# NETWORK MONITORING PROJECT REPORT

## a) Introduction

- This network monitoring project utilizes the ***scapy*** python library to capture network packets and analyze them at different layers*(Ethernet, IP, TCP, and UDP).*

- The scapy python library extracts important information about network traffic, including source and destination addresses, packet sizes, and protocol types.

- This project aims to create a system that captures network packets, logs relevant information and calculates important network metrics such as ***throughput***, ***latency***, ***protocol usage*** and also tracks unique IP addresses and MAC addresses, and visualizes this data in graphs to aid in the analysis.

## b) Project Outline

### *1. Packet Capture and Parsing*

- Using the **Scapy** library, packets are captured and parsed across multiple layers: Ethernet, IP, TCP, and UDP.

- Essential packet details (source/destination addresses, protocol type, packet size) are logged for analysis.

### *2. Logging System*

- A logging mechanism that stores each packet's details in a file (***network_events.log*** ) with a timestamp is created.
- The log entries include:
    - *Timestamp*
    - *Protocol (Ethernet, IP, TCP, UDP)*
    - *Source and destination addresses*
    - *Packet size*

### *3. Throughput Calculation:*

- Throughput is calculated for each protocol (Ethernet, IP, TCP, UDP) at an interval of 10 seconds.

• The throughput is displayed in **bits per second (bps)** and data stored in a dictionary for plotting.)

### *4. Latency Measurement:*

- Latency is calculated for each TCP and UDP connection by recording timestamps for request and response packets. .

- The latency is computed as the difference between the response and request timestamps in milliseconds(time it takes data to travel from the source to the destination)

### 5. *Network Metrics Calculation:*

Network metrics such as the Average packet size,Total packets captured per protocol,Rate of new connections (connections over time are tracked and displayed in real-time updating every 30 seconds.

### 6. *Real-Time Statistics Display and Analysis:*

Summarized Statistics such as the total number of connections per protocol,Average packet size per protocol, unique IP addresses, and unique MAC addresses are tracked and displayed in real-time updating every 30 seconds

### 7. *Visualization*:

- The following graphs are generated and saved as .png files:
    1. ***Throughput Over Time***: Line plot of throughput (bps) for each protocol over time.

2. ***Latency Distribution***: Histogram showing latency across all connections.
3. ***Protocol Usage***: Bar chart displaying the number of packets per protocol and unique counts of IP and MAC addresses.

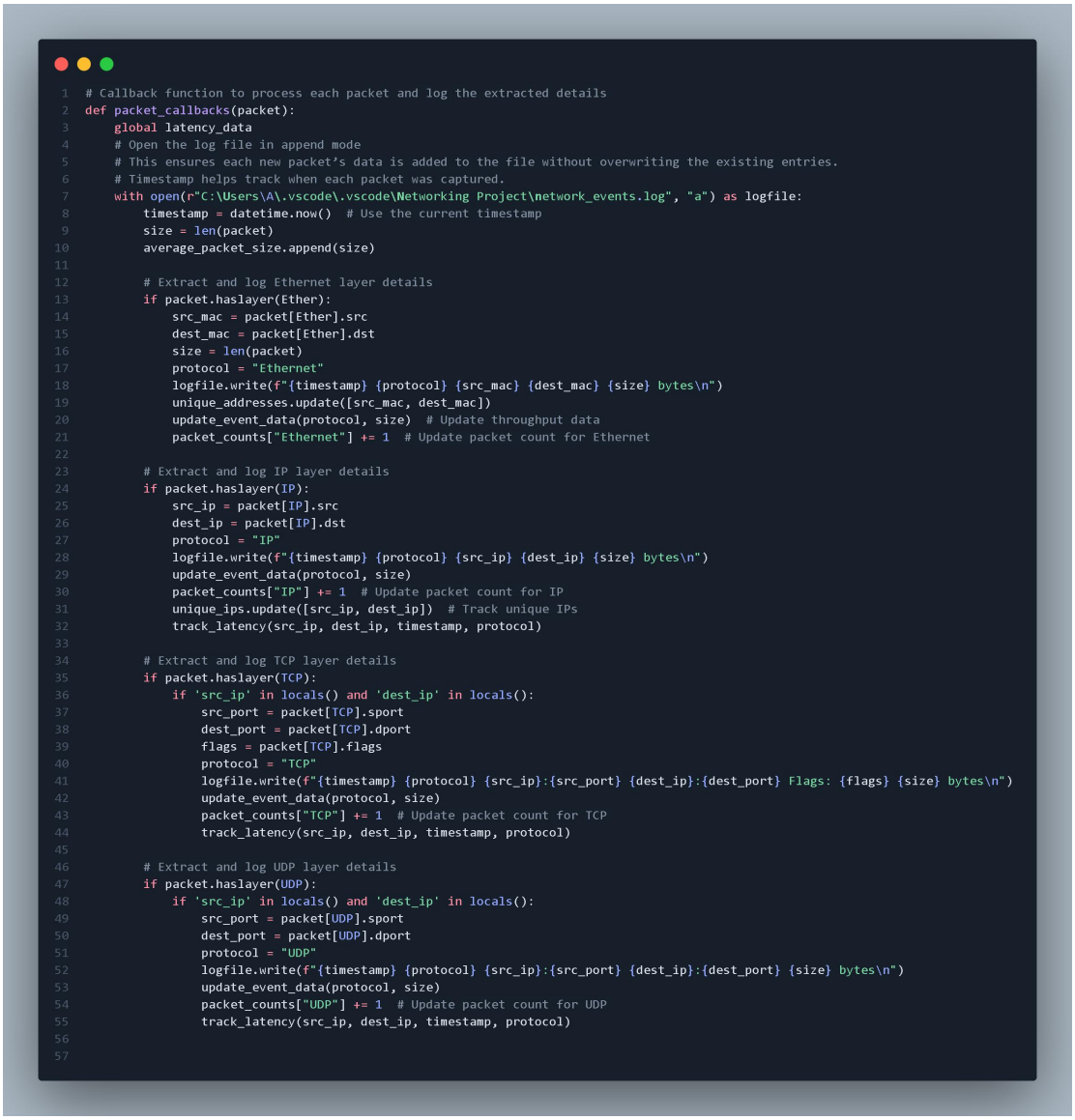
**8. Graceful Termination**:

- The program supports graceful termination on pressing **Ctrl+C**. Upon termination, the program outputs final statistics and saves the plots.

# c)Functions Used

## 1. packet_callbacks()

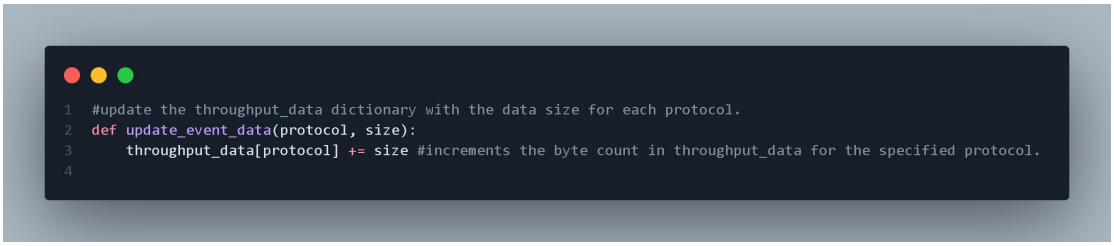
- This function is called each time a packet is captured by **Scapy**.
- This callback function extracts relevant data from each packet, logs it to a file, and updates counters for throughput and latency.



```python
# Callback function to process each packet and log the extracted details
def packet_callbacks(packet):
    global latency_data
    # Open the log file in append mode
    # This ensures each new packet's data is added to the file without overwriting the existing entries.
    # Timestamp helps track when each packet was captured.
    with open(r"C:\Users\A\.vscode\.vscode\Networking Project\network_events.log", "a") as logfile:
        timestamp = datetime.now()  # Use the current timestamp
        size = len(packet)
        average_packet_size.append(size)

        # Extract and log Ethernet layer details
        if packet.haslayer(Ether):
            src_mac = packet[Ether].src
            dest_mac = packet[Ether].dst
            size = len(packet)
            protocol = "Ethernet"
            logfile.write(f"{timestamp} {protocol} {src_mac} {dest_mac} {size} bytes\n")
            unique_addresses.update([src_mac, dest_mac])
            update_event_data(protocol, size)  # Update throughput data
            packet_counts["Ethernet"] += 1  # Update packet count for Ethernet

        # Extract and log IP layer details
        if packet.haslayer(IP):
            src_ip = packet[IP].src
            dest_ip = packet[IP].dst
            protocol = "IP"
            logfile.write(f"{timestamp} {protocol} {src_ip} {dest_ip} {size} bytes\n")
            update_event_data(protocol, size)
            packet_counts["IP"] += 1  # Update packet count for IP
            unique_ips.update([src_ip, dest_ip])  # Track unique IPs
            track_latency(src_ip, dest_ip, timestamp, protocol)

        # Extract and log TCP layer details
        if packet.haslayer(TCP):
            if 'src_ip' in locals() and 'dest_ip' in locals():
                src_port = packet[TCP].sport
                dest_port = packet[TCP].dport
                flags = packet[TCP].flags
                protocol = "TCP"
                logfile.write(f"{timestamp} {protocol} {src_ip}:{src_port} {dest_ip}:{dest_port} Flags: {flags} {size} bytes\n")
                update_event_data(protocol, size)
                packet_counts["TCP"] += 1  # Update packet count for TCP
                track_latency(src_ip, dest_ip, timestamp, protocol)

        # Extract and log UDP layer details
        if packet.haslayer(UDP):
            if 'src_ip' in locals() and 'dest_ip' in locals():
                src_port = packet[UDP].sport
                dest_port = packet[UDP].dport
                protocol = "UDP"
                logfile.write(f"{timestamp} {protocol} {src_ip}:{src_port} {dest_ip}:{dest_port} {size} bytes\n")
                update_event_data(protocol, size)
                packet_counts["UDP"] += 1  # Update packet count for UDP
                track_latency(src_ip, dest_ip, timestamp, protocol)


```

## 2. update_event_data()

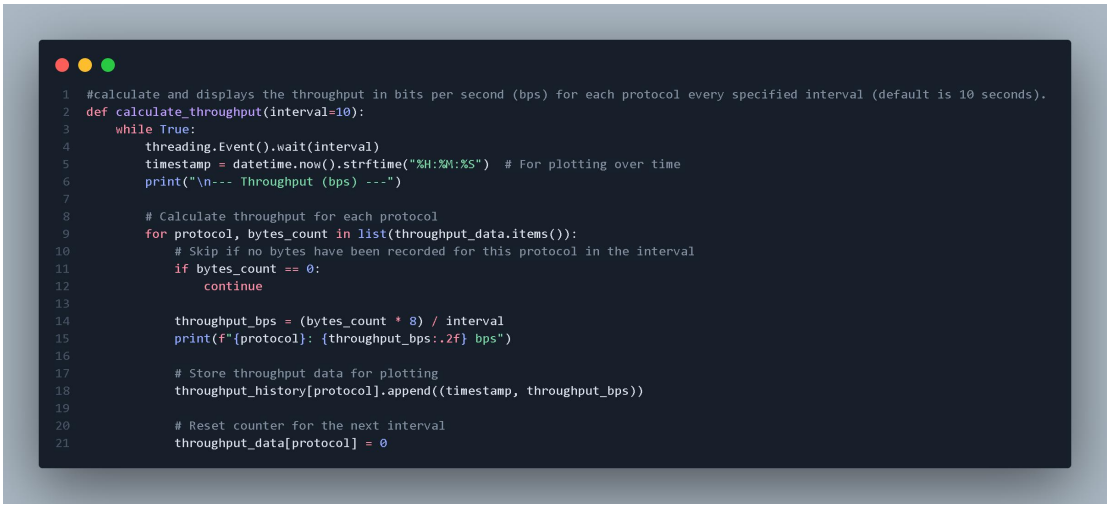
- This function updates the throughput data for each protocol, incrementing the byte count for the protocol every time a packet is processed.



```python
#update the throughput_data dictionary with the data size for each protocol.
def update_event_data(protocol, size):
    throughput_data[protocol] += size #increments the byte count in throughput_data for the specified protocol.

```

## 3. calculate_throughput()

- Calculates throughput for each protocol (Ethernet, IP, TCP, UDP) every 10 seconds in **bits per second (bps)**.
- Stores throughput data for plotting over time.



```python
#calculate and displays the throughput in bits per second (bps) for each protocol every specified interval (default is 10 seconds).
def calculate_throughput(interval=10):
    while True:
        threading.Event().wait(interval)
        timestamp = datetime.now().strftime("%H:%M:%S")  # For plotting over time
        print("\n--- Throughput (bps) ---")

        # Calculate throughput for each protocol
        for protocol, bytes_count in list(throughput_data.items()):
            # Skip if no bytes have been recorded for this protocol in the interval
            if bytes_count == 0:
                continue

            throughput_bps = (bytes_count * 8) / interval
            print(f"{protocol}: {throughput_bps:.2f} bps")

            # Store throughput data for plotting
            throughput_history[protocol].append((timestamp, throughput_bps))

            # Reset counter for the next interval
            throughput_data[protocol] = 0
```

## 4. track_latency()

- Tracks the start and end timestamps for each TCP and UDP connection to measure latency.
- Computes the latency in milliseconds by subtracting the request timestamp from the response timestamp.

```python
# Function to track latency
def track_latency(src_ip, dest_ip, timestamp, protocol):
    conn_key = (src_ip, dest_ip, protocol)
    if conn_key not in latency_data:
        # Mark the beginning of a new connection
        latency_data[conn_key] = {"start": timestamp}
    else:

    # Calculate latency if end timestamp is available
        latency_data[conn_key]["end"] = timestamp
        latency = (latency_data[conn_key]["end"] - latency_data[conn_key]["start"]).total_seconds() * 1000  # ms
        protocol_latencies[protocol].append(latency)  # Store latency by protocol
        all_latencies.append(latency)  # Store in global list for final statistics
        del latency_data[conn_key]  # Remove the connection entry after calculation

```

## 5. calculate_latency()

- Computes the average latency for each protocol and displays it every 10 seconds.
- Calculates the overall average latency across all connections.

```python
def calculate_latency(latency_interval=10):
    while not stop_event.is_set():
        stop_event.wait(latency_interval)
        if stop_event.is_set():
            break

        # Display protocol-specific average latencies
        for protocol, latencies in protocol_latencies.items():
            avg_protocol_latency = sum(latencies) / len(latencies) if latencies else 0
            print(f"\n--- Latency for {protocol} ---")
            print(f"Average Latency: {avg_protocol_latency:.2f} ms")
            # Clear latencies for next interval
            protocol_latencies[protocol].clear()

        # Calculate and display overall average latency
        total_latency = sum(all_latencies)
        count = len(all_latencies)
        overall_avg_latency = total_latency / count if count > 0 else 0
        print("\n--- Overall Latency ---")
        print(f"Average Latency across all protocols: {overall_avg_latency:.2f} ms")

```

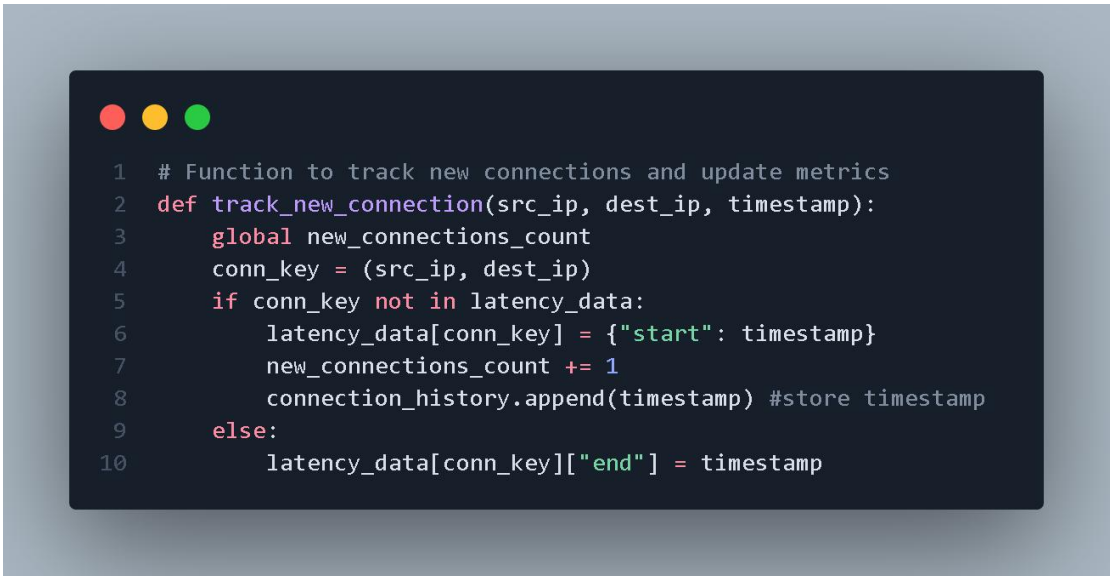
## 6. display_real_time_stats()

- Displays real-time statistics, including unique IP addresses, unique MAC addresses, packet counts per protocol, and average packet size every 30 seconds.

```python
# Real-time statistics display every 30 seconds and network metrics display
def display_real_time_stats():
    while not stop_event.is_set():
        stop_event.wait(30)  # Display stats every 30 seconds
        if stop_event.is_set():
            break
        avg_packet_size = sum(average_packet_size) / len(average_packet_size) if average_packet_size else 0

        avg_packet_size_per = {protocol: throughput_data[protocol] / packet_counts[protocol]
                               for protocol in packet_counts if packet_counts[protocol] > 0}
        new_connection_rate = calculate_new_connection_rate()
        print("\n---------- Network Metrics --------------")

        #Network Metrics
        print(f"Average Packet Size: {avg_packet_size:.2f} bytes")
        print(f"Packets per Protocol: {dict(packet_counts)}")
        print(f"Rate Of New Connections: {new_connection_rate:.2f} connections/second")

        #Real-time Statistics
        print("\n---------- Real-Time Statistics ----------")
        print("Connections per Protocol:", dict(packet_counts))
        print("Average Packet Size per Protocol:", {k: f"{v:.2f} bytes" for k, v in avg_packet_size_per.items()})
        print(f"Unique MAC Addresses: {len(unique_addresses)}")
        print(f"Unique IPs: {len(unique_ips)}")

```

## 7. track_new_connection()

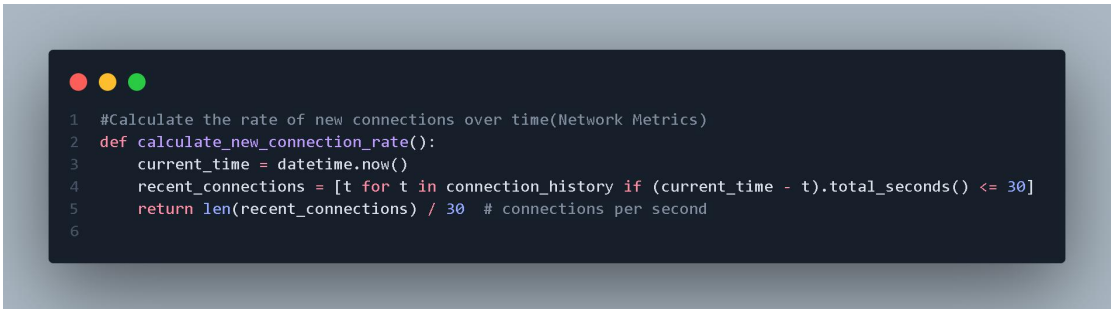
- Designed to track new connections between source and destination IP addresses and update network metrics related to connection events.



```python
# Function to track new connections and update metrics
def track_new_connection(src_ip, dest_ip, timestamp):
    global new_connections_count
    conn_key = (src_ip, dest_ip)
    if conn_key not in latency_data:
        latency_data[conn_key] = {"start": timestamp}
        new_connections_count += 1
        connection_history.append(timestamp) #store timestamp
    else:
        latency_data[conn_key]["end"] = timestamp
```

## 8. calculate_new_connection_rate()

- This function calculates the rate of new connections per second within the past 30 seconds. This rate is useful for monitoring network activity and identifying periods of increased connection establishment, which may indicate network load spikes or potential anomalies.



```python
#Calculate the rate of new connections over time(Network Metrics)
def calculate_new_connection_rate():
    current_time = datetime.now()
    recent_connections = [t for t in connection_history if (current_time - t).total_seconds() <= 30]
    return len(recent_connections) / 30  # connections per second

```

## 8. plot_throughput()

- Plots the throughput data over time for each protocol using **matplotlib**.

```python
# Throughput for each protocol over time(line plot)
def plot_throughput():
    if not throughput_history:
        print("No throughput data to plot.") #Debugging
        return
    plt.figure(figsize=(12, 6))
    for protocol, data in throughput_history.items():
        if data:
            times, throughput_values = zip(*data)
            plt.plot(times, throughput_values, label=f"{protocol} Throughput (bps)")
    plt.xlabel("Time")
    plt.ylabel("Throughput (bps)")
    plt.title("Network Throughput Over Time")
    if len(throughput_history) > 0:
        plt.legend()
    plt.xticks(rotation=45)
    plt.tight_layout()
    plt.savefig(r"C:\Users\A\.vscode\.vscode\Networking Project\throughput_over_times.png")
    plt.close()

```

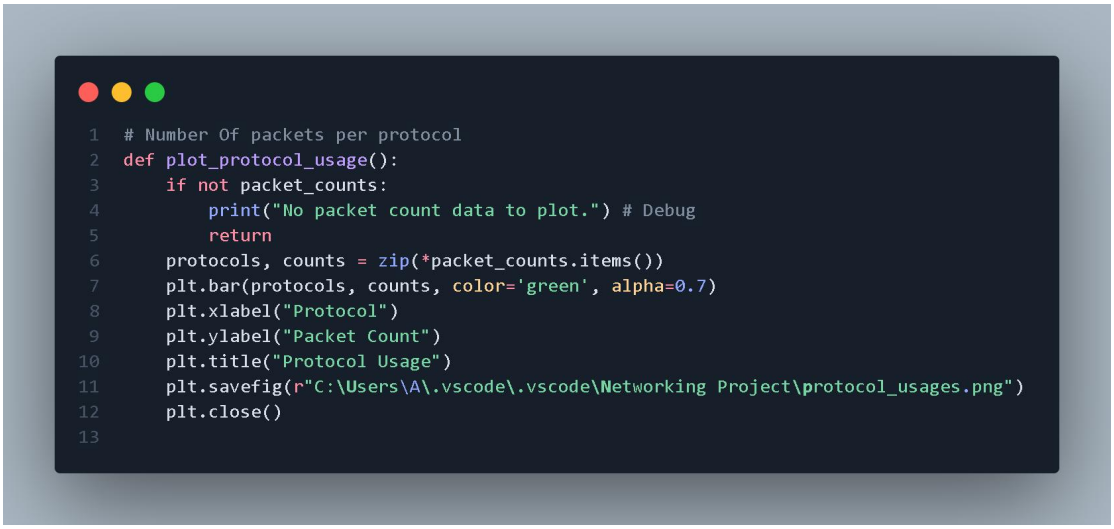
## 9. plot_latency_distribution()

- Plots the distribution of latency using a histogram.

```python
# Latency Distribution across all connections
def plot_latency_distribution():
    if not all_latencies:
        print("No latency data to plot.")
        return
    plt.hist(all_latencies, bins=20, color='blue', alpha=0.7)
    plt.xlabel("Latency (ms)")
    plt.ylabel("Frequency")
    plt.title("Latency Distribution")
    plt.savefig(r"C:\Users\A\.vscode\.vscode\Networking Project\latency_distribution.png")
    plt.close()
```

## 10. plot_protocol_usage()

- Plots the number of packets per protocol using a bar chart, showing the usage of Ethernet, IP, TCP, and UDP.



```python
# Number Of packets per protocol
def plot_protocol_usage():
    if not packet_counts:
        print("No packet count data to plot.") # Debug
        return
    protocols, counts = zip(*packet_counts.items())
    plt.bar(protocols, counts, color='green', alpha=0.7)
    plt.xlabel("Protocol")
    plt.ylabel("Packet Count")
    plt.title("Protocol Usage")
    plt.savefig(r"C:\Users\A\.vscode\.vscode\Networking Project\protocol_usages.png")
    plt.close()

```

## 11. if __name__ == "__main__":

- This code snippet is the main execution block for running your **network monitoring program**. It initializes and starts threads for various tasks, initiates packet sniffing,handles **CTRL+C** interrupts to gracefully terminate the program, stops packet capture, and generates final statistics and plots.

```python
if __name__ == "__main__":

    try:
        # Start the threads for each function
        throughput_thread = threading.Thread(target=calculate_throughput, args=(interval,), daemon=True)
        throughput_thread.start()

        stats_thread = threading.Thread(target=display_real_time_stats, daemon=True)
        stats_thread.start()

        latency_thread = threading.Thread(target=calculate_latency, args=(latency_interval,), daemon=True)
        latency_thread.start()

        # Run sniff in the main thread
        print("Starting packet sniffing... Press Ctrl+C to stop.")
        sniff(prn=packet_callbacks, store=0, stop_filter=lambda x: stop_event.is_set())

    except KeyboardInterrupt:
        # Trigger stop event to terminate all threads
        print("\nGracefully terminating...")
        stop_event.set()

    # Save the plots
    plot_throughput()
    plot_latency_distribution()
    plot_protocol_usage()

    # Display Final Statistics
    avg_packet_size_val = sum(average_packet_size) / len(average_packet_size) if average_packet_size else 0
    print(f"\n---------Final Statistics------------")
    print(f"Total Connections: {len(latency_data)}")
    print(f"Unique Addresses: {len(unique_addresses)}")
    print(f"Unique IPs: {len(unique_ips)}")
    print(f"Average Packet Size: {avg_packet_size_val:.2f} bytes")
```

# d) Results

The results of the project are presented in both real-time statistics and graphical visualizations.

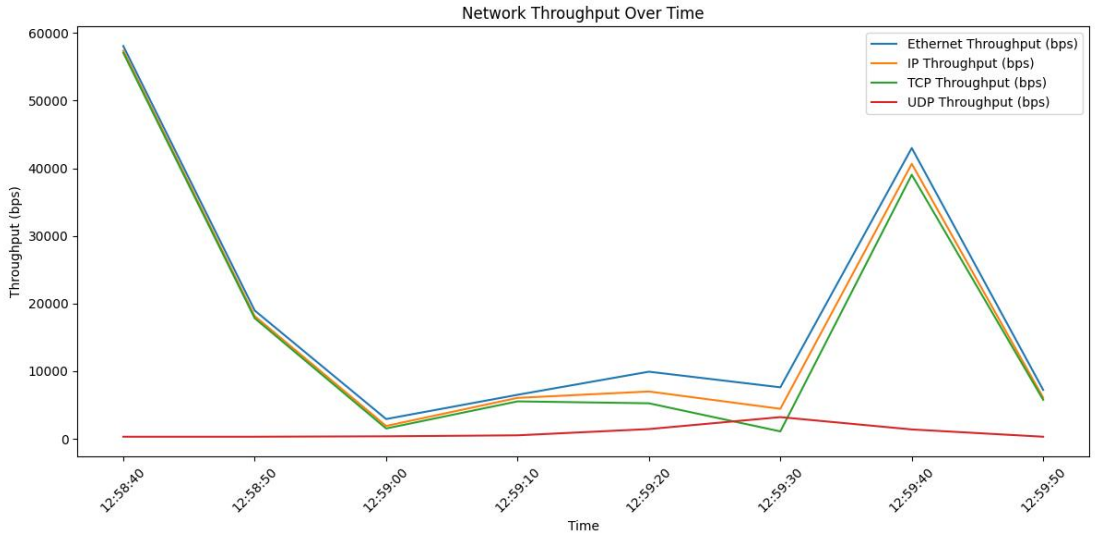
## 1. ***Sample Outputs***(Throughput, Latency, Network Metrics, Real-Time Statistics)

```
--- Throughput (bps) ---
Ethernet: 7231.20 bps
IP: 6080.00 bps
TCP: 5768.80 bps

--- Latency for IP ---UDP: 311.20 bps

Average Latency: 673.90 ms

--- Latency for TCP ---
Average Latency: 673.90 ms

--- Latency for UDP ---
Average Latency: 0.00 ms

--- Overall Latency ---
Average Latency across all protocols: 559.51 ms

---------- Network Metrics --------------
Average Packet Size: 233.81 bytes
Packets per Protocol: {'Ethernet': 832, 'IP': 643, 'TCP': 592, 'UDP': 41}
Rate Of New Connections: 0.00 connections/second

---------- Real-Time Statistics ----------
Connections per Protocol: {'Ethernet': 832, 'IP': 643, 'TCP': 592, 'UDP': 41}
Average Packet Size per Protocol: {'Ethernet': '2.09 bytes', 'IP': '1.06 bytes', 'TCP': '0.00 bytes', 'UDP': '16.61 bytes'}Unique MAC Addresses: 35
Unique IPs: 23

---------Final Statistics-----------
Total Connections: 24
Unique Addresses: 35
Unique IPs: 23
Average Packet Size: 233.81 bytes
```

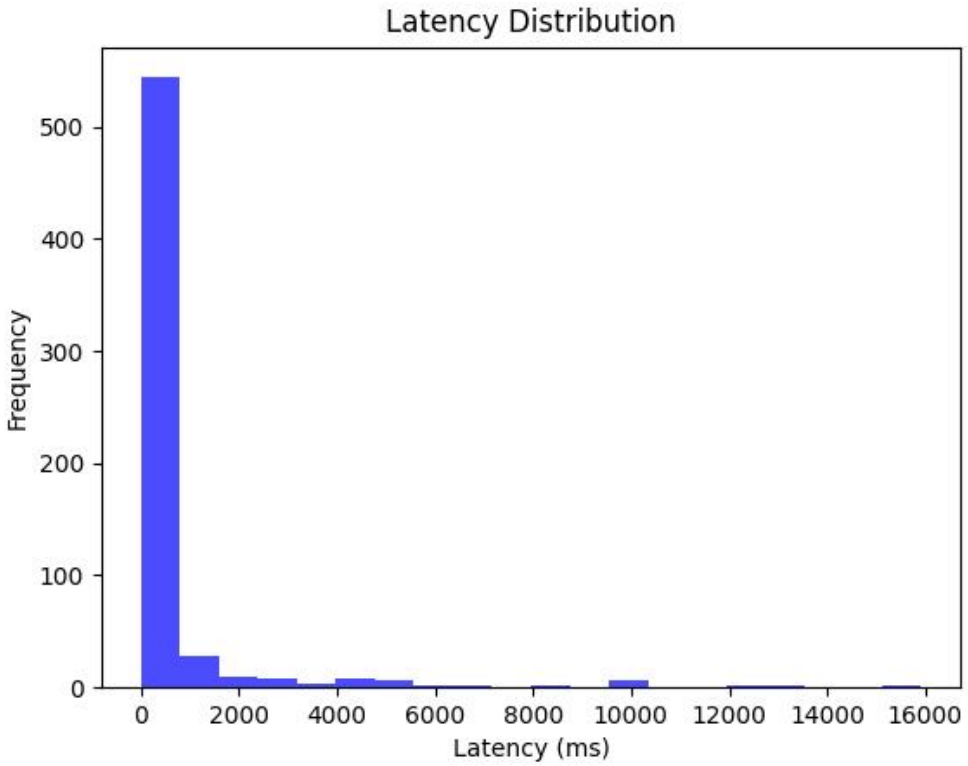
## 2. *Sample logs*

```
2024-11-12 12:58:30.112349 Ethernet e8:2a:44:d5:c7:97 54:7f:ee:5c:20:01 655 bytes
2024-11-12 12:58:30.112349 IP 172.28.164.99 185.252.220.114 655 bytes
2024-11-12 12:58:30.112349 TCP 172.28.164.99:51109 185.252.220.114:443 Flags: PA 655 bytes
2024-11-12 12:58:30.164859 Ethernet 54:7f:ee:5c:20:01 e8:2a:44:d5:c7:97 665 bytes
2024-11-12 12:58:30.164859 IP 185.252.220.114 172.28.164.99 665 bytes
2024-11-12 12:58:30.164859 TCP 185.252.220.114:443 172.28.164.99:51109 Flags: PA 665 bytes
2024-11-12 12:58:30.168596 Ethernet e8:2a:44:d5:c7:97 54:7f:ee:5c:20:01 54 bytes
2024-11-12 12:58:30.168596 IP 172.28.164.99 185.252.220.114 54 bytes
2024-11-12 12:58:30.168596 TCP 172.28.164.99:51109 185.252.220.114:443 Flags: A 54 bytes
2024-11-12 12:58:30.175356 Ethernet e8:2a:44:d5:c7:97 54:7f:ee:5c:20:01 132 bytes
2024-11-12 12:58:30.175356 IP 172.28.164.99 185.252.220.114 132 bytes
2024-11-12 12:58:30.175356 TCP 172.28.164.99:51109 185.252.220.114:443 Flags: PA 132 bytes
2024-11-12 12:58:30.235953 Ethernet 54:7f:ee:5c:20:01 e8:2a:44:d5:c7:97 66 bytes
2024-11-12 12:58:30.235953 IP 185.252.220.114 172.28.164.99 66 bytes
2024-11-12 12:58:30.235953 TCP 185.252.220.114:443 172.28.164.99:51109 Flags: A 66 bytes
2024-11-12 12:58:30.283949 Ethernet e8:2a:44:d5:c7:97 54:7f:ee:5c:20:01 205 bytes
2024-11-12 12:58:30.283949 IP 172.28.164.99 185.252.220.114 205 bytes
2024-11-12 12:58:30.283949 TCP 172.28.164.99:51109 185.252.220.114:443 Flags: PA 205 bytes
2024-11-12 12:58:30.330113 Ethernet b6:8b:a9:dc:cb:c6 e8:2a:44:d5:c7:97 66 bytes
2024-11-12 12:58:30.330113 IP 185.252.220.114 172.28.164.99 66 bytes
2024-11-12 12:58:30.330113 TCP 185.252.220.114:443 172.28.164.99:51109 Flags: A 66 bytes
2024-11-12 12:58:30.355100 Ethernet b6:8b:a9:dc:cb:c6 e8:2a:44:d5:c7:97 120 bytes
2024-11-12 12:58:30.355100 IP 185.252.220.114 172.28.164.99 120 bytes
```

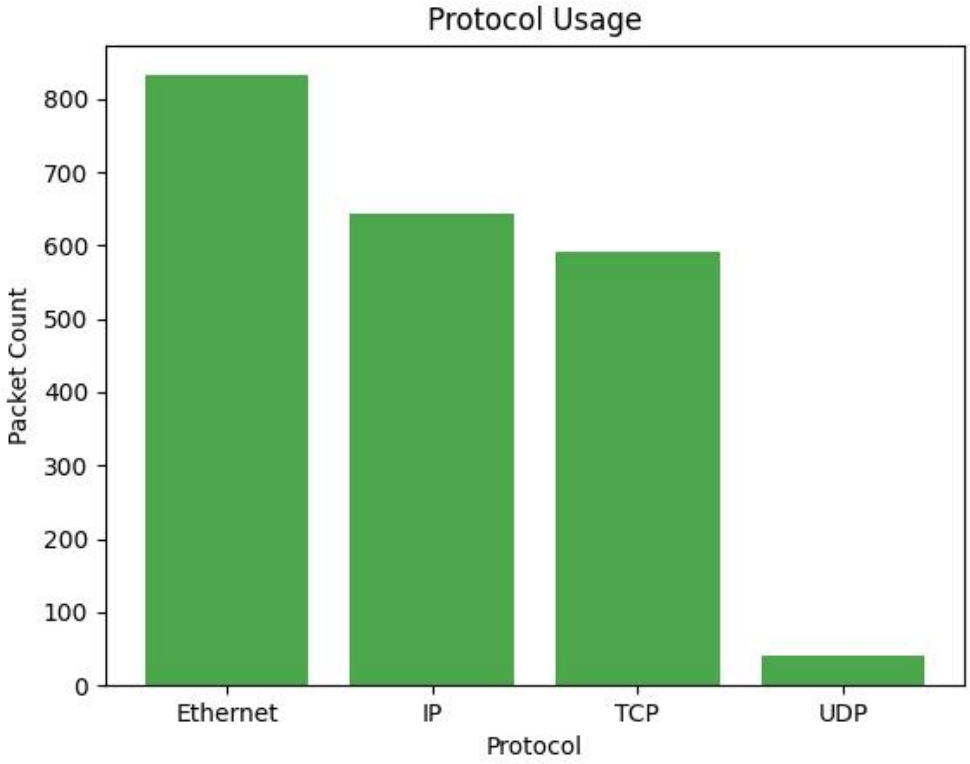
## *a) Throughput Over Time*




## *b) Latency Distribution*:



***c) Protocol Usage***:



# e) Discussion Section on the Network Environment

### *Testing Environment*

- **Network Environment**: The program was tested on a home Wi-Fi network  with a computer to simulate typical network traffic.
- **Packet Capture**: The program captured network traffic, including HTTP requests, DNS queries, and general data exchange on the network.
- **Latency Measurement**: Latency was measured for both **TCP** and **UDP** connections, with TCP connections ( HTTP traffic) showing a more noticeable latency compared to UDP connections (DNS queries).
- **Protocol Usage**: The most frequent protocols observed were **Ethernet**, **IP**, and **TCP**, with a significant portion of traffic being HTTP-based.

### *Key Findings:*

1. **Network Throughput**: The throughput was relatively stable with peaks corresponding to heavy web browsing and streaming activities.
2. **Latency**: Latency was higher during periods of heavy traffic or network congestion. However, latency for UDP packets was lower due to its connectionless nature.
3. **Protocol Usage**: **Ethernet** and **IP** were the most frequently encountered protocols, followed by **TCP** and **UDP**. This is because Ethernet and IP are fundamental for packet delivery.