1. What is SYN, SYN-ACK, PSH-ACK, and ACK?

1.1. SYN (Synchronize): This flag is set when initiating a connection. When a client wants to establish a TCP connection with a server, it sends a packet with the SYN flag set to the server. The SYN flag indicates that the client is requesting synchronization with the server to establish a connection.

1.2. SYN-ACK (Synchronize-Acknowledge): When the server receives a SYN packet from a client, it responds by sending a packet with both the SYN and ACK flags set. This indicates that the server has acknowledged the client's synchronization request and also wants to synchronize its own sequence numbers for the connection.

1.3. PSH-ACK (Push-Acknowledge): The PSH-ACK flag is used during the data transmission phase. When a sender (either client or server) wants to send data to the other party, it sets the PSH (Push) flag in the TCP header. This flag indicates that the receiving party should process the data immediately rather than buffering it. The ACK flag is also set to acknowledge the receipt of previous data.

1.4. ACK (Acknowledgment): The ACK flag is used to acknowledge received packets. It indicates that the sender has successfully received the data up to a particular sequence number. In various stages of the TCP connection, the ACK flag is used to confirm the receipt of SYN, SYN-ACK, or PSH packets.

2. Why not just use 0 as the Initial Sequence Number (ISN)?

By using a random ISN, TCP enhances security, supports concurrent connections, enables stateful networking devices, and maintains backward compatibility with the existing infrastructure.

3. Why do we need Three-Way Handshake when establishing a connection, but 4-Way Say Goodbye when ending the connection?

The three-way handshake ensures that both parties agree to establish a connection, synchronize their initial sequence numbers, and confirm that communication channels are working properly.

The four-way handshake allows both parties to ensure that all the data has been transmitted and received correctly before terminating the connection. It provides a reliable and orderly way to close the connection, allowing any remaining data to be exchanged and ensuring that both sides are aware of the termination.

In summary, the three-way handshake establishes a connection, while the four-way handshake terminates it in a controlled manner, allowing for proper

data transmission and acknowledgment before the connection is closed.

4. What is Cumulative ACK?

   In TCP (Transmission Control Protocol), a Cumulative ACK (Acknowledgment) refers to the acknowledgment of received data up to a certain sequence number.

   When data is transmitted over a TCP connection, it is divided into segments, each with a unique sequence number. The receiving party acknowledges the receipt of data by sending an ACK packet with an acknowledgment number indicating the next expected sequence number.

   In the case of a Cumulative ACK, the acknowledgment number represents the highest sequence number received successfully, along with an implicit acknowledgment of all preceding bytes or segments. It means that the receiver has received and successfully processed all data up to that sequence number.

   For example, if a receiver receives segments with sequence numbers 1, 2, 3, and 4, it may send a Cumulative ACK with acknowledgment number 5. This indicates that it has received and processed all data up to sequence number 4, and it is now expecting the next segment with sequence number 5.

   The Cumulative ACK mechanism simplifies the acknowledgment process by reducing the number of ACK packets sent. Instead of sending individual acknowledgments for every received segment, the receiver can acknowledge multiple segments at once, improving the efficiency of the TCP connection.

   Cumulative ACKs are an essential part of TCP's reliable data transfer mechanism, ensuring that data is successfully delivered and allowing the sender to keep track of the received and acknowledged segments.

5. Why do we need the pseudoheader?

   The main purpose of the pseudoheader is to provide additional information necessary for the integrity check of the IP (Internet Protocol) datagram, which encapsulates the TCP or UDP segment. It includes specific fields from the IP header, such as source and destination IP addresses, protocol type, and the length of the TCP or UDP segment.

   The pseudoheader plays a crucial role in ensuring data integrity, verifying the correct routing of segments, and enabling efficient multiplexing of TCP and UDP connections. By including relevant information from the IP header, it facilitates

accurate checksum calculations and enhances the reliability of the overall communication.

6. We know the IP protocol has checksum, and so do the TCP protocol. What's different? And why do we need both?

1. IP Checksum:

The IP protocol includes a checksum that covers the entire IP packet, including the IP header and its encapsulated data (which can be TCP, UDP, or other protocols). The IP checksum is calculated at the network layer and is primarily responsible for detecting errors in the IP packet during its transmission across the network. It helps ensure the integrity of the IP packet as it is routed through various network devices.

2. TCP Checksum:

The TCP checksum is a field within the TCP header. It is calculated over the entire TCP segment, including the TCP header and payload (data). The purpose of the TCP checksum is to verify the integrity of the TCP segment during its transmission between the source and destination hosts. It ensures that the TCP segment has not been corrupted or modified in route.

The IP checksum operates at the network layer and provides integrity checking for the IP packet as a whole. It detects errors that may occur in the IP header, payload, or during transmission through network devices. The IP checksum ensures that the IP packet is delivered correctly across the network.

The TCP checksum, which is part of the TCP header, focuses on the integrity of the TCP segment specifically. It verifies the correctness of the TCP header and payload (data) within the TCP segment. The TCP checksum ensures that the TCP segment arrives intact at the destination, protecting the reliability of the TCP connection.

By having separate checksums at different layers, the IP and TCP protocols work together to ensure end-to-end data integrity. The IP checksum checks the integrity of the IP packet throughout the network, while the TCP checksum verifies the integrity of the TCP segment within that IP packet. This layered approach allows for more comprehensive error detection and ensures the reliability of both the IP and TCP protocols.

7. Explain your code.

```
void initS(Segment* seg, uint16_t srcPort, uint16_t desPort) {
    seg->l4info.SourcePort = srcPort;
    seg->l4info.DesPort = desPort;
    seg->l4info.WindowSize = 65535;
}

void replyS(Segment* seg, uint32_t seqNum, uint32_t ackNum, uint8_t flags) {
    seg->l4info.SeqNum = seqNum;
    seg->l4info.AckNum = ackNum;
    seg->l4info.Flag = flags;
}
```

Set up the segment for sending messages to the server.

```
void sendpacket(int socket_fd, char* buffer, size_t length, Segment* seg, char *unknown, uint16_t flag) {
    myheadercreater(seg);
    memcpy(buffer, seg->header, sizeof(seg->header));
    send(socket_fd, buffer, length, 0);
}
```

Create the header of the segment by l4info which set up by initS and replyS, copy the header to the buffer, and then send this buffer to the server.

```
ssize_t recvpacket(int socket_fd, char* buffer, size_t length, Segment* seg, char *unknown) {
    ssize_t byterecv = recv(socket_fd, buffer, length, 0);
    memcpy(seg->header, buffer, sizeof(char) * 20);
    memcpy(seg->payload, buffer + 20, sizeof(char) * 1000);
    myheaderreverse(seg); seg->p_len = ((byterecv) ? (byterecv - 20) : 0);
    return byterecv;
}
```

Receive packet from the server side, copy the header and payload sent by the server, and then change the header to Acknum and SeqNum which is the integer type. The length of the payload is the value return by the recv() function.

```
bool packet_corrupt(char* buffer, ssize_t length, const char* role) {
    printf("length: %ld\n", length);
    if(length == 0 || length == -1) return true;
    char check[1032]; memset(check, 0, sizeof(check));
    memcpy(check, buffer, sizeof(char) * 16);
    check[20] = 127; check[21] = 0; check[22] = 0; check[23] = 1;
    check[24] = 127; check[25] = 0; check[26] = 0; check[27] = 1;
    check[28] = 0; check[29] = 6; check[30] = 0; check[31] = 20;
    memcpy(check + 32, buffer + 20, sizeof(char) * (length - 20));
    uint16_t *p = (uint16_t *)buffer;
    uint16_t checksum = (*(p + 8));
    if(mychecksum(check, sizeof(check)) == ntohs(checksum)) return false;
    else return true;
}
```

Decide whether the packet is corrupted or not by checking the checksum passed by server and checksum we computed. That is, check whether the checksum we received and the checksum we computed by received data is the same or not.

```
void myheadercreater(Segment *s)
{
    s->header[0] = (s->l4info.SourcePort >> 8);
    s->header[1] = (s->l4info.SourcePort & 255);
    s->header[2] = (s->l4info.DesPort >> 8);
    s->header[3] = (s->l4info.DesPort & 255);
    s->header[4] = (s->l4info.SeqNum >> 24);
    s->header[5] = ((s->l4info.SeqNum >> 16) & 255);
    s->header[6] = ((s->l4info.SeqNum >> 8) & 255);
    s->header[7] = (s->l4info.SeqNum & 255);
    s->header[8] = (s->l4info.AckNum >> 24);
    s->header[9] = ((s->l4info.AckNum >> 16) & 255);
    s->header[10] = ((s->l4info.AckNum >> 8) & 255);
    s->header[11] = (s->l4info.AckNum & 255);
    s->header[12] = s->l4info.HeaderLen << 4;
    s->header[13] = s->l4info.Flag;
    s->header[14] = (s->l4info.WindowSize >> 8);
    s->header[15] = (s->l4info.WindowSize & 255);
}
```

Create the header(Called by sendpacket).

```
void myheaderreverse(Segment *s)
{
    s->l4info.SeqNum = 0, s->l4info.AckNum = 0;
    for(int i = 4, leftshift = 24; i <= 7; ++i, leftshift -= 8) {
        uint32_t tmp = ((s->header[i] < 0) ? 255 + s->header[i] + 1 : s->header[i]);
        s->l4info.SeqNum += (tmp << leftshift);
    }
    for(int i = 8, leftshift = 24; i <= 11; ++i, leftshift -= 8) {
        uint32_t tmp = ((s->header[i] < 0) ? 255 + s->header[i] + 1 : s->header[i]);
        s->l4info.AckNum += (tmp << leftshift);
    }
    s->l4info.Flag = s->header[13];
}
```

Change the data contained in header into integer type, especially Acknum, SeqNum and Flag(Called by recvpacket).

```
char o_buffer[20], i_buffer[1020];
uint32_t currentSeg, currentAck;
uint16_t sPort, Dport;
Segment sendS, recvS;
currentSeg = rand();
currentAck = 0;
sPort = rand()%65535 + 1;
Dport = SERVER_PORT;
initS(&sendS,sPort,Dport);
replyS(&sendS,currentSeg,currentAck,SYN);
sendpacket(socket_fd,o_buffer,sizeof(o_buffer),&sendS,"client",0);
recvpacket(socket_fd,i_buffer,sizeof(i_buffer),&recvS,"client");
currentAck = recvS.l4info.SeqNum+1;
currentSeg = recvS.l4info.AckNum;
replyS(&sendS,currentSeg,currentAck,ACK);
sendpacket(socket_fd,o_buffer,sizeof(o_buffer),&sendS,"client",0);
```

Three way hand shake.

```
Segment last_recv_packet = recvS;
int base = sendS.l4info.AckNum;
printf("base = %d\n", base);
FILE* file = fopen("received_image.jpg", "wb");
if (file == NULL) {
    perror("Fail to open");
    exit(1);
}
while(1){
    ssize_t byterecv = recvpacket(socket_fd,i_buffer, sizeof(i_buffer),&recvS,"client");
    if(byterecv == 0) break;
    if((CORRUPT_DETECT) ? (!packet_corrupt(i_buffer,byterecv,"client")) : 1){
        if(recvS.l4info.SeqNum == sendS.l4info.AckNum){
            last_recv_packet = recvS;
            fwrite(recvS.payload, 1, recvS.p_len, file);
            currentAck = last_recv_packet.l4info.SeqNum + last_recv_packet.p_len;
            currentSeg = last_recv_packet.l4info.AckNum;
            replyS(&sendS,currentSeg,currentAck,ACK);
            sendpacket(socket_fd,o_buffer,sizeof(o_buffer),&sendS,"client",0);

            sleep(0);
        }else{
            printf("recv seqnum == %d, send acknum == %d\n", recvS.l4info.SeqNum, sendS.l4info.AckNum);
            printf("Rdt client : cumulative ack, send last ack packet!\n");
            sendpacket(socket_fd,o_buffer,sizeof(o_buffer),&sendS, "client", 0);
            sleep(0);
        }
    }else{
        printf("Rdt client : Dropped corrupt packet\n");
        sendpacket(socket_fd,o_buffer,sizeof(o_buffer),&sendS,"client",0);
        sleep(0);
    }
}
```

Receive data.

8. What did you learn in LAB 3?

In Lab3, I have learned how to initiate three way hand shake, and how to implement data transmission between server and client with TCP protocol. Three way hand shake will ensure that server and client can have proper connection,

and then the server and client can transmit data in this way. The mechanism of TCP connection and data transmission with pipeline can be discovered by observing the log when the program is running. When receiving the entire picture without corruption, I feel I know more about the conception taught in class.