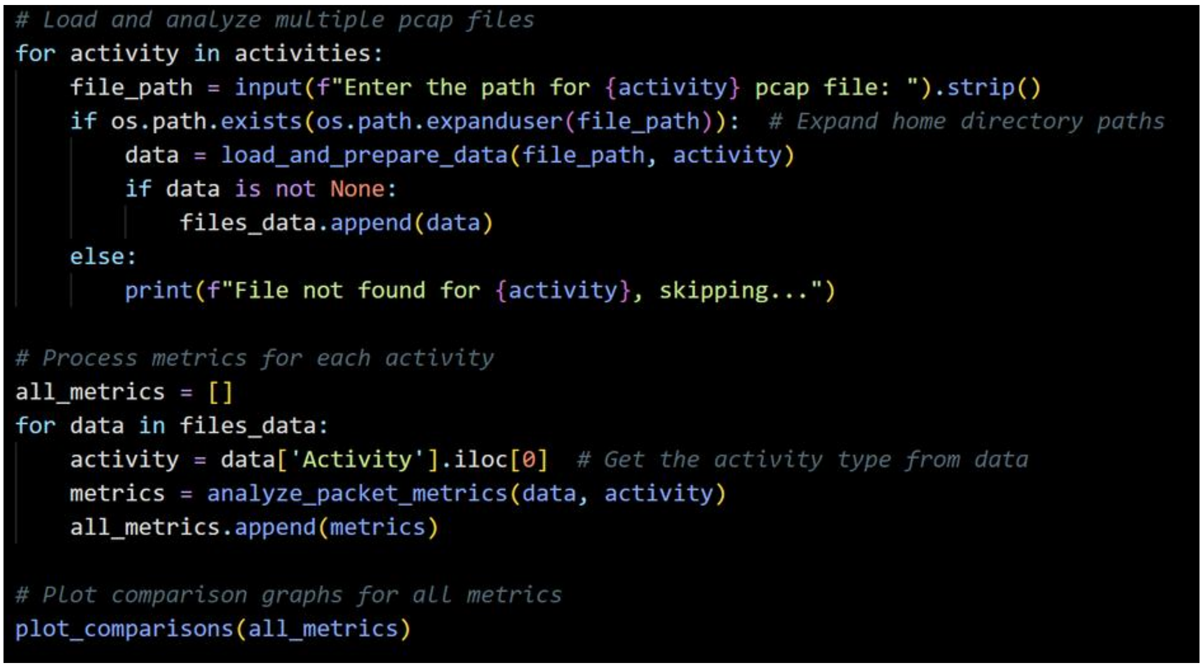
The title and labels specify that this is a comparison of the total bytes transferred during each activity.

**Layout Adjustment**:

We adjust the layout using plt.subplots_adjust to prevent overlap between the subplots.

**Displaying the Plot**:

Finally, we call plt.show() to render and display all the subplots at once.



```python
# Load and analyze multiple pcap files
for activity in activities:
    file_path = input(f"Enter the path for {activity} pcap file: ").strip()
    if os.path.exists(os.path.expanduser(file_path)):  # Expand home directory paths
        data = load_and_prepare_data(file_path, activity)
        if data is not None:
            files_data.append(data)
    else:
        print(f"File not found for {activity}, skipping...")

# Process metrics for each activity
all_metrics = []
for data in files_data:
    activity = data['Activity'].iloc[0]  # Get the activity type from data
    metrics = analyze_packet_metrics(data, activity)
    all_metrics.append(metrics)

# Plot comparison graphs for all metrics
plot_comparisons(all_metrics)
```

Next, we created the process to load, analyze, and visualize multiple pcap files this sums up what happens during runtime.

First, for each of the 5 activities, we ask the user to input the name of its pcap file and store that in file_path, and if it exists in the os, we set its data using the load_and_prepare_data method explained above.

If data is not empty, we append it to the list called “files_data” which we declared at the beginning of the code, and as usual, we print an error if the file doesn’t exist in the os.

Next, we iterate over the data for each pcap file, during each iteration we get the name of the pcap capture’s activity, and retrieve it’s analyzed metrics, and append those metrics to the “all_metrics” list.

After processing the metrics for all activities, we pass the all_metrics list to the plot_comparisons(all_metrics) function, this function, which is explained above, generates comparison plots for each network metric (throughput, PPS, inter-arrival times, flow size, and flow volume) across the different activities.

## Section 3:

Comparisons: Microsoft edge generates more distinct connections or sessions compared to other apps, it also generates the most number of packets most of the time, thus relying on more storage.

Using zoom generated the fewest total packets, which could indicate that the packets it sends are more efficient or that the data sent in each packet is larger.

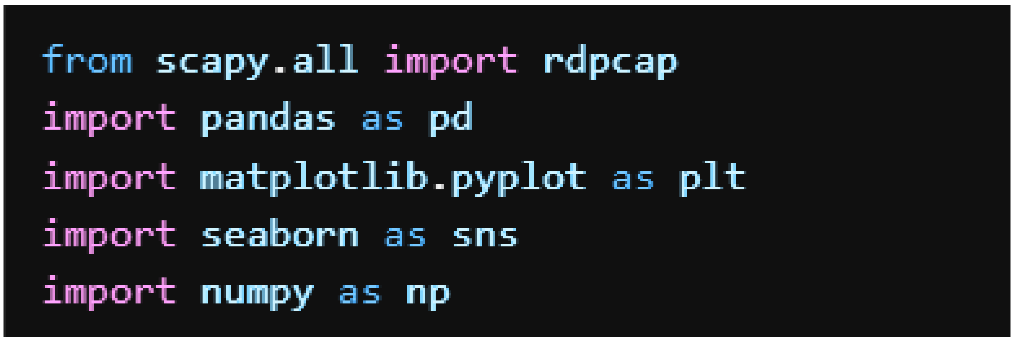
## Section 4:

### Required installations in order to run code on fresh python installation

pip install scapy pandas matplotlib seaborn numpy

## Code explanation

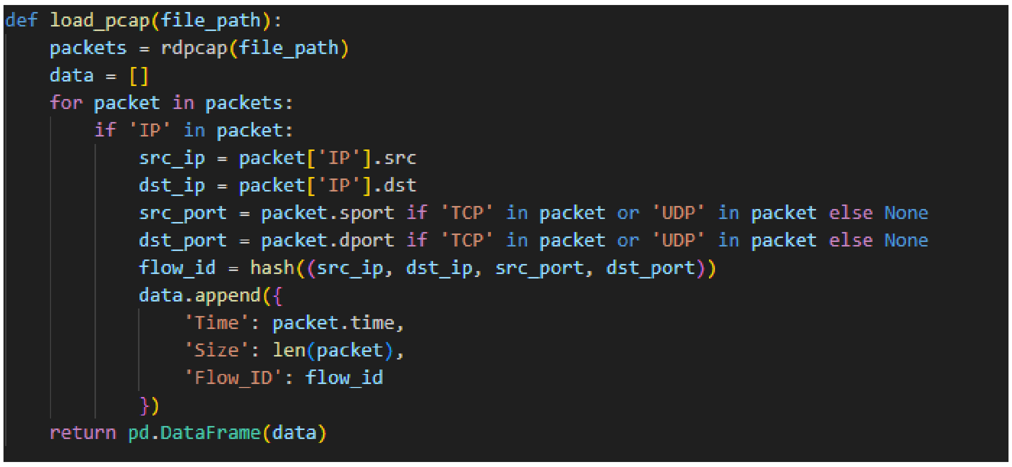
### Importing Required Libraries



```python
from scapy.all import rdpcap
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
```

- **Scapy**: Used to read and parse PCAP files.
- **Pandas**: Helps organize extracted network traffic data into DataFrames for analysis.
- **Matplotlib & Seaborn**: Used for visualizing packet size and inter-arrival time distributions.
- **NumPy**: Assists in numerical computations like calculating inter-arrival times.

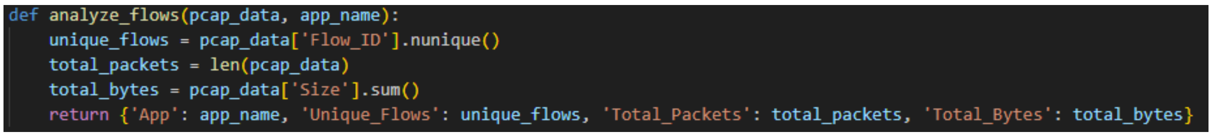
### Loading and Extracting Packet Data



```python
def load_pcap(file_path):
    packets = rdpcap(file_path)
    data = []
    for packet in packets:
        if 'IP' in packet:
            src_ip = packet['IP'].src
            dst_ip = packet['IP'].dst
            src_port = packet.sport if 'TCP' in packet or 'UDP' in packet else None
            dst_port = packet.dport if 'TCP' in packet or 'UDP' in packet else None
            flow_id = hash((src_ip, dst_ip, src_port, dst_port))
            data.append({
                'Time': packet.time,
                'Size': len(packet),
                'Flow_ID': flow_id
            })
    return pd.DataFrame(data)
```

This function reads a PCAP file and extracts relevant packet details from IP packets. It captures source and destination IPs, ports, packet size, and timestamp while generating a Hashed 4-tuple (flow identifier derived from src_ip, dst_ip, src_port, and dst_port) to uniquely identify network flows. The extracted data is stored in a Pandas DataFrame for further analysis.

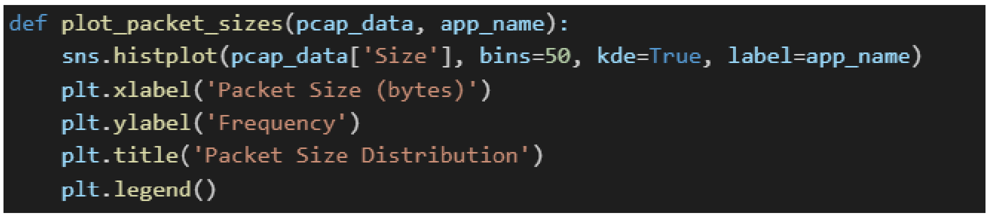
## Analyzing Unique Network Flows



```python
def analyze_flows(pcap_data, app_name):
    unique_flows = pcap_data['Flow_ID'].nunique()
    total_packets = len(pcap_data)
    total_bytes = pcap_data['Size'].sum()
    return {'App': app_name, 'Unique_Flows': unique_flows, 'Total_Packets': total_packets, 'Total_Bytes': total_bytes}
```

This function computes key properties such as the number of unique flows, the total number of packets, and the sum of all packet sizes, summarizing network traffic statistics for each analyzed application.

## Packet Size Distribution Visualization



```python
def plot_packet_sizes(pcap_data, app_name):
    sns.histplot(pcap_data['Size'], bins=50, kde=True, label=app_name)
    plt.xlabel('Packet Size (bytes)')
    plt.ylabel('Frequency')
    plt.title('Packet Size Distribution')
    plt.legend()
```

A histogram of packet sizes is plotted using Seaborn, incorporating kde to smooth the probability distribution curve, making visualizing packet size variations across different applications simpler.

## Inter-Arrival Time Analysis

```python
def plot_inter_arrival_times(pcap_data, app_name):
    inter_arrival_times = np.diff(pcap_data['Time'].sort_values().values)
    sns.histplot(inter_arrival_times, bins=50, kde=True, label=app_name)
    plt.xlabel('Inter-arrival Time (seconds)')
    plt.ylabel('Frequency')
    plt.title('Packet Inter-arrival Time Distribution')
    plt.legend()
```

This function calculates the time differences between consecutive packets, and visualizes them using a histogram. This analysis helps spot traffic patterns, whether the flow is steady or comes in bursts.

## Comparing Multiple Applications Using PCAP Files

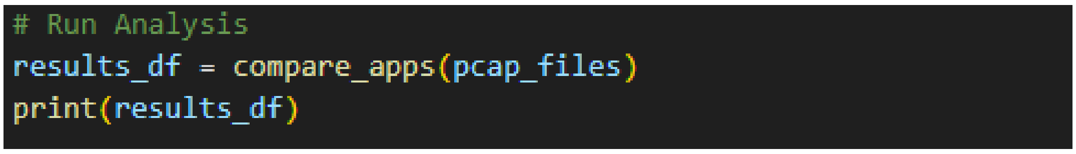
```python
def compare_apps(pcap_files):
    results = []
    plt.figure(figsize=(12, 5))
    for idx, (file_path, app_name) in enumerate(pcap_files.items()):
        print(f"Processing {app_name}...")
        pcap_data = load_pcap(file_path)
        results.append(analyze_flows(pcap_data, app_name))
        plt.subplot(1, 2, 1)
        plot_packet_sizes(pcap_data, app_name)
        plt.subplot(1, 2, 2)
        plot_inter_arrival_times(pcap_data, app_name)
    plt.show()
    return pd.DataFrame(results)
```

This function processes multiple PCAP files, each file corresponding to the required(browser media player etc) loads and analyzes them, calls the packet size and packet timing visualization functions to visualize the differences between applications. It then compiles flow statistics into a summary DataFrame for simpler comparison.

## User Input for PCAP Files and Running the Analysis

```python
pcap_files = {}
for app in ['Web-surfing 1', 'Web-surfing 2', 'Audio Streaming', 'Video Streaming', 'Video Conferencing']:
    path = input(f"Enter the pcap file path for {app}: ")
    pcap_files[path] = app
```

This prompts the user to enter file paths for different network applications, allowing customized analysis.



```
# Run Analysis
results_df = compare_apps(pcap_files)
print(results_df)
```

The script then prints the unique flows and Total packets and Total size in bytes for each pcap file processed during runtime.

## section 4 answer:

### Observations from the PCAP Files

#### Video Conferencing App

The traffic is steady and continuous with packets being sent and received at regular intervals.
There are fewer pauses, since the app needs to keep the video and audio running in real time.
If video is enabled, the packets are heavier and in rate, while audio-only calls use less bandwidth.

Even without seeing the content, the constant, back-and-forth nature of the traffic makes it clear that this is a live communication app.

#### Video Streaming App

Traffic starts with a large burst of data as the video loads.

After that, data comes in steadily, depending on buffering and video quality.

If the user skips forward or changes quality, there are more bursts of data.

The combination of an initial spike followed by smoother streaming makes it easy to tell that a video is being played.

#### Audio Streaming App

Packets weight less compared to video streaming, with smaller packets.instead of a continuous stream, data is sent in parts, loading chunks of audio ahead of time. If using offline mode, there is little to no network activity.

The smaller packet sizes and periodic data flow make this traffic look different from video streaming or web browsing.

#### Web Browsing on Two Different Browsers

Traffic happens in short bursts, especially when loading a new webpage. Some background traffic may appear, depending on browser settings.

While it may not be clear exactly which sites are visited, the bursty nature of web browsing traffic is different from streaming or conferencing apps.

## How an Attacker Could Use information up to this point

Even though the traffic is encrypted, an attacker could still:

- Tell which type of app is being used based on packet sizes and timing, for example: An
- Guess when a person is watching a video, listening to music, or in a live call just from the way packets are sent and received.
- Identify differences between browsers, since, for example: Microsoft edge has way higher Packet Size distribution than Google.

## Ways to Make This Harder for an Attacker

Using a VPN, enabling privacy features in apps and browsers or using tools that add random delays or extra traffic.

A VPN to mix traffic with other users, making patterns harder to track. Enabling privacy features to reduce unnecessary background traffic. Using aforementioned tools to make patterns less obvious.

## Answer Summary

Even with encryption, traffic analysis can reveal a lot about what someone is doing online. Maybe they won’t know the specific activities being done(i.e what kind of video you’re watching), but they’ll be able to somewhat guess what kind of activity you’re doing.