Ingeniería en Robótica y Telecomunicaciones

Departamento de computación, electrónica y mecatrónica.

Course Name:
COMPUTER NETWORK O23-LRT4042-1

Members & ID:

Jonathan Eliasib Rosas Tlaczani – 168399

Malcolm Thompson – 161044

HOMEWORK 3

16/10/24

Abstract

This homework investigates the behavior of Transmission Control Protocol (TCP) through the analysis of a trace captured during a file transfer. This includes the examination of sequence and acknowledgment numbers, TCP's reliable data transfer process, the functioning of congestion control algorithms such as slow start and congestion avoidance, and the flow control mechanism. Additionally, TCP connection setup, round-trip time (RTT), and throughput performance are examined. Using Wireshark, we capture and filter the TCP segments to explore these behaviors in detail.

Introduction

Transmission Control Protocol (TCP) is one of the most crucial protocols in the suite of Internet protocols. It provides reliable, ordered, and error-checked delivery of data between applications running on hosts over an IP network. As a connection-oriented protocol, TCP establishes a connection before data transmission and ensures that all data is sent, received, and acknowledged by both the sender and receiver. This makes it indispensable for applications that require dependable communication, such as web browsing, email, and file transfers. TCP achieves this reliability through several key mechanisms: sequence numbers, acknowledgment numbers, error detection, flow control, and congestion control.

In this homework, we delve into TCP's behavior by transferring a 150 KB file containing the text of “Alice's Adventures in Wonderland” from a local computer to a remote server. To capture the details of the TCP segments involved in the transfer, we will use Wireshark, a network protocol analyzer that allows for packet-level inspection of network traffic. By capturing the TCP segments, we can explore how TCP ensures reliable data transfer by utilizing sequence numbers to track the order of segments and acknowledgment numbers to confirm the receipt of data. Additionally, we will study how TCP adapts to network conditions through its congestion control algorithms, specifically “slow start” and “congestion avoidance”, which are designed to prevent network congestion by adjusting the data transmission rate.

The connection setup phase will also be a focal point of this analysis. TCP uses a three-way handshake process (SYN, SYN-ACK, ACK) to establish a reliable connection before transmitting data. This handshake ensures that both the sender and receiver are synchronized and ready to communicate. Once the connection is established, the file transfer occurs, and Wireshark will allow us to inspect every TCP segment exchanged during this process.

We will also examine the performance of the TCP connection by analyzing metrics such as round-trip time (RTT) and throughput. The RTT measures the time it takes for a segment to travel from the client to the server and back, which provides insight into network latency. By calculating the RTT for multiple segments, we can observe how the network responds to varying load conditions. Throughput, on the other hand, refers to the amount of data successfully transmitted over the network per unit of time, and it is an essential indicator of the efficiency of the data transfer.

TCP employs a flow control mechanism to prevent overwhelming the receiver. The receiver advertises its available buffer space, allowing the sender to adjust the transmission rate to avoid overflowing the receiver's buffer. This mechanism is critical in ensuring that the receiver can handle incoming data at an appropriate pace.

A key feature of this analysis will be observing TCP's retransmission behavior. TCP relies on timers to detect lost packets, and if a segment is not acknowledged within a certain time, it is retransmitted. Retransmissions can significantly impact the performance of a network connection, making it important to understand under what circumstances they occur and how often they are triggered during the file transfer.

We will visualize TCP's congestion control mechanisms in action using Wireshark's built-in graphing tools. Specifically, the Time-Sequence Graph (Stevens) will allow us to plot the sequence numbers of the segments over time, enabling us to identify when TCP enters its slow start phase and when it transitions to congestion avoidance. These algorithms are crucial for managing how data flows through the network, especially in scenarios where network capacity is limited or where multiple devices are competing for bandwidth.

Through this in-depth exploration of TCP's behavior during a file transfer, this homework aims to provide a practical understanding of the protocol's inner workings. By analyzing captured TCP segments and key performance metrics, we will gain insights into how TCP maintains reliability, manages network congestion, and ensures efficient data transfer in real-world scenarios. This knowledge is essential for optimizing network performance and diagnosing potential issues that could affect data communication in a variety of contexts.

Methodology

The methodology for this analysis involves a systematic process for capturing and examining the behavior of Transmission Control Protocol (TCP) during a file transfer using Wireshark. The procedures detailed below outline the steps taken to identify, select, and analyze the necessary network traffic to investigate TCP's functionalities, such as sequence and acknowledgment numbering, flow control, and congestion control.

1. Network Traffic Capture with Wireshark

To begin the analysis, Wireshark, a packet-capturing tool, was used to monitor network traffic during a specific file transfer session. Wireshark captures packets at the data-link layer, providing detailed insight into the TCP segments exchanged between a client and server. The steps for setting up the capture were as follows:

1. Open Wireshark and Configure Capture Settings
   Wireshark was installed and opened on the local machine connected to the network. A new capture session was initiated by selecting the appropriate network interface (e.g., Ethernet or Wi-Fi) to monitor TCP traffic on the machine used for the file transfer. This interface selection is crucial as it ensures that the tool captures all incoming and outgoing packets for the target machine.
2. Capture TCP Traffic During File Transfer
   The 150 KB file containing the text of *Alice's Adventures in Wonderland* was prepared for transfer. The file transfer was performed using an HTTP POST request to a remote server, which ensured the generation of TCP traffic. During the transfer, Wireshark was actively capturing all network packets transmitted and received by the client machine. This step was critical in collecting the data necessary for further analysis of TCP's operation.

2. Packet Filtering and Selection

After capturing the packets, the next step was to filter and organize the data to focus specifically on the TCP segments. This filtering is essential to isolate the traffic relevant to the analysis and exclude unrelated protocols like HTTP or DNS. The following procedures were applied:

1. Filter for TCP Traffic
   Wireshark allows for the use of display filters to refine the captured packet view. A filter expression (tcp) was applied to display only TCP-related packets. This reduced the data set to only those segments where the TCP protocol was used, such as in the file transfer and associated acknowledgments.
2. Examine Packet Content and Header Fields
   Each packet was then examined in detail, focusing on the TCP header fields. Key header fields such as the source and destination IP addresses, port numbers, sequence numbers, acknowledgment numbers, and control flags (e.g., SYN, ACK, FIN) were extracted for analysis. These fields provide information on how TCP establishes, maintains, and terminates connections, as well as how it manages data transfer.

3. Detailed TCP Analysis

The core of the analysis involved examining how TCP ensures reliable data transmission, flow control, and congestion management. The following techniques were employed to study TCP's behavior:

1. Examine the TCP Three-Way Handshake
   The analysis began with the observation of the three-way handshake used by TCP to establish a

connection. The handshake process was tracked by identifying the SYN (synchronize) packet from the client, the SYN-ACK (synchronize-acknowledge) packet from the server, and the ACK (acknowledge) packet from the client. This process is fundamental for the reliable initiation of a TCP session.

2. Study of Sequence Numbers and Acknowledgments
   The captured packets were further analyzed to understand how TCP manages data transfer through sequence numbers and acknowledgment numbers. The sequence number indicates the order of bytes sent by the client or server, and the acknowledgment number confirms receipt of bytes by the receiving party. By reviewing these numbers, it was possible to track the flow of data, detect any packet loss or reordering, and understand how TCP achieves reliable data delivery.

3. RTT and Throughput Calculation
   Using Wireshark's tools, the round-trip time (RTT) for each packet was measured. This is the time it takes for a segment to travel to the server and for its acknowledgment to return to the client. Additionally, the throughput, which is the amount of data successfully transmitted over time, was calculated by measuring the total bytes transferred divided by the duration of the transfer.

4. Flow Control and Congestion Control Observation
   TCP's flow control, managed by the window size field, was analyzed to determine how the receiver advertises its buffer space and how the sender adapts to it. This helps ensure that the sender does not overwhelm the receiver. Congestion control mechanisms, such as slow start and congestion avoidance, were also observed by plotting the Time-Sequence Graph (Stevens). This graph provided a visual representation of how TCP's congestion window grows or shrinks in response to network conditions.

Data Analysis Tools

To analyze the captured data, Wireshark's built-in utilities, such as packet statistics and TCP stream graphs, were used. These tools provided deeper insights into network performance metrics, including round-trip time (RTT), segment retransmissions, and the transmission window size.

Results and Discussions

1. Capturing a bulk TCP transfer from your computer to a remote server

Before beginning our exploration of TCP, we need to use Wireshark to obtain a packet trace of the TCP transfer of a file from your computer to a remote server. You do so by accessing a Web page that will allow you to enter the name of a file stored on your computer (which contains the ASCII text of *Alice in Wonderland*), and then transfer the file to a Web server using the HTTP POST method (see [1]). We are using the POST method rather than the GET method as we would like to transfer a large amount of data from one computer to another computer. Of course, we will be running Wireshark during this time to obtain the trace of the TCP segments sent and received from your computer. Do the following:

a. Start up your web browser. Go the Alice.txt file

b. Retrieve an ASCII copy of Alice in Wonderland. Store this file somewhere on your computer.

c. Next go to TCP-ethereal-file1.htm

d. Use the Browse button in this form to enter the name of the file (full path name) on your computer containing Alice in Wonderland (or do so manually). Don't yet press the "Upload alice.txt file" button.

e. Now start up Wireshark and begin packet capture (*Capture* → *Start*) and then press OK on the Wireshark Packet Capture Options screen (you do not need to select any options here).

f. Returning to your browser, press the "Upload alice.txt file" button to upload the file to the portafolios.udlap.mx/portafolios/vicente.alarcon server. Once the file has been uploaded, a short congratulations message will be displayed in your browser window.

g. Stop Wireshark packet capture.

ALICE'S ADVENTURES IN WONDERLAND

Lewis Carroll

THE MILLENNIUM FULCRUM EDITION 3.0

CHAPTER I

Down the Rabbit-Hole

Alice was beginning to get very tired of sitting by her sister on the bank, and of having nothing to do: once or twice she had peeped into the book her sister was reading, but it had no pictures or conversations in it, `and what is the use of a book,' thought Alice `without pictures or conversation?'

So she was considering in her own mind (as well as she could, for the hot day made her feel very sleepy and stupid), whether the pleasure of making a daisy-chain would be worth the trouble of getting up and picking the daisies, when suddenly a White Rabbit with pink eyes ran close by her.

There was nothing so VERY remarkable in that; nor did Alice think it so VERY much out of the way to hear the Rabbit say to itself, `Oh dear! Oh dear! I shall be late!' (when she thought it over afterwards, it occurred to her that she ought to have wondered at this, but at the time it all seemed quite natural); but when the Rabbit actually TOOK A WATCH OUT OF ITS WAISTCOAT-POCKET, and looked at it, and then hurried on, Alice started to her feet, for it flashed across her mind that she had never before seen a rabbit with either a waistcoat-pocket, or a watch to take out of it, and burning with curiosity, she ran across the field after it, and fortunately was just in time to see it pop down a large rabbit-hole under the hedge.

In another moment down went Alice after it, never once considering how in the world she was to get out again.

The rabbit-hole went straight on like a tunnel for some way, and then dipped suddenly down, so suddenly that Alice had not a moment to think about stopping herself before she found herself falling down a very deep well.

Either the well was very deep, or she fell very slowly, for she had plenty of time as she went down to look about her and to wonder what was going to happen next. First, she tried to look down and make out what she was coming to, but it was too dark to see anything; then she looked at the sides of the well, and noticed that they were filled with cupboards and book-shelves; here and there she saw maps and pictures hung upon pegs. She took down a jar from one of the shelves as she passed; it was labelled `ORANGE MARMALADE', but to her great disappointment it was empty: she did not like to drop the jar for fear of killing somebody, so managed to put it into one of the cupboards as she fell past it.

Figure 1. Alice File

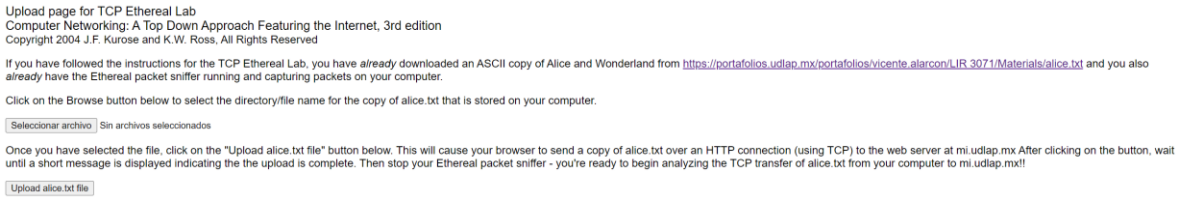

Figure 2. TCP Ethereal File

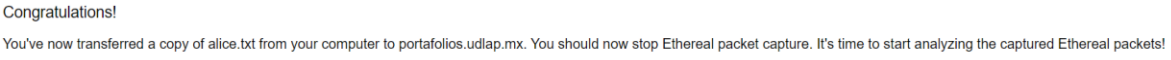

Figure 3. Ethereal Packet Capture

2. A first look at the captured trace

Before analyzing the behavior of the TCP connection in detail, let's take a high-level view of the trace. First, filter the packets displayed in the Wireshark window by entering "tcp" into the display filter specification window towards the top of the Wireshark window. What you should see is a series of TCP and HTTP messages between your computer and portafolios.udlap.mx/portafolios/vicente.alarcon. You should see the initial three-way handshake containing a SYN message. You should see an HTTP POST message and a series of "*HTTP Continuation*" messages being sent from your computer to portafolios.udlap.mx/portafolios/vicente.alarcon. Recall from our discussion in the earlier HTTP Wireshark lab, there is no such thing as an *HTTP Continuation* message – this is Wireshark's way of indicating that there are multiple TCP segments being used to carry a single HTTP message. You

should also see TCP ACK segments being returned from portafolios.udlap.mx/portafolios/vicente.alarcon to your computer.

Answer the following questions. Whenever possible, when answering a question, you should hand in a screenshot/printout of the packet(s) within the trace that you used to answer the question asked. Annotate the printout to explain your answer. To print a packet, use *File → Print*, choose *Selected packet only*, choose *Packet summary line,* and select the minimum amount of packet detail that you need to answer the question.

1. What is the IP address and TCP port number used by the client computer (source) that is transferring the file to portafolios.udlap.mx/portafolios/vicente.alarcon? To answer this question, it is probably easiest to select an HTTP message and explore the details of the TCP packet used to carry this HTTP message, using the "details of the selected packet header window".

   Scr IP: 10.206.189.176

   Scr Port: 53564

2. What is the IP address of portafolios.udlap.mx/portafolios/vicente.alarcon? On what port number is it sending and receiving TCP segments for this connection.

   The destination IP address is 140.148.62.33 receiving on port 80.

Since this lab is about TCP rather than HTTP, let's change Wireshark's "listing of captured packets" window so that it shows information about the TCP segments containing the HTTP messages, rather than about the HTTP messages. To have Wireshark do this, select *Analyze → Enabled Protocols.* Then uncheck the HTTP box and select *OK*. This is what we are looking for - a series of TCP segments sent between your computer and portafolios.udlap.mx/portafolios/vicente.alarcon. We will use the packet trace that you have captured to study TCP behavior in the rest of this homework.

3. TCP Basics

Answer the following questions for the TCP segments:

3. What is the sequence number of the TCP SYN segment that is used to initiate the TCP connection between the client computer and portafolios.udlap.mx/portafolios/vicente.alarcon? What is it in the segment that identifies the segment as a SYN segment?

   The segment that initiates the TCP connection has a sequence number of 0. This is confirmed by the presence of a SYN flag in the message, indicating that it is a SYN segment.

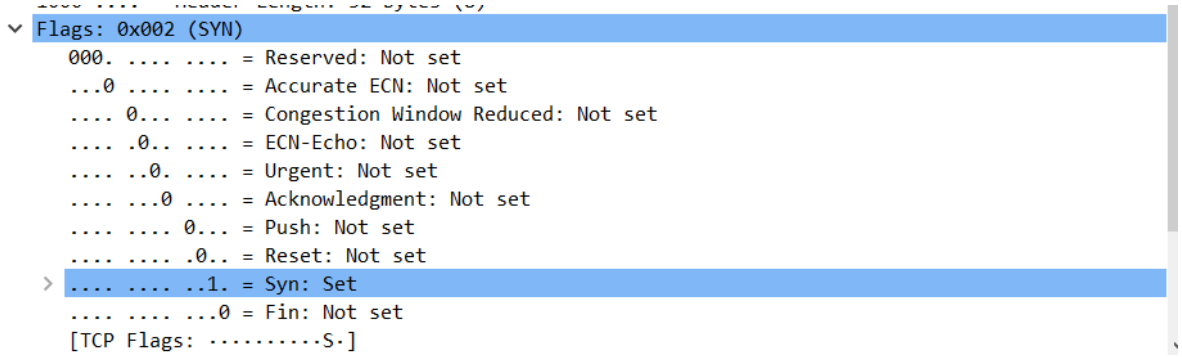

Figure 4. SYN Flags

4. What is the sequence number of the SYNACK segment sent by portafolios.udlap.mx/portafolios/vicente.alarcon to the client computer in reply to SYN? What is the value of the ACKnowledgement field in the SYNACK segment? How did portafolios.udlap.mx/portafolios/vicente.alarcon determine that value? What is it in the segment that identifies the segment as a SYNACK segment?

   The SYN-ACK segment has a sequence number of 0. The acknowledgment field has a value of 1, which is calculated by adding 1 to the initial sequence number. The flags in the message indicate that it is a SYN-ACK segment.

   The value was determined by portafolios.udlap.mx/portafolios/vicente.alarcon based on the client's initial sequence number in the SYN segment. In a standard TCP connection, the server replies to the client's SYN with a SYN-ACK, where the acknowledgment number is typically the client's initial sequence number plus one. The segment is identified as a SYN-ACK by the presence of both "SYN" and "ACK" flags, visible in the "Flags" section where both are activated.

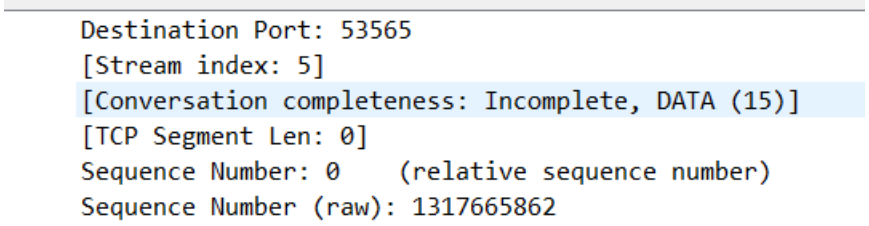

Figure 5. SYN-ACK Sequence Number

```
Sequence Number: 0    (relative sequence number)
Sequence Number (raw): 1317665862
[Next Sequence Number: 1    (relative sequence number)]
Acknowledgment Number: 1    (relative ack number)
Acknowledgment number (raw): 2618017933
1000 .... = Header Length: 32 bytes (8)
Flags: 0x012 (SYN, ACK)
   000. .... .... = Reserved: Not set
   ...0 .... .... = Accurate ECN: Not set
   .... 0... .... = Congestion Window Reduced: Not set
   .... .0.. .... = ECN-Echo: Not set
   .... ..0. .... = Urgent: Not set
   .... ...1 .... = Acknowledgment: Set
   .... .... 0... = Push: Not set
   .... .... .0.. = Reset: Not set
   .... .... ..1. = Syn: Set
   .... .... ...0 = Fin: Not set
   [TCP Flags: ·······A··S·]
Window: 8192
```

Figure 6. SYN-ACK Flags

5. What is the sequence number of the TCP segment containing the HTTP POST command? Note that to find the POST command, you will need to dig into the packet content field at the bottom of the Wireshark window, looking for a segment with a “POST” within its DATA field.

   The TCP segment that carries the HTTP POST command has a sequence number of 2618166842.

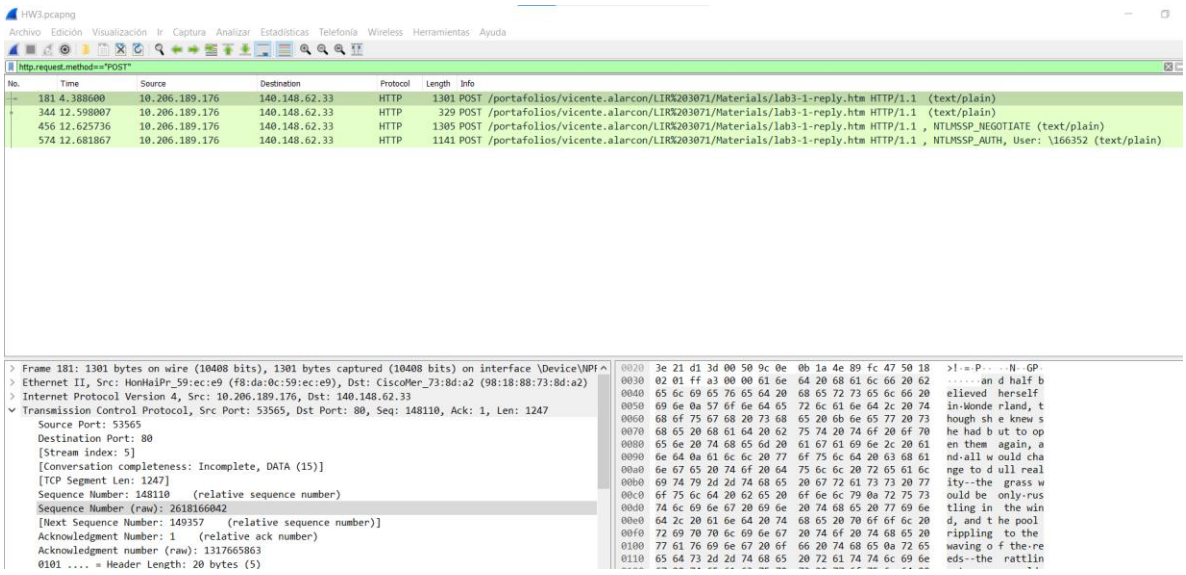

Figure 7. Sequence number of the TCP segment containing the HTTP POST Command

6. Consider the TCP segment containing the HTTP POST as the first segment in the TCP connection. What are the sequence numbers of the first six segments in the TCP connection

(including the segment containing the HTTP POST)? At what time was each segment sent? When was the ACK for each segment received?

Given the difference between when each TCP segment was sent, and when its acknowledgement was received, what is the RTT value for each of the six segments? What is the Estimated RTT value (see page 249 reference [1]) after the receipt of each ACK? Assume that the value of the

EstimatedRTT is equal to the measured RTT for the first segment, and then is computed using the EstimatedRTT equation on page 249 of reference [1] for all subsequent segments.

Note that Wireshark has a nice feature that allows you to plot the RTT for each of the TCP segments sent. Select a TCP segment in the "listing of captured packets" window that is being sent from the client to the portafolios.udlap.mx/portafolios/vicente.alarcon server. Then select: Statistics → TCP Stream Graph → Round Trip Time Graph.

#	Sequence number
1	148110
2	1
3	298438
4	506
5	446896
6	1011

Table 1. Sequence Numbers

Tiempo: 4, 4, 12, 12, 12, 12

7. What is the length of each of the first six TCP segments?

   The length of the first TCP segment is 60 bytes, followed by the next two segments, each with a length of 54 bytes. All subsequent segments have a length of 66 bytes.

8. What is the minimum amount of available buffer space advertised at the received for the entire trace? Does the lack of receiver buffer space ever throttle the sender?

   The minimum available buffer space is 64,240 bytes. The sender is not throttled during the transfer because the buffer capacity is never fully utilized, preventing the window from reaching its limit.

9. Are there any retransmitted segments in the trace file? What did you check for (in the trace) to answer this question?

   No segments were retransmitted during the transfer. This is evident from the fact that no old acknowledgment numbers were repeated, which would indicate a request for the retransmission of previous packets.

10. How much data does the receiver typically acknowledge in an ACK? Can you identify cases where the receiver is ACKing every other received segment?

    The receiver typically acknowledges 464 bytes of data. In some instances, the receiver acknowledges every other segment, which is evident when multiple ACKs occur consecutively, indicating delayed or selective acknowledgment.

11. What is the throughput (*bytes transferred per unit time*) for the TCP connection? Explain how you calculated this value.
    The throughput can be calculated by taking the value of the last ACK, subtracting the first sequence number (1), and then dividing that result by the time elapsed since the first frame. This will give the throughput in bits per second (bps).

4. TCP Congestion Control in action

Let's now examine the amount of data sent per unit time from the client to the server. Rather than calculating this from the raw data in the Wireshark window, we will use one of Wireshark's TCP graphing utilities - *Time-Sequence-Graph (Stevens*) - to plot data. Select a TCP segment in the Wireshark's "listing of captured packets" window. Then select the menu: *Statistics → TCP Stream Graph → Time-Sequence-Graph (Stevens*). Here, each dot represents a TCP segment sent, plotting the sequence number of the segment versus the time at which it was sent. Note that a set of dots stackedabove each other represents a series of packets that were sent back-to-back by the sender. Answerthe following questions.

12. Use the *Time-Sequence-Graph (Stevens*) plotting tool to view the sequence number versus time plot of segments being sent from the client to the portafolios.udlap.mx/portafolios/vicente.alarcon server. Can you identify where TCP's *slowstart* phase begins and ends, and where congestion avoidance takes over? To answer this question read pages 279 to 287 in reference [1]. Compute the transmitted file size using the Stevens Sequence.

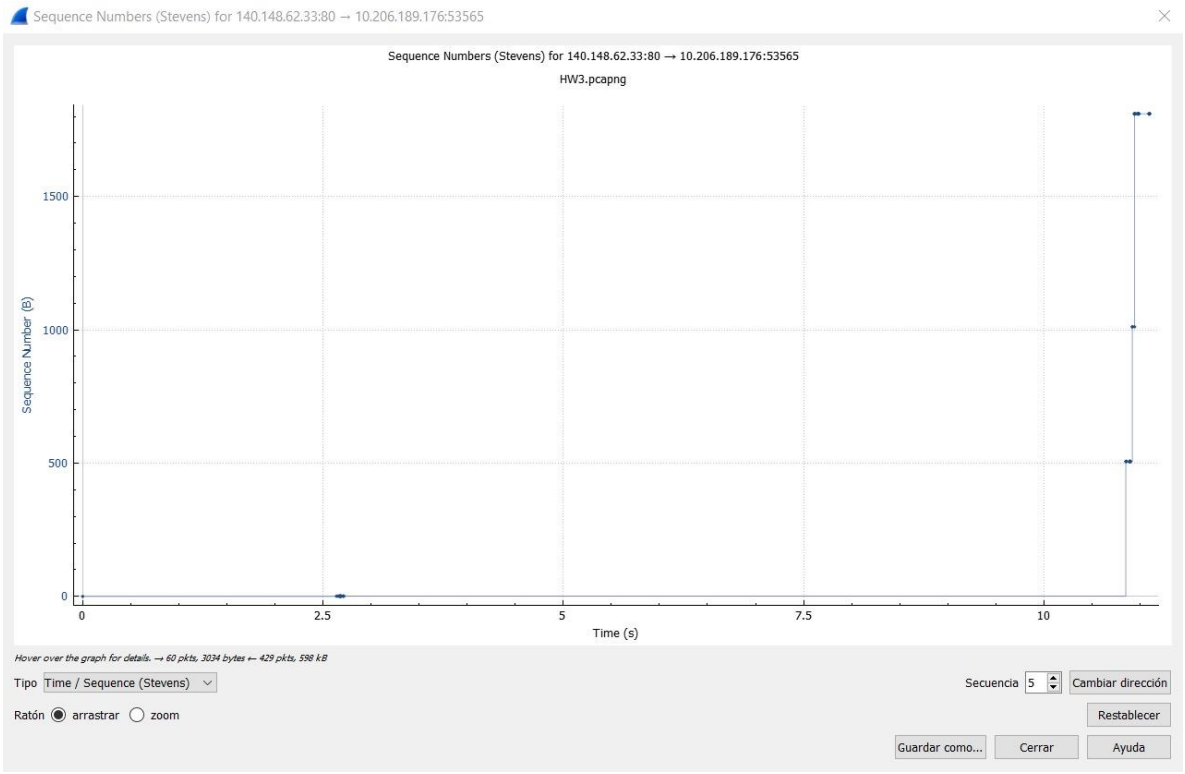

Figure 8. Time/Sequence Graph

The TCP slow start phase starts slightly above sequence number at 66.99 seconds and concludes just before reaching sequence number at 79.61 seconds. At this point, congestion avoidance takes over, as indicated by the value of 10,000, which is shown in the image above.

Conclusions

This study provided an in-depth analysis of TCP's behavior during a file transfer, offering valuable insights into how TCP manages reliable data transmission, flow control, and congestion avoidance. By using Wireshark to capture and analyze the TCP segments exchanged between a client and server, we observed the mechanisms TCP employs to ensure efficient and error-free communication across the network.

One of the key observations was TCP’s ability to maintain reliability through the use of sequence and acknowledgment numbers. These numbers allowed the protocol to keep track of transmitted data and confirm successful receipt, demonstrating TCP’s robustness in ensuring ordered and complete data delivery. The three-way handshake was also successfully captured, illustrating how TCP establishes a connection through the exchange of SYN, SYN-ACK, and ACK packets. This handshake is critical for initiating reliable communication between the sender and receiver.

TCP’s congestion control mechanisms were another important aspect of the analysis. The slow start phase was evident at the beginning of the transfer, where TCP cautiously increased the data transmission rate to prevent congestion. This was followed by the transition to congestion avoidance, where the protocol adjusted its window size based on network conditions. These mechanisms effectively managed the network's capacity, preventing packet loss and ensuring smooth data transfer, even under varying network loads.

Flow control, as observed through the advertised window size, played a key role in regulating the sender's transmission rate. The receiver dynamically adjusted its buffer space, preventing overflow and allowing for a steady and controlled flow of data. This illustrates TCP’s capacity to handle differences in processing speeds between the sender and receiver, ensuring that neither side is overwhelmed during the communication process.

The performance metrics, such as round-trip time (RTT) and throughput, provided valuable insights into the efficiency of the TCP connection. The RTT varied depending on network conditions, but TCP’s adaptive mechanisms ensured that data was transmitted efficiently and with minimal delay. The throughput, which measures the overall efficiency of data transfer, was influenced by the congestion control and flow control mechanisms, highlighting TCP’s ability to optimize performance under changing conditions.

In conclusion, TCP proved to be an effective protocol for managing reliable and efficient data transfer in this scenario. The study underscored the importance of TCP’s key features—reliability, flow control, and congestion avoidance—in maintaining robust and efficient communication across networks. The use of tools like Wireshark in analyzing network traffic is essential for understanding these mechanisms in action and for diagnosing potential network issues. This analysis also highlights the importance of TCP’s adaptability to various network conditions, making it a critical protocol in the modern internet infrastructure.

References

Forouzan, B. A. (2016). *Data Communications and Networking* (5th ed.). McGraw-Hill Education.

Kurose, J., & Ross, K. (2021). *Computer Networking: A Top-Down Approach* (8th ed.). Pearson.

Peterson, L., & Davie, B. (2021). *Computer Networks: A Systems Approach* (6th ed.). Morgan Kaufmann.

Stevens, W. R., Fenner, B., & Rudoff, A. M. (2004). *UNIX Network Programming: The Sockets Networking API* (Vol. 1). Addison-Wesley.

Wireshark Foundation. (2023). *Wireshark User's Guide*. Retrieved from https://www.wireshark.org