CSDS 425: Computer Networks

Project #5

Name: Aditi Mondal
Case Network ID: axm1849

1. Introduction

   Theme: Understanding TCP connections: reliability, flow control and connection management

   The network analysis I have performed focuses on understanding the intricacies of the TCP (Transmission Control Protocol) connections in my packet capture file with the major theme centered around understanding its reliability, flow control mechanism, and connection management. In the analysis I intend to majorly look into aspects such as TCP reliability by checking the retransmission statistics, identifying and addressing packet loss issues(if any). The TCP protocol also provides flow control in networks which involves managing the amount of data sent before receiving an acknowledgment, by analyzing the window sizes I intend to provide better insights into the efficiency of this control. Additionally, a study of TCP's connection management will involve analyzing TCP handshakes, session durations, and the overall stability of connections. The main goal is to gain comprehensive insights into TCP's behavior. By examining various parameters such as session duration, round-trip time, window size, the project aims to uncover insights into the network performance.
   The TCP protocol plays a crucial role in ensuring secure and efficient delivery of data across networks which is a major aspect that motivated me to look more into it and study its behavior to get a profound understanding of the TCP protocol and provide insights into the reliable and secure data communication in computer networks.

2. Procedure

   The details regarding the data collection method I employed for this network analysis project and the tools I utilized for analyzing the captured data have been discussed below.

   2.1. Data Collection

   For the data collection methodology, I have utilized Wireshark, an open-source network protocol analyzer, to capture network packets over my home Wi-fi network. The Wi-Fi traffic was captured continuously for a period of 6 hours without applying any specific filters, to gather a substantial amount of raw data that would ensure there is enough information available for thorough analysis. Capturing packets without filters helped in getting a comprehensive view of the Wi-Fi traffic, including various protocols and communication patterns.
   After completing the packet capture, I applied the "tcp" filter to the original file,

which initially contained a total of 1,122,265 packets. Upon applying the filter, the total number of packets was reduced to 442,232, representing 39.4% of the original packets captured in the file which I saved as a separate file, named **TCPfile.pcap(main file for analysis)**. This is the file that contains the refined dataset with only TCP packets that has been used for the subsequent analysis.

2.2. Data Analysis Methods

➔ Filters within Wireshark

As part of the data analysis, I utilized different filters within Wireshark to selectively isolate specific packets relevant to particular aspects of the TCP protocol I intended to analyze.

➔ Custom columns in Wireshark

I further added custom columns in Wireshark to the TCPfile.pcap file that were required for the network analysis which includes:

- Source port- the source port number from which the TCP segment is sent
- Destination Port- the destination port number to which the TCP segment is directed
- Advertized Window- the advertised window size in each TCP packet.
- TCP Segment Len- the length of the TCP segment, indicating the amount of data (in bytes) carried by the segment.
- RTT- measures the Round Trip time ; the time between sending a packet and receiving the acknowledgment (ACK) for that packet from the destination
- Calculated window size- actual window size computed after applying the window scaling factor

Finally, saved the final packets in a csv format in the **TCPfileanalysis.csv** file.

➔ Statiscal analysis and Visualization using Python program in Jupyter Notebook (refer to **proj5.py**)

For the statistical analysis and visualizing the results, I utilized the Jupyter notebook to analyze these packets to identify patterns, trends, or anomalies in the data. The .csv format proved to be particularly convenient for processing and interpreting packet data in the Python programming language within the Jupyter Notebook.

3. Results

Listed below are the results of the analysis of the TCP connections in the packet capture file.

3.1. Protocol Distribution

```
Top Source Ports used for TLSv1.2:
   Source Port  Packets
0          443    11778
1        50780      226
2        51024      153

Top Source Ports used for TLSv1.3:
   Source Port  Packets
0          443    49011
1        51297      877
2        50790      370
```

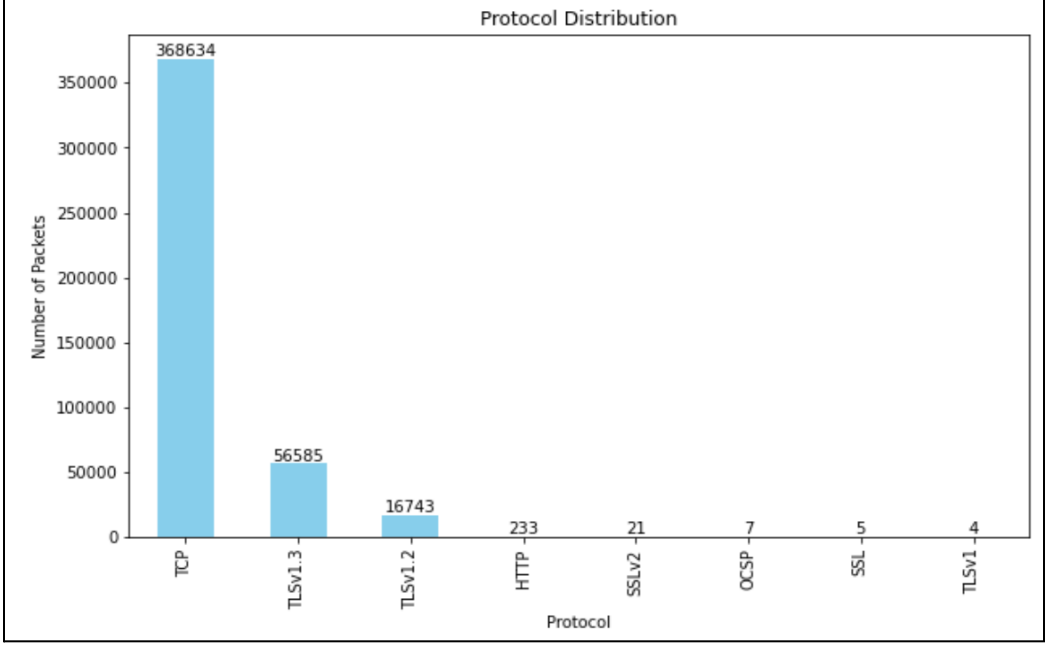


Fig-1: Protocol Distribution over the entire packet capture file

The graph above shows the count of the protocols used for the TCP connections based on Wireshark's inbuilt Protocol hierarchy. Even if the main transport layer protocol used for all the connections is TCP, using the Wireshark's Protocol column we can get more precise information on the network traffic.

We can see that the significant number protocols used by the packets are TCP, TLSv1.3 and TLSv1.2.

The large number of packets (56585) using TLSv1.3 indicates a significant usage of the latest version of the TLS protocol. TLSv1.3 is designed to provide improved security and performance compared to its previous versions..The presence of 16743 packets using TLSv1.2 suggests that there are still systems or applications in the network that use the slightly older version of TLS.

Overall, the high count of TLS packets indicates secure communications in the network. It provides some insights for privacy and data protection, especially if these TLS connections are used for secure data transfer - as we can also see for

this network file that the TLS protocol is most used by the source ports 443 which is used for secure web communications using HTTPS.

3.2. Latency during the establishment of TCP connections

Applied the filter “tcp.flags.syn ==1 || (tcp.flags.syn ==1 && tcp.flags.ack ==1)” on TCPfile.pcap to extract packets involved in the TCP connection establishment and stored in a .csv file named **TCP_RTT.csv** .

A total of 3162 (out of the original 442,232) packets were extracted using the above filter to measure the RTT distribution over time for the TCP connection establishment phase, that is, when a packet sends SYN and receives a corresponding SYN-ACK back.

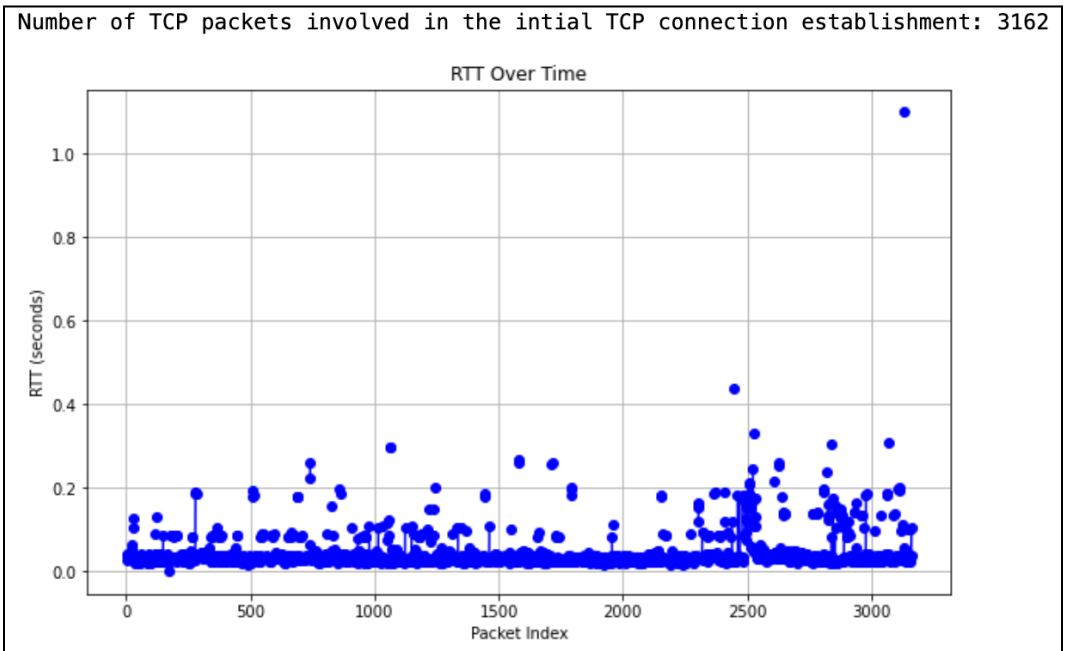
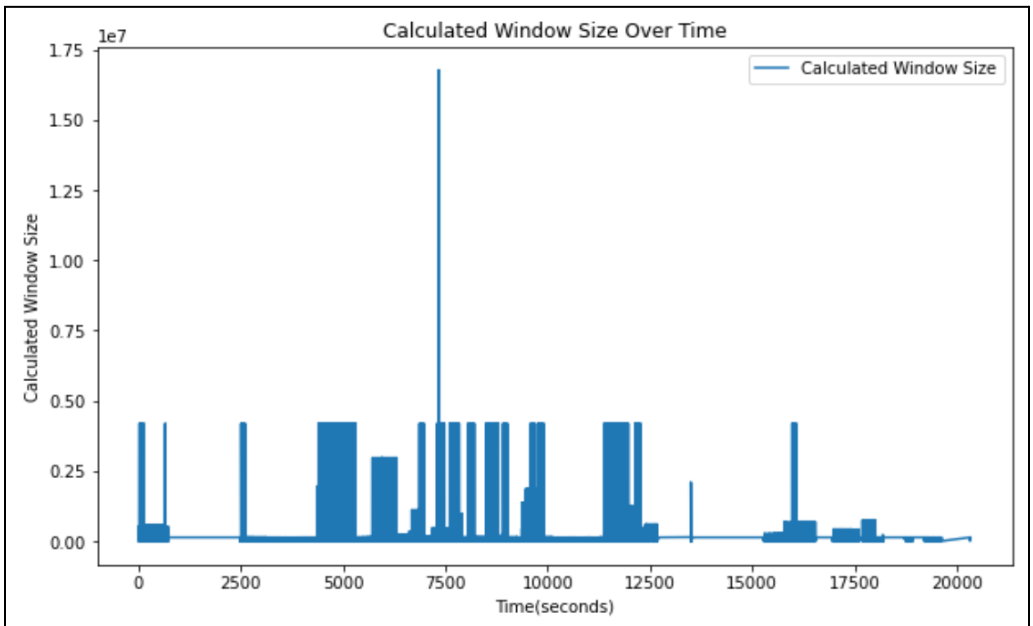


Fig-2: RTT Distribution for TCP Connection Establishment

In the above figure, the mean RTT is approximately 0.0469 seconds, which suggests that, on average, the round-trip time for acknowledgments(SYN-ACK here) during the TCP connection establishment is relatively low. The standard deviation is also moderate with low variability in the RTT values, while the maximum RTT of 1.099023 seconds indicate occasional instances of longer

delays. The quartile values also indicate that the majority of RTT values fall within a reasonably tight range for the initial TCP connection establishment with some outliers as also displayed in the graph(Fig-2)above which is a scatter plot to mark the Round-trip time taken by each packet over time.
Overall, the network demonstrates good responsiveness during TCP connection establishment, with low to moderate latency.

3.3. Analyzing TCP Flow Control from the 'Calculated window size' distribution




```
 Statistics:
count    4.422320e+05
mean     4.159968e+05
std      6.771506e+05
min      0.000000e+00
25%      1.223680e+05
50%      2.851840e+05
75%      4.992000e+05
max      1.677696e+07
```

Fig-3: Calculated Window Size distribution over time

The above graph plots the calculated window sizes with respect to the time of the packet capture for the entire trace of the file. The calculated window size was computed as a custom column after applying the window scaling factor to get the true window size (as also mentioned above as part of the data analysis methods). As we can see in the statistics: the mean value (4.16e+05) indicates the central tendency of the window sizes, while the standard deviation (6.77e+05) suggests a relatively high degree of variability or dispersion around the mean. This could indicate that there is a significant range of window sizes and the variability could be attributed to the dynamic adjustments in the TCP flow control parameters.

Also the sudden spike in the graph suggests a transient increase in window size, indicating a momentary enhancement in network conditions such as high available bandwidth during that time. Overall, the statistics highlight the dynamic nature of TCP flow control in response to network conditions.

3.4. TCP reset packets; Examining TCP's ability to handle abnormal connection termination

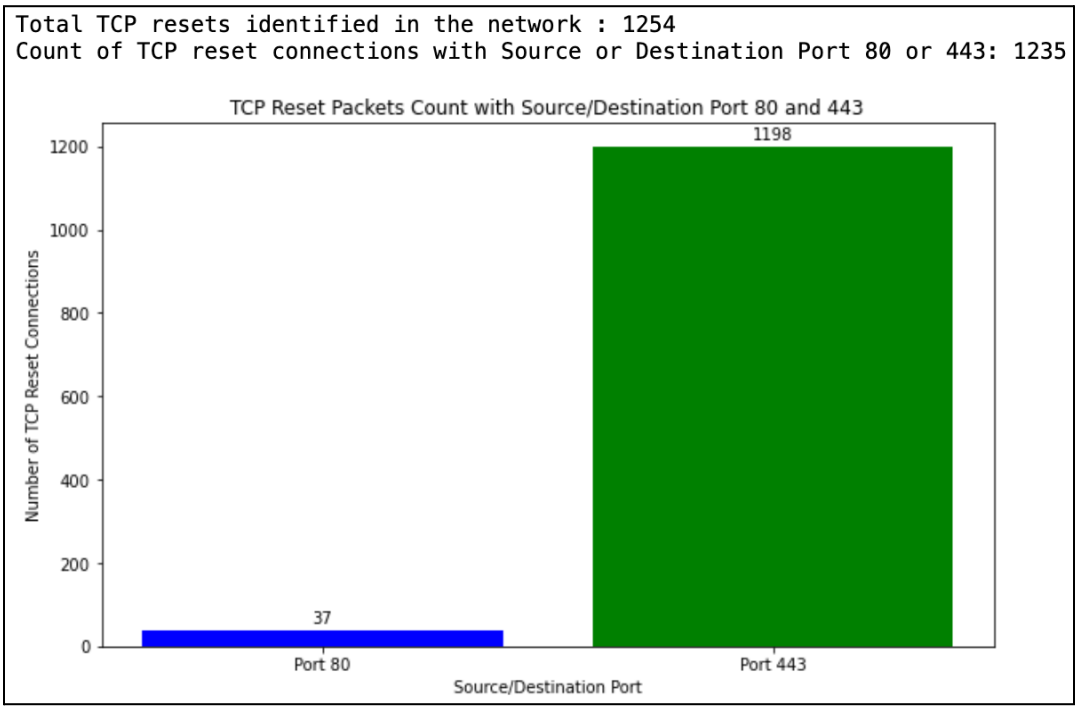


Fig-4: Count of TCP resets for ports 80 and 443

From the above statistics, the total count of TCP resets (1254) indicates the number of times a connection was reset during the capture period since it is a negligible number compared to the total packets captured, it may not be an alarming aspect however I further analyzed the ports for which the maximum resets occurred and as we see from the above figure , a total of 1235 out of 1254 reset connections used ports 80 or 443 which are commonly associated with HTTP and HTTPS traffic.This suggests that there were disruptions or terminations mostly in web-related traffic. This could be due to various reasons, including server or client issues, network problems, or even security-related events. In fact, we can see from the graph above that the highest resets were for the ports 443.

While the overall count of TCP resets might not be significant, the focus on ports 80 and 443 highlights the significance of disruptions in web-related traffic.

3.5. TCP DUP ACKs; Examining Packet loss in Network

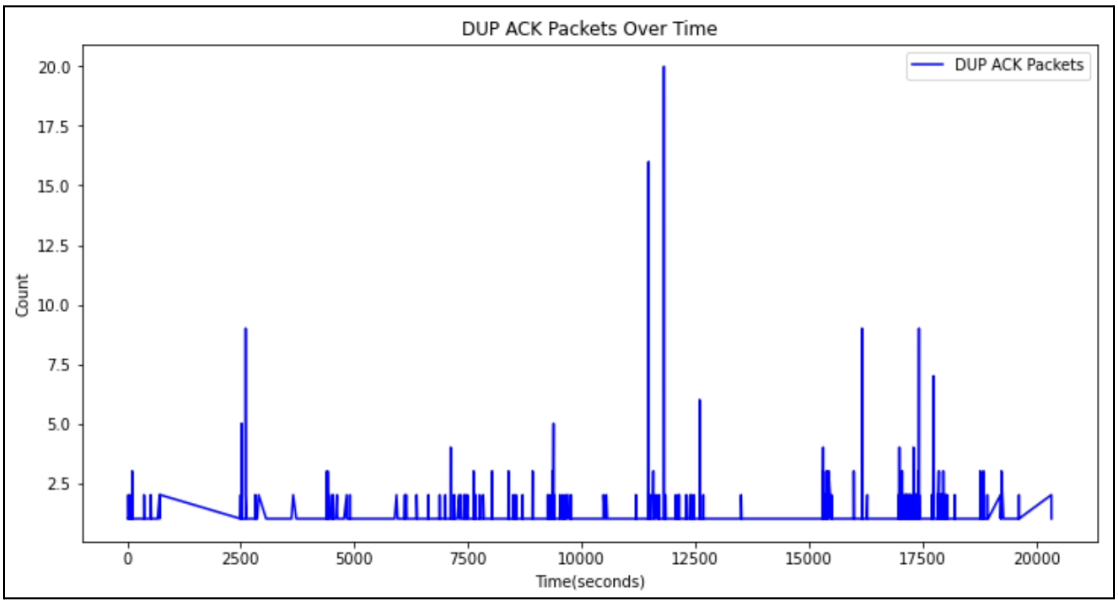


Fig-5: Count of TCP DUP ACKs over time

The above graph displays the count of the TCP duplicate acknowledgements sent each time in the entire packet capture file. The time period near 12500 seconds shows elevated duplicate ACK counts which may suggest a moment of network congestion. During congestion, packets might be dropped or delayed, leading to the need for additional acknowledgments.

This can be correlated with packet loss in the network, since DUP ACKs are often a result of receivers expecting a packet that was not successfully delivered or out-of-order packets sent.

Thus, the large spike could indicate a short-lived episode of network congestion, other than that the DUP ACKs sent each time represent intermittent bursts of packet loss but we see the network quickly recovers from these bursts resulting in a return of DUP ACK counts to their normal or comparatively lower levels.

3.6. TCP Retransmissions; Examining TCP's Packet Recovery Mechanism

```
Count of Packets Retransmitted: 3206
Out of 3206 retransmitted packets the count of packets that were spuriously retransmitted: 692
Out of 3206-692= 2514 retransmitted packets the count of packets that were part of fast retransmissions: 132
Count of Retransmission Packets with TCP Segment Length equal to zero: 530
```

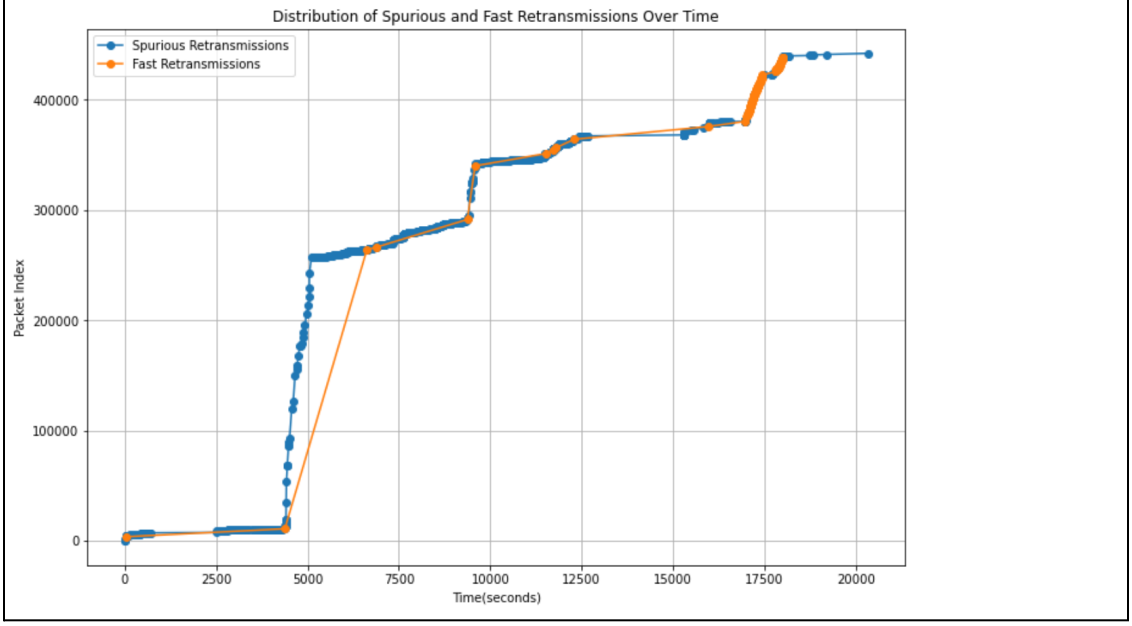


Fig-6: Spurious Retransmissions and Fast Retransmissions distribution over time

From the above figure we can see, out of a total of 442232 packets 3206 packets were retransmitted that is 0.7 % of the packets were retransmitted, some were spuriously retransmitted believing it was not successfully received by the receiver, even though the original packet had reached its destination. Additionally, there were instances of fast retransmissions, which occurred promptly in response to multiple duplicate acknowledgments. In the graph the instances of too many spurious retransmissions at certain time periods suggest short-lived network disruptions or fluctuations that contributed to lost packets.

Also, the correlation with the graph in Fig-5 suggests that fast retransmissions were triggered during periods of heightened duplicate acknowledgments. The sender quickly retransmitted packets upon detecting multiple duplicate acknowledgments during those time periods. Overall, the TCP retransmissions help provide a mechanism for improving reliability in the face of potential network issues.

3.7. TCP Window Updates Over time

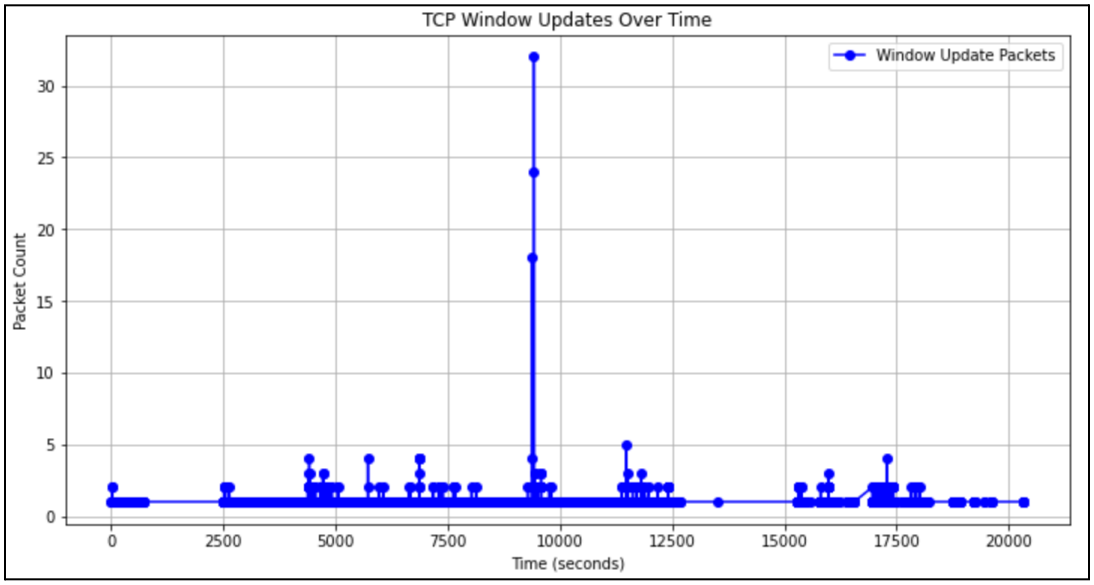


Fig-7: Count of TCP Window Updates over time

Utilized the TCP window update characteristic in Wireshark to plot the above graph that shows the count of TCP window updates that occurred over time. These "TCP Window Update" packets are a normal part of TCP's flow control mechanism. So, when the receiving application extracts data from the receive buffer, creating more available space, the sender updates the TCP Window Size field in the packet header and Wireshark marks these "TCP Window Update" packets to signify the increased buffer space.

Now, as we can see from the above graph the highest window updates have occurred near 10000 seconds of the packet capture which indicates that during this period the sender's TCP receive buffer space increased. This adjustment suggests dynamic changes in data flow and network conditions during this specified period of the packet capture.