

רשתות תקשורת מטלה 3

1. פירוט של הפתרון המוצע:

1.1 הסבר תמציתי של התוכנית - מבט על:

התוכנית בונה מערכת של תקשורת בין לקוח לשרת, שמוודאת שההודעות שמגיעות יישארו בסדר הנכון וגם יגיעו בצורה אמינה, בלי לאבד מידע.

מה הלקוח עושה?

פיצול הודעות: אם ההודעה ארוכה מדי, הלקוח מחלק אותה לפקטות שונות לפי הגודל שמותר לשלוח, כמו שהשרת קובע בעת תחילת התקשורת. (בין עם ע"י הזנת קלט של המשתמש או ע"י קריאה מקובץ חיצוני)

שליחה בשלבים: הלקוח שולח כל פעם כמה חבילות לפי גודל חלון מסוים מבלי לחכות לאישור מהשרת, יצירת חלון וקביעת גודלו נעשית על מנת לא לשלוח יותר מדי הודעות בבת אחת מה שעלול ליצור עומס ברשת או בזבז משאבים מיותרים במידה והשרת לא עונה.

מעקב אחרי ACK-ים: הלקוח מחכה לאישור מהשרת על כל חבילה שהגיעה, ואם האישור לחבילה הראשונה שנשלחה באותו החלון לא מגיע בזמן, הוא שולח את החבילה שוב.

שידור חוזר: אם יש בעיה, כמו שהשרת לא ענה בזמן המוגדר (timeout) או שחבילה הלכה לאיבוד, הלקוח שולח מחדש את החבילות שלא קיבלו אישור.

מה השרת עושה?

קבלת ההודעות: השרת מקבל את החבילות מהלקוח, ומוודא כי הן מגיעות בסדר הנכון.

ניהול חבילות out-of-order: אם חבילה מגיעה לפני החבילה שציפו לה, השרת שומר אותה בצד (ב-buffer) ומחכה לחבילה שחסרה.

שליחת אישורים (ACK): השרת שולח אישור (ACK) ללקוח אחרי כל חבילה שהגיעה בסדר הנכון. אם חבילה חסרה, הוא שולח אישור רק על החבילה האחרונה שקיבלה אישור.

שחרור חבילות מה-buffer: ברגע שחבילה חסרה מגיעה, השרת מוסיף אותה ואת כל החבילות ששמר בצד לרצף התקין. (ומוציא ACK על החבילה בעל המספר הסדורי הגבוה ביותר שהתקבלה)

בהסתכלות כללית:

הלקוח והשרת עובדים יחד כדי לוודא שההודעות מגיעות בשלמותן בסדר הנכון ובזמן המוגדר ומבצעים התאמות ותיקונים בכדי לדאוג לכך במידה ומשהו לא קורה כמו שצריך.

הלקוח לא שולח יותר מדי הודעות בבת אחת, והשרת מנהל את הטיפול בחבילות לא מסודרות או לא שלמות. אם יש בעיה, כמו איבוד חבילות, המערכת יודעת להתמודד עם זה בעזרת שידורים חוזרים ושימוש בטיימר. זה עוזר לוודא שכל המידע יגיע, אפילו אם יש בעיות בתקשורת.

1.2. ארכיטקטורה:

המנגנונים אותם מימשנו בפרויקט:

- 1.2.1 - מנגנון קריאת קובץ קונפיגורציה
- 1.2.2 - מנגנון חלון הזזה (Sliding Window)
- 1.2.3 - מנגנון אישורים (Acknowledgments - ACKs)
- 1.2.4 - מנגנון טיימר ו Timeout
- 1.2.5 - מנגנון חלוקה להודעות, שליחתן וקבלתן בגודל המתאים (Segmentation)
- 1.2.6 - מנגנון אגירת הודעות מחוץ לסדר (Out-of-Order Buffer)
- 1.2.7 - מנגנון פירוק ACK-ים סמוכים

1.2.1 מנגנון קריאת קובץ קונפיגורציה:**מה המנגנון עושה?**

גם הלקוח וגם השרת יודעים לקרוא פרמטרים חשובים מתוך קובץ קונפיגורציה (configuration file) במקום לקבל אותם מהמשתמש ישירות. זה מאפשר להגדיר מראש ערכים כמו גודל חלון, גודל הודעה מקסימלי, זמן timeout.

חלוקת המערך נעשית לפי הסימון ":", כך שהמערך במקום ה-0 הוא המפתח ובמיקום ה-1 הוא הערך של אותו מפתח. ערך ההחזרה של הפונקציה היא מערך מסוג DICT עם כל הערכים שנקראו מהקובץ טקסט.

```
def read_parameters_from_file(file_path): // or recv
    with open(file_path, 'r') as file:
        data = file.read()
    parameters = {}
    for line in data.splitlines():
        key, value = line.split(':', 1)
        parameters[key.strip()] = value.strip().strip('"')
    return parameters
```

פורמט הקובץ יהיה מסוג TXT ויהיה מוגדר בצורה הבאה:

```
message:"<string>"
maximum_msg_size:<int>
window_size:<int>
timeout:<int>
```

מימוש בלקוח:

הלקוח פונה אל המשתמש ומאפשר למשתמש לבחור ע"י הזנת file/input במידה ובחר file- תקרא את הנתונים מתוך קובץ קונפיגורציה שנבנה מראש בשימוש בפונקציה הנ"ל. במידה ובחר input - תקרא את הערכים לפי הזנתיו של המשתמש לתוך המשתנים וישתמש בהם. לאחר שקבענו את ההגדרות לפי השיטה הרצויה התוכנית תמתין לקבלת גודל הודעה מקסימלי מהשרת.

```
source = input("Enter 'file' to read parameters from a file or 'input' to
provide manually: ").strip().lower()
if source == 'file':
    file_path = input("Enter the file path: ").strip()
    params = read_parameters_from_file(file_path)
    message = params['message'].replace('"', '')
    window_size = int(params['window_size'])
    timeout = int(params['timeout'])
else:
    message = input("Enter the message: ").strip()
    window_size = int(input("Enter the sliding window size: ").strip())
    timeout = int(input("Enter the timeout in seconds: ").strip())
```

מימוש בשרת:

השרת מקבל בקשה מהלקוח לקבל את גודל ההודעה המקסימלי, קורא את גודל ההודעה המקסימלי מתוך קובץ קונפיגורציה או מאפשר למשתמש להזין אותו. (בהתאם לבחירת המשתמש) ולאחר מכן מחזיר ללקוח את גודל ההודעה המקסימלי שיוכל לשלוח.

במידה ובחר file- תקרא את הנתון מתוך קובץ קונפיגורציה שנבנה מראש בשימוש בפונקציה הנ"ל (שנמצאת גם אצלו כי לשניהם צריכה יכולת לקרוא מקובץ) ותשלח אותו ללקוח.
במידה ובחר input - תבקש מהמשתמש להזין את גודל ההודעה המקסימלי ותשלח אותו ללקוח.

```
data = client_socket.recv(1024).decode('utf-8')
if data.startswith("MAX_SIZE_REQUEST"):
    # Get parameters from user or file
    source = input("Enter 'file' to read MAX SIZE of the message from a file or
'input' to provide manually: ").strip().lower()
    if source == 'file':
        file_path = input("Enter the file path: ").strip()
        params = read_parameters_from_file(file_path)
        max_size = int(params['maximum_msg_size'])
    else:
        max_size = int(input("Enter the maximum message size: ").strip())

    client_socket.send(str(max_size).encode('utf-8'))
```

1.2.2 מנגנון חלון הזזה (Sliding Window)

מה המנגנון עושה?

מאפשר שליחה של מספר הודעות בו-זמנית עד למגבלה מוגדרת מראש של גודל החלון בין עם על ידי המשתמש או בין עם על ידי הנתונים בקובץ. בנוסף מנגנון חלון ההזזה מספק בקרה על כמות ההודעות שיכולות להישלח מבלי לקבל אישור (ACK) על הודעות קודמות.

מימוש בלקוח:

המנגנון ימשיך לעבוד כל עוד החבילה האחרונה שלא קיבלה ACK לא חורגת מגודל החלון. בכל איטרציה נבדוק שאנחנו נמצאים בגבולות החלון והאם יש עוד חבילות שעליו לשלוח. לאחר שנשלחו כל הפקטות שניתן בהתאם לגודל החלון נמתין לקבל ACK מהשרת, במידה ונקבל ACK על חבילה בחלון נקדם את החלון על מנת לשלוח את החבילות הבאות. ברגע שנשלח את כל ההודעות ונענה עליהן ב ACK (ע"י השרת) המנגנון יפסיק לעבוד.

```
while unacknowledged < total_messages:
    # Send messages within the window
    while next_to_send < unacknowledged + window_size and next_to_send < total_messages:
        send_message_with_boundary(client_socket, next_to_send, message_parts[next_to_send])
        print(f"Sent message {next_to_send}: {message_parts[next_to_send]}")

    try:
        # Receive ACKs
        ack_response = client_socket.recv(1024).decode('utf-8')
        ack_buffer += ack_response

        while "ACK" in ack_buffer:
            ack_index = ack_buffer.find("ACK")
            try:
                start_index = ack_index + 3
                end_index = start_index
                while end_index < len(ack_buffer) and ack_buffer[end_index].isdigit():
                    end_index += 1
                ack_number = int(ack_buffer[start_index:end_index])
                print(f"Server acknowledged up to message {ack_number}")
                unacknowledged = max(unacknowledged, ack_number + 1)
                ack_buffer = ack_buffer[end_index:]

            if unacknowledged == total_messages: # All messages acknowledged
                break
```

1.2.3 מנגנון אישורים (Acknowledgments - ACKs)

מה המנגנון עושה?

השרת שולח ACK ללקוח עבור החבילה האחרונה שהתקבלה ברצף.
אם הודעה הגיעה מחוץ לסדר, השרת שולח את ה-ACK האחרון התקף.

מימוש בשרת:

השרת מקבל חבילה, מפריד אותה לפי המספר שלה ולפי תוכן ההודעה ולאחר מכן מבצע בדיקה לגביה, נחלק את המימוש לשני תרחישים:

1. **החבילה לא הגיעה בסדר הנכון** - במצב כזה השרת ידפיס הודעה שהחבילה הגיעה בסדר לא נכון ומאחסן אותה בתוך buffer שיצרנו בשביל לשמור בו הודעות שהגיעו בסדר הלא נכון. השרת יוציא ACK רק על ההודעה האחרונה שהגיעה לפי הסדר.

```
else:
    # Out of order, store the message
    print(f"Message {message_number} out of order, storing: {message_data}")
    received_messages[message_number] = message_data
    # Resend the last valid ACK
    ack_message = f"ACK{highest_contiguous_ack}"
    client_socket.send(ack_message.encode('utf-8'))
    print(f"Resent: {ack_message}")
```

2. **החבילה התקבלה בסדר שנכון או שלפניה היו הודעות שלא הגיעו בסדר הנכון** - בדיקה שהחבילה היא אכן החבילה הבאה שלה אנו מצפים, במידה וכן נבדוק האם קיימות הודעות שנמצאות בbuffer, כלומר הודעות שהגיעו לפני ההודעה שקיבלנו אך לא הגיעו בסדר המתאים, לאחר מכן תתבצע שליחת ACK עם מספר החבילה הגבוה ביותר מבין ההודעות שהתקבלו.

```
if message_number == highest_contiguous_ack + 1:
    # Correct order, update highest_contiguous_ack
    print(f"Message {message_number} arrived in the correct order: {message_data}")
    highest_contiguous_ack = message_number

    # Check for any buffered messages that can now be acknowledged
    while highest_contiguous_ack + 1 in received_messages:
        highest_contiguous_ack += 1
        print(f"Adding buffered message {highest_contiguous_ack}: {received_messages.pop(highest_contiguous_ack)}")

    # Send a single ACK for the highest contiguous sequence number
    ack_message = f"ACK{highest_contiguous_ack}"
    client_socket.send(ack_message.encode('utf-8'))
    print(f"Sent: {ack_message}")
```

1.2.4 מנגנון טיימר ו Timeout

מה המנגנון עושה?

הלקוח מפעיל טיימר לאחר שהוא שולח את הפקטה הראשונה בחלון הנוכחי לשרת. הלקוח ממשיך את הטיימר כל עוד הפקטה הראשונה לא קיבלה ACK, כלומר הטיימר לא מתאפס עד שהפקטה הראשונה בחלון לא קיבלה את הACK שלה. אם אחרי הזמן המוגדר כ Timeout ע"י המשתמש/הקובץ לא קיבלנו ACK לפקטה הראשונה בחלון, נצהיר על TIMER EXPIRED ונשלח את כל סגמנט ההודעה מחדש עבור כל החבילות שלא הגיעו ליעדן ונאפס את הטיימר.

מימוש בלקוח:

בתחילת ההודעה הראשונה בכל חלון, אנחנו פותחים טיימר ע"י בדיקה האם קיים טיימר וגם אם ההודעה היא בגבולות חלון הזזה (window_size), אם הגענו לקצה חלון הזזה או שאם הגענו לפקטה האחרונה (מצב זה מתקיים אם $total_packets < window_size$) אז תתחיל את הטיימר.

```
while unacknowledged < total_messages:
    # Send messages within the window
    while next_to_send < unacknowledged + window_size and next_to_send <
total_messages:
        send_message_with_boundary(client_socket, next_to_send,
message_parts[next_to_send])
        print(f"Sent message {next_to_send}: {message_parts[next_to_send]}")

        if start_time is None and next_to_send == unacknowledged: # Start
timer for the first message in the window
            start_time = time.time()
            next_to_send += 1
```

לאחר ששלחנו את כל ההודעות בגבולות חלון ההזזה, אנחנו נרצה לבדוק האם התקבלו כל ה ACK's לכלל ההודעות הנמצאים בגבולות החלון. אם כן, תבטל את ריצת הטיימר עד חלון ההזזה החדש. מצב זה מתקיים אם ההודעות הגיעו בסדר הנכון ולא התקיימה בעיה במהלך הטיימר המוגדר.

```
while "ACK" in ack_buffer:
    ack_index = ack_buffer.find("ACK")
    try:
        start_index = ack_index + 3
        end_index = start_index
        while end_index < len(ack_buffer) and ack_buffer[end_index].isdigit():
            end_index += 1
        ack_number = int(ack_buffer[start_index:end_index])
        print(f"Server acknowledged up to message {ack_number}")
        unacknowledged = max(unacknowledged, ack_number + 1)
        ack_buffer = ack_buffer[end_index:]

        if unacknowledged == total_messages: # All messages acknowledged
            break

        # Reset timer if the first message in the window is acknowledged
        if start_time is not None and unacknowledged >= next_to_send:
            start_time = None
```

אחרת נבדוק האם הטיימר הגיעה לסופו, נשלח את כל ההודעות מההודעה הראשונה שלא קיבלה ACK's עד ההודעה האחרונה בחלון הזמן. ונכריז על TIMER EXPIRED. לאחר שליחת ההודעות נאפס את הטיימר.

```
# Check timer expiration for the first unacknowledged message
if start_time is not None and time.time() - start_time > timeout:
    print("Timer expired, resending unacknowledged messages...")
    for i in range(unacknowledged, next_to_send):
        send_message_with_boundary(client_socket, i, message_parts[i])
        print(f"Resent message {i}: {message_parts[i]}")
    start_time = time.time() # Restart the timer
```

בנוסף הלקוח מממש **מנגנון Timeout** שיהיה אחראי על בדיקה של קבלת הACK-ים, במידה ולא מתקבלת תגובת ACK מהשרת נדע לקבל פידבק מהפונקציה `recv_with_timeout` שאחרי זמן `timeout` שהגדרנו מראש, יש לשלוח את המידע מחדש על כל החבילות שלא קיבלו ACK החל מהראשונה:

```
try:
    # Receive ACKs with timeout
    ack_response = recv_with_timeout(client_socket, 1024, timeout)
    if ack_response is None:
        print("Timeout occurred, resending all unacknowledged messages...")
        for i in range(unacknowledged, next_to_send):
            send_message_with_boundary(client_socket, i, message_parts[i])
            print(f"Resent message {i}: {message_parts[i]}")
        start_time = time.time() # Restart the timer
        continue
```

```
def recv_with_timeout(client_socket, buffer_size, timeout):
    """
    Receives data with a timeout.
    """
    client_socket.settimeout(timeout)
    try:
        data = client_socket.recv(buffer_size)
        client_socket.settimeout(None) # Reset to blocking mode
        return data
    except socket.timeout:
        print("Timeout occurred while waiting for data.")
        return None
```


1.2.5 מנגנון חלוקה להודעות , שליחתן וקבלתן בגודל המתאים (Segmentation)

מה המנגנון עושה?

מכיוון שהפרויקט הגדיר שיש להגביל את הגודל של כל הודעה יצרנו מנגנון המפצל הודעות ארוכות להודעות קטנות יותר כדי לעמוד במגבלת הגודל שהשרת הגדיר ע"י חלוקה למספר הודעות כאשר גודל כל הודעה יהיה מקסימום הגודל המוגדר מראש.

מימוש בלקוח:

הלקוח מחלק הודעה גדולה לחלקים שגודלם הוא לכל היותר הגודל המקסימלי שקיבל מהשרת, ושולח אותם בנפרד:

```
# Define the message to be sent
# Comment left unchanged to preserve your original notes
message_parts = [message[i:i + max_size] for i in range(0, len(message),
max_size)]
print(f"Message chunks: {message_parts}")
```

לאחר מכן בעת שליחת כל "חלק" מן ההודעה לשרת מתבצע מספור והפרדה להודעות לפי אורכה של כל הודעה ע"י **מנגנון השליחה** המבוקר שיצרנו על מנת לייצר בקרה על הגעת ההודעות בגודל המותר ולפי סדר , המנגנון שומר את האורך של ההודעה במין header , מצמיד את מספר ההודעה לתוכן שלה ובכך דואג שנדע "לפרק" אותה נכון בשרת.

```
def send_message_with_boundary(client_socket, message_number, message_data):
    message = f"{message_number}:{message_data}"
    message_bytes = message.encode('utf-8')
    message_length = len(message_bytes)
    length_prefix = f"{message_length:04}".encode('utf-8')
    client_socket.send(length_prefix + message_bytes)
```

מימוש המנגנון בשרת:

בצד השרת, המנגנון מקבל קודם את 4 הבתים הראשונים של ההודעה שהוגדרו לייצג את האורך של ההודעה במימוש בצד הלקוח, לכן נקבל אותם ונבין מה גודל ההודעה שנרצה לקבל. לאחר מכן נגדיר קבלה של ההודעה באורך שנמסר לנו ובכך המנגנון דואג שנקבל רק את ההודעה ששלחנו ללא הודעות סמוכות. בנוסף, הוספנו חלק שבדק האם נשלחה הודעה ריקה או שהחיבור נסגר על מנת למנוע תקלות.

```
def recv_message_with_boundary(client_socket):
    """
    Reads a message with a 4-byte length prefix.
    """
    try:
        length_prefix = client_socket.recv(4)
        if not length_prefix:
            print("Client closed the connection.")
            return None # Connection closed by the client
        length_prefix = length_prefix.decode('utf-8')
        message_length = int(length_prefix)
        # Read the exact number of bytes for the message
        message = client_socket.recv(message_length).decode('utf-8')
        return message
    except ValueError:
        print("Error: Received an invalid message length prefix.")
        return None
    except ConnectionResetError:
        print("Connection was reset by the client.")
        return None
```

1.2.6 מנגנון אגירת הודעות מחוץ לסדר (Out-of-Order Buffer)

מה המנגנון עושה?

השרת מבצע שאילתה: האם ההודעה שהתקבלה היא שווה למספר ההודעה הקודמת שאושרה, אם כן אזי סדר ההודעות תקין, אחרת נכריז על ההודעה שהגיעה בסדר הלא נכון. "נאגור" את ההודעה במבנה נתונים LIST לטובת מנגנון ה"אישורים". לאחר מכן השרת שולח הודעת ACK להודעה האחרונה שנשלחה ובכך הודיע ללקוח לאיזו הודעה הוא מחכה שתשלח. כאשר הודעה חסרה מתקבלת, השרת משחרר את ההודעות מה-buffer ברצף.

מימוש בשרת:

```
if message_number == highest_contiguous_ack + 1:
    #CODE IS IMPLEMENTED HERE IN THE REAL FILE
else:
    # Out of order, store the message
    print(f"Message {message_number} out of order, storing: {message_data}")
    received_messages[message_number] = message_data
    # Resend the last valid ACK
    ack_message = f"ACK{highest_contiguous_ack}"
    client_socket.send(ack_message.encode('utf-8'))
    print(f"Resent: {ack_message}")
```

1.2.7 מנגנון פירוק ACK-ים סמוכים

מה המנגנון עושה?

המנגנון נועד לטפל במצב שבו הלקוח מקבל ACK-ים מהשרת ומתקבלים מספר ACK-ים סמוכים (למשל, "ACK1ACK2ACK3") בתוך אותו ה-buffer. כל ACK מפורק ומטופל בנפרד, כך שהלקוח יוכל להתקדם במעקב אחרי ההודעות שאושרו.

מימוש בלקוח:

הנתונים שמתקבלים מהשרת נשמרים בתוך ack_buffer, שהוא מחרוזת המכילה את כל ה-ACK-ים שנשלחו, כולל כאלה שעדיין לא טופלו. לאחר מכן המנגנון ימצא את האינדקס שבו מתחיל הרצף ACK ויבצע חילוץ למספר שנמצא אחרי המילה ACK בכל איטרציה כך שגם אם יהיה יותר ממספר אחד רצוף הוא יחולץ באופן תקין, לאחר כל חילוץ של מספר מ-ACK האינדקס יקודם למעבר ל-ACK הנוכחי ונודיע על קבלת ה-ACK עבור ההודעה.

```
while "ACK" in ack_buffer:
    ack_index = ack_buffer.find("ACK")
    try:
        start_index = ack_index + 3
        end_index = start_index
        while end_index < len(ack_buffer) and ack_buffer[end_index].isdigit():
            end_index += 1
        ack_number = int(ack_buffer[start_index:end_index])
        print(f"Server acknowledged up to message {ack_number}")
        unacknowledged = max(unacknowledged, ack_number + 1)
        ack_buffer = ack_buffer[end_index:]
```

2. צילומי מסך (תרחישי קצה ותרחישי בסיס):

בפרק זה נתמקד בניתוח שיטות תקשורת ובקרת תעבורה במערכות מבוססות TCP. נבחן שני סוגי תרחישים: תרחישי בסיס ותרחישי קצה.

2.1.0 תרחישי בסיס: נציג איך ניתן לחלק הודעה גדולה לפקטות קטנות יותר, ונדון בהעברתן תוך שמירה על הגבלות גודל הפקטה וגודל החלון. נראה דוגמאות לתהליכים של שליחה וקבלה בהגדרות שונות של גודל חלון וטיים-אאוט.

2.2.0 תרחישי קצה: נתייחס למצבים בהם נוצרות שגיאות כגון אובדן פקטות, קבלת פקטות מחוץ לסדר, והשפעות של טיימ-אאוט. נבחן איך המערכת מגיבה לאירועים אלה ואילו טכניקות ניתן לנקוט כדי לשפר את אמינות התקשורת.

הפרק יכלול צילומי מסך מ-Wireshark, שמראים באופן חזותי את פרטי התקשורת והתגובות של המערכת לשונות התרחישים. הצילומים יסייעו להדגים ולהפנים את התהליכים המתוארים בתרחיש, אשר יתנו מבט פנימי לתופעות המתרחשות במהלך תקשורת רשת פעילה.

2.1.0 תרחישי בסיס:

2.1.1. - תרחיש בסיס 1: חלוקת הודעה לפקטות (1)

2.1.2. - תרחיש בסיס 2: חלוקת הודעה לפקטות (2)

2.1.1. תרחיש בסיס 1: חלוקת הודעה לפקטות (1)

נציג כיצד טקסט בגודל 30 בתים נשלח ומתקבל כשגודל ההודעה המקסימלי הינו 10 בתים, נגדיר :

ההודעה עצמה : !!!DOR AND BARUH ARE THE BESTS

גודל ההודעה - 30 בתים.

גודל הודעה מקסימלית - 10 בתים

גודל החלון - 3 חבילות

טיים-אאוט - 5 שניות (לא רלוונטי)

תיאור פעילות הלקוח והשרת:

Server:

```
Server is listening on 127.0.0.1:12345
Connection from ('127.0.0.1', 62021) has been established.
Enter 'file' to read MAX SIZE of the message from a file or 'input' to provide manually: input
Enter the maximum message size: 10
Received: 0:DOR AND BA
Message 0 arrived in the correct order: DOR AND BA
Sent: ACK0
Received: 1:RUH ARE TH
Message 1 arrived in the correct order: RUH ARE TH
Sent: ACK1
Received: 2:E BESTS!!!
Message 2 arrived in the correct order: E BESTS!!!
Sent: ACK2
Connection closed.
```

Client:

```
Enter 'file' to read parameters from a file or 'input' to provide manually: input
Enter the message: DOR AND BARUH ARE THE BESTS!!!
Enter the sliding window size: 3
Enter the timeout in seconds: 5
Max message size from server is: 10 bytes
Adjusted maximum message size: 10 bytes
Timeout has set to: 5 seconds
Message chunks: ['DOR AND BA', 'RUH ARE TH', 'E BESTS!!!']
Sent message 0: DOR AND BA
Sent message 1: RUH ARE TH
Sent message 2: E BESTS!!!
Server acknowledged up to message 0
Server acknowledged up to message 1
Server acknowledged up to message 2
All messages sent and acknowledged!

Process finished with exit code 0
```

ניתוח פקטות ב-Wireshark:

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	127.0.0.1	127.0.0.1	TCP	56	62021 → 12345 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
2	0.000159	127.0.0.1	127.0.0.1	TCP	56	12345 → 62021 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
3	0.000204	127.0.0.1	127.0.0.1	TCP	44	62021 → 12345 [ACK] Seq=1 Ack=1 Win=65280 Len=0
4	14.046305	127.0.0.1	127.0.0.1	TCP	60	62021 → 12345 [PSH, ACK] Seq=1 Ack=1 Win=65280 Len=16
5	25.502195	127.0.0.1	127.0.0.1	TCP	46	12345 → 62021 [PSH, ACK] Seq=1 Ack=17 Win=65280 Len=2
6	25.502337	127.0.0.1	127.0.0.1	TCP	60	62021 → 12345 [PSH, ACK] Seq=17 Ack=3 Win=65280 Len=16
7	25.502427	127.0.0.1	127.0.0.1	TCP	60	62021 → 12345 [PSH, ACK] Seq=33 Ack=3 Win=65280 Len=16
8	25.502440	127.0.0.1	127.0.0.1	TCP	48	12345 → 62021 [PSH, ACK] Seq=3 Ack=33 Win=65280 Len=4
9	25.502481	127.0.0.1	127.0.0.1	TCP	60	62021 → 12345 [PSH, ACK] Seq=49 Ack=7 Win=65280 Len=16
10	25.502532	127.0.0.1	127.0.0.1	TCP	48	12345 → 62021 [PSH, ACK] Seq=7 Ack=65 Win=65280 Len=4
11	25.502562	127.0.0.1	127.0.0.1	TCP	48	12345 → 62021 [PSH, ACK] Seq=11 Ack=65 Win=65280 Len=4
12	25.502692	127.0.0.1	127.0.0.1	TCP	44	62021 → 12345 [FIN, ACK] Seq=65 Ack=15 Win=65280 Len=0
13	25.502720	127.0.0.1	127.0.0.1	TCP	44	12345 → 62021 [ACK] Seq=15 Ack=66 Win=65280 Len=0
14	25.502784	127.0.0.1	127.0.0.1	TCP	44	12345 → 62021 [FIN, ACK] Seq=15 Ack=66 Win=65280 Len=0
15	25.502816	127.0.0.1	127.0.0.1	TCP	44	62021 → 12345 [ACK] Seq=66 Ack=16 Win=65280 Len=0

1,2,3- לחיצת יד משולשת של הסרבר והקליינט.

4- השרת שולח לקליינט את הMAX_SIZE_REQUEST על מנת שידע שהוא רוצה לקבל את גודל ההודעה המקסימלית.

0000	02 00 00 00 45 00 00 38	fe 61 40 00 80 06 00 00E..8 .a@.....
0010	7f 00 00 01 7f 00 00 01	f2 45 30 39 d4 5d bc 3cE09.]<
0020	89 c9 23 2d 50 18 00 ff	b0 3e 00 00 4d 41 58 5f	..#-P...>..MAX
0030	53 49 5a 45 5f 52 45 51	55 45 53 54	SIZE_REQUEST

5- הלקוח מחזיר לשרת את גודל החבילה המקסימלי - במקרה שלנו 10

0000	02 00 00 00 45 00 00 2a	fe 69 40 00 80 06 00 00E..* .i@.....
0010	7f 00 00 01 7f 00 00 01	30 39 f2 45 89 c9 23 2d09.E.-#-
0020	d4 5d bc 4c 50 18 00 ff	1f 79 00 00 31 30	..]LP...y..10

6- שליחת החבילה הראשונה לאחר חלוקת החבילה הכוללת ל3 חלקים שכל אחד מהם בגודל 10 byte, להלן החבילה:

0000	02 00 00 00 45 00 00 38	fe 6b 40 00 80 06 00 00E..8 .k@.....
0010	7f 00 00 01 7f 00 00 01	f2 45 30 39 d4 5d bc 4cE09.]L
0020	89 c9 23 2f 50 18 00 ff	60 dd 00 00 30 30 31 32	..#/P...`...0012
0030	30 3a 44 4f 52 20 41 4e	44 20 42 41	0:DOR AN D BA

```
Sequence Number: 17 (relative sequence number)
Sequence Number (raw): 3562912844
[Next Sequence Number: 33 (relative sequence number)]
Acknowledgment Number: 3 (relative ack number)
```

7- שליחת החבילה השנייה מתוך ה 3

0000	02 00 00 00 45 00 00 38	fe 6d 40 00 80 06 00 00E..8 .m@.....
0010	7f 00 00 01 7f 00 00 01	f2 45 30 39 d4 5d bc 5cE09.]\\
0020	89 c9 23 2f 50 18 00 ff	48 bc 00 00 30 30 31 32	..#/P...H...0012
0030	31 3a 52 55 48 20 41 52	45 20 54 48	1:RUH AR E TH

```
Sequence Number: 33 (relative sequence number)
Sequence Number (raw): 3562912860
[Next Sequence Number: 49 (relative sequence number)]
Acknowledgment Number: 3 (relative ack number)
```

8- קבלת ACK0 לחבילה הראשונה מתוך 3

0000	02 00 00 00 45 00 00 2c	fe 6e 40 00 80 06 00 00E.., .n@.....
0010	7f 00 00 01 7f 00 00 01	30 39 f2 45 89 c9 23 2f09.E.-#/#
0020	d4 5d bc 5c 50 18 00 ff	c4 21 00 00 41 43 4b 30	..]LP...!..ACK0

9-שליחת החבילה השלישית מתוך ה 3

```

0000  02 00 00 00 45 00 00 38 fe 6f 40 00 80 06 00 00  ....E..8..o@.....
0010  7f 00 00 01 7f 00 00 01 f2 45 30 39 d4 5d bc 6c  ....E09..].1
0020  89 c9 23 33 50 18 00 ff 6d dc 00 00 30 30 31 32  ..#3P...m...0012
0030  32 3a 45 20 42 45 53 54 53 21 21 21 2:E BEST S!!!

```

```

Sequence Number: 49      (relative sequence number)
Sequence Number (raw): 3562912876
[Next Sequence Number: 65      (relative sequence number)]
Acknowledgment Number: 7      (relative ack number)

```

10-קבלת ACK1 לחבילה השנייה מתוך 3

```

0000  02 00 00 00 45 00 00 2c fe 70 40 00 80 06 00 00  ....E..,..p@.....
0010  7f 00 00 01 7f 00 00 01 30 39 f2 45 89 c9 23 33  ....E09..E..#3
0020  d4 5d bc 7c 50 18 00 ff c3 fc 00 00 41 43 4b 31  ..].|P...ACK1

```

11-קבלת ACK2 לחבילה השלישית מתוך 3

```

0000  02 00 00 00 45 00 00 2c fe 72 40 00 80 06 00 00  ....E..,..r@.....
0010  7f 00 00 01 7f 00 00 01 30 39 f2 45 89 c9 23 37  ....E09..E..#7
0020  d4 5d bc 7c 50 18 00 ff c3 f7 00 00 41 43 4b 32  ..].|P...ACK2

```

12,13,14,15 - סיום התקשורת - השרת מול הלקוח והלקוח אל מול השרת.

2.1.2. תרחיש בסיס 2: חלוקת הודעה לפקטות (2)

נציג כיצד טקסט בגודל 30 בתים נשלח ומתקבל כשגודל ההודעה המקסימלי הינו 5 בתים, זאת אומרת 6 חבילות, נגדיר את גודל החלון ל 4, כלומר 4 חבילות יעברו ולאחר מכן ה 2 הנוספות.
נגדיר :

ההודעה עצמה : !!!DOR AND BARUH ARE THE BESTS

גודל ההודעה - 30 בתים.

גודל הודעה מקסימלית - 5 בתים.

גודל החלון - 4 חבילות

טיים-אאוט - 5 שניות (לא רלוונטי)

תיאור פעילות הלקוח והשרת:

Server:

```
Server is listening on 127.0.0.1:12345
Connection from ('127.0.0.1', 63061) has been established.
Enter 'file' to read MAX SIZE of the message from a file or 'input' to provide manually: input
Enter the maximum message size: 5
Received: 0:DOR A
Message 0 arrived in the correct order: DOR A
Sent: ACK0
Received: 1:ND BA
Message 1 arrived in the correct order: ND BA
Sent: ACK1
Received: 2:RUH A
Message 2 arrived in the correct order: RUH A
Sent: ACK2
Received: 3:RE TH
Message 3 arrived in the correct order: RE TH
Sent: ACK3
Received: 4:E BES
Message 4 arrived in the correct order: E BES
Sent: ACK4
Received: 5:TS!!!
Message 5 arrived in the correct order: TS!!!
Sent: ACK5
Connection closed.
```

Client:

```
Enter 'file' to read parameters from a file or 'input' to provide manually: input
Enter the message: DOR AND BARUH ARE THE BESTS!!!
Enter the sliding window size: 4
Enter the timeout in seconds: 5
Max message size from server is: 5 bytes
Adjusted maximum message size: 5 bytes
Timeout has set to: 5 seconds
Message chunks: ['DOR A', 'ND BA', 'RUH A', 'RE TH', 'E BES', 'TS!!!']
Sent message 0: DOR A
Sent message 1: ND BA
Sent message 2: RUH A
Sent message 3: RE TH
Server acknowledged up to message 0
Server acknowledged up to message 1
Server acknowledged up to message 2
Sent message 4: E BES
Sent message 5: TS!!!
Server acknowledged up to message 3
Server acknowledged up to message 4
Server acknowledged up to message 5
All messages sent and acknowledged!

Process finished with exit code 0
```

ניתוח פקטות ב-Wireshark:

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	127.0.0.1	127.0.0.1	TCP	56	63061 → 12345 [SYN, Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
2	0.000116	127.0.0.1	127.0.0.1	TCP	56	12345 → 63061 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
3	0.000155	127.0.0.1	127.0.0.1	TCP	44	63061 → 12345 [ACK] Seq=1 Ack=1 Win=65280 Len=0
4	26.594980	127.0.0.1	127.0.0.1	TCP	60	63061 → 12345 [PSH, ACK] Seq=1 Ack=1 Win=65280 Len=16
5	38.259738	127.0.0.1	127.0.0.1	TCP	45	12345 → 63061 [PSH, ACK] Seq=1 Ack=17 Win=65280 Len=1
6	38.259903	127.0.0.1	127.0.0.1	TCP	55	63061 → 12345 [PSH, ACK] Seq=17 Ack=2 Win=65280 Len=11
7	38.259969	127.0.0.1	127.0.0.1	TCP	48	12345 → 63061 [PSH, ACK] Seq=2 Ack=28 Win=65280 Len=4
8	38.259970	127.0.0.1	127.0.0.1	TCP	55	63061 → 12345 [PSH, ACK] Seq=28 Ack=2 Win=65280 Len=11
9	38.260011	127.0.0.1	127.0.0.1	TCP	48	12345 → 63061 [PSH, ACK] Seq=6 Ack=39 Win=65280 Len=4
10	38.260037	127.0.0.1	127.0.0.1	TCP	55	63061 → 12345 [PSH, ACK] Seq=39 Ack=10 Win=65280 Len=11
11	38.260069	127.0.0.1	127.0.0.1	TCP	48	12345 → 63061 [PSH, ACK] Seq=10 Ack=50 Win=65280 Len=4
12	38.260081	127.0.0.1	127.0.0.1	TCP	55	63061 → 12345 [PSH, ACK] Seq=50 Ack=14 Win=65280 Len=11
13	38.260158	127.0.0.1	127.0.0.1	TCP	55	63061 → 12345 [PSH, ACK] Seq=61 Ack=14 Win=65280 Len=11
14	38.260194	127.0.0.1	127.0.0.1	TCP	55	63061 → 12345 [PSH, ACK] Seq=72 Ack=14 Win=65280 Len=11
15	38.260307	127.0.0.1	127.0.0.1	TCP	48	12345 → 63061 [PSH, ACK] Seq=14 Ack=83 Win=65280 Len=4
16	38.260340	127.0.0.1	127.0.0.1	TCP	48	12345 → 63061 [PSH, ACK] Seq=18 Ack=83 Win=65280 Len=4
17	38.260364	127.0.0.1	127.0.0.1	TCP	48	12345 → 63061 [PSH, ACK] Seq=22 Ack=83 Win=65280 Len=4
18	38.260445	127.0.0.1	127.0.0.1	TCP	44	63061 → 12345 [FIN, ACK] Seq=83 Ack=26 Win=65280 Len=0
19	38.260485	127.0.0.1	127.0.0.1	TCP	44	12345 → 63061 [ACK] Seq=26 Ack=84 Win=65280 Len=0
20	38.260593	127.0.0.1	127.0.0.1	TCP	44	12345 → 63061 [FIN, ACK] Seq=26 Ack=84 Win=65280 Len=0
21	38.260634	127.0.0.1	127.0.0.1	TCP	44	63061 → 12345 [ACK] Seq=84 Ack=27 Win=65280 Len=0

1,2,3- לחיצת יד משולשת של הסרבר והקליינט.

4- השרת שולח לקליינט את הMAX_SIZE_REQUEST על מנת שידע שהוא רוצה לקבל את גודל ההודעה המקסימלית.

0000	02 00 00 00 45 00 00 38	3f 06 40 00 80 06 00 00E..8.?@.....
0010	7f 00 00 01 7f 00 00 01	f6 55 30 39 dc 75 da d2U09.u..
0020	b8 f3 74 1d 50 18 00 ff	05 66 00 00 4d 41 58 5f	..t.P...f.MAX_
0030	53 49 5a 45 5f 52 45 51	55 45 53 54	SIZE_REQ UEST

5- הלקוח מחזיר לשרת את גודל החבילה המקסימלי - במקרה שלנו- 5

0000	02 00 00 00 45 00 00 29	3f 0a 40 00 80 06 00 00E..) ? @
0010	7f 00 00 01 7f 00 00 01	30 39 f6 55 b8 f3 74 1d09.U..t..
0020	dc 75 da e2 50 18 00 ff	70 d1 00 00 35	..u..P...p...5

6- שליחת החבילה הראשונה לאחר חלוקת החבילה הכוללת 6 חלקים שכל אחד מהם בגודל 5 byte , להלן החבילה:

0000	02 00 00 00 45 00 00 33	3f 0c 40 00 80 06 00 00E..3.?@.....
0010	7f 00 00 01 7f 00 00 01	f6 55 30 39 dc 75 da e2U09.u..
0020	b8 f3 74 1e 50 18 00 ff	3d b5 00 00 30 30 30 37	..t.P...=...0007
0030	30 3a 44 4f 52 20 41		0:DOR A

```
Sequence Number: 17      (relative sequence number)
Sequence Number (raw): 3698711266
[Next Sequence Number: 28      (relative sequence number)]
Acknowledgment Number: 2      (relative ack number)
```

7- שליחת ACK0 לחבילה הראשונה, מצד השרת ללקוח:

0000	02 00 00 00 45 00 00 2c	3f 0e 40 00 80 06 00 00E..,.?@.....
0010	7f 00 00 01 7f 00 00 01	30 39 f6 55 b8 f3 74 1e09.U..t..
0020	dc 75 da ed 50 18 00 ff	19 4f 00 00 41 43 4b 30	..u..P...O..ACK0

8- שליחת החבילה השנייה מתוך ה-6:

```

0000  02 00 00 00 45 00 00 33 3f 0f 40 00 80 06 00 00  ....E..3 ?.@.....
0010  7f 00 00 01 7f 00 00 01 f6 55 30 39 dc 75 da ed  ....U09.u..
0020  b8 f3 74 1e 50 18 00 ff 64 93 00 00 30 30 30 37  ..t.P...d...0007
0030  31 3a 4e 44 20 42 41                                1:ND BA

```

```

Sequence Number: 28      (relative sequence number)
Sequence Number (raw): 3698711277
[Next Sequence Number: 39      (relative sequence number)]
Acknowledgment Number: 2      (relative ack number)

```

9-שליחת ACK1 לחבילה השנייה, מצד השרת ללקוח:

```

0000  02 00 00 00 45 00 00 2c 3f 12 40 00 80 06 00 00  ....E.., ?.@.....
0010  7f 00 00 01 7f 00 00 01 30 39 f6 55 b8 f3 74 22  ....09.U..t"
0020  dc 75 da f8 50 18 00 ff 19 3f 00 00 41 43 4b 31  ..u..P...?..ACK1

```

10- שליחת החבילה השלישית מתוך ה-6: (עדיין כחלק מהשליחה של החלון הראשון)

```

0000  02 00 00 00 45 00 00 33 3f 13 40 00 80 06 00 00  ....E..3 ?.@.....
0010  7f 00 00 01 7f 00 00 01 f6 55 30 39 dc 75 da f8  ....U09.u..
0020  b8 f3 74 26 50 18 00 ff 37 91 00 00 30 30 30 37  ..t&P...7...0007
0030  32 3a 52 55 48 20 41                                2:RUH A

```

```

Sequence Number: 39      (relative sequence number)
Sequence Number (raw): 3698711288
[Next Sequence Number: 50      (relative sequence number)]
Acknowledgment Number: 10      (relative ack number)

```

11-שליחת ACK2 לחבילה השלישית, מצד השרת ללקוח:

```

0000  02 00 00 00 45 00 00 2c 3f 15 40 00 80 06 00 00  ....E.., ?.@.....
0010  7f 00 00 01 7f 00 00 01 30 39 f6 55 b8 f3 74 26  ....09.U..t&
0020  dc 75 db 03 50 18 00 ff 19 2f 00 00 41 43 4b 32  ..u..P.../..ACK2

```

12-שליחת החבילה הרביעית והאחרונה לחלון הראשון מתוך ה-6:

```

0000  02 00 00 00 45 00 00 33 3f 17 40 00 80 06 00 00  ....E..3 ?.@.....
0010  7f 00 00 01 7f 00 00 01 f6 55 30 39 dc 75 db 03  ....U09.u..
0020  b8 f3 74 2a 50 18 00 ff 57 5e 00 00 30 30 30 37  ..t*P...W^...0007
0030  33 3a 52 45 20 54 48                                3:RE TH

```

```

[... Sequence Length 11]
Sequence Number: 50      (relative sequence number)
Sequence Number (raw): 3698711299
[Next Sequence Number: 61      (relative sequence number)]
Acknowledgment Number: 14      (relative ack number)

```

13-שליחת החבילה החמישית מתוך ה-6: (תישלח מכיוון שקיבלנו ACK-ים קודמים)

```

0000 02 00 00 00 45 00 00 33 3f 19 40 00 80 06 00 00  ....E..3 ?.@.....
0010 7f 00 00 01 7f 00 00 01 f6 55 30 39 dc 75 db 0e  ....U09.u..
0020 b8 f3 74 2a 50 18 00 ff 36 87 00 00 30 30 30 37  ..t*P...6...0007
0030 34 3a 45 20 42 45 53 4:E BJS

```

```

Sequence Number: 61      (relative sequence number)
Sequence Number (raw): 3698711310
[Next Sequence Number: 72      (relative sequence number)]
Acknowledgment Number: 14      (relative ack number)

```

14-שליחת החבילה השישית והאחרונה מתוך ה-6:

```

0000 02 00 00 00 45 00 00 33 3f 1b 40 00 80 06 00 00  ....E..3 ?.@.....
0010 7f 00 00 01 7f 00 00 01 f6 55 30 39 dc 75 db 19  ....U09.u..
0020 b8 f3 74 2a 50 18 00 ff 79 6d 00 00 30 30 30 37  ..t*P...ym..0007
0030 35 3a 54 53 21 21 21 5:TS!!!

```

```

Sequence Number: 72      (relative sequence number)
Sequence Number (raw): 3698711321
[Next Sequence Number: 83      (relative sequence number)]
Acknowledgment Number: 14      (relative ack number)

```

15-שליחת ACK3 לחבילה הרביעית, מצד השרת ללקוח:

```

0000 02 00 00 00 45 00 00 2c 3f 1d 40 00 80 06 00 00  ....E.., ?.@.....
0010 7f 00 00 01 7f 00 00 01 30 39 f6 55 b8 f3 74 2a  ....09.U..t*
0020 dc 75 db 24 50 18 00 ff 19 09 00 00 41 43 4b 33  ..u.$P...ACK3

```

16-שליחת ACK4 לחבילה השישית, מצד השרת ללקוח:

```

0000 02 00 00 00 45 00 00 2c 3f 1f 40 00 80 06 00 00  ....E.., ?.@.....
0010 7f 00 00 01 7f 00 00 01 30 39 f6 55 b8 f3 74 2e  ....09.U..t.
0020 dc 75 db 24 50 18 00 ff 19 04 00 00 41 43 4b 34  ..u.$P...ACK4

```

17-שליחת ACK5 לחבילה החמישית, מצד השרת ללקוח:

```

0000 02 00 00 00 45 00 00 2c 3f 21 40 00 80 06 00 00  ....E.., ?!@.....
0010 7f 00 00 01 7f 00 00 01 30 39 f6 55 b8 f3 74 32  ....09.U..t2
0020 dc 75 db 24 50 18 00 ff 18 ff 00 00 41 43 4b 35  ..u.$P...ACK5

```

18,19,20,21 - סיום התקשורת - השרת מול הלקוח והלקוח אל מול השרת.

2.2.0 תרחישי קצה לבדיקות:

2.2.1 - תרחיש קצה 1: איבוד חבילה

2.2.2 - תרחיש קצה 2: חבילות מחוץ לסדר

2.2.3 - תרחיש קצה 3: timeout

2.2.1 תרחיש קצה 1: איבוד חבילה

תיאור התרחיש: חבילה אובדת במהלך השידור - נגרום לחבילה מספר 2 מתוך 3 לא להגיע בשידור הראשון.
 *נבדוק שאכן קיבלנו ACK רק על החבילה הראשונה ושהיא נשלחת שוב.
 *נבדוק שקיבלנו רק את ACK ה 3 שהוא הגבוה מבניהם.

נגדיר:

ההודעה עצמה : ale
 גודל ההודעה - 3 בתים.
 גודל הודעה מקסימלית - 1 בתים.
 גודל החלון - 3 חבילות
 טיים-אאוט - 3 שניות

שינוי קוד שליחת ההודעה כך שיפספס לשלוח את ההודעה האי-זוגית:

```
def send_message_with_boundary(client_socket, message_number,
message_data, extreme_case_index):
    """
    Sends a message with a number and a 4-byte length prefix.
    """
    message = f"{message_number}:{message_data}"
    message_bytes = message.encode('utf-8')
    message_length = len(message_bytes)
    length_prefix = f"{message_length:04}".encode('utf-8')
    if message_number % 2 == 0 or extreme_case_index['index'] % 3 == 0:
        client_socket.send(length_prefix + message_bytes)
        extreme_case_index['index'] += 1
def start_client():
    extreme_case_index = {"index": 0}
```

תיאור פעילות הלקוח והשרת:

Server:

Server is listening on 127.0.0.1:12345
Connection from ('127.0.0.1', 54228) has been established.
Enter 'file' to read MAX SIZE of the message from a file or 'input' to provide manually: input
Enter the maximum message size: 1
Received: 0:a
Message 0 arrived in the correct order: a
Sent: ACK0
Received: 2:e
Message 2 out of order, storing: e
Resent: ACK0
Received: 1:l
Message 1 arrived in the correct order: l
Adding buffered message 2: e
Sent: ACK2
Received: 2:e
Message 2 out of order, storing: e
Resent: ACK2
Client closed the connection.
Connection closed.

Client:

Enter 'file' to read parameters from a file or 'input' to provide manually: input
Enter the message: ale
Enter the sliding window size: 3
Enter the timeout in seconds: 3
Max message size from server is: 1 bytes
Adjusted maximum message size: 1 bytes
Timeout has set to: 3 seconds
Message chunks: ['a', 'l', 'e']
Sent message 0: a
Sent message 1: l
Sent message 2: e
Server acknowledged up to message 0
Server acknowledged up to message 0
Timer expired, resending unacknowledged messages...
Resent message 1: l
Resent message 2: e
Server acknowledged up to message 2
All messages sent and acknowledged!

ניתוח פקטות ב-Wireshark:

Time	Source	Destination	Protocol	Length	Info
1 0.000000	127.0.0.1	127.0.0.1	TCP	56	54228 → 12345 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
2 0.000038	127.0.0.1	127.0.0.1	TCP	56	12345 → 54228 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
3 0.000050	127.0.0.1	127.0.0.1	TCP	44	54228 → 12345 [ACK] Seq=1 Ack=1 Win=2161152 Len=0
4 35.187285	127.0.0.1	127.0.0.1	TCP	60	54228 → 12345 [PSH, ACK] Seq=1 Ack=1 Win=2161152 Len=16
5 39.979549	127.0.0.1	127.0.0.1	TCP	45	12345 → 54228 [PSH, ACK] Seq=1 Ack=17 Win=2161152 Len=1
6 44.189386	127.0.0.1	127.0.0.1	TCP	51	54228 → 12345 [PSH, ACK] Seq=17 Ack=2 Win=2161152 Len=7
7 44.189480	127.0.0.1	127.0.0.1	TCP	48	12345 → 54228 [PSH, ACK] Seq=2 Ack=24 Win=2161152 Len=4
8 51.464840	127.0.0.1	127.0.0.1	TCP	51	54228 → 12345 [PSH, ACK] Seq=24 Ack=6 Win=2161152 Len=7
9 51.464934	127.0.0.1	127.0.0.1	TCP	48	12345 → 54228 [PSH, ACK] Seq=6 Ack=31 Win=2161152 Len=4
10 75.136640	127.0.0.1	127.0.0.1	TCP	51	54228 → 12345 [PSH, ACK] Seq=31 Ack=10 Win=2161152 Len=7
11 75.136771	127.0.0.1	127.0.0.1	TCP	48	12345 → 54228 [PSH, ACK] Seq=10 Ack=38 Win=2161152 Len=4
12 77.455034	127.0.0.1	127.0.0.1	TCP	51	54228 → 12345 [PSH, ACK] Seq=38 Ack=14 Win=2161152 Len=7
13 77.455121	127.0.0.1	127.0.0.1	TCP	48	12345 → 54228 [PSH, ACK] Seq=14 Ack=45 Win=2161152 Len=4
14 86.356676	127.0.0.1	127.0.0.1	TCP	44	54228 → 12345 [FIN, ACK] Seq=45 Ack=18 Win=2161152 Len=0
15 86.356691	127.0.0.1	127.0.0.1	TCP	44	12345 → 54228 [ACK] Seq=18 Ack=46 Win=2161152 Len=0
16 86.356753	127.0.0.1	127.0.0.1	TCP	44	12345 → 54228 [FIN, ACK] Seq=18 Ack=46 Win=2161152 Len=0
17 86.356766	127.0.0.1	127.0.0.1	TCP	44	54228 → 12345 [ACK] Seq=46 Ack=19 Win=2161152 Len=0

1,2,3- לחיצת יד משולשת של הסרבר והקליינט.

4- השרת שולח לקליינט את ה-MAX_SIZE_REQUEST על מנת שידע שהוא רוצה לקבל את גודל ההודעה המקסימלית.

0000	02 00 00 00 45 00 00 38	3a 33 40 00 80 06 00 00E..8 :3@.....
0010	7f 00 00 01 7f 00 00 01	d3 d4 30 39 ee b9 a2 7209...r
0020	24 89 6c 49 50 18 20 fa	ca 46 00 00 4d 41 58 5f	\$.LIP...F..MAX
0030	53 49 5a 45 5f 52 45 51	55 45 53 54	SIZE_REQ UEST

5- הלקוח מחזיר לשרת את גודל החבילה המקסימלי - במקרה שלנו- 1

0000	02 00 00 00 45 00 00 29	3a 62 40 00 80 06 00 00E..):b@.....
0010	7f 00 00 01 7f 00 00 01	30 39 d3 d4 24 89 6c 4909...\$.LI
0020	ee b9 a2 82 50 18 20 fa	39 b2 00 00 31P...9...1

6- שליחת החבילה ה-0 לאחר חלוקת החבילה הכוללת 3 חלקים שכל אחד מהם בגודל 1 byte, להלן החבילה:

0000	02 00 00 00 45 00 00 2f	3a e7 40 00 80 06 00 00E../ :@.....
0010	7f 00 00 01 7f 00 00 01	d3 d4 30 39 ee b9 a2 8209.....
0020	24 89 6c 4a 50 18 20 fa	79 0d 00 00 30 30 30 33	\$.LJP...y...0003
0030	30 3a 61		0:a

```
Sequence Number: 17      (relative sequence number)
Sequence Number (raw): 4005143170
[Next Sequence Number: 24      (relative sequence number)]
Acknowledgment Number: 2      (relative ack number)
```

7- שליחת ACK0 לחבילה ה-0, מצד השרת ללקוח:

0000	02 00 00 00 45 00 00 2c	3a eb 40 00 80 06 00 00E., :@.....
0010	7f 00 00 01 7f 00 00 01	30 39 d3 d4 24 89 6c 4a09...\$.LJ
0020	ee b9 a2 89 50 18 20 fa	de 33 00 00 41 43 4b 30P...3...ACK0

8- מכיוון שמנענו באופן יזום את שליחת חבילה 1, נשלח את החבילה 2 וניצור מצב של קבלת חבילות מחוץ לסדר: (out of order)

```

0000  02 00 00 00 45 00 00 2f 3c 0c 40 00 80 06 00 00  ....E.. / <.@.....
0010  7f 00 00 01 7f 00 00 01 d3 d4 30 39 ee b9 a2 89  ....09.....
0020  24 89 6c 4e 50 18 20 fa 73 02 00 00 30 30 30 33  $.lNP. . s...0003
0030  32 3a 65                                     2:e

```

```

Sequence Number: 24      (relative sequence number)
Sequence Number (raw): 4005143177
[Next Sequence Number: 31      (relative sequence number)]
Acknowledgment Number: 6      (relative ack number)

```

9-נקבל בתגובה ACK0 לחבילה השנייה, מצד השרת ללקוח, מכיוון שהגדרנו לשרת שכאשר אין רצף של חבילות הוא ישלח את הACK של החבילה האחרונה שהתקבלה ברצף:

```

0000  02 00 00 00 45 00 00 2c 3c 10 40 00 80 06 00 00  ....E.. , <.@.....
0010  7f 00 00 01 7f 00 00 01 30 39 d3 d4 24 89 6c 4e  ....09...$.lN
0020  ee b9 a2 90 50 18 20 fa de 28 00 00 41 43 4b 30  ....P. . .(..ACK0

```

10- לאחר מכן נקבל timeout מהשרת מכיוון שאנחנו מחכים ל ACK על שאר החבילות שלא נשלחו , מה שיגרם לשליחה מחודשת של כל החבילות שלא קיבלו ACK , כלומר חבילה 1 וחבילה 2.

```

0000  02 00 00 00 45 00 00 2f 3d 65 40 00 80 06 00 00  ....E.. / =e@.....
0010  7f 00 00 01 7f 00 00 01 d3 d4 30 39 ee b9 a2 90  ....09.....
0020  24 89 6c 52 50 18 20 fa 6c f7 00 00 30 30 30 33  $.lRP. . 1...0003
0030  31 3a 6c                                     1:1

```

```

Sequence Number: 31      (relative sequence number)
Sequence Number (raw): 4005143184
[Next Sequence Number: 38      (relative sequence number)]
Acknowledgment Number: 10      (relative ack number)

```

11-מכיוון שהשרת אוגר בbuffer את ההודעות שהגיעו אליו ולא קיבלו ACK ברגע שקיבלנו את הודעה 1 נוציא מהbuffer את הודעה 2 ונשלח ACK על ההודעה עם המספר הסידורי הגדול ביותר שהגיע כלומר ACK2

```

0000  02 00 00 00 45 00 00 2c 3d 69 40 00 80 06 00 00  ....E.. , =i@.....
0010  7f 00 00 01 7f 00 00 01 30 39 d3 d4 24 89 6c 52  ....09...$.lR
0020  ee b9 a2 97 50 18 20 fa de 1b 00 00 41 43 4b 32  ....P. . ...ACK2

```

12-נשלח גם את חבילה 2 שוב מכיוון שכך עובד מנגנון Timeout - שולח את כל ההודעות עליהן לא התקבל ACK , הוא מקבל אינדקציה לפי ההודעה הראשונה.

```

0000  02 00 00 00 45 00 00 2f 3d ab 40 00 80 06 00 00  ....E.. / =@.....
0010  7f 00 00 01 7f 00 00 01 d3 d4 30 39 ee b9 a2 97  ....09.....
0020  24 89 6c 56 50 18 20 fa 72 ec 00 00 30 30 30 33  $.lVP. . r...0003
0030  32 3a 65                                     2:e

```

```

Sequence Number: 38      (relative sequence number)
Sequence Number (raw): 4005143191
[Next Sequence Number: 45      (relative sequence number)]
Acknowledgment Number: 14      (relative ack number)

```

13-נקבל שוב את ACK2, יש התנגשות בין המנגנונים אך המצב טבעי עקב ההוראות אליהן התבקשנו במטלה.

0000	02 00 00 00 45 00 00 2c 3d af 40 00 80 06 00 00	...E., =@...
0010	7f 00 00 01 7f 00 00 01 30 39 d3 d4 24 89 6c 5609..\$.1V
0020	ee b9 a2 9e 50 18 20 fa de 10 00 00 41 43 4b 32	...P.ACK2

לאחר מכן קיבלנו ACK-ים על כל ההודעות וניתן לסיים את התקשורת.

14,15,16,17 - סיום התקשורת - השרת מול הלקוח והלקוח אל מול השרת.

2.2.2 תרחיש קצה 2: timeout

תיאור התרחיש: טיימר של הלקוח פג מבלי שהתקבל ACK -נגדיר את ה-timeout בשרת להיות 4 שניות ולפני שליחת ה-ACK נגדיר sleep למשך 5 שניות מה שיצור חריגה מה-timeout שהגדרנו לשרת, ולאחר מכן: *נבטל את ה-sleep ונבדוק שכל החבילות נשלחו מחדש לאחר תפוגת זמן הTimeout וקיבלו ACK.

נגדיר:

ההודעה עצמה : !!!DOR AND BARUH ARE THE BESTS

גודל ההודעה - 30 בתים.

גודל הודעה מקסימלית - 10 בתים.

גודל החלון - 3 חבילות

טיים-אאוט - 4 שניות

תיאור פעילות הלקוח והשרת:

Client:

```
Enter 'file' to read parameters from a file or 'input' to provide manually: input
Enter the message: DOR AND BARUH ARE THE BESTS!!!
Enter the sliding window size: 3
Enter the timeout in seconds: 4
Max message size from server is: 10 bytes
Adjusted maximum message size: 10 bytes
Timeout has set to: 4 seconds
Message chunks: ['DOR AND BA', 'RUH ARE TH', 'E BESTS!!!']
Sent message 0: DOR AND BA
Sent message 1: RUH ARE TH
Sent message 2: E BESTS!!!
Timeout occurred while waiting for data.
Timeout occurred, resending all unacknowledged messages...
Resent message 0: DOR AND BA
Resent message 1: RUH ARE TH
Resent message 2: E BESTS!!!
Server acknowledged up to message 0
Server acknowledged up to message 1
Server acknowledged up to message 2
All messages sent and acknowledged!

Process finished with exit code 0
```

Baruh Ifraimov - 208526012

Dor Cohen - 211896279

Server:

```
Connection from ('127.0.0.1', 57108) has been established.
Enter 'file' to read MAX SIZE of the message from a file or 'input' to provide manually: input
Enter the maximum message size: 10
Received: 0:DOR AND BA
Message 0 arrived in the correct order: DOR AND BA
Simulating timeout by delaying response...
Sent: ACK0
Received: 1:RUH ARE TH
Message 1 arrived in the correct order: RUH ARE TH
Sent: ACK1
Received: 2:E BESTS!!!
Message 2 arrived in the correct order: E BESTS!!!
Sent: ACK2
Received: 0:DOR AND BA
Message 0 out of order, storing: DOR AND BA
Resent: ACK2
Received: 1:RUH ARE TH
Message 1 out of order, storing: RUH ARE TH
Resent: ACK2
Received: 2:E BESTS!!!
Message 2 out of order, storing: E BESTS!!!
Resent: ACK2
Connection was reset by the client.
No more data from the client. Closing connection.
Connection closed.
```

ניתוח פקטות ב-Wireshark:

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	127.0.0.1	127.0.0.1	TCP	56	57108 → 12345 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
2	0.000060	127.0.0.1	127.0.0.1	TCP	56	12345 → 57108 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
3	0.000084	127.0.0.1	127.0.0.1	TCP	44	57108 → 12345 [ACK] Seq=1 Ack=1 Win=65280 Len=0
4	15.019949	127.0.0.1	127.0.0.1	TCP	60	57108 → 12345 [PSH, ACK] Seq=1 Ack=1 Win=65280 Len=16
5	20.811356	127.0.0.1	127.0.0.1	TCP	46	12345 → 57108 [PSH, ACK] Seq=1 Ack=17 Win=65280 Len=2
6	20.811531	127.0.0.1	127.0.0.1	TCP	60	57108 → 12345 [PSH, ACK] Seq=17 Ack=3 Win=65280 Len=16
7	20.811602	127.0.0.1	127.0.0.1	TCP	60	57108 → 12345 [PSH, ACK] Seq=33 Ack=3 Win=65280 Len=16
8	20.811637	127.0.0.1	127.0.0.1	TCP	60	57108 → 12345 [PSH, ACK] Seq=49 Ack=3 Win=65280 Len=16
9	24.827105	127.0.0.1	127.0.0.1	TCP	60	57108 → 12345 [PSH, ACK] Seq=65 Ack=3 Win=65280 Len=16
10	24.827154	127.0.0.1	127.0.0.1	TCP	60	57108 → 12345 [PSH, ACK] Seq=81 Ack=3 Win=65280 Len=16
11	24.827170	127.0.0.1	127.0.0.1	TCP	60	57108 → 12345 [PSH, ACK] Seq=97 Ack=3 Win=65280 Len=16
12	25.812107	127.0.0.1	127.0.0.1	TCP	48	12345 → 57108 [PSH, ACK] Seq=3 Ack=113 Win=65280 Len=4
13	25.812231	127.0.0.1	127.0.0.1	TCP	48	12345 → 57108 [PSH, ACK] Seq=7 Ack=113 Win=65280 Len=4
14	25.812269	127.0.0.1	127.0.0.1	TCP	48	12345 → 57108 [PSH, ACK] Seq=11 Ack=113 Win=65280 Len=4
15	25.812303	127.0.0.1	127.0.0.1	TCP	48	12345 → 57108 [PSH, ACK] Seq=15 Ack=113 Win=65280 Len=4
16	25.812335	127.0.0.1	127.0.0.1	TCP	48	12345 → 57108 [PSH, ACK] Seq=19 Ack=113 Win=65280 Len=4
17	25.812367	127.0.0.1	127.0.0.1	TCP	48	12345 → 57108 [PSH, ACK] Seq=23 Ack=113 Win=65280 Len=4
18	25.812382	127.0.0.1	127.0.0.1	TCP	44	57108 → 12345 [RST] Seq=113 Win=0 Len=0
19	25.812382	127.0.0.1	127.0.0.1	TCP	44	57108 → 12345 [RST, ACK] Seq=113 Ack=23 Win=0 Len=0

1,2,3- לחיצת יד משולשת של הסרבר והקליינט.

4- השרת שולח לקליינט את הMAX_SIZE_REQUEST על מנת שידע שהוא רוצה לקבל את גודל ההודעה המקסימלית.

0000	02 00 00 00 45 00 00 38	29 c5 40 00 80 06 00 00E..8).@.....
0010	7f 00 00 01 7f 00 00 01	df 14 30 39 d2 cd 3e c509...>..
0020	a8 4c 63 7b 50 18 00 ff	e3 a5 00 00 4d 41 58 5f	..Lc{P...MAX
0030	53 49 5a 45 5f 52 45 51	55 45 53 54	SIZE_REQUEST

5- הלקוח מחזיר לשרת את גודל החבילה המקסימלי - במקרה שלנו- 10

0000	02 00 00 00 45 00 00 2a	29 cb 40 00 80 06 00 00E..*)..@.....
0010	7f 00 00 01 7f 00 00 01	30 39 df 14 a8 4c 63 7b09...Lc{
0020	d2 cd 3e d5 50 18 00 ff	52 e0 00 00 31 30	..>..P...R...10

6- שליחת החבילה ה-0 לאחר חלוקת החבילה הכוללת 3 חלקים שכל אחד מהם בגודל 10 byte, להלן החבילה: (במקביל עם בוא השרת לשלוח ACK מופעל מנגנון sleep שמעכב את הACK0 5ב שניות).

0000	02 00 00 00 45 00 00 38	29 cd 40 00 80 06 00 00E..8).@.....
0010	7f 00 00 01 7f 00 00 01	df 14 30 39 d2 cd 3e d509...>..
0020	a8 4c 63 7d 50 18 00 ff	94 44 00 00 30 30 31 32	..Lc}P...D...0012
0030	30 3a 44 4f 52 20 41 4e	44 20 42 41	0:DOR AND BA

7- שליחת החבילה ה-1 לאחר חלוקת החבילה הכוללת 3 חלקים שכל אחד מהם בגודל 10 byte, להלן החבילה:

0000	02 00 00 00 45 00 00 38	29 cf 40 00 80 06 00 00E..8).@.....
0010	7f 00 00 01 7f 00 00 01	df 14 30 39 d2 cd 3e e509...>..
0020	a8 4c 63 7d 50 18 00 ff	7c 23 00 00 30 30 31 32	..Lc}P... #...0012
0030	31 3a 52 55 48 20 41 52	45 20 54 48	1:RUH AR E TH

8- שליחת החבילה ה-2 לאחר חלוקת החבילה הכוללת 3 חלקים שכל אחד מהם בגודל 10 byte, להלן החבילה:

```

0000 02 00 00 00 45 00 00 38 29 d1 40 00 80 06 00 00 .....E..8 ).@.....
0010 7f 00 00 01 7f 00 00 01 df 14 30 39 d2 cd 3e f5 .....09...>
0020 a8 4c 63 7d 50 18 00 ff a1 47 00 00 30 30 31 32 ..Lc}P...G..0012
0030 32 3a 45 20 42 45 53 54 53 21 21 21 2:E BEST S!!!

```

9- כעת מנגנון ה TIMEOUT נכנס לפעולה , לאחר 4 שניות החבילות מתחילות להישלח שוב , אך מכיוון שלא מנענו מהחבילות להגיע לשרת אלא רק עיכבנו אותן תיווצר בעתיד קבלה כפולה , כעת נשלח את החבילות שוב מכיוון שהופעל המנגנון.

```

0000 02 00 00 00 45 00 00 38 29 d7 40 00 80 06 00 00 .....E..8 ).@.....
0010 7f 00 00 01 7f 00 00 01 df 14 30 39 d2 cd 3f 05 .....09...?
0020 a8 4c 63 7d 50 18 00 ff 94 14 00 00 30 30 31 32 ..Lc}P...0012
0030 30 3a 44 4f 52 20 41 4e 44 20 42 41 0:DOR AN D BA

```

10- חבילה 1:

```

0000 02 00 00 00 45 00 00 38 29 d9 40 00 80 06 00 00 .....E..8 ).@.....
0010 7f 00 00 01 7f 00 00 01 df 14 30 39 d2 cd 3f 15 .....09...?
0020 a8 4c 63 7d 50 18 00 ff 7b f3 00 00 30 30 31 32 ..Lc}P...{...0012
0030 31 3a 52 55 48 20 41 52 45 20 54 48 1:RUH AR E TH

```

11- חבילה 2:

```

0000 02 00 00 00 45 00 00 38 29 db 40 00 80 06 00 00 .....E..8 ).@.....
0010 7f 00 00 01 7f 00 00 01 df 14 30 39 d2 cd 3f 25 .....09...?%
0020 a8 4c 63 7d 50 18 00 ff a1 17 00 00 30 30 31 32 ..Lc}P...0012
0030 32 3a 45 20 42 45 53 54 53 21 21 21 2:E BEST S!!!

```

12- שליחת ACK0 לחבילה ה-0 , כתוצאה מפעילות המנגנון:

```

0000 02 00 00 00 45 00 00 2c 29 dd 40 00 80 06 00 00 .....E.., ).@.....
0010 7f 00 00 01 7f 00 00 01 30 39 df 14 a8 4c 63 7d .....09...Lc}
0020 d2 cd 3f 35 50 18 00 ff f7 38 00 00 41 43 4b 30 ..?5P...8..ACK0

```

13- שליחת ACK1 לחבילה ה-1, כתוצאה מפעילות המנגנון:

```

0000 02 00 00 00 45 00 00 2c 29 df 40 00 80 06 00 00 .....E.., ).@.....
0010 7f 00 00 01 7f 00 00 01 30 39 df 14 a8 4c 63 81 .....09...Lc
0020 d2 cd 3f 35 50 18 00 ff f7 33 00 00 41 43 4b 31 ..?5P...3..ACK1

```

14- שליחת ACK2 לחבילה ה-1, כתוצאה מפעילות המנגנון:

```

0000 02 00 00 00 45 00 00 2c 29 e1 40 00 80 06 00 00 .....E.., ).@.....
0010 7f 00 00 01 7f 00 00 01 30 39 df 14 a8 4c 63 85 .....09...Lc
0020 d2 cd 3f 35 50 18 00 ff f7 2e 00 00 41 43 4b 32 ..?5P...ACK2

```

15,16,17 - מכיוון שביצענו שליחה כפולה לחבילות עקב Timeout שלא מנע את הגעת החבילות אלא רק עיכב אותן , נקבל ACK2 על כל אחת מהחבילות , מכיוון שכבר הגיעו לנו החבילות והן לא בסדר המצופה (כי לא מצפים לכלום) השרת ישלח עליהן ACK2 ולאחר מכן הקליינט שהבין ששלח את כל ההודעות וקיבל עליהן ACK יסגור את החיבור : x3

```

0000 02 00 00 00 45 00 00 2c 29 e5 40 00 80 06 00 00 .....E.., ).@.....
0010 7f 00 00 01 7f 00 00 01 30 39 df 14 a8 4c 63 8d .....09...Lc
0020 d2 cd 3f 35 50 18 00 ff f7 26 00 00 41 43 4b 32 ..?5P...&..ACK2

```

18,19 - סיום התקשורת - ה"סגירה" תבצע באופן לא צפוי , השרת ישלח לקליינט בקשת RST מכיוון שהקליינט סוגר את החיבור אחרי ששלח את כל החבילות ושרת רוצה לקבל ACK.

2.2.3 תרחיש קצה 3: חבילות מחוץ לסדר

נשלח את חבילה 0, לאחר מכן חבילה 2 ולאחר מכן חבילה 3 ולאחר מכן חבילה 1 ונראה מה קורה.
 התרחיש המצופה: חבילה 1 תגיע, לא ישלח ACK אך היא תיכנס לbuffer, לאחר מכן חבילה 0 תגיע וישלח ACK1, לאחר מכן חבילה 2 תגיע וישלח ACK2.
 *נוודא שהשרת מכניס את החבילות ל-buffer.
 *נוודא שהשרת שולח ACK רק עבור ההודעות ברצף.
 נגדיר:

ההודעה עצמה: DOR AND BARUH LIKE TO EAT

גודל ההודעה - 25 בתים.

גודל הודעה מקסימלית - 7 בתים.

גודל החלון - 4 חבילות

טיים-אאוט - 5 שניות

השינוי בקוד על מנת לבדוק את המקרה: (נשלח את החבילות לא לפי סדר באופן יזום)

```
send_message_with_boundary(client_socket, 0, message_parts[0]) # Message 0
if len(message_parts) > 2:
    send_message_with_boundary(client_socket, 2, message_parts[2]) # Message 2
if len(message_parts) > 3:
    send_message_with_boundary(client_socket, 3, message_parts[3]) # Message 3
if len(message_parts) > 1:
    send_message_with_boundary(client_socket, 1, message_parts[1]) # Message 1
```

תיאור פעילות הלקוח והשרת:**Client:**

```
Enter 'file' to read parameters from a file or 'input' to provide manually: input
Enter the message: DOR AND BARUH LIKE TO EAT
Enter the sliding window size: 4
Enter the timeout in seconds: 5
Max message size from server is: 7 bytes
Adjusted maximum message size: 7 bytes
Timeout has set to: 5 seconds
Message chunks: ['DOR AND', ' BARUH ', 'LIKE TO', ' EAT']
Server acknowledged up to message 0
Server acknowledged up to message 0
Server acknowledged up to message 0
Server acknowledged up to message 3
All messages sent and acknowledged!

Process finished with exit code 0
```


Server:

```

Connection from ('127.0.0.1', 59630) has been established.
Enter 'file' to read MAX SIZE of the message from a file or 'input' to provide manually: input
Enter the maximum message size: 7
Received: 0:DOR AND
Message 0 arrived in the correct order: DOR AND
Sent: ACK0
Received: 2:LIKE TO
Message 2 out of order, storing: LIKE TO
Resent: ACK0
Received: 3: EAT
Message 3 out of order, storing: EAT
Resent: ACK0
Received: 1: BARUH
Message 1 arrived in the correct order: BARUH
Adding buffered message 2: LIKE TO
Adding buffered message 3: EAT
Sent: ACK3
Client closed the connection.
Client closed the connection. Exiting loop.
Connection closed.

```

ניתוח פקטות ב-Wireshark:

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	127.0.0.1	127.0.0.1	TCP	56	59630 → 12345 [SYN, Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
2	0.000081	127.0.0.1	127.0.0.1	TCP	56	12345 → 59630 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
3	0.000107	127.0.0.1	127.0.0.1	TCP	44	59630 → 12345 [ACK] Seq=1 Ack=1 Win=65280 Len=0
4	40.012703	127.0.0.1	127.0.0.1	TCP	60	59630 → 12345 [PSH, ACK] Seq=1 Ack=1 Win=65280 Len=16
5	59.138955	127.0.0.1	127.0.0.1	TCP	45	12345 → 59630 [PSH, ACK] Seq=1 Ack=17 Win=65280 Len=1
6	59.139155	127.0.0.1	127.0.0.1	TCP	57	59630 → 12345 [PSH, ACK] Seq=17 Ack=2 Win=65280 Len=13
7	59.139207	127.0.0.1	127.0.0.1	TCP	57	59630 → 12345 [PSH, ACK] Seq=30 Ack=2 Win=65280 Len=13
8	59.139227	127.0.0.1	127.0.0.1	TCP	54	59630 → 12345 [PSH, ACK] Seq=43 Ack=2 Win=65280 Len=10
9	59.139243	127.0.0.1	127.0.0.1	TCP	57	59630 → 12345 [PSH, ACK] Seq=53 Ack=2 Win=65280 Len=13
10	59.139359	127.0.0.1	127.0.0.1	TCP	48	12345 → 59630 [PSH, ACK] Seq=2 Ack=66 Win=65280 Len=4
11	59.139475	127.0.0.1	127.0.0.1	TCP	48	12345 → 59630 [PSH, ACK] Seq=6 Ack=66 Win=65280 Len=4
12	59.139590	127.0.0.1	127.0.0.1	TCP	48	12345 → 59630 [PSH, ACK] Seq=10 Ack=66 Win=65280 Len=4
13	59.139638	127.0.0.1	127.0.0.1	TCP	48	12345 → 59630 [PSH, ACK] Seq=14 Ack=66 Win=65280 Len=4
14	60.140303	127.0.0.1	127.0.0.1	TCP	44	59630 → 12345 [FIN, ACK] Seq=66 Ack=18 Win=65280 Len=0
15	60.140408	127.0.0.1	127.0.0.1	TCP	44	12345 → 59630 [FIN, ACK] Seq=18 Ack=67 Win=65280 Len=0

1,2,3- לחיצת יד משולשת של הסרבר והקליינט.

4- השרת שולח לקליינט את הMAX_SIZE_REQUEST על מנת שידע שהוא רוצה לקבל את גודל ההודעה המקסימלית.

0000	02 00 00 00 45 00 00 38	b6 47 40 00 80 06 00 00	... E . 8 . G@
0010	7f 00 00 01 7f 00 00 01	e8 ee 30 39 bb 4c f3 a5 09 . L . .
0020	7c 06 4c e5 50 18 00 ff	7f 48 00 00 4d 41 58 5f	. L . P H . . MAX_
0030	53 49 5a 45 5f 52 45 51	55 45 53 54	SIZE_REQ UEST

5- הלקוח מחזיר לשרת את גודל החבילה המקסימלי - במקרה שלנו 10

0000	02 00 00 00 45 00 00 29	b6 59 40 00 80 06 00 00E..).Y@.....
0010	7f 00 00 01 7f 00 00 01	30 39 e8 ee 7c 06 4c e509.. ..L..
0020	bb 4c f3 b5 50 18 00 ff	e8 b3 00 00 37	..L..P... ..7

6- שליחת החבילה ה-0 לאחר חלוקת החבילה הכוללת 4 חלקים שכל אחד מהם בגודל של עד 7 byte, להלן החבילה:

0000	02 00 00 00 45 00 00 38	fe 6b 40 00 80 06 00 00E..8..k@.....
0010	7f 00 00 01 7f 00 00 01	f2 45 30 39 d4 5d bc 4cE09..]..L..
0020	89 c9 23 2f 50 18 00 ff	60 dd 00 00 30 30 31 32	..#/P... ..0012
0030	30 3a 44 4f 52 20 41 4e	44 20 42 41	0:DOR AN D BA

```
Sequence Number: 17      (relative sequence number)
Sequence Number (raw): 3142382517
[Next Sequence Number: 30      (relative sequence number)]
Acknowledgment Number: 2      (relative ack number)
```

7- כמעט נדלג על החבילה ה-1 ובמקום נבצע שליחת החבילה ה-2 מתוך ה-4: נוכל לראות שהack number נשאר כמו הקודם מכיוון שלא שלחנו ACK על שום חבילה חדשה שהגיעה, החבילה תיכנס לbuffer, בהנחיות הפרויקט נאמר לשלוח ACK על חבילה מסוימת רק במידה והחבילות הגיעו ברצף, אחרת נחזיר ACK לחבילה האחרונה שהגיעה ברצף ובכך נדע שאנחנו צריכים לשלוח את החבילה שאחריה.

0000	02 00 00 00 45 00 00 35	b6 5d 40 00 80 06 00 00E..5..]@.....
0010	7f 00 00 01 7f 00 00 01	e8 ee 30 39 bb 4c f3 c209..L..
0020	7c 06 4c e6 50 18 00 ff	86 13 00 00 30 30 30 39	..L..P... ..0009
0030	32 3a 4c 49 4b 45 20 54	4f	2:LIKE T O

```
Sequence Number: 30      (relative sequence number)
Sequence Number (raw): 3142382530
[Next Sequence Number: 43      (relative sequence number)]
Acknowledgment Number: 2      (relative ack number)
```

8- שליחת החבילה ה-3 והאחרונה, בהבנה ששוב נקבל עליה ACK0 אם הכל עובד כמתוכנן, אך החבילה תיכנס לbuffer בשביל שברגע שנשלים את החבילה החסרה נוכל להשתמש בה על מנת לשלוח ACK על קבלת כל החבילות שנמצאות ברצף עד החבילה הגבוהה ביותר. נוכל לראות שהack number נשאר כמו הקודם מכיוון שלא שלחנו ACK על שום חבילה חדשה שהגיעה.

0000	02 00 00 00 45 00 00 32	b6 5f 40 00 80 06 00 00E..2.._@.....
0010	7f 00 00 01 7f 00 00 01	e8 ee 30 39 bb 4c f3 cf09..L..
0020	7c 06 4c e6 50 18 00 ff	2a 56 00 00 30 30 30 36	..L..P... *V..0006
0030	33 3a 20 45 41 54		3: EAT

```
Sequence Number: 43      (relative sequence number)
Sequence Number (raw): 3142382543
[Next Sequence Number: 53      (relative sequence number)]
Acknowledgment Number: 2      (relative ack number)
```

9-כעת , נשלח את חבילה מספר 1 בסדר לא נכון ובתור החבילה האחרונה מתוך הארבע , עכשיו נסתכל על התגובות מהצד של השרת.

```

0000 02 00 00 00 45 00 00 35 b6 61 40 00 80 06 00 00 .....E..5.a@.....
0010 7f 00 00 01 7f 00 00 01 e8 ee 30 39 bb 4c f3 d9 .....09..|..L..
0020 7c 06 4c e6 50 18 00 ff b7 02 00 00 30 30 30 39 |..L..P.....0009
0030 31 3a 20 42 41 52 55 48 20 1: BARUH

```

```

Sequence Number: 53      (relative sequence number)
Sequence Number (raw): 3142382553
[Next Sequence Number: 66      (relative sequence number)]
Acknowledgment Number: 2      (relative ack number)

```

10-קבלת ACK0 לחבילה ה-0 מתוך 3:

```

0000 02 00 00 00 45 00 00 2c b6 63 40 00 80 06 00 00 .....E.., .c@.....
0010 7f 00 00 01 7f 00 00 01 30 39 e8 ee 7c 06 4c e6 .....09..|..L..
0020 bb 4c f3 e6 50 18 00 ff 93 0b 00 00 41 43 4b 30 |..L..P.....ACK0

```

11-קבלת ACK0 לחבילה ה-1 מתוך 3:(השרת קיבל אותה והכניס אותה לbuffer אך הוא לא קיבל אותן ברצף ולכן לא יוחזר עליה ACK).

```

0000 02 00 00 00 45 00 00 2c b6 65 40 00 80 06 00 00 .....E.., .e@.....
0010 7f 00 00 01 7f 00 00 01 30 39 e8 ee 7c 06 4c ea .....09..|..L..
0020 bb 4c f3 e6 50 18 00 ff 93 07 00 00 41 43 4b 30 |..L..P.....ACK0

```

12- קבלת ACK0 לחבילה-2 מתוך 3:(השרת קיבל אותה והכניס אותה לbuffer אך הוא לא קיבל אותן ברצף ולכן לא יוחזר עליה ACK)

```

0000 02 00 00 00 45 00 00 2c b6 67 40 00 80 06 00 00 .....E.., .g@.....
0010 7f 00 00 01 7f 00 00 01 30 39 e8 ee 7c 06 4c ee .....09..|..L..
0020 bb 4c f3 e6 50 18 00 ff 93 03 00 00 41 43 4b 30 |..L..P.....ACK0

```

13- קבלת ACK3 - לאחר שהכנסנו את חבילה 2 וחבילה 3 לbuffer כעת מגיעה חבילה 1 לפי הסדר , השרת מעדכן שהיא התקבלה לפי הסדר ובודק בbuffer האם יש חבילות שעוד לא עודכן עליהן ACK בגלל שהגיעו לא בסדר, הוא רואה שנמצאות שם חבילות 2 ו 3 מוציא אותן ואז שולח ACK על החבילה עם המספר הסידורי הכי גבוה , כלומר , חבילה מס' 3. ולכן ישלח ACK3.

```

0000 02 00 00 00 45 00 00 2c b6 69 40 00 80 06 00 00 .....E.., .i@.....
0010 7f 00 00 01 7f 00 00 01 30 39 e8 ee 7c 06 4c f2 .....09..|..L..
0020 bb 4c f3 e6 50 18 00 ff 92 fc 00 00 41 43 4b 33 |..L..P.....ACK3

```

14,15 - סיום התקשורת - השרת מודיע ללקוח שקיבל את כל החבילות וההתקשרות בניהם מסתיימת בסגירת הקשר משני הצדדים.

3. קישור למקורות אונליין ששימשו לטובת פתרון המטלה:

3.1 ביבליוגרפיה:

- [Python Network Sockets Part 2: Message Length](#)
- [Sockets Tutorial with Python 3 part 1 - sending and receiving data](#)
- [טל בן שושן – UDP ו TCP הסבר על](#)

3.2 שאלות לגPT:

A. Unexpected error: [WinError 10053] An established connection was

aborted by the software in your host machine מה זה??

A תשובה:

השגיאה "[WinError 10053] An established connection was aborted by the software in your host machine" מתרחשת בדרך כלל כשמשו משפיק את החיבור בין השרת ללקוח. זה עשוי לנבוע מכמה סיבות:

1. **בעיה בחומת אש או אנטי-וירוס:** ייתכן שחומת האש או האנטי-וירוס שלך חסמו את החיבור. נסה להשבית אותם זמנית ולבדוק אם הבעיה נפתרת.
2. **בעיות בקוד:** אם אתה עובד על פרויקט הכולל חיבורי רשת, ייתכן שהלקוח או השרת סגרו את החיבור באופן לא צפוי. בדוק את הלוגיקה של הקוד שלך לוודא שהחיבורים מטופלים בצורה נכונה.
3. **בעיות זמן קצוב (Timeout):** ייתכן שהחיבור הופסק בגלל זמן קצוב. ודא שהגדרות הזמן הקצוב בקוד מתאימות ושאינ עיכובים משמעותיים בעיבוד.
4. **שגיאות בתקשורת:** בעיה ברשת המקומית או בתשתית האינטרנט עשויה לגרום לשגיאה. נסה להריץ את התוכנית במכשיר אחר או על רשת אחרת.
5. **מגבלות בשרת:** השרת עשוי להגביל חיבורים מסוימים או לסגור חיבורים עקב עומס. בדוק את הגדרות השרת או יומני השגיאות שלו.

בפרויקט שלך, בהתבסס על הדרישות שנמסרו בקובץ, ודא שאתה:

- מטפל בכל החריגות (exceptions) כשאתה יוצר או מנהל חיבורים.
- מאתחל את הטיימרים ומטפל בהודעות חוזרות כהלכה.
- עוקב אחרי הודעות ה-Ack מהשרת כדי למנוע שליחה לא רצויה.

אם אתה צריך עזרה בפתרון בעיה ספציפית בקוד, תוכל להעלות קטעים ממנו או להסביר את הסיטואציה המלאה.

B. אשמח שתעבור על הקובץ ותעזור לי להבין מה בדיוק צריך לעשות במטלה

B תשובה:

על בסיס התוכן בקובץ, הנה הסבר ברור על המשימות שצריך לבצע במטלה:

מטרת המטלה:

ליישם גרסה פשוטה של פרוטוקול העברת נתונים אמין ומסודר, בדומה לעקרונות של TCP. מדובר ביצירת שתי תוכניות:

1. לקוח (Client).

2. שרת (Server).

שני הצדדים יתקשרו דרך חיבור רשת, תוך שימוש ב-sockets של TCP.

שלבי העבודה:

1. שלב ראשוני - קביעת גודל הודעה מקסימלי:

- הלקוח שולח לשרת בקשה לגודל הודעה מקסימלי (בבתים) אותו השרת יכול לקבל.
- השרת מחזיר מספר המייצג את גודל ההודעה המקסימלי.

2. שליחת הודעות מהלקוח לשרת:

- לאחר קבלת גודל ההודעה המקסימלי, הלקוח מחלק את הנתונים להודעות בגודל זה (לדוגמה: אם גודל ההודעה המקסימלי הוא 20 בתים והנתונים הם 80 בתים, אז יש לחלק ל-4 הודעות: M0, M1, M2, M3).

3. יישום גישה של "חלון הזזה" (Sliding Window):

- הלקוח ישתמש בחלון הזזה לשליחה אמינה ומסודרת של הודעות.
- השרת שולח אישורים (Acknowledgments) על ההודעות שהתקבלו.
- אם ההודעות התקבלו לא בסדר הנכון, השרת מחזיר אישור רק להודעה הרציפה האחרונה שהתקבלה בסדר.

4. ניהול טיימרים וזמן השהיה (Timeouts):

- הלקוח שומר טיימר לכל הודעה שנשלחה.
- אם הודעה לא קיבלה אישור בזמן, הלקוח שולח שוב את כל ההודעות שבחלון.

דרישות ותנאים:

● מבנה הקוד:

- כתוב בצורה ברורה, עם שמות משתנים משמעותיים.
- הוספת הערות בקוד.

● שימוש בקלט:

- הקוד צריך לתמוך בקבלת משתנים (כגון הודעה, גודל הודעה מקסימלי, גודל חלון, וזמן השהיה) משני מקורות:
 1. קלט ידני מהמשתמש.
 2. קובץ טקסט בפורמט מוגדר (ראה דוגמת פורמט בקובץ).

שלבים בתהליך התכנות:

1. כתיבת תוכנית שרת:

- מאזין לבקשות TCP.
 - מחזיר את גודל ההודעה המקסימלי.
 - מנהל קבלת הודעות ושליחת אישורים.
2. כתיבת תוכנית לקוח:

- פותח חיבור לשרת.
 - מבקש את גודל ההודעה המקסימלי.
 - מחלק הודעות ושולח אותן.
 - מנהל אישורים וטיימרים.
3. שילוב "חלון הזזה":

- מוודא שכל ההודעות נשלחות ומאושרות בצורה רציפה.
4. בדיקות ואימות:

- בדיקה שהמערכת עובדת במגוון מצבים:
 - הודעות בגודל קטן או גדול.
 - זמן השהיה שונה.
 - נתוני קלט שונים.

דגשים נוספים:

- יש לציין בקובץ PDF המסירה את כל האתרים, הכלים והמשאבים שהשתמשת בהם.
- אסור להעתיק קוד באופן ישיר מאתרים כמו GitHub.

אם יש צורך בהבהרה או פירוט לגבי אחד השלבים, אני כאן לעזור!

- C. The command "client.recv" will not get me all the data, for example if sent three messages in a row and after finishing the command i use "client.recv" i'm only getting single response, i can tell that there is a buffer in client.recv but it doesn't send me all the buffer

C תשובה:

The behavior you're observing with `client.recv` is due to the way TCP sockets and the `recv` function handle incoming data. `recv` retrieves up to the number of bytes you specify but does not guarantee to receive an entire message, as TCP is a stream-oriented protocol. Messages might be fragmented across multiple calls to `recv`, or multiple messages could be combined into one read due to buffering.

Solution

You need to implement a message protocol that ensures you can reconstruct messages accurately. Here's how you can approach this:

1. Prefixing Messages with Their Length

- Each message can be prefixed with its length (e.g., as a fixed-size header).
- When reading data, first read the length header to determine how many bytes to read for the complete message.

2. Loop Until the Entire Message Is Received

- Use a loop to call `recv` repeatedly until you've received all the bytes you expect.

3. Using a Delimiter

- If messages are of variable length, use a special delimiter to signify the end of each message. The receiver can accumulate data in a buffer until it detects the delimiter.

Example Code

Client-Side Sending Messages:

```
import socket

def send_message(client_socket, message):
    # Encode the message and get its length
    message = message.encode('utf-8')
    message_length = len(message)
    # Send the length as a fixed-size header (e.g., 4 bytes)
    client_socket.sendall(message_length.to_bytes(4, 'big'))
    # Send the actual message
    client_socket.sendall(message)
```

Server-Side Receiving Messages:

```
def recv_message(server_socket):
    # Read the length header (4 bytes)
    header = server_socket.recv(4)
    if not header:
        return None
    message_length = int.from_bytes(header, 'big')

    # Read the message based on its length
    message = b''
    while len(message) < message_length:
        chunk = server_socket.recv(message_length - len(message))
        if not chunk:
            raise ConnectionError("Connection lost")
        message += chunk

    return message.decode('utf-8')
```

Why This Works

- The **length-prefixed approach** ensures that you know exactly how many bytes to expect for a complete message.
- The loop for reading guarantees that even if the message is fragmented, you accumulate all parts until the entire message is received.

If you're dealing with multiple messages sent in quick succession, this structure will help manage them effectively, ensuring no data is lost or misinterpreted.