<u>רשתות תקשורת מטלה 3-</u> מגישות: נדב רוני- 325730190 ואיזביצקי בר- 212138457

במטלה זו התבקשנו לכתוב קוד של שרת ולקוח שמתקשרים לפי פרוטוקול TCP עם עקרונות חלון ההזזה וtimeOut.
השרת מגדיר את גודל ההודעה המקסימלי שהוא יכול לקבל מהלקוח והלקוח מגדיר את גודל החלון ואת זמן ה timeOut (הזמן שהלקוח מחכה לקבלת ACK).
הלקוח מחלק את ההודעה שהוא רוצה לשלוח לשרת לסגמנטים לפי גודל ההודעה המקסימלי שהשרת יכול לקבל ועל מנת לעקוב אחר סדר הסגמנטים לכל אחד מהסגמנטים יש Index.
עבור מטלה זו הגדרנו שהheader של ההודעה לשרת יהיה קבוע והוא 8 תווים (7 ספרות שמעידות על הindex ותו אחד של : שמסיים את הHeader (מבדיל בין הheader לתוכן הסגמנט).
במידה ונרצה לשלוח הודעות יותר ארוכות ממשפטים ארוכים (לדוג'- פסקאות) נגדיל את מספר הספרות בheader.
לאחר שהלקוח חילק את ההודעה לסגמנטים הוא שולח סגמנטים לשרת ככמות גודל החלון, ובכל פעם שמקבל ACK ברצף שהוא מצפה לו הוא מזיז את החלון index קדימה (לפי הACK השהתקבל) ושלוח את הסגמנטים שלא נשלחו בחלון החדש כך ממשיך עד ששולח את כל הסגמנטים ומקבל עליהם ACK אלא אם בזמן שהוא מחכה לACK נגמר הזמן (timeOut) אז הוא מתחיל לשלוח לשרת שוב את הסגמנטים שנמצאים בחלון שהוא נמצא בו כרגע וממנו ממשיך.
נציין שהשרת שולח כל פעם ACK עם הIndex שעד אליו יש לו רצף של סגמנטים.

<u>מעבר על הקוד -</u>

השרת-

פעולת run_server הפעולה שהmain של השרת קורא לה וממנה פותחים את השרת ומנהלים את ההאזנות וקבלת ההודעות.

```
server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server_socket.bind((host, port))
server_socket.listen()
print(f"Server is listening on {host}:{port}")

con, adr = server_socket.accept()
print(f"Connection established with {adr}")
```

פותח סוקט ומאזין ללקוחות שרוצים להתחבר.
ברגע שמקבל בקשה להתחברות – מתחבר ללקוח

```
max_size = settingM()
con.sendall(f"{max_size}".encode())  # Send max_size to the client
print(f"Maximum message size ({max_size} bytes) sent to the client.")
```

מגדיר את גודל ההודעה המקסימלי -
דרך קריאה מקובץ או דרך קלט מהמשתמש.
לאחר מכן שולח את הגודל ללקוח.

```python
received_segments = {}  # Store received segments by index
expected_segment = 0    # Expected segment to complete the sequence
while True:
    try:
        data = con.recv(max_size+8).decode() # Receive data from the client

        if not data:
            print("No data received.")
            break

        if data.strip().lower() == "exit":
            print("Exit command received. Closing connection.")
            break

        if data == "-1":  # the client signals that all segments have been received
            print("Client finished sending segments-resetting server for the next message.")
            # Reset server state to prepare for the next message
            received_segments = {}
            expected_segment = 0
            continue
        index, segment = data.split(sep: ":", maxsplit: 1)
        index = int(index)

        if index not in received_segments: # check if the segment is already in the received_segments
            received_segments[index] = segment
            print(f"Adding segment {index} to the received segments.")
            expected_segment += 1

            while expected_segment in received_segments: # update the last index in the sequence
                expected_segment += 1

            con.send(f"ACK {index}".encode()) # send ACK for each segment received
            print(f"Sending ACK for segment {index}")

    except Exception as e:
        print(f"Error occurred: {e}")
        break

con.close()
server_socket.close()
print("Server shut down.")
```

בלולאה זו השרת מחכה לdata מהלקוח(8 ביטים של של הheader+ הmax_size) ולפיו מתנהל-
אם לא קיבל בו כלום- יוצא מהלולאה וסוגר את הקשר.
אם מקבל exit- יוצא מהלולאה וסוגר את הקשר.
אם מקבל '1-' – מאתחל את כל המערכים כדי להיות מוכן להודעה חדשה שהולכת להישלח.
אחרת, מפצל את ה Index מהתוכן של הסגמנט, במידה והסגמנט (חיפוש לפי אינדקס) שיתקבל לא התקבל עד רגע זה -מוסיף אותו למילון ששומר על הסגמנטים שהגיעו.
לאחר מכן, מעדכן את האינדקס של הסגמנט שהוא צריך לקבל (הבא ברצף) .
שולח לבסוף ACK ללקוח עם הסגמנט הבא ברצף שהוא צריך לקבל.

פעולת setting מחזירה את גודל ההודעה המקסימלי – או שקוראת מקובץ או כקלט מהמשתמש.

```python
def settingM():
    choice = input("enter 1 to load the max_message_size from a file or 2 to enter it by yourself :").strip().lower()
    if choice == "1":
        file_path = input("enter the file path: ")
        try:
            with open(file_path, "r", encoding="utf-8") as file:
                for line in file:
                    if line.startswith("maximum_msg_size:"):
                        value = int(line[17:-1])
                if value is None:
                    raise ValueError("Some settings are missing in the file.")
                return value
        except (FileNotFoundError, ValueError) as e:
            print(f"Error reading settings from file: {e}. Please enter the settings manually.")

    # If file reading fails or user chooses not to use a file
    max_message_size = int(input("Enter the maximum message size (in bytes): "))
    return max_message_size
```

הפונקציה settingM מגדירה את גודל ההודעה המקסימלי לתקשורת. הפונקציה מאפשרת למשתמש שתי אפשרויות: לקורא את הערך מתוך קובץ או להזין אותו באופן ידני. אם המשתמש בוחר לקורא מתוך קובץ, הפונקציה מבקשת את כתובת הקובץ ומחפשת שורה שמתחילה ב-maximum_msg_size: כדי לחלץ את הערך. אם הערך לא נמצא או שיש שגיאה (למשל, הקובץ חסר), הפונקציה מציגה הודעת שגיאה ומבקשת מהמשתמש להזין את הערך ידנית. במידה והמשתמש בוחר להזין את הערך בעצמו מלכתחילה, הפונקציה תבקש ממנו להכניס את הגודל ישירות.

הלקוח-

```python
def run_client():
    host = 'localhost'
    port = 12345
    client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    client_socket.connect((host, port))
    print(f"Connected to server at {host}:{port}")

    max_msg_size = int(client_socket.recv(1024).decode())  # receive the max size message from the server
    print(f"maximum message size allowed by server: {max_msg_size} bytes.")

    # Get settings and send timeout to the server
    window_size, timeout = settingsVal()
    print(f"server allows messages up to {max_msg_size} bytes, sliding window size of {window_size}, timeout of {

    while True:
        message = settingMsg()

        if message.strip().lower() == "exit":
            # Exit the client when user inputs "exit"
            client_socket.send("exit".encode())
            ans = client_socket.recv(1024).decode()
            if ans.strip().lower() == "exit":
                print("Server acknowledged exit. Closing connection.")
            break
        sliding_window(client_socket, message, max_msg_size, window_size, timeout)  # send the message using the s

    client_socket.close()
    print("Connection closed.")
```

הפונקציה המרכזית שמפעילה את הלקוח. תחילה מתחבר לשרת, לאחר מכן מקבל את גודל ההודעה המרבי המותר מהשרת, ומגדיר את פרמטרי התקשורת (גודל חלון הזזה ו- timeOut). אחרי שהגדיר את כל הפרמטרים לתקשורת הוא נכנס ללולאה שבה המשתמש יכול להזין / לקרוא מקובץ הודעות לשליחה או לסיים את החיבור באמצעות הקלדת ."exit" במידה וקיבל הודעה הוא קורא לפעולת sliding_window.

```python
def sliding_window(client_socket, message, max_msg_size, window_size, timeout):
    segments = []  # Split the message into segments based on the maximum size
    for i in range(0, len(message), max_msg_size):
        segment_index = i // max_msg_size
        segment = message[i:i + max_msg_size]
        if len(segment) < max_msg_size:
            segment = segment.ljust(max_msg_size, " ")  #If the length of the segment is less than the defined maximum size
        formatted_index = str(segment_index).zfill(7)  # header- 7 digits
        segments.append((formatted_index, segment))  # Store segment index and data
    print(f"Message split into {len(segments)} segments: {segments}")

    next_ack = 0  # Start with the first segment
    next_to_send = 0
    while next_ack < len(segments):
        window_end = min(next_ack + window_size, len(segments))
        for i in range(next_ack, window_end):  # send all segments in the current window
            index, segment = segments[i]
            client_socket.sendall(f"{index}:{segment}".encode())
            next_to_send += 1
            print(f"Sending segment {int(index)}: {segment}")

        next_ack,next_to_send = receive_acks(client_socket, segments, next_ack, window_size,timeout, window_end, next_to_send)
```

בפעולה זו תחילה מחלק את ההודעה לסגמנטים לפי גודל ההודעה המקסימלי שהשרת יכול לקבל.
אם ההודעה האחרונה קטנה יותר מגודל ההודעה המקסימלית מוסיף לה " " כדי שתהיה כגודל ההודעה המקסימלית (למניעת בעיות בהמשך של מקרי קיצון).

נכנס ללולאה שבה הוא בודק באיזה אידנקס החלון מסתיים (בוחר את המינימלי בין האינדס הסופי של מערך הסגמנטים לבין האינדקס של סוף החלון) הוא שולח את כמות הסגמנטים כגודל החלון הזזה וקורא לפעולת receive_acks שבה הוא מנהל את קבלת הACK ופועל לפי הם.

```python
def receive_acks(client_socket, segments, next_ack, window_size,timeout, window_end, next_to_send):
    client_socket.settimeout(timeout)  # Set timeout for receiving ACK
    while True:
        try:
            ack = client_socket.recv(1024).decode()
            ack_segments = ack.split("ACK ")[1:]
            ack_segments = [segment.strip() for segment in ack_segments]  # filter empty values if any

            for a in ack_segments:
                ack_segment = int(a.split()[-1])  # extract segment number
                print(f"received ACK: {ack_segment}")
                next_ack = ack_segment+1

                if window_end < len(segments) - 1:
                    window_end = min(next_ack + window_size - 1, len(segments) - 1)
                    while next_to_send <= window_end:  # if we have segment in the window that we don't send
                        index, segment = segments[next_to_send]
                        print(f"Sending new segment {index}: {segment}")
                        client_socket.sendall(f"{index}:{segment}".encode())  # send the segment
                        client_socket.settimeout(timeout)  # reset the timeout for receiving ACKs
                        next_to_send += 1

            # Exit if all segments are acknowledged
            if next_ack >= len(segments):
                print("All segments have been sent and the server receive")
                client_socket.sendall("-1".encode())  # notify the server - complete
                return next_ack,next_to_send

        except socket.timeout:
            print(f"Timeout occurred while waiting for ACK. Exiting receive_acks.")
            break

        except Exception as e:
            print(f"Error while receiving ACK: {e}")
            break

    print(f"Exiting receive_acks with next_to_send={next_ack}")
    return next_ack,next_to_send
```

פונקציה זו מחכה לACK מהשרת וברגע שמתקבל הוא בודק אם הוא הACK שציפה לקבל(= הACK ברצף הנכון של הסגמנטים).אם כן מקדם את החלון לאינקדס אחד יותר מהACK שהתקבל ושולח את הסגמנטים שלא נשלחו בחלון.
במידה ולא מתקבל ACK רלוונטי או בכלל ועובר זמן הtimeOut שהוגדר אז יוצא מהפעולה ומחזיר לפעולה שקראה לו sliding_window את האינדקס בו נעצר ולא קיבל עליו ACK.

לבסוף, בודק אם הגענו לסוף מערך הסגמנטים- אם כן יוצא מהפעולה ומדפיס שכל הACKים הגיעו.

```python
def settingMsg():
    choice = input("enter 1 to read from a file or 2 to write the message (if you want to end write in the message ex
    if choice == "1":
        file_path = input("enter the file path: ")
        try:
            with open(file_path, "r", encoding="utf-8") as file:
                for line in file:
                    if line.startswith("message:"):  # check if the line contains the message
                        ans = line[9:-2]  # extract the message content
                if ans is None:
                    raise ValueError("No message in file.")
                return ans
        except (FileNotFoundError, ValueError) as e:
            print(f"Error reading settings from file: {e}. Please enter the settings manually.")

    # if the user chooses to write manually
    message = input("enter the message: ")
    return message
```

פונקציה זו אחראית לקביעת ההודעה שהלקוח ישלח לשרת.
תחילה היא מבקשת מהמשתמש לבחור בין שתי אפשרויות: קריאה של ההודעה מתוך קובץ או הזנת ההודעה באופן ידני. במקרה של בחירה בקובץ, הפונקציה מנסה לפתוח את הקובץ ולחפש שורה שמתחילה ב-message: ולחלץ את ההודעה מתוך השורה. אם לא נמצא תוכן מתאים או שהקובץ לא קיים, המשתמש יתבקש להזין את ההודעה ידנית.

```python
def settingsVal():
    choice = input("enter 1 to read from a file or 2 to enter the details by yourself: ").strip().lower()
    if choice == "1":
        file_path = "file.txt"
        settings = {"window_size": None, "timeout": None}
        try:
            with open(file_path, "r", encoding="utf-8") as file:
                for line in file:
                    if not line.startswith("message:") and ":" in line: # Skip lines starting with "message:" and pars
                        key, value = line.split(sep: ":", maxsplit: 1)
                        key = key.strip()
                        value = value.strip()
                        if key in settings:
                            settings[key] = int(value)
                if None in settings.values():
                    raise ValueError("Some settings are missing in the file.")
                return settings.values()
        except (FileNotFoundError, ValueError) as e:
            print(f"Error reading settings from file: {e} please enter the settings manually.")
```

פונקציה זו אחראית להגדרת פרמטרים הקשורים לתקשורת כמו גודל חלון ההזזה (window_size) וזמן ההמתנה (timeout). המשתמש בוחר אם לקרוא את ההגדרות מקובץ או להזין אותן ידנית. במקרה של טעינה מקובץ, הפונקציה מחפשת ערכים בפורמט key: value כמו window_size: 5. אם חסר ערך או שהקובץ אינו תקין, המשתמש יתבקש להזין את הנתונים באופן ידני.

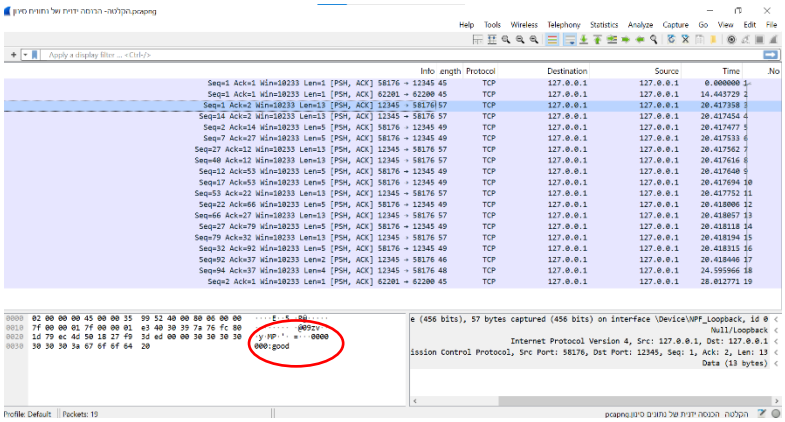
הקלטות Wireshark וצילומי קוד -

בתקית המטלה מצורפות 8 הקלטות ל4 מקרים (הרצות), עבור כל מקרה יש את ההקלטה הרגילה ואת ההקלטה עם הסינון שבצענו על מנת לראות שהקוד שלנו פועל כרואי.
את הסינון בצענו לפי דגל הpush.כך שנוכל לראות רק את ההודעות שנשלחות בין השרת ללקוח מכיוון שה-אקים (האישורים) שהשרת שולח ללקוח נמצאים אצלו בהודעה שהוא שולח ללקוח ולא ולא האקים המובנים בתקשורתTCP.
בהקלטות ללא הסינון ניתן לראות את ההתחברות בין השרת ללקוח (מה שלא רואים בהקלטה עם הסינון).

מקרה 1- הכנסת נתונים ידנית ללקוח והשרת:

הנתונים שהוכנסו- גודל הודעה מקסימלי- 5, גודל חלון-4, משך זמן ההמתנה- 3,
הודעה- "good luck for roni and bar today"

```
Sending segment 0: good
Sending segment 1: luck
Sending segment 2: for r
Sending segment 3: oni a
received ACK: 0
Sending new segment 0000004: nd ba
received ACK: 1
Sending new segment 0000005: r tod
received ACK: 2
Sending new segment 0000006: ay
received ACK: 3
received ACK: 4
received ACK: 5
received ACK: 6
All segments have been sent and the server receive
enter 1 to read from a file or 2 to write the message (if you want to end write in the message exit): 2
enter the message: exit
Connection closed.
```



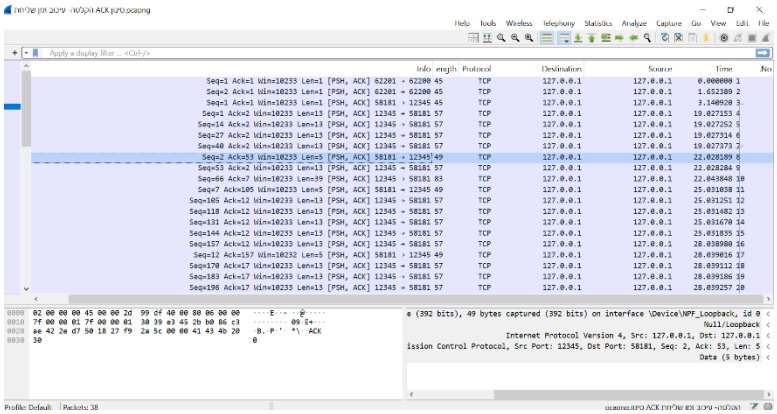
בהקלטה ניתן לראות שהסגמנטים 0 ו1 נשלחים ואז מגיעים אקים לפני ששלחנו את כל החלון וזאת מכיוון שהשרת והלקוח רצים במקביל ולכן האקים נשלחים תוך כדי- אך הלקוח מנהל את השליחה לפי קבלת האקים בסוף החלון.
מצורף למעלה צילום מההקלטה לדוגמא שבו רואים איפה ניתן לראות מה נשלח (מוקף באדום).

מקרה 2- הגעה לזמן הtimeOut-

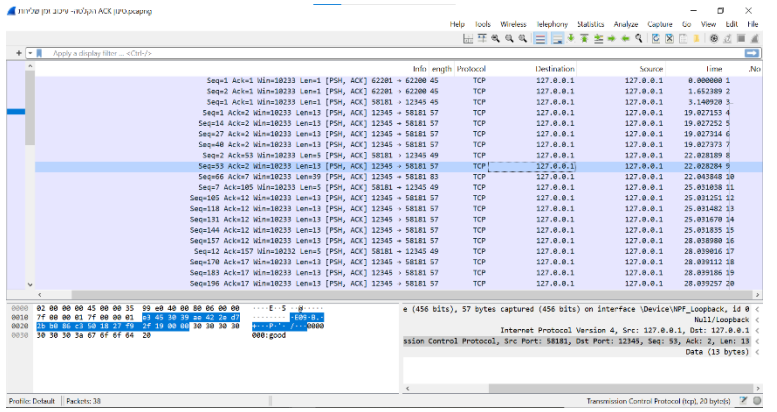
מקרה קצה זה הוא כאשר הלקוח מחכה ל-אק רלוונטי או בכלל ולא מקבל ואז נגמר הזמן ההמתנה שלו.
במקרה זה הוא צריך לשלוח מחדש את כל החלון ולאפס את הזמן.
כדי לדמות מקרה זה הוספנו בשרת לפני כל שליחה של אק המתנה של 3 שניות (כדי להאריך את זמן החזרת התגובה).
הפקודה שהוספנו- time.sleep(3)

(לצרוך ההדגמה הנתונים הוכנסו ידנית (לא מקובץ) והם אותם נתונים מההקלטה הקודמת).

```
enter the message: gool luck for bar and roni today
Message split into 7 segments: [('0000000', 'gool '), ('0000001', 'luck '), ('0000002', 'for b'), ('0000003', 'ar an'), ('0000004', 'd ron'), ('
Sending segment 0: gool
Sending segment 1: luck
Sending segment 2: for b
Sending segment 3: ar an
Timeout occurred while waiting for ACK. Exiting receive_acks.
Exiting receive_acks with next_to_send=0
Sending segment 0: gool
Sending segment 1: luck
Sending segment 2: for b
Sending segment 3: ar an
received ACK: 0
Timeout occurred while waiting for ACK. Exiting receive_acks.
Exiting receive_acks with next_to_send=1
Sending segment 1: luck
Sending segment 2: for b
Sending segment 3: ar an
Sending segment 4: d ron
received ACK: 1
Timeout occurred while waiting for ACK. Exiting receive_acks.
Exiting receive_acks with next_to_send=2
```



```
Sending segment 2: for b
Sending segment 3: ar an
Sending segment 4: d ron
Sending segment 5: i tod
received ACK: 2
Timeout occurred while waiting for ACK. Exiting receive_acks.
Exiting receive_acks with next_to_send=3
Sending segment 3: ar an
Sending segment 4: d ron
Sending segment 5: i tod
Sending segment 6: ay
received ACK: 3
Timeout occurred while waiting for ACK. Exiting receive_acks.
Exiting receive_acks with next_to_send=4
Sending segment 4: d ron
Sending segment 5: i tod
Sending segment 6: ay
received ACK: 4
Timeout occurred while waiting for ACK. Exiting receive_acks.
Exiting receive_acks with next_to_send=5
Sending segment 5: i tod
Sending segment 6: ay
received ACK: 5
Timeout occurred while waiting for ACK. Exiting receive_acks.
Exiting receive_acks with next_to_send=6
Sending segment 6: ay
received ACK: 6
All segments have been sent and the server receive
enter 1 to read from a file or 2 to write the message (if you want to end write in the message exit): 2
enter the message: exit
Connection closed.

Process finished with exit code 0
```



ניתן לראות בהקלטה שנשלח אק עבור 0 אבל הוא נשלח אחרי הזמן ולכן החלון כולו נשלח מההתחלה. וברגע שמתקבל אק בזמן החלון זז בהתאם.

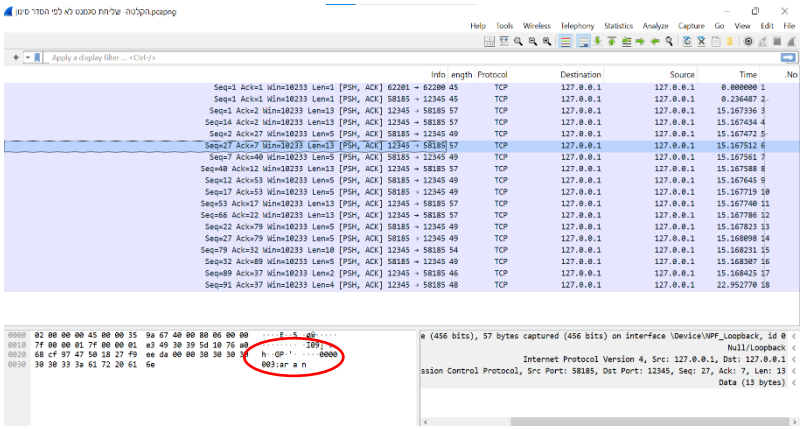
מקרה 3- ACK לא נשלח בסדר נכון –

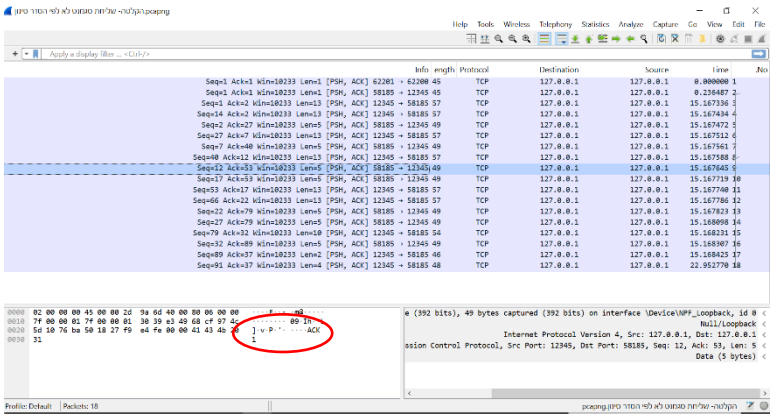
מקרה קצה זה הוא כאשר הסגמנטים לא הגיעו בסדר המצופה(לפי האינדקסים).
במקרה זה ה-אק שישלח ללקוח הוא עם האינדקס של הסגמנט האחרון שהגיע אליו ברצף.
במידה והגיע לשרת סגמנט לא בסדר הנכון הוא שומר אותו בכל מקרה וברגע שיגיע הסגמנט שחסר ברצף ה-אק שישלח הוא של הסדר עד כה.
לדוגמא בבדיקה שלנו (עם הנתונים של המקרה הראשון)- שלחנו את הסגמנטים בסדר הבר- 0,1,3,2 ורצינו לראות שלאחר שליחת סגמנט 3 נקבל אק 1 ולאחר 2 נקבל את אק 3.

בשביל לדמות את שינוי סדר הסגמנטים במקום לקורא לפעולת
sliding_window
הרגילה קראנו לפעולה
sliding_window_lose2
שהיא מבצעת אותו הדבר רק שולחת את הסגמנטים הראשונים בסדר שציינו קודם.

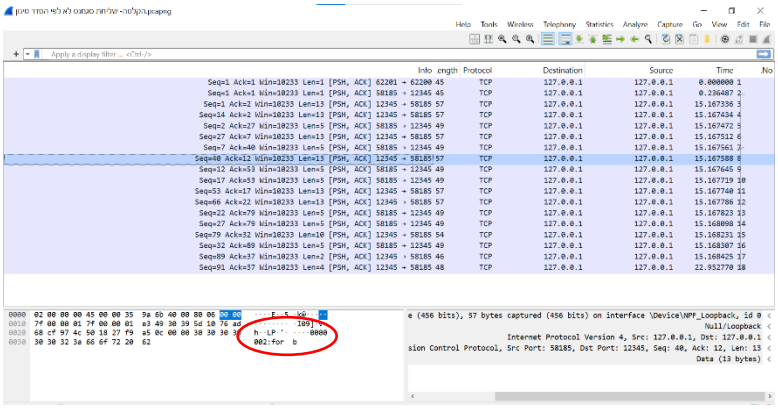
```python
def sliding_window_lose2(client_socket, message, max_msg_size, window_size, timeout):
    segments = []  # Split the message into segments based on the maximum size
    for i in range(0, len(message), max_msg_size):
        segment_index = i // max_msg_size
        segment = message[i:i + max_msg_size]
        if len(segment) < max_msg_size:
            segment = segment.ljust(max_msg_size,
                                    " ")  # If the length of the segment is less than the defined maximum size
        formatted_index = str(segment_index).zfill(7)  # header- 7 digits
        segments.append((formatted_index, segment))  # Store segment index and data
    print(f"Message split into {len(segments)} segments: {segments}")

    next_ack = 0  # Start with the first segment
    missing_segment_index = 2  # Index of the segment to simulate loss
    next_to_send = 0
    while next_to_send < len(segments):
        window_end = min(next_ack + window_size, len(segments))
        for i in range(next_to_send, window_end):  # send all segments in the current window
            index, segment = segments[i]
            if int(index) == missing_segment_index:  # Skip sending the missing segment
                print(f"Skipping segment {int(index)} to simulate loss.")
            else:
                client_socket.sendall(f"{index}:{segment}".encode())
                next_to_send += 1
                print(f"Sending segment {int(index)}: {segment}")

        # After sending the window, send the missing segment
        if missing_segment_index >= next_ack and missing_segment_index < window_end:
            index, segment = segments[missing_segment_index]
            print(f"Sending missing segment {int(index)}: {segment}")
            client_socket.sendall(f"{index}:{segment}".encode())
            next_to_send += 1

        next_ack,next_to_send = receive_acks(client_socket, segments, next_ack, window_size, timeout, window_end,next_to_send)
```

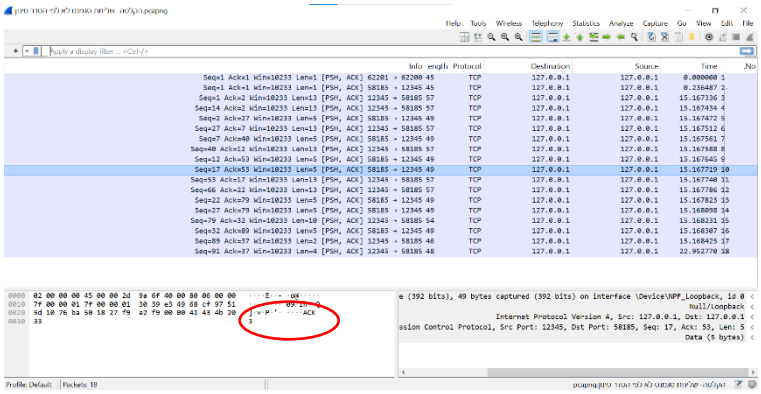
```
Message split into 7 segments: [('0000000', 'good '), ('0000001', 'luck '), ('0000002', 'for b'), ('0000003', 'ar an'), ('0000004', 'd ron'),
Sending segment 0: good
Sending segment 1: luck
Skipping segment 2 to simulate loss.
Sending segment 3: ar an
Sending missing segment 2: for b
received ACK: 0
Sending new segment 0000004: d ron
received ACK: 1
Sending new segment 0000005: i tod
received ACK: 1
received ACK: 3
Sending new segment 0000006: ay
received ACK: 4
received ACK: 5
received ACK: 6
All segments have been sent and the server receive
enter 1 to read from a file or 2 to write the message (if you want to end write in the message exit): 2
enter the message: exit
Connection closed.
```





ניתן לראות שלאחר שליחת סגמנט 3 קיבלנו ACK של 1





ניתן לראות שלאחר שליחת סגמנט 2 קיבלנו ACK של 3

<u>מקרה 4- קריאה מקובץ-</u>

הוספנו הקלטה שבה רואים שהלקוח והשרת עובדים גם מקריאה מקובץ על ידי הכנסת כתובת הקובץ על ידי המשתמש במידה והקובץ לא תקין המשתמש יתבקש להכניס את הנתונים ידנית.

הקוד שלנו תומך בקריאת קבצי- txt(windows)

מקורות שנעזרנו בהם במהלך המטלה-

כדי להבין איך מחשבים את הזמן שהלקוח מחכה לתשובה מהשרת ואיך לאפסו –

https://stackoverflow.com/questions/3432102/python-socket-connection-timeout

וכדי לדעת איך לעקב את הזמן של שליחת התגובה מהשרת -
https://www.geeksforgeeks.org/python-time-module/