Communication Networks Final Project

Simon Kirschenbaum-327073763 Dvir Opatowsky -326063260
Harel Weinberg -322351883 Shira Duek-325175115

Project Due Date: March 3, 2025

Table of Contents

Part I---------- Pages 3-8

Part II---------- Pages 9-16

Part III---------- Pages 17-38

Bonus---------- Pages 37-38

PART I

Question 1:

A user reports that their file transfer is slow, and you need to analyze the transport layer to identify possible causes. What are the factors that could cause slow transfer, and how would you resolve the issue?

Answer:

There are several possible factors at the transport layer that can cause file transfer slowdowns:

- **Network Congestion**
  Congestion in the network may lead to packet loss and retransmissions, which limits the traffic flow. In addition, network congestion can lead to reduced TCP Window Size due to failure to receive ACKs (acknowledgments) which also limits traffic flow.
  Solution: expand bandwidth and use alternative paths.

- **Sender Window Size**
  A small window size limits the amount of packets that can be sent without confirmation which reduces bandwidth utilization.
  Solution: increase the window size

  A big window size can cause too many retransmitted packets to be sent in the case of a network issue, which could worsen network congestion.
  Solution: decrease the window size

- **Receiver Window Size**
  The receiver window size can limit the amount of traffic one can receive without sending confirmation, which limits the amount of data that can be sent from both sides of the conversation in each RTT
  Solution: managing the window size on the receiver's end to allow optimal use of the bandwidth and allow the most amount of data in each RTT

- **TCP Congestion Control**
  TCP can slow down traffic when it recognizes network congestion, sometimes even when there is no actual packet loss.
  Solution: switch the algorithm for congestion control.

  In certain situations, the algorithm for widening the sender window size can be slow, which limits bandwidth usage.

Solution: widen the bandwidth to prevent congestion.
Algorithms like “Slow Start” and “Fast Recovery” can cause the data transfer rate to grow too gradually and not use bandwidth optimally.
Solution: use a different protocol (UDP, QUIC).

- **Short Timeout**
  If the timeout is too short the sender may retransmit too early before getting the acknowledgment (ACK) from the receiver, causing double packets which creates network congestion and in certain cases causes TCP Congestion Control (which may slow down as explained in the bullet-point above).
  Solution: tune the timeout with respect to the network conditions to prevent early and unnecessary retransmissions.

Question 2

Analyze the effects of TCP’s flow control mechanism on data transmission. How would it impact performance when the sender has significantly higher processing power than the receiver?

Answer:

TCP’s flow control mechanism is intended to prevent the sending computer from flooding the receiving computer with too much data. The mechanism uses the Receive Window to make sure the sender doesn’t send so much information that the receiver can’t process and store it to memory.

Impact on performance when sender has higher processing power:

When the sending computer has significantly higher processing power than the receiver it will create a situation where the receiver will receive more packets than it can actually process, ultimately causing packet loss. This is because the receiver has a slower processing speed which limits the amount of packets it can process at a time, meaning the rate of incoming traffic is bigger than the rate of processing which is the source of the packet loss.

The TCP’s Flow Control Mechanism prevents this by using the receiver window size as a guideline and makes sure that the sender doesn’t send more packets than the receiver can receive and process.

In addition, the low processing of the receiver slows the sending of confirmation, which also slows the sending of the next window.

Question 3

Analyze the role of routing in a network where multiple paths exist between the source and destination. How does the path choice affect network performance, and what factors should be considered in routing decisions?

Answer

Impact of routing on network performance:

- **Round Trip Time - RTT**
  RTT is the time required to send a message from the source to the destination and receive a reply back. A route with fewer Hops and stable connectivity creates a shorter RTT which improves response time

- **Bandwidth**
  A route with a high capacity can move more data simultaneously. For example, a route that contains a router with a low Maximum Transmission Unit (MTU) will require message fragmentation, resulting in longer transmission times compared to a router with a higher MTU.

- **Packet Loss**
  congested routes with high traffic can cause packet loss and reduced performance. For example, a short route with heavy congestion can cause packet loss and reduced performance, meaning it might be more efficient to take a longer route.

  Instability in the route may impact real-time communication.

Important factors in when choosing a route:

1. **Routing Policy:** For example, preference for a faster path or a more stable one.
2. **Connection Quality:** Measuring bandwidth, congestion, and packet loss.
3. **Redundancy and Reliability:** Smart routing ensures that the connection will continue to work in case of failure.
4. **Load Balancing:** Distributing the load across multiple paths.
5. **Quality of Service:** Certain paths will provide better performance for specific types of services, for example, for video streaming, we would prefer to choose paths that optimally support the service type we are using.

Question 4

How does MPTCP improve network performance?

Answer

Multipath TCP (MPTCP) allows the use of multiple routers simultaneously in the same TCP connection.

How MPTCP improves network performance:

- **Improved usage of bandwidth**
  It utilizes multiple connections to increase the transfer rate, using more paths than TCP, thereby allowing for increased bandwidth usage compared to using a single path.

- **Reliability**
  In case of a failure in one of the routers, the connection remains and uses an alternative route.

- **Load Balancing and Reducing Network Congestion Impact**
  - Splits the congestion between multiple routers to prevent bottle-necks.
  - If a route is congested, MPTCP routes traffic to another router with less congestion.

Question 5

You are monitoring network traffic and notice high packet loss between two routers. Analyze the potential causes for packet loss at the Network and Transport Layers and recommend steps to resolve the issue.

Answer

Packet loss can occur both at the network layer (IP) and at the transport layer (TCP/UDP).

Potential factors at the network layer:

1. **Network congestion**

   Routers drop packets when their queues are full.

   Solution:

a. We can use more powerful routers with larger memory and greater computing power, thereby increasing the queue size and reducing processing time.
b. We can use a better routing algorithm to distribute the load more efficiently.
c. Network expansion – Adding new routers to the network will create new paths for packet transmission, reducing the number of packets each router receives.

2. **Mismatched MTU:**

   Packets requiring fragmentation may be lost.

   Solution: Configure routers in the network to use the same MTU or send smaller packets. An algorithm similar to those used in flow networks can be utilized to determine the maximum MTU size that can be transmitted across the network.

3. **TTL Expiration:**

   If the TTL (Time-To-Live) of a packet is insufficient for the required number of hops to reach its destination, the packet is considered lost.

   Solution: Set a sufficiently high TTL value to ensure the packet reaches its destination.

Possible Causes in the Transport Layer (TCP/UDP):

1. **Receiver Window Size Smaller than Sender Window Size:**

   If the receiver's window size is smaller than the sender's window size, additional incoming packets will be lost.

   Solution: Adjust the sender's window size to match the receiver's window size.

2. **Timing Mechanism Issues:**

   If the timing mechanism does not match the network's RTT (Round Trip Time), the sender may not receive acknowledgment messages from the receiver. As a result, the sender assumes the packets were lost and retransmits them, even though no actual packet loss occurred.

   Solution: Increase the timeout duration by calculating the RTT.

PART II

Article 1: FlowPic: Encrypted Internet Traffic Classification is as Easy as Image Recognition

Question 1: What is the main contribution of the paper?

- This paper introduces FlowPic, a novel approach for encrypted internet traffic classification by converting flow data into images and running them through Convolutional Neural Networks (CNNs) to determine its classification. Unlike previous methods that rely on handcrafted statistical features and shallow machine learning models, FlowPic transforms packet size and arrival time into a two-dimensional histogram (image) and applies CNN-based image recognition techniques. This approach allows us to classify traffic into categories such as VoIP, video, chat, browsing, file transfer; very quickly and accurately, even in the face of encryption techniques like VPNs or Tor.

Question 2: What traffic features does the paper use, and which are novel?

- FlowPic is a method that uses multiple traffic features to create an image that is then classified using CNN-based image recognition techniques. The traffic features it uses include:
    - Packet size
    - Packet inter-arrival time
    - Flow directionality

  The novelty of the FlowPic method comes from their use of these ideas:

    - Use of histograms to archive the data for analysis
    - CNN (and deep learning in general) based classification
    - Encryption resilience: the method works without inspection the payload and can be used to classify traffic encrypted with VPN or Tor.

Question 3: What are the main results (you may copy the figures from the paper), and what are the insights from their results?

- The FlowPic method achieves high accuracy in encrypted traffic classification by converting flow data into images and using CNNs for deep learning-based classification. For traffic categorization, it achieves 85.0% for non-VPN, 98.4% for VPN, and 67.8% for Tor traffic. The lower accuracy for Tor suggests that its encryption is more effective at obfuscating flow patterns, whereas VPN traffic still retains identifiable characteristics.

  In application identification, FlowPic achieves 99.7% accuracy, significantly outperforming traditional machine learning methods (e.g., k-NN at 93.9%), showing that different applications have unique traffic flow signatures that CNNs can learn effectively. Similarly, encryption detection reaches 88.4% accuracy, with Tor traffic being the easiest to identify (97.7% recall), while VPN and non-VPN are harder to separate due to shared characteristics.

Problem	FlowPic Acc. (%)	Best Previous Result	Remark
Non-VPN Traffic Categorization	85.0	84.0 % Pr., Gil *et al.* [15]	Different categories. [15] used unbalanced dataset
VPN Traffic Categorization	98.4	98.6 % Acc., Wang *et al.* [7]	[7] Classify raw packets data. Not including browsing category
Tor Traffic Categorization	67.8	84.3 % Pr., Gil *et al.* [15]	Different categories. [15] used unbalanced dataset
Non-VPN Class vs. All	97.0 (Average)	No previous results	
VPN Class vs. All	99.7 (Average)	No previous results	
Tor Class vs. All	85.7 (Average)	No previous results	
Encryption Techniques	88.4	99. % Acc., Wang *et al.* [7]	[7] Classify raw packets data, not including Tor category
Applications Identification	99.7	93.9 % Acc., Yamansavascilar *et al.* [10]	Different classes

  A key strength of FlowPic is its generalization ability. Even when the model is not trained on a specific application, it still classifies it into the correct category (e.g., Facebook Video -> 99.9% accuracy), proving that it captures category-level behaviors rather than app-specific features. This also extends to encryption: a model trained only on non-VPN traffic can still classify VPN traffic with up to 99.4% accuracy, revealing that VPN does not fully hide traffic patterns, while Tor remains more difficult to classify.

  Overall, FlowPic outperforms traditional feature-based methods, generalizes well to new traffic, and enables real-time, privacy-preserving classification of encrypted internet traffic.

<u>Article 2:</u> Early Traffic Classification With Encrypted ClientHello: A Multi-Country Study

<u>Question 1:</u> <u>What is the main contribution of the paper?</u>

- The article presents an innovative and first-of-its-kind method for classifying TLS 1.3 traffic with ECH, as most existing methods have become irrelevant due to the new encryption.

  It is the first to prove that the type of service can be identified even when the SNI is encrypted!

  The article challenges the assumption that ECH completely prevents traffic identification and demonstrates that valuable information about users can still be inferred.

<u>Question 2:</u> <u>What traffic features does the paper use, and which are novel?</u>

- The article uses two types of features to identify services:

  Base Features:

- **Packet Size** – Minimum, maximum, and average.
- **Inter-Packet Timing** – Timing of data transmission between the client and the server.
- **Forward & Backward Traffic** – The ratio of sent to received packets.

  New Features:

- **TLS Cipher Suites** – The encryption algorithms supported by the client and server.
- **Session Length & Extensions** – The number of supported TLS extensions and the session ID length.
- **QUIC Packet Analysis** – Analyzing packets in the QUIC protocol, including QUIC Padding, which complicates traffic identification.
- **Identification Based on Initial TLS Packet Patterns** – Enables classification even without SNI information.

Question 3: What are the main results (you may copy the figures from the paper), and what are the insights from their results?

Research Stages:

Step 1: Dataset Collection

- 600,000 TLS flows were captured from various countries (USA, Germany, Russia, Spain, Kazakhstan, and Turkey).
- Data was collected from YouTube, Netflix, Facebook, TikTok, and Spotify traffic.
- Wireshark was used to capture traffic while controlled browsing was performed using Selenium.

Step 2: Feature Extraction

- Unencrypted TLS features were analyzed, including supported cipher suites and visible TLS extensions.
- Statistical flow data was integrated, such as packet size, inter-packet timing, and traffic direction.
- Researchers emphasized that statistical features allow for accurate identification even when SNI is encrypted.

Step 3: Machine Learning Training

- A Random Forest algorithm was used, combining TLS features with statistical features.
- Data was split into 70% for training and 30% for testing.
- Performance was compared against existing algorithms like MATEC, BGRUA, and CESNET.

Step 4: Evaluation & Testing

- Accuracy was measured using F-score, a standard classification quality metric.
- The hRFTC model achieved 94.6% accuracy, compared to only 38.4% for previous TLS-based methods.
- The study found that using statistical flow data significantly improves traffic classification.

hRFTC Algorithm Overview & Dataset Details

hRFTC Workflow:

- **Step 1:** Data preparation – Each sample is labeled based on the service from which it was captured.
- **Step 2:** Feature extraction – Combining TLS metadata with statistical data such as packet size and timing.
- **Step 3:** Model training & evaluation – 5-Fold Cross Validation was used to prevent overfitting.

Key Findings:

- The model successfully classified services with 94.6% accuracy!
- Adding TLS metadata and flow statistics significantly improved performance.

- Even though content is encrypted, a lot can still be inferred from metadata alone.

Privacy Concerns:

- **Malicious entities** could use this method to determine which services and apps a user is accessing – even when the content is encrypted!
- **Internet providers (ISPs)** could track users and serve targeted ads based on their traffic.

- Even when using a VPN, the method can still distinguish between different types of services.

The conclusion from the Results:

- TLS features alone are not enough! Flow data must be included to achieve high accuracy.
- QUIC makes classification more difficult, but analyzing packet size can help mitigate this challenge.
- Deep learning-based models are less effective than the hybrid hRFTC approach.
- Despite advancements in encryption, traffic can still be classified with high accuracy – based solely on metadata analysis!

Classification accuracy	Feature type
38.4%	TLS only
91.2%	Statistical Features Only
94.6%	TLS + Flow Statistics

<u>Article 3:</u> Analyzing HTTPS Encrypted Traffic to Identify User's Operating System, Browser and Application

<u>Question 1:</u> <u>What is the main contribution of the paper?</u>

The article presents an innovative method for identifying a user's operating system, browser, and application based solely on encrypted HTTPS traffic.

Until now, most research has focused either on identifying applications alone or on analyzing unencrypted traffic.

This study introduces, for the first time, a method that allows the identification of not only the application but also the user's operating system and browser—even when all data is encrypted.

The method does not require direct access to the data itself but instead analyzes traffic patterns (metadata) to extract unique characteristics of each operating system and browser.

Unlike previous studies, the researchers examined the external characteristics of the packets—which can be accessed even when the data is encrypted.

<u>Question 2:</u> <u>What traffic features does the paper use, and which are novel?</u>

- **Basic Features (Base Features)**

  These are statistical features that have been used in previous research for traffic analysis. They are useful for the basic identification of applications and browsers but do not provide high accuracy for identifying the operating system.

  Number of packets sent and received.
  Packet size – minimum, maximum, average, and standard deviation.
  Inter-arrival time between packets – for example, the minimum, average, and maximum time between two consecutive packets.
  Forward and backward traffic – identifying the directionality of data flow.

- **New Features**

  SSL/TLS Features

  When connecting to a website via HTTPS, a Handshake occurs where the browser sends information about the encryption versions it supports.
  Different operating systems and browsers send different SSL identifiers!
  Therefore, it is possible to identify them based solely on the handshake process without reading the actual data.

- **SSL Extension Count** – The number of extensions the browser sends during the SSL handshake.
- **SSL Session ID Length** – The session ID of the connection, which has different values for each browser.
- **SSL Cipher Methods Count** – The number of encryption methods the browser offers during the handshake process.

TCP Features

Different operating systems manage their TCP connections differently.
As a result, it is possible to identify the operating system based only on how it manages network connections!

- **TCP Initial Window Size** – Determines how much data can be sent before receiving further acknowledgment.
- **TCP Window Scaling Factor** – Used to increase the window size in network connection management.

Bursty Behavior of Browsers

Why is Bursty Behavior important?

Browsers do not send data at a constant rate but instead transmit in bursts. Each browser has a unique signature of data transmission patterns.
Thus, it is possible to identify the browser solely based on transmission patterns, even when the content is encrypted!

- **Forward & Backward Bursts Count** – The number of times the browser sends a large burst of data and then pauses.
- **Mean & Max Peak Throughput** – How fast the browser sends packets in a single burst.

Question 3: What are the main results (you may copy the figures from the paper), and what are the insights from their results?

Conclusion from the table: Using the new features improved accuracy to 96.06%! This means that adding SSL/TLS, TCP, and burst traffic (Bursts) features significantly enhances user identification.

Combined classification accuracy <OS, browser, app>	Types of Features
93.51%	Basic only
94.85%	New only
96.06%	Basic + New

Operating System (OS) Identification – 100% accuracy
Browser Identification – 97-99% accuracy.
Application Identification – 96% accuracy.
This means that even without direct access to the data, the user can still be identified

PART III

We recorded the following data as a Pcap file and imported it as CSV for easier processing:

- Chrome - about 194 seconds of searching on Google, clicking on links, and lingering on websites.
- Edge - about 165 seconds of searching on Microsoft Bing, clicking on links, and lingering on websites.
- Spotify_signin - about 221 seconds, on Chrome, includes signing into spotify.com (which also includes going into Gmail for 2FA) then listening to a song with no video for about 120 seconds then switching to another song with no video for the rest of the recording.
- Spotify_no_signin - about 185 seconds, on Chrome, was already in Spotify when the recording started, then listening to a song for about 120 seconds with no video then switching to another song with no video for the rest of the recording.
- Youtube_240 - about 189 seconds, watching 2 YouTube videos on Chrome at a resolution of 240p.
- Youtube_720- about 188 seconds, watching 2 YouTube videos on Chrome at a resolution of 720p. (the same videos as youtube_240)
- Youtube_1080- about 197 seconds, watching 2 YouTube videos on Chrome at a resolution of 1080p. (the same videos as youtube_240)
- Zoom - about 270 seconds of a Zoom meeting. Includes sharing screen, muting, turning off the camera, speaking, drawing on the shared screen, and ending the meeting.

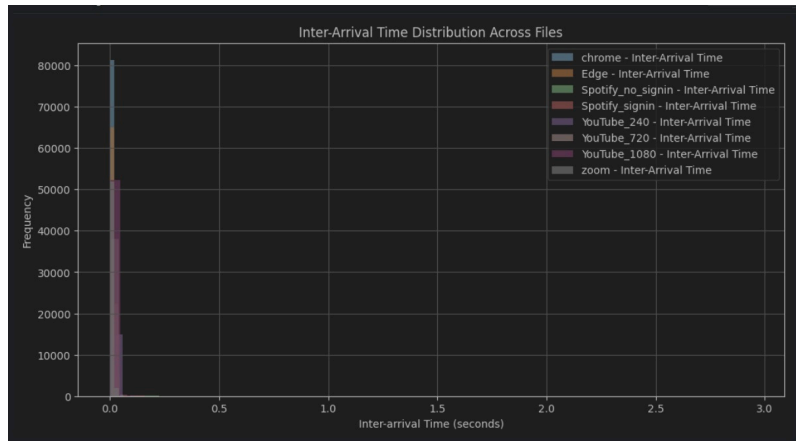
We also created a key file in order to decrypt HTTPS and other TLS-encrypted traffic.

- In this part, we added graphs that we thought were relevant to our explanation but other graphs, and the code to create those graphs, exist in the code portion of this project on the GitHub page.

Inter-arrival times

In this plot, we calculated the time in between each consecutive packet and marked it on a graph as well as logged the average inter-arrival time for each file.

We can see that this graph does not help us distinguish types of files from each other because most internet traffic comes in quick succession which leads to all the graphs looking very similar when compared to one another.



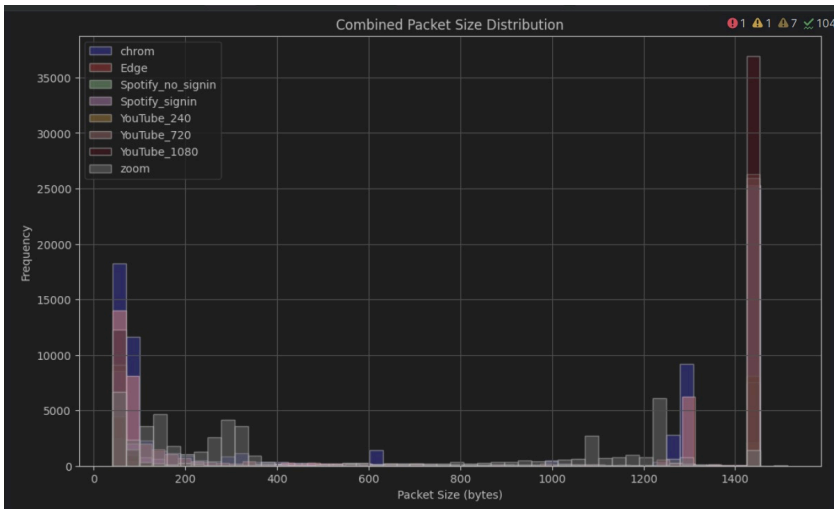
Although the graphs themselves aren't much help in distinguishing traffic, we can learn some things from the average inter-arrival time. For example: we can see that the Spotify_not_pros.csv and Spotify_not_pros_oneSonge.csv have significantly higher average inter-arrival time compared to the other files. This is significant especially when we compare it to the Spotify_with_pros.csv where the average inter-arrival time is lower despite them both being similar recordings. This is because when looking at the Chrome and Edge recordings we can see that the average inter-arrival time is the lowest, meaning a lot of packets are being sent in quick succession – when we cross reference that with the fact that the Spotify_with_pros.csv used Chrome to log in, we can discern that browsing has a higher average packet-per-second rate than Spotify.

```
File: chrome.csv
    Average Inter-Arrival Time: 0.002352 seconds
File: Edge.csv
    Average Inter-Arrival Time: 0.002488 seconds
File: Spotify_no_signin.csv
    Average Inter-Arrival Time: 0.032001 seconds
File: Spotify_signin.csv
    Average Inter-Arrival Time: 0.009611 seconds
File: YouTube_240.csv
    Average Inter-Arrival Time: 0.012356 seconds
File: YouTube_720.csv
    Average Inter-Arrival Time: 0.004891 seconds
File: YouTube_1080.csv
    Average Inter-Arrival Time: 0.003742 seconds
File: zoom.csv
    Average Inter-Arrival Time: 0.004999 seconds
```

When looking at the average times for the YouTube recordings, we can see that as the video resolution increases- the average inter-arrival time decreases. This is because the higher resolution requires more information to be sent to the user. In order to prevent buffering, YouTube needs to send more information at the same time as it is needed for lower resolutions, meaning the time between packets will be noticeably lower- leading to a change in average inter-arrival time.

Packet sizes

In this plot, we graphed the packet sizes that appear in the traffic recordings. The graph is very varied (which is expected). This is because of the different file sizes and types of protocols for the different services needed for all the different types of recordings.

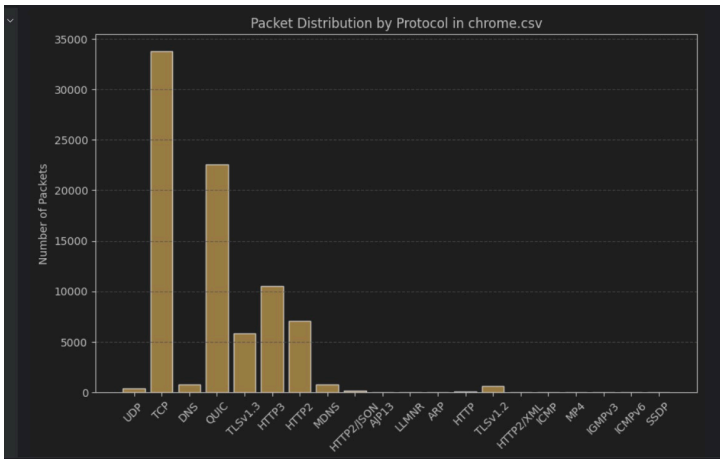


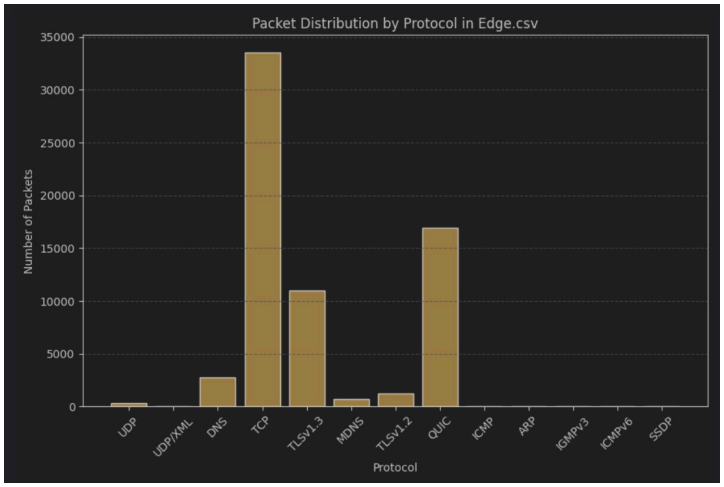
We can see more interesting points when looking at the data in numbers, for example: YouTube (in all resolutions) has a higher average packet size than all the other types of recordings. The average packet size also increases based on the resolution set in the YouTube video. The size difference between Spotify and YouTube is also obvious when you consider that Spotify is only streaming audio while YouTube streams audio and video.
In addition to that, zoom's standard deviation is also a lot lower than the rest of the recordings. This is because Zoom is trying to keep a steady and uniform flow of information in order to make sure that the call between participants is eligible (no jumps or cuts in voice or video) and do it all in real-time. It's also worth noting that the average packet size of Zoom is also the smallest of any of the recordings, which ties into the narrative of the steady flow of information- in order for Zoom to accurately and quickly send information it sacrifices quality (meaning smaller packet size).

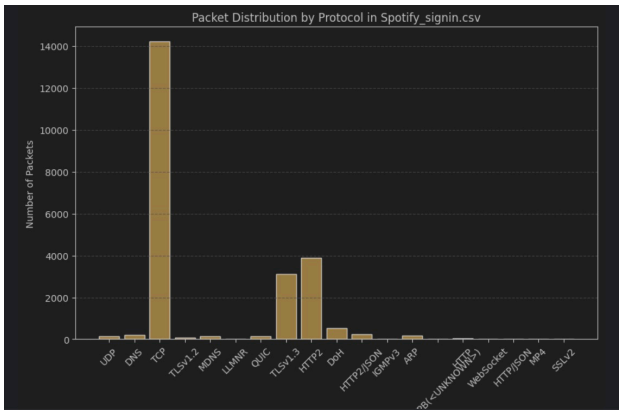
```
File: chrome.csv
    Average Packet Size: 736.66 bytes
    Standard Deviation: 632.49 bytes
File: Edge.csv
    Average Packet Size: 797.19 bytes
    Standard Deviation: 647.14 bytes

File: Spotify_no_signin.csv
    Average Packet Size: 614.07 bytes
    Standard Deviation: 658.93 bytes
File: Spotify_signin.csv
    Average Packet Size: 611.64 bytes
    Standard Deviation: 639.39 bytes

File: YouTube_240.csv
    Average Packet Size: 846.81 bytes
    Standard Deviation: 667.68 bytes
File: YouTube_720.csv
    Average Packet Size: 1024.62 bytes
File: YouTube_1080.csv
    Average Packet Size: 1055.19 bytes
    Standard Deviation: 621.89 bytes
File: zoom.csv
    Average Packet Size: 541.90 bytes
    Standard Deviation: 477.98 bytes
```

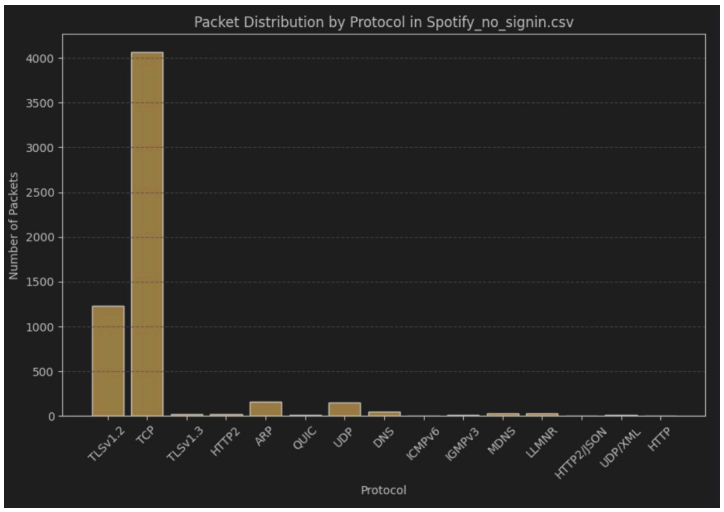
protocols

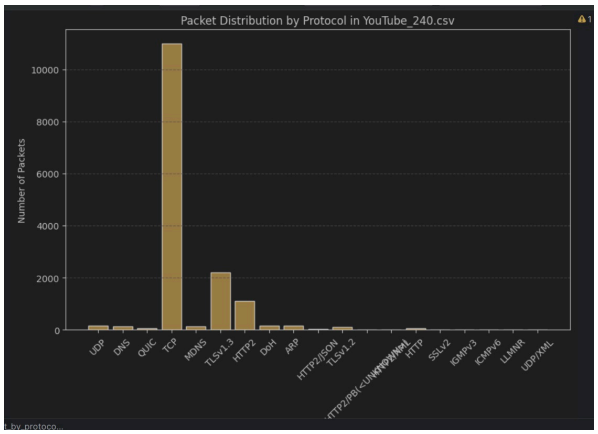
We plotted out the protocol distribution in each recording. While recording we made a key file.
We can see that in most of the graphs (the exception being the zoom graph) TCP is the most common protocol. This is because TCP is a secure protocol that can safely and accurately relay all the information to the user. For this reason, it is very common when using a browser.
Because the Spotify and YouTube recordings are of the browser versions of each program we can see the TCP appears many times.
Zoom, unlike the rest of the applications, uses mostly UDP. This is because Zoom prefers speed and efficiency over reliability.

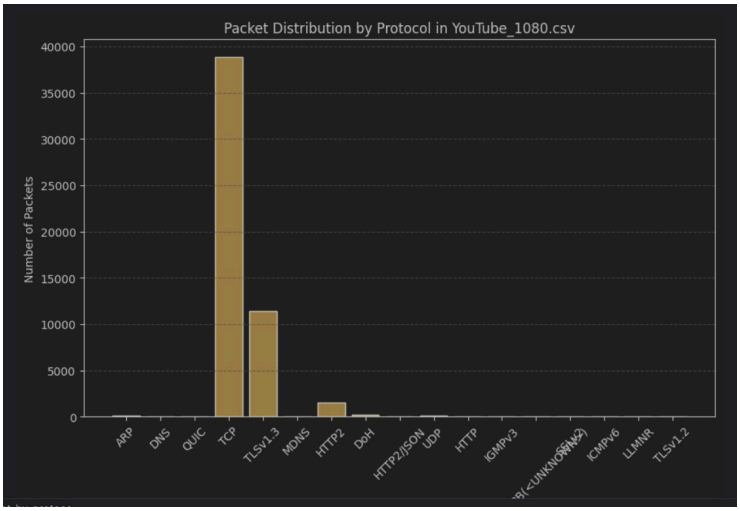


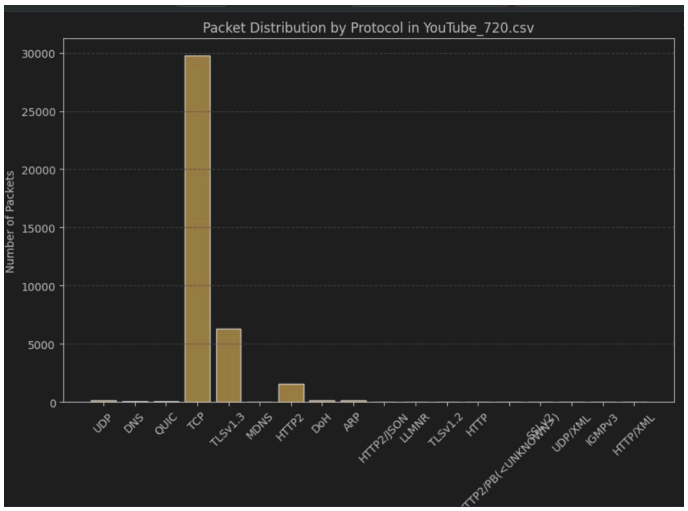


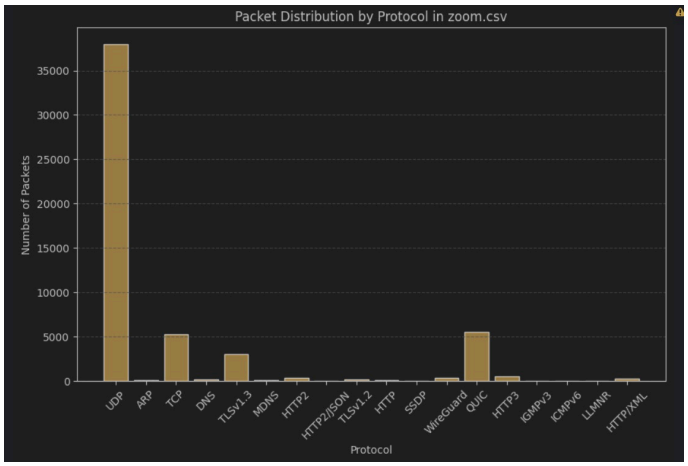






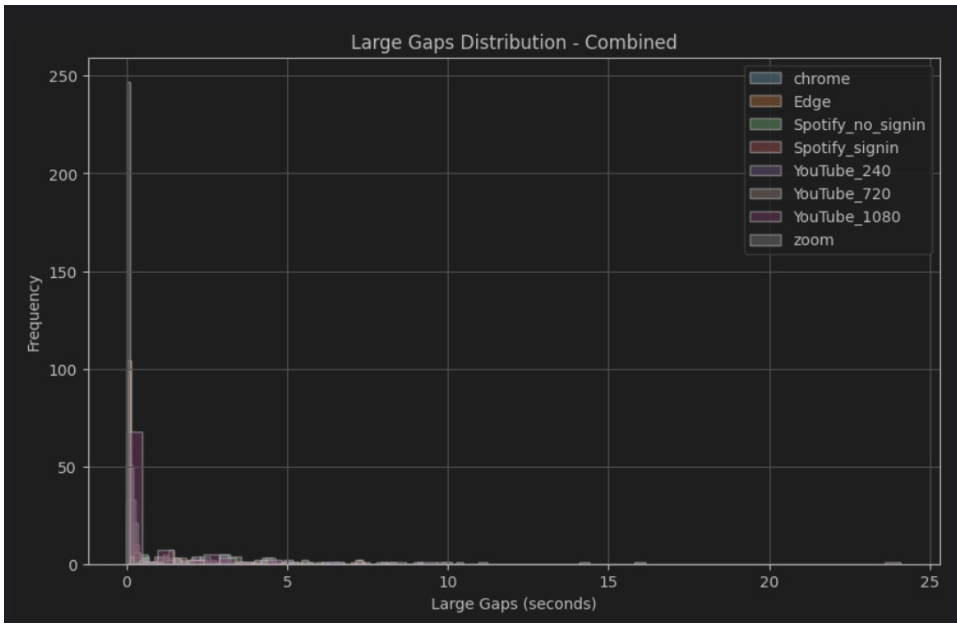

Packet Distribution by Protocol in YouTube_1080.csv
Number of Packets
40000
35000
30000
25000
20000
15000
10000
5000
0
ARP
DNS
QUIC
TCP
TLSv1.3
MDNS
HTTP2
DoH
HTTP2/JSON
UDP
HTTP
IGMPv3
ICMPv6
LLMNR
TLSv1.2


Packet Distribution by Protocol in YouTube_720.csv
Number of Packets
30000
25000
20000
15000
10000
5000
0
UDP
DNS
QUIC
TCP
TLSv1.3
MDNS
HTTP2
DoH
ARP
HTTP2/JSON
LLMNR
TLSv1.2
HTTP
UDP/XML
IGMPv3
HTTP/XML


Packet Distribution by Protocol in zoom.csv
Number of Packets
35000
30000
25000
20000
15000
10000
5000
0
UDP
ARP
TCP
DNS
TLSv1.3
MDNS
HTTP2
HTTP2/JSON
TLSv1.2
HTTP
SSDP
WireGuard
QUIC
HTTP3
IGMPv3
ICMPv6
LLMNR
HTTP/XML
Protocol

Large gaps

Here we plotted out "large gaps" between packages being sent. The way "large gaps" are categorized is as follows: All packets that are labeled in a single second in a pcap file (e.g. a packet that was recorded at 17.3 seconds and a packet that was recorded at 17.8 seconds) would be put into the same bin. If a bin has less than 15 members it gets deleted. After this, we calculate the time between the last packet of a bin and the first packet of the following bin. plotting the "big gaps" serves to try and look at the downtime in each type of pcap recording.




```
File: chrome.csv - Average Large Gap: 0.317550 seconds
File: Edge.csv - Average Large Gap: 0.284707 seconds
File: Spotify_no_signin.csv - Average Large Gap: 2.198915 seconds
File: Spotify_signin.csv - Average Large Gap: 1.252155 seconds
File: YouTube_240.csv - Average Large Gap: 1.911645 seconds
File: YouTube_720.csv - Average Large Gap: 1.350778 seconds
File: YouTube_1080.csv - Average Large Gap: 1.214111 seconds
File: zoom.csv - Average Large Gap: 0.067869 seconds
```

The graph which consists of all the different recordings as a single histogram is quite hard to read, therefore more individual graphs are needed. We can also use the average “large gaps” per pcap to help discern important information from each recording.

We can see that the average large gap in zoom.csv is considerably smaller than in the rest of the files. This is because there is a more consistent flow of traffic on Zoom to keep a stable and understandable conversation between users. This is in contrast to YouTube for example where it’s sufficient to download the video in pieces to be played back to the user.

Spotify has the highest average large gap. This is because Spotify quickly downloads pieces of the song (which is a much smaller file than a video) and then has a lot more downtime than any other of the recorded software.

Flow lengths

Next, we plotted out flow lengths. Flow is categorized as traffic that shares a 5-tuple:

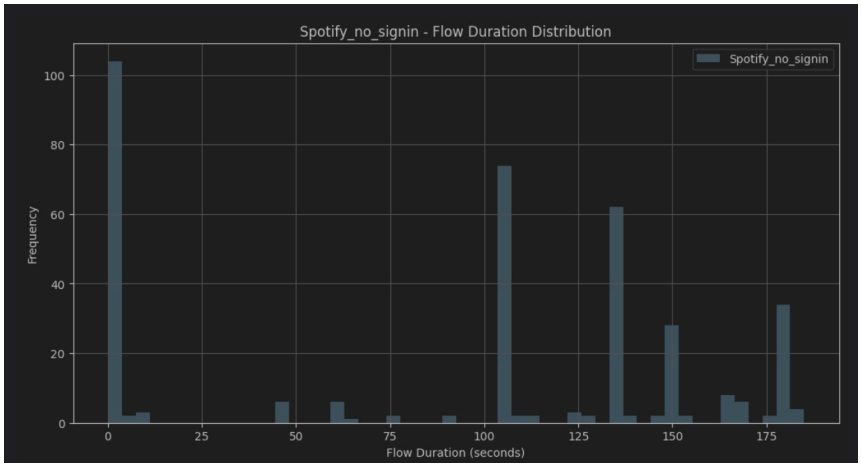
1. Source IP – The IP address of the sender
2. Destination IP – The IP address of the receiver
3. Source Port – The sender's port number
4. Destination Port – The receiver's port number
5. Protocol – TCP, UDP, ICMP, etc.

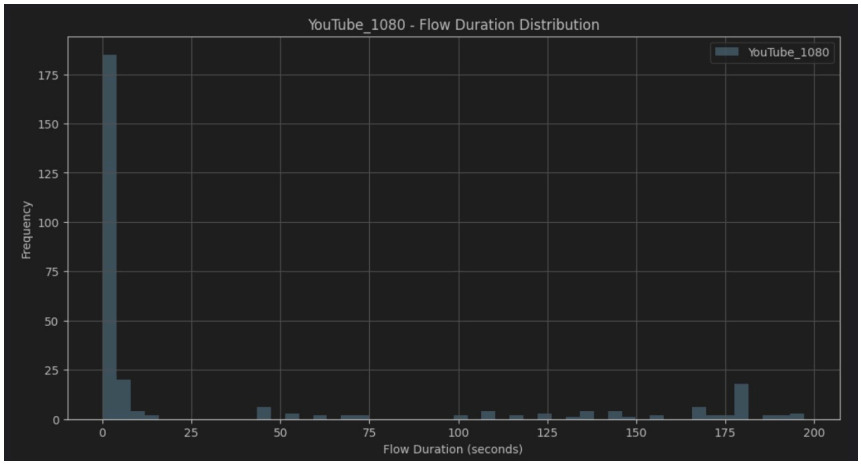
When plotting out flow lengths we look for flow and see how long in seconds it takes for a specific flow to end. A flow is considered finished when:

- For TCP: Connection closes (via FIN/ACK or RST packets).
- For UDP: Timeout occurs (since UDP is connectionless, a flow ends if no packets are seen for a set time).
- For ICMP: Single request-response (e.g., ping request/reply).

Timeouts:

- Active Timeout: If a flow is long-lived, it is split into chunks (e.g., every 5 minutes).
- Inactive Timeout: If no packets are seen for a duration (e.g., 30 seconds), the flow is closed.





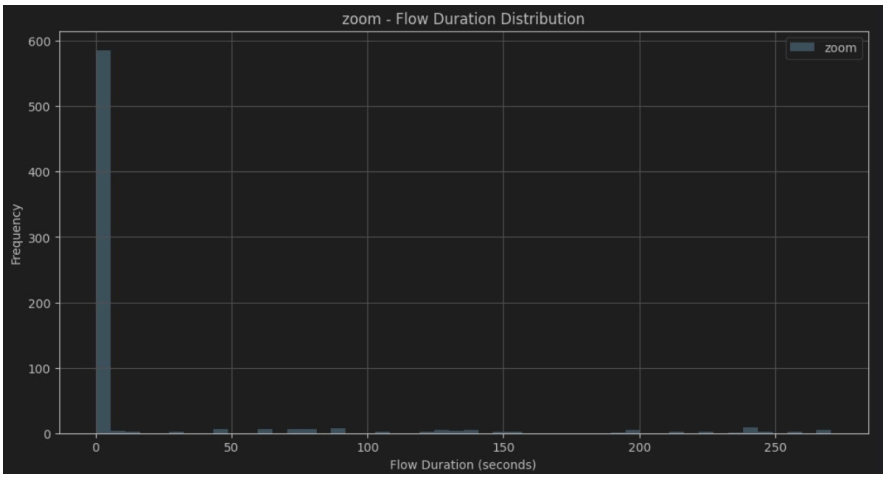
We can see from these graphs that Spotify and YouTube handle audio/video streaming very differently, spotify has big spikes while YouTube is more spread out (save for the spike at the beginning). This is because of the way the applications handle their data: YouTube uses a buffering system to provide the user with video streaming, while Spotify keeps an open connection to continuously stream the audio to the user. Spotify is able to stream the audio constantly instead of buffering like YouTube because audio uses considerably less bandwidth to transfer.

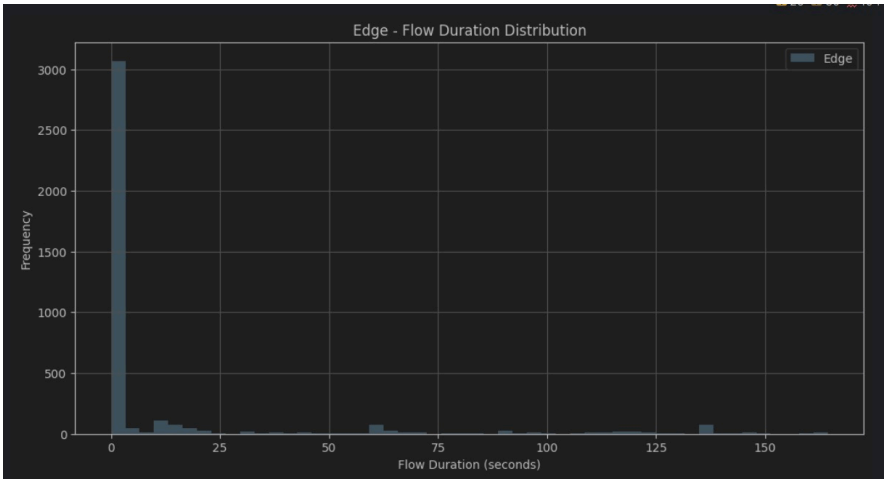
The buffering system means that the connection between YouTube and the client is rarely consistent and most connections are short, while the spotify approach creates large spikes

because of all the long connections that take place.

```
chrome: 26.80 seconds
Edge: 12.38 seconds
Spotify_no_signin: 91.64 seconds
Spotify_signin: 57.15 seconds
YouTube_240: 26.87 seconds
YouTube_720: 38.63 seconds
YouTube_1080: 36.52 seconds
zoom: 17.93 seconds
```

If we look at the other graphs we can see that most applications look similar to YouTube in terms of the graph – most of their connections are around the 0-second mark on the graph. This is because most of the traffic is short. If we look at the average flow time for each app we will see that compared to YouTube and Spotify the flow sizes are quite small. While Spotify and YouTube are services that require relatively large file sizes to be sent to a user, browsing and using Zoom focuses more on “bite-size” data.

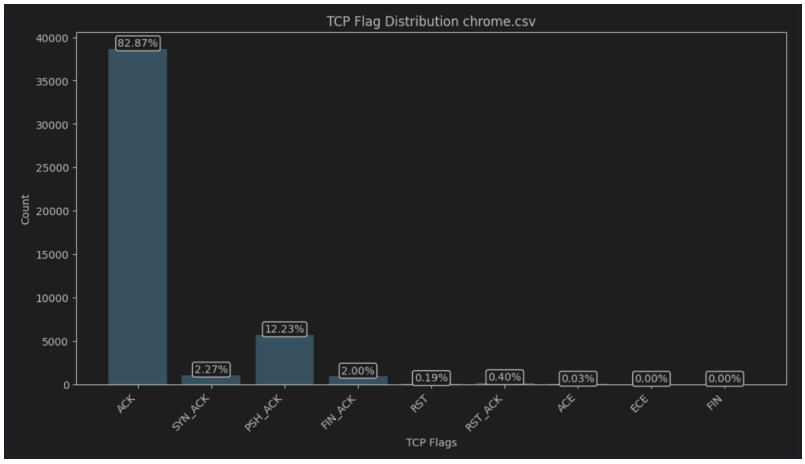




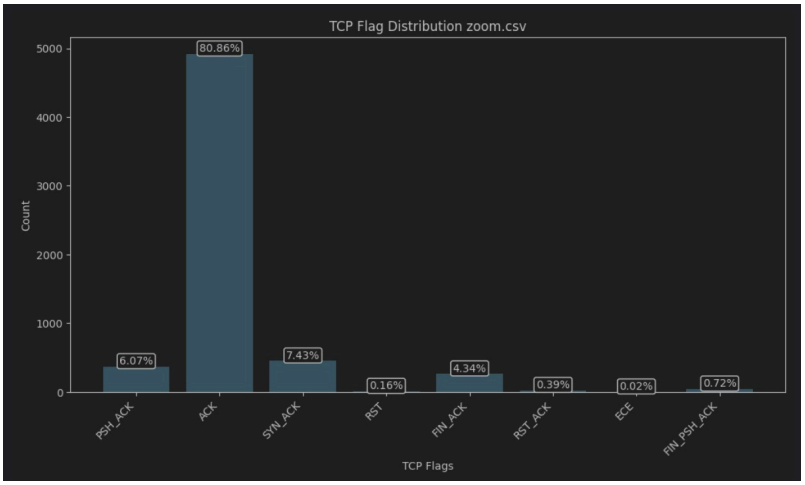
<u>TCP Flags</u>

Here we plotted how many flags of each appear in TCP connections. It's immediately obvious that in each graph the ACK flag appears the most. This is in the nature of the TCP, where ACK is used to convey acknowledgments and is used in almost every packet barring specific cases like SYN and RST packets.

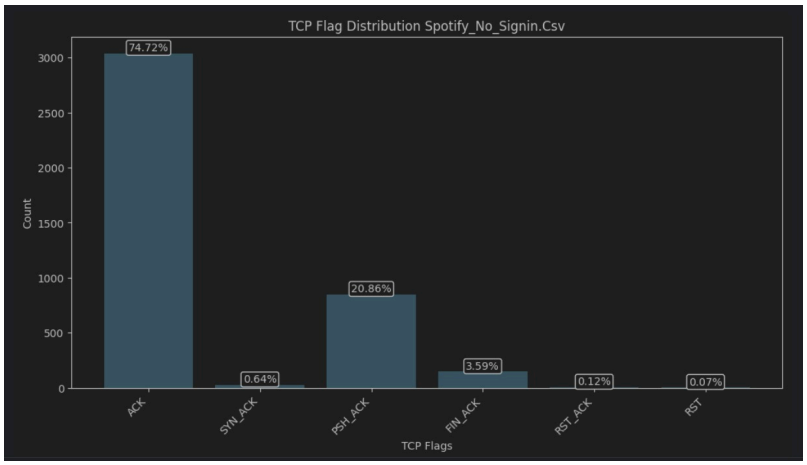
In the chrome graph, we can see a similar amount of SYN/ACK and FIN/ACK flags. This is evidence of a stable connection. When these flags are relatively even it means that connections are being opened and closed evenly.



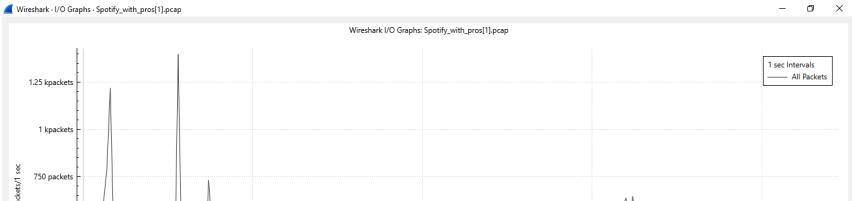
In the zoom graph, we can see the opposite, there are more SYN/ACK flags than FIN/ACK, meaning more connections being opened than being closed. It's worth mentioning that connections can also be closed with RST but even so, there are still more connection openings than closings. This is evidence of connections being left open for long periods of time in order to stream information from one place to another over time.



In the Spotify graph, we see the third option: more FIN/ACK than SYN/ACK. This is probably because of the nature of the recording. The connections being closed by the FIN/ACK flag were probably opened before the recording started. Because the recording does not include the login to Spotify, it probably missed the opening of those connections.



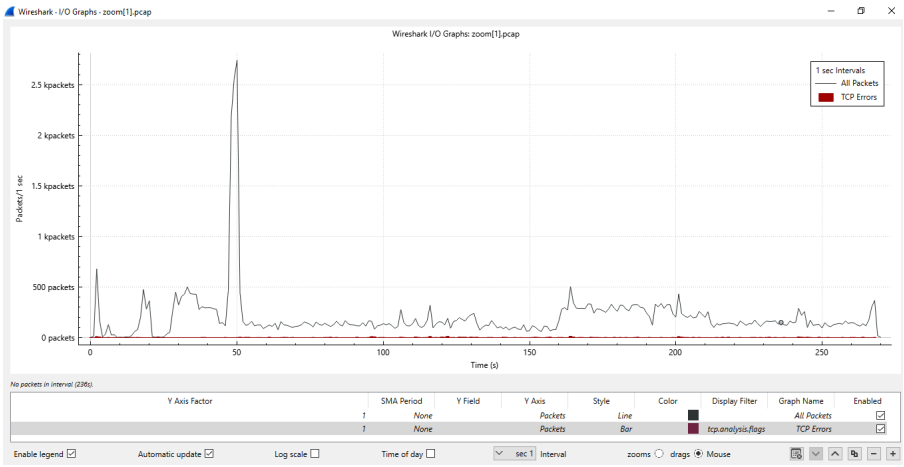
I/O Graphs from Wireshark



Spotify:

In this recording, we can see that Spotify allows a steady stream of packets in order to play its music. Spotify opens a connection and slowly downloads pieces of the song as it needs it. The big spike at the beginning is characteristic of logging into Spotify with Google.

Zoom:



One characteristic of this recording is that most of the time, Zoom does not have many large peaks.
At the beginning of the connection, there is some activity, but afterward, there are not many packets, with most staying below 500 at any given second.
This is because Zoom has significantly lower quality compared to YouTube, for example.
It is important to note that if the available bandwidth allows, the quality in Zoom—and consequently, the number of packets—will increase.

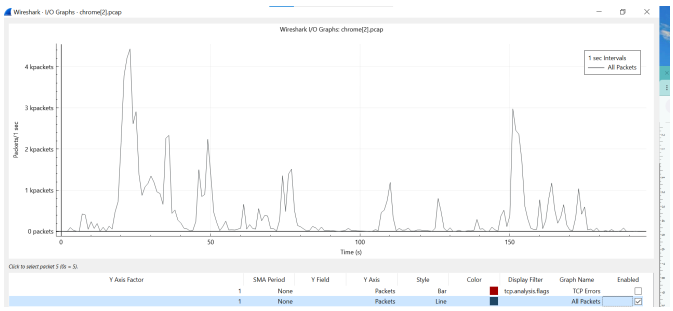
Chrome + Edge:

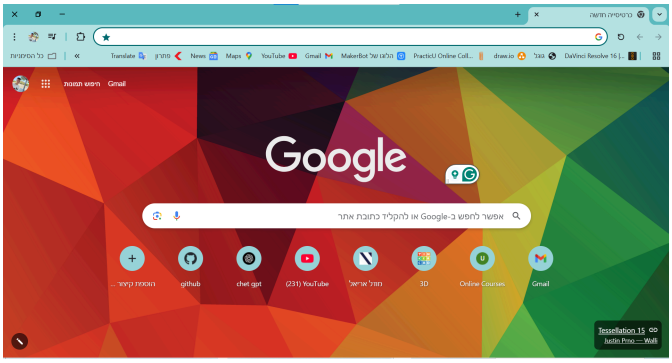
The graphs for Chrome and Edge are fundamentally similar. When a search is performed, there is a noticeable increase in the number of packets due to the large amount of data being loaded from various websites and advertisements.

Between searches, there is very little traffic because users are primarily scrolling through web pages. For example, if reading an article on Ynet, once the data has been loaded, there are very few additional packets until navigating to the next article.

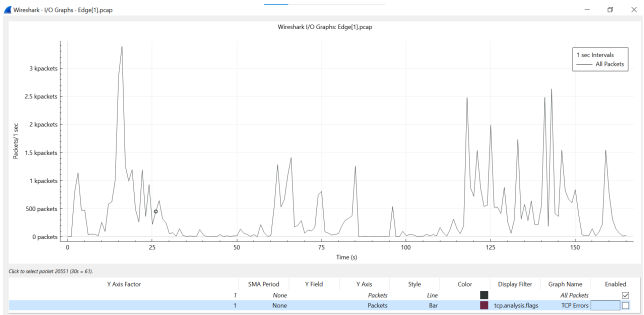
A key difference is that Edge loads content at startup, as it initially displays articles, unlike Chrome, which does not.

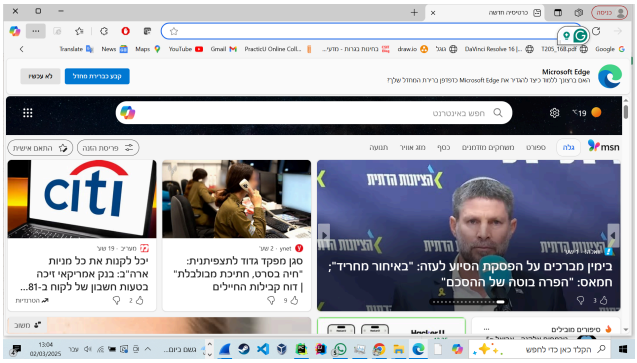
Chrome





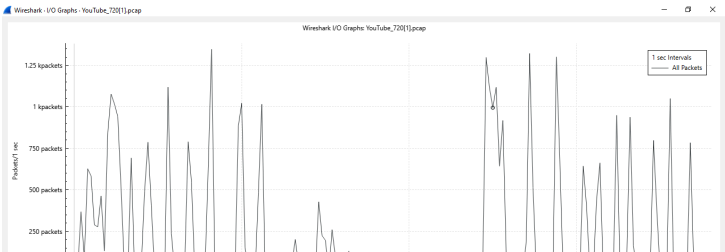
Edge





<u>YouTube:</u>

Several insights can be drawn from this graph. First, there are multiple large peaks with relatively short intervals between them. The gap in the middle



indicates that the video has already been loaded.

In other words, there is no ongoing data download at that point, making the graph appear static, even though the video is actively playing.

<u>Question 4:</u>

Examining Behavioral Patterns

By analyzing the graphs of the applications we used, we discovered that each application has its own unique behavioral signature, which consists of several key characteristics:

- Protocol type
- Packet transmission rate
- Packet size
- Pause durations between packet bursts
- Number of connections

We discovered that each application has a unique traffic signature, meaning that based on the various characteristics mentioned above, it is possible to classify the traffic and accurately associate it with the corresponding application.

Examples:

- **Packet Size** – Small requests from the client and variable-sized responses from the involved servers, as different objects are loaded, such as images, text, and redirects from other websites.

- **Protocol Type** – Most web browsing traffic is HTTPS, encrypted using TLS over a TCP connection, typically on port 443. An attacker could identify this by analyzing the port usage.

- **Message Transmission Rate** – The transmission rate is not particularly high. Many items are sent over a short period, followed by most interactions occurring within the client's browser until the user decides to perform another action, such as navigating to another page or website. This is characterized by short bursts of messages, followed by relatively long pauses, during which Keep Alive and ACK messages are sent.

Video Streaming on <u>YouTube</u>:

- **Packet Size:** The packet size varies depending on the video quality, but it is generally large.

- **Protocol Type:**
  The traffic is HTTPS, encrypted using TLS over a TCP connection, typically transmitted over port 443.
  In TLS records, there are usually multiple TLS records due to the large data size.

- **Message Transmission Rate:**
  When watching a YouTube video, the buffer loads a portion of the video, and once sufficiently filled, it stops downloading for a certain interval—measured at around 4 seconds in our sample—before resuming to download the next segment.
  The transmission rate is characterized by short bursts of large data packets occurring every few seconds.

- **Message Types:**
  The session is characterized by many TCP and TLS messages sent from the server, with numerous acknowledgment messages from the client.
  The TCP window size increases rapidly and reaches a large size quickly.

Audio Streaming on <u>Spotify</u>:

- **Packet Size:** Packets are relatively uniform in size, typically around 1,454 bytes (medium size).

- **Protocol Type:**
  The traffic is HTTPS, encrypted using TLS over a TCP connection, transmitted over port 443.

- **Message Transmission Rate:**
  When playing a song on Spotify, the entire song is downloaded as soon as the object is requested.
  This results in a single, short burst where most of the song data is transferred, followed by long pauses with only small data transmissions, such as Keep-Alive packets and ACKs.

  The next significant transmission likely occurs when the next song is loaded, approximately 3–5 minutes later.

- **Message Types:**
  The session consists of moderate amounts of TCP and TLS messages sent by the server and many acknowledgment messages from the client.
  After the song is downloaded, the client and server primarily exchange Keep Alive messages to maintain the connection.

Video Call on <u>Zoom</u>:

- **Packet Size:**
  Zoom uses small packets, despite transmitting both audio and video, because the call occurs in real time. The priority is to transmit data quickly rather than in large quantities.

- **Protocol Type:**

    - The traffic is primarily UDP for audio and video, using ports 8801–8810 for UDP-based video connections.
    - UDP is used to minimize latency and avoid delays in message transmission.
    - If UDP is unavailable, Zoom falls back to TCP, transmitting through HTTPS over port 443 using TLS.
- **Message Transmission Rate:**
  The transmission rate is continuous and high, as both sides are constantly exchanging data in real time, such as:

    - **Speaking to each other**
    - **Camera streaming**
    - **Screen sharing**

      This results in an extremely high message transmission rate.

- **TLS Packet Size:**
  The TLS packet size in Zoom is relatively small, typically between 30–100 bytes per TLS header.
  This is because, as mentioned earlier, most data is transmitted over UDP without encryption, prioritizing low latency and minimizing delays.

Searches in <u>Chrome Browser</u>:

Packet Size

The packet size in Chrome's network traffic varies based on the type of request and the amount of data returned from servers.

- When performing a search, small requests are sent to the search engine (e.g., Google), but responses may be larger, depending on the data retrieved.

Examples:

- **While typing in the search bar**, Chrome sends small requests for autocomplete suggestions. These packets are typically very small.
- **When loading a search results page**, large packets are received as more data is loaded.

In our analysis, the average sent packet size was 736.66 bytes, though actual packets may be significantly smaller or larger, depending on the data being exchanged.

Protocol Type

From our network traffic observations in Chrome, we found that:

- TCP is the dominant protocol, especially for loading traditional web pages.
- However, there is widespread use of QUIC, which improves speed and performance.

Message Transmission Rate

The data transmission rate in Chrome is very fast and depends on user interaction:

- **When typing in the search bar**, Chrome **sends real-time requests** to fetch autocomplete suggestions.
- **During a search**, data is **continuously transmitted** until the search results are fully received from the servers.
- **When navigating between pages or scrolling search results**, additional requests are sent to **load more content**.
- **Traffic pauses only when the user is inactive**, meaning no active searches or interactions are taking place.

How and to What Extent Can an Attacker Identify the Services in Use with the Following Data?

If an attacker has access to the following fields:

- Source IP Address
- Destination IP Address
- Source Port
- Destination Port
- Packet Size
- Packet Timing

They can classify messages based on the source and destination addresses and port numbers, allowing them to identify ongoing communications on the network.

How Can an Attacker Use This Data?

IP addresses and port numbers can help identify specific services with known addresses or by comparing them to an existing database of known addresses.
An attacker can also infer the service in use by analyzing the port number, even if the destination IP is unknown or not in their database.

For example, if a user is making a Zoom video call, the attacker may not recognize the destination IP, but if the traffic is passing through port 8801, they can reasonably infer that the service in use is Zoom.

Examples:

- IP Address 8.8.8.8 is associated with Google's DNS servers.
- Port 443 is used for HTTPS services, indicating secure web browsing.
- Port 8801 is commonly used by Zoom.
- Port 554 is used for live streaming services (RTSP).

Even with encrypted traffic, an attacker can infer service usage based on metadata alone, highlighting a potential privacy risk.

If an Attacker Lacks Access to These Fields

If the attacker does not have access to source IP, destination IP, source port, and destination port, they will face significant obstacles in classifying the traffic and associating it with specific services or conversations. However, this does not entirely prevent identification, just makes it harder.

After classifying the data, the attacker can still use packet sizes and packet timing to compare them with known application patterns. They can analyze traffic volume, message transmission rates, and patterns observed in previously recorded application traffic.

From packet size and timing, the attacker can also infer whether a message is a data acknowledgment (ACK) or an actual data transmission.

Machine Learning for Traffic Classification

An attacker can use machine learning algorithms to differentiate between traffic patterns and identify the type of service being used.

When these fields are available, the attacker can better categorize traffic, distinguish between sent and received messages and improve the accuracy of application identification.

When Only Packet Size and Timing Are Available

If the attacker has access only to packet size and timing, they can still attempt to infer the type of application, but they will not be able to classify the traffic based on sender, receiver, or port number.

- Traffic from multiple services may mix together, making it harder to isolate specific applications.
- To overcome this, the attacker can analyze burst patterns and check for consistent message sizes and known burst intervals seen in common applications.
- By examining ACK message frequency, they can estimate whether the service is real-time (e.g., Zoom) or streaming-based (YouTube).

Example:

- Zoom traffic can still be identified due to its consistent packet sizes, continuous transmission, and high message frequency.
- Streaming services like YouTube or Spotify, however, show bursts of traffic followed by pauses, making them distinguishable from real-time communication.
- Web browsing traffic would be much harder to pinpoint without IP and port information, as the attacker would lack site-specific details.

How to Mitigate This Attack?

1. Padding Messages
    - TLS supports encrypted padding, which makes it harder for the attacker to determine the original message size.
    - By adding randomized padding, message size patterns become less predictable.
2. Modifying Transmission Timing
    - Randomizing message intervals or adding artificial delays can disrupt predictable traffic patterns.
    - This makes it more difficult for an attacker to match timing to known applications.
3. Using a VPN
    - A VPN routes all traffic through a secure server, masking the original destination.
    - The attacker will only see encrypted traffic going to the VPN server, rather than identifying the actual service being used.

By applying these defensive measures, users can significantly reduce the ability of attackers to infer network activity based on traffic metadata.

After Our Analysis, We Implemented a K-Nearest Neighbors (KNN) Machine Learning Model

https://github.com/dviropa/Communication-Networks-Final-Project

Unfortunately, the model's accuracy is low, and one of the main reasons for this is a lack of data. If we had a larger and more diverse dataset, the model would be able to learn better and achieve significantly higher accuracy.

About the Model:

The K-Nearest Neighbors (KNN) model is a proximity-based algorithm. Unlike neural networks or regression models, it does not "learn" in the traditional sense but instead stores all training examples and uses them to classify new instances.

The principle is simple:

1. When a new sample arrives, KNN measures its distance from all existing examples.
2. It identifies the k closest examples—i.e., the k nearest neighbors.
3. The model assigns the label of the new sample based on the majority class of the closest neighbors.

One of the key advantages of KNN is its flexibility—the value of K can be adjusted based on specific needs, allowing for optimal accuracy depending on the dataset and model performance.

For example:
If k = 3 and the three closest neighbors are Spotify, Spotify, and YouTube,
the model will classify the new sample as Spotify, as it is the most frequent class among the neighbors.

If k = 5 and the three closest neighbors are Spotify, Spotify, YouTube, YouTube and YouTube
the model will classify the new sample as YouTube, as it is the most frequent class among the neighbors.

Bonus:

We conducted a recording of Spotify, during which we occasionally performed a general search in Google Chrome.

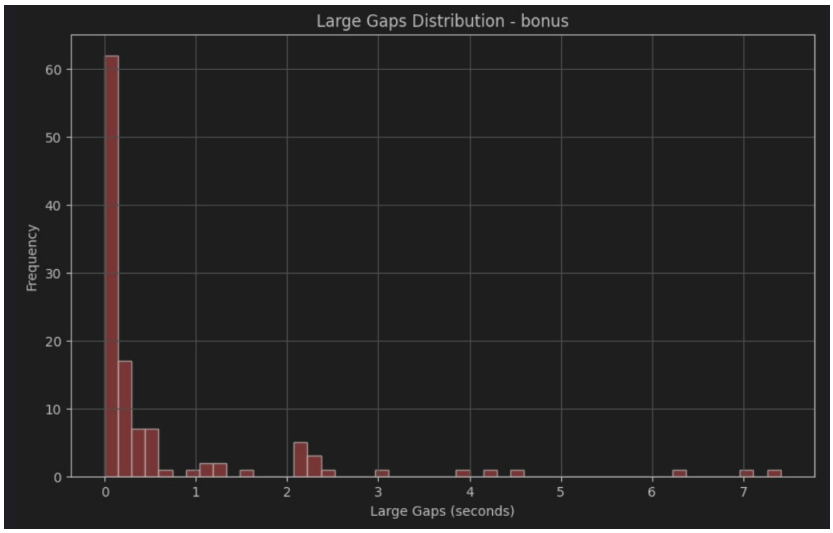
- Recording name: Bonus.pcap
- Average packet size: 708.45 bytes

After running the KNN model with k = 4, the model classified the traffic as a general search.

Ultimately, our model is not designed to predict that it will detect both Spotify and Chrome simultaneously, as it is a relatively simple model and is also limited by the amount of available data.

However, it is very interesting that the model was able to predict the search activity in Google, suggesting that the search pattern was dominant in the data.

This indicates that the new recording resembles search traffic more than Zoom traffic in terms of network behavior.

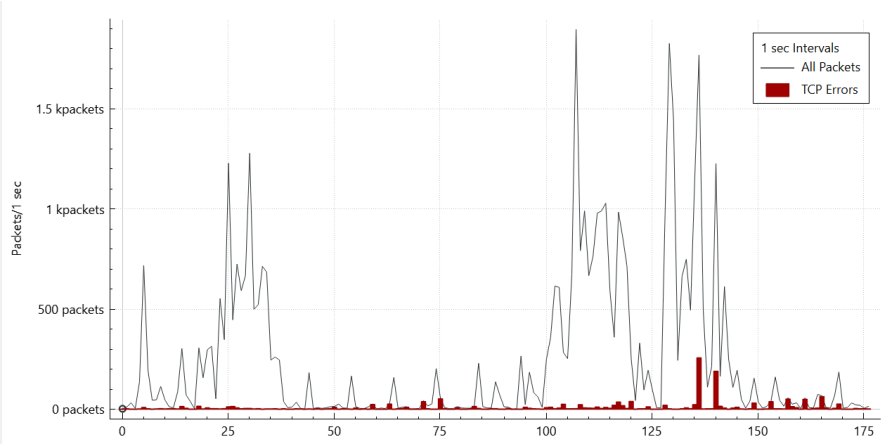


Graph combination and analysis:

It can be observed that this graph resembles the pattern of a general internet search, as there are occasional peaks indicating data-intensive loading typically associated with searches. However, there are also indicators of Spotify traffic within the data.

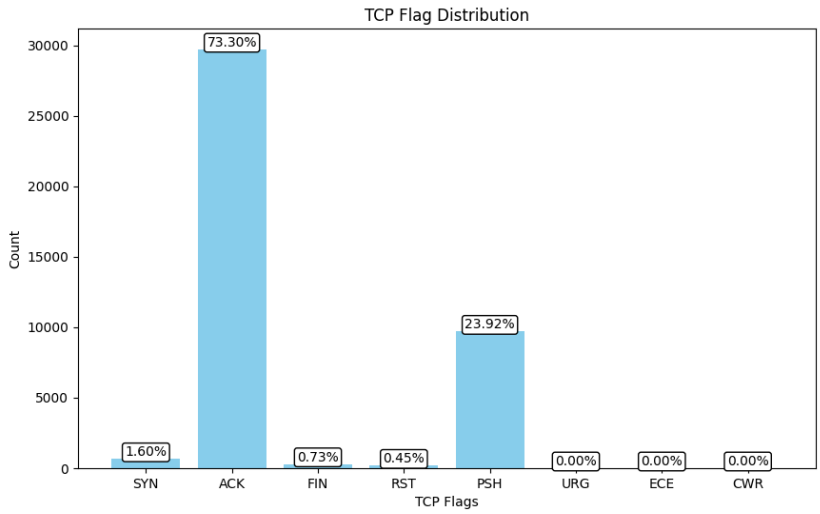
For example, in Spotify, there is typically a large initial data load, followed by relatively small fluctuations as the streaming continues.

In our case, this Spotify pattern is somewhat overshadowed by the intermittent search activity, making it less distinguishable but still present in the data.



This graph displays the distribution of “large gaps” (Inter-Bin Gaps), representing the time intervals between consecutive data packets in the network.

Most packets arrive with very small gaps, indicating a continuous data flow, which is expected in Spotify music streaming. Additionally, larger gaps (1–7 seconds) appear but with low frequency, suggesting web browsing activity, where page requests or irregular data loading patterns occur.

The low presence of SYN, FIN, and RST flags suggests a persistent connection with minimal interruptions, typical of streaming services.

This structure reinforces the hypothesis that the captured traffic includes simultaneous Spotify music streaming and web browsing activity.