Q 1. The UDP checksum provides for error detection. Consider the following word with 32 bits:

01100110011000000101010101010101 (1)

(a) Compute the checksum. (Recall that UDP computes checksum based on 16-bit word.) Break the 32-bit word into two 16-bit words, and sum their up.

01100110011000000101010101010101="0110011001100000"+"0101010101010101"

checksum=~(0110011001100000+0101010101010101)=0100010001001010

(b) How does the receiver detect errors?

At the receiver side, all four 16-bit words are added, including the checksum：

If no errors are introduced into the packet, then clearly the sum at the receiver will be 1111111111111111.

If one of the bits is a 0, then we know that errors have been introduced into the packet.

(c) If the receiver does not detect any error using the checksum, does it mean that the message was transmitted without any error? Please explain the reason and provide an example.

Nope, this is a kind of hash, may let Hash Collision

Q 2. Fill in the blanks (B1) and (B2) in the below figure for go-back-N and selective repeat.

(B1) and (B2) for go-back-N

B1:resend ack0 B2:234

(B1) and (B2) for selective repeat

B1:send ack2 B2:456

Q 3.The following figure illustrates the convergence of TCP’s additive-increase multiplicative-decrease (AIMD)algorithm.

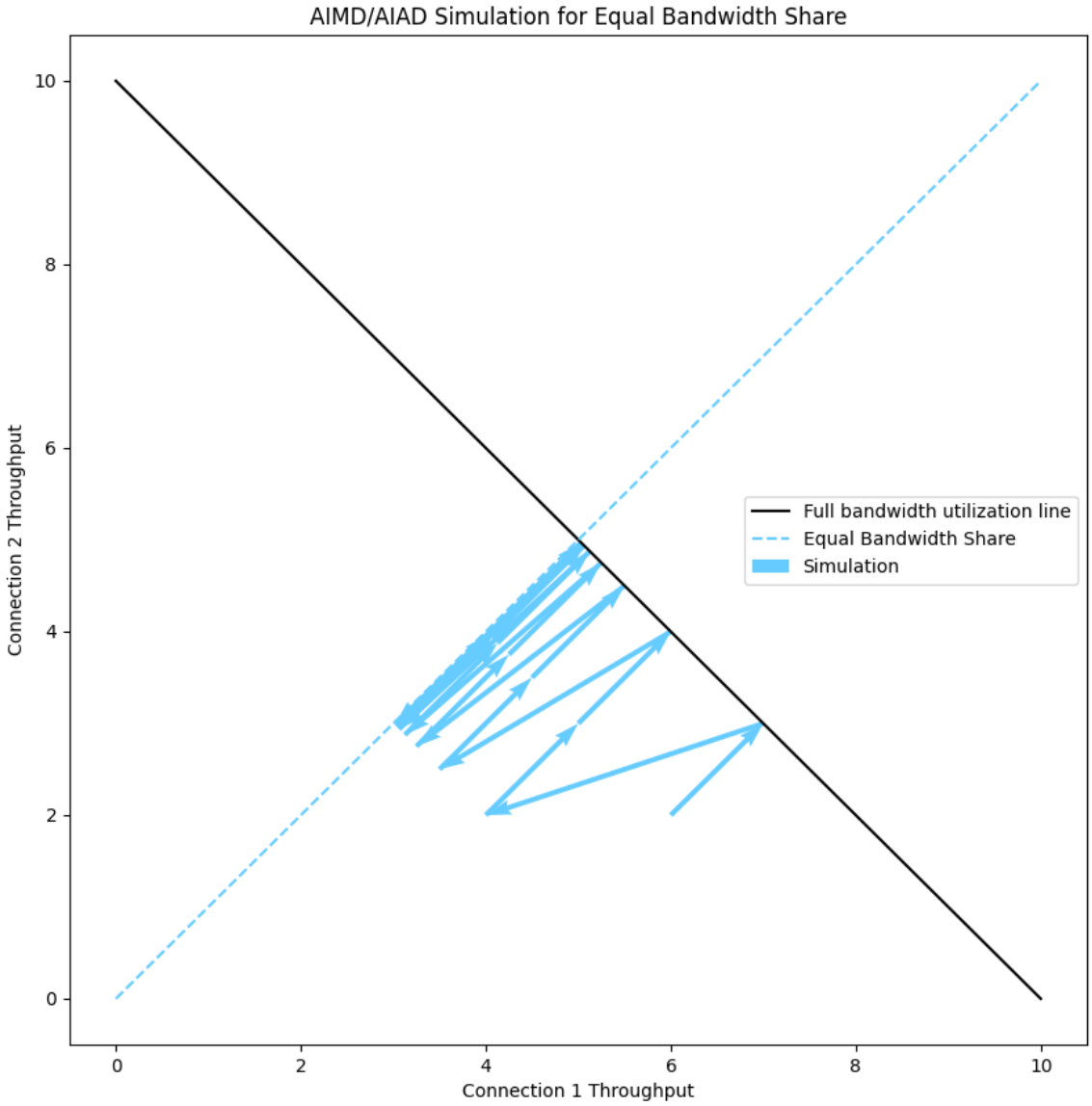
Suppose that instead of a multiplicative decrease, TCP decreased the window size by a constant amount. Would the resulting additive-increase additive-decrease (AIAD) algorithm converge to an equal share

algorithm? Justify your answer using a diagram similar to the above figure. (Note: Simply draw the diagram is not sufficient. You need to explain what the diagram shows.)

Talk is cheap, I will show you the code: this is the code for AIMD

```
import matplotlib.pyplot as plt
def aimd_simulation(R, max_iterations):

    conn1_throughput_values = [6]
    conn2_throughput_values = [2]
    conn1_throughput=6
    conn2_throughput=2
    router_buffer_sizes = []

    for _ in range(max_iterations):
        # AI
        conn1_throughput += 1
        conn2_throughput += 1

        # MD
        if conn1_throughput + conn2_throughput > R:
            conn1_throughput *= 0.5
            conn2_throughput *= 0.5

        conn1_throughput_values.append(conn1_throughput)
        conn2_throughput_values.append(conn2_throughput)
        router_buffer_sizes.append(R - conn1_throughput - conn2_throughput)

    return conn1_throughput_values, conn2_throughput_values, router_buffer_sizes

def plot_graph(R, conn1_throughput_values, conn2_throughput_values):
    plt.figure(figsize=(10,10))

    # y=-x+R
    plt.plot([0, R], [R, 0],color='black',label='Full bandwidth utilization line')

    # y=x
    plt.plot([0, R], [0, R], color="#66ccff",label='Equal Bandwidth Share',
linestyle='--')

    # simulations
    dx = [conn1_throughput_values[i + 1] - conn1_throughput_values[i] for i in
range(len(conn1_throughput_values) - 1)]
    dy = [conn2_throughput_values[i + 1] - conn2_throughput_values[i] for i in
range(len(conn2_throughput_values) - 1)]
    plt.quiver(conn1_throughput_values[:-1], conn2_throughput_values[:-1], dx, dy,
               angles='xy', scale_units='xy', scale=1, color="#66ccff",
width=0.005, label='Simulation')

    plt.xlabel('Connection 1 Throughput')
    plt.ylabel('Connection 2 Throughput')
    plt.title('AIMD/AIAD Simulation for Equal Bandwidth Share')
```

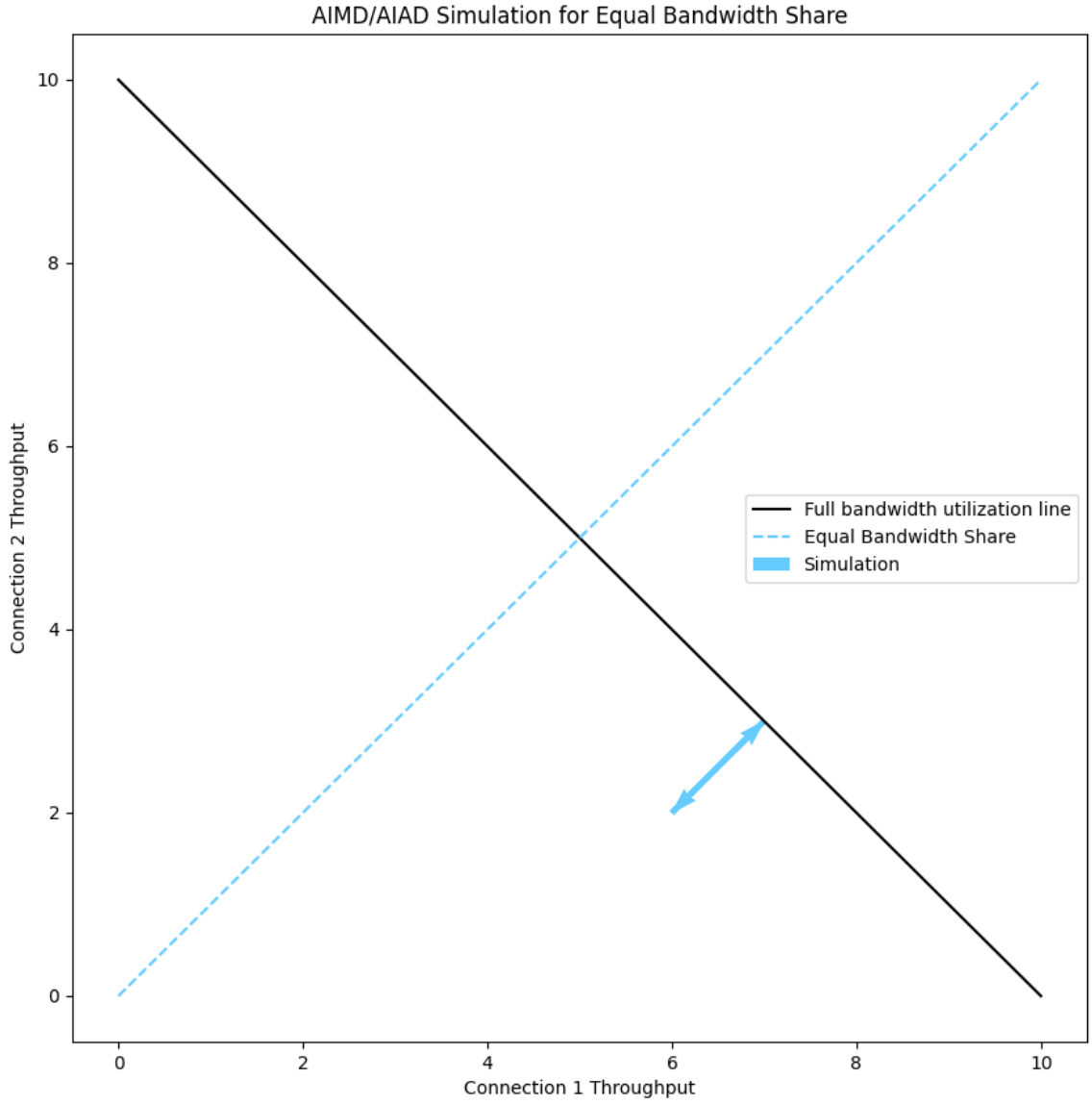
```
    plt.legend()
    plt.show()

conn1_throughput_values, conn2_throughput_values, router_buffer_sizes =
aimd_simulation(10,20)
plot_graph(10, conn1_throughput_values, conn2_throughput_values)
```



This is AIAD

```
import matplotlib.pyplot as plt
def aiad_simulation(R, max_iterations):

    conn1_throughput_values = [6]
    conn2_throughput_values = [2]
```

```
    conn1_throughput=6
    conn2_throughput=2
    router_buffer_sizes = []

    for _ in range(max_iterations):
        # AI
        conn1_throughput += 1
        conn2_throughput += 1

        # AD
        if conn1_throughput + conn2_throughput > R:
            conn1_throughput -=2
            conn2_throughput -=2

        conn1_throughput_values.append(conn1_throughput)
        conn2_throughput_values.append(conn2_throughput)
        router_buffer_sizes.append(R - conn1_throughput - conn2_throughput)

    return conn1_throughput_values, conn2_throughput_values, router_buffer_sizes

def plot_graph(R, conn1_throughput_values, conn2_throughput_values):
    plt.figure(figsize=(10,10))

    # y=-x+R
    plt.plot([0, R], [R, 0],color='black',label='Full bandwidth utilization line')

    # y=x
    plt.plot([0, R], [0, R], color="#66ccff",label='Equal Bandwidth Share',
linestyle='--')

    # simulations
    dx = [conn1_throughput_values[i + 1] - conn1_throughput_values[i] for i in
range(len(conn1_throughput_values) - 1)]
    dy = [conn2_throughput_values[i + 1] - conn2_throughput_values[i] for i in
range(len(conn2_throughput_values) - 1)]
    plt.quiver(conn1_throughput_values[:-1], conn2_throughput_values[:-1], dx, dy,
               angles='xy', scale_units='xy', scale=1, color="#66ccff",
width=0.005, label='Simulation')

    plt.xlabel('Connection 1 Throughput')
    plt.ylabel('Connection 2 Throughput')
    plt.title('AIMD/AIAD Simulation for Equal Bandwidth Share')
    plt.legend()
    plt.show()

conn1_throughput_values, conn2_throughput_values, router_buffer_sizes =
aiad_simulation(10,20)
plot_graph(10, conn1_throughput_values, conn2_throughput_values)
```



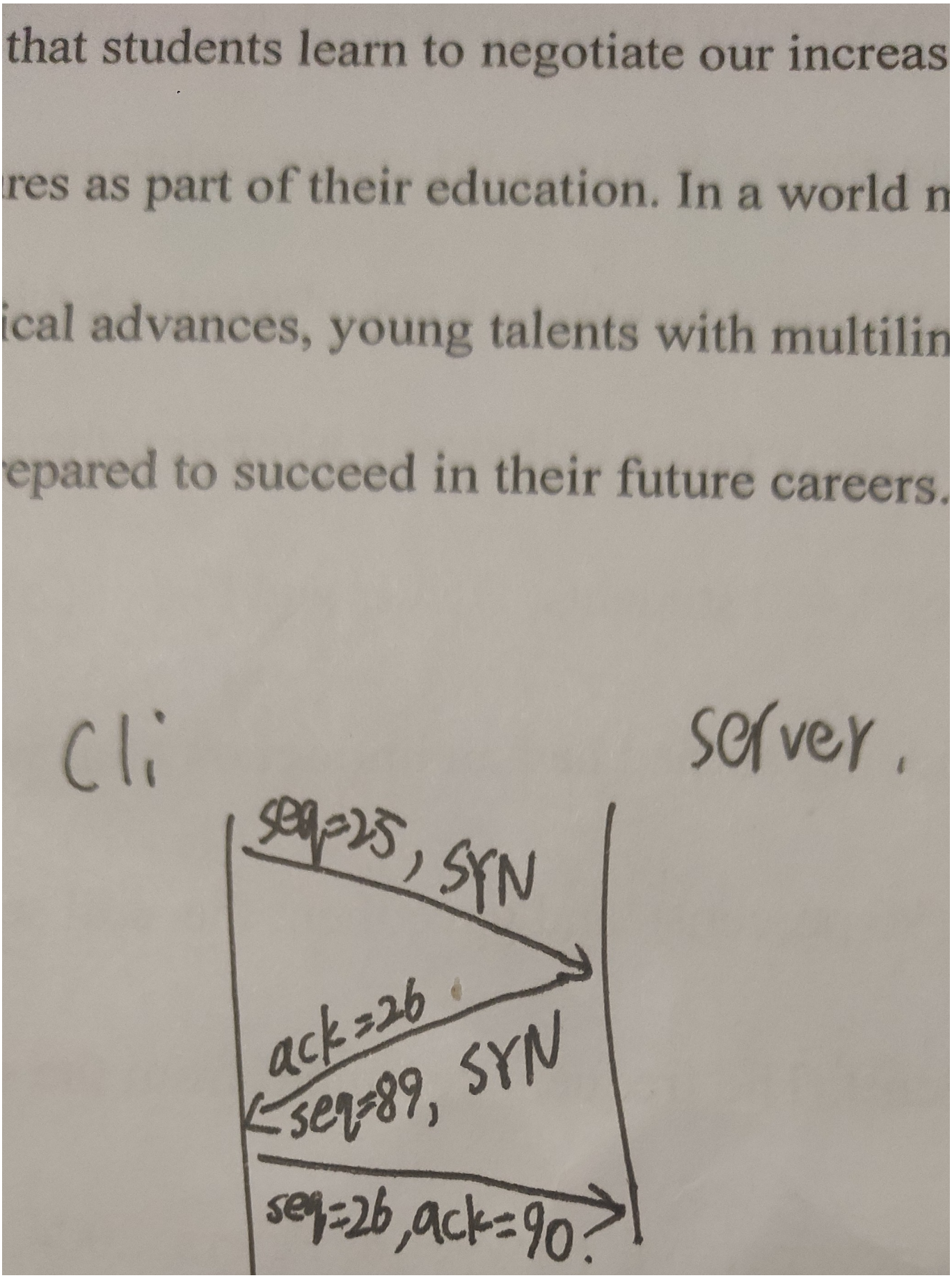
as is shown in these figures, AIMD and AIAD can reduce packet loss , but AIAD is not fair

The convergence of AIMD is provable and can be approximated(emotionally) by Newton's iteration of a function (a process that looks like Newton's iteration of a function)

Q 4. Draw the TCP connection-establishment procedure (that is, TCP three-way handshaking) between a client host and a server host. Suppose the initial sequence number of the client host is 25, and that of the server host is 89. For each segment

exchange between the client and server, please indicate

(1) the SYN bit, sequence number, acknowledgement number (if necessary);



(2) whether the segment can carry data in the segment payload (that is, in the segment data filed).

No. During the handshake, the packet does not carry the actual application layer data.

Q 5. Consider the TCP procedure for estimating RTT. Suppose that α = 0.1, and Estimated RTT is initialized as Estimated RTT0.

Recall that Estimated RTT[n] = (1 − α)Estimated RTT[n-1] + αSample RTT.

(a) For a given TCP connection, suppose four acknowledgments have been returned in sequence with corresponding sample RTTs: SampleRTT1, SampleRTT2, SampleRTT3, and SampleRTT4. Express EstimatedRTT in terms of EstimatedRT T0 and the four sample RTTs.

EstimatedRTT1=(1−0.1)×EstimatedRTT0+0.1×SampleRTT1 EstimatedRTT2= (1−0.1)×EstimatedRTT1+0.1×SampleRTT2 EstimatedRTT3=(1−0.1)×EstimatedRTT2+0.1×SampleRTT3 EstimatedRTT4=(1−0.1)×EstimatedRTT3+0.1×SampleRTT4

(b) Generalize your formula for n sample RTTs

$\text{EstimatedRTT}*n = (1 - *alpha) *times *text{EstimatedRTT}*{n-1} + \alpha \times \text{SampleRTT}_n$ (Recursive)

or

$\text{EstimatedRTT}_n = (1 - \alpha)^n \times \text{EstimatedRTT}*0 + *sum*{i=1}^{n} \alpha^{n} \times \text{SampleRTT}_i$