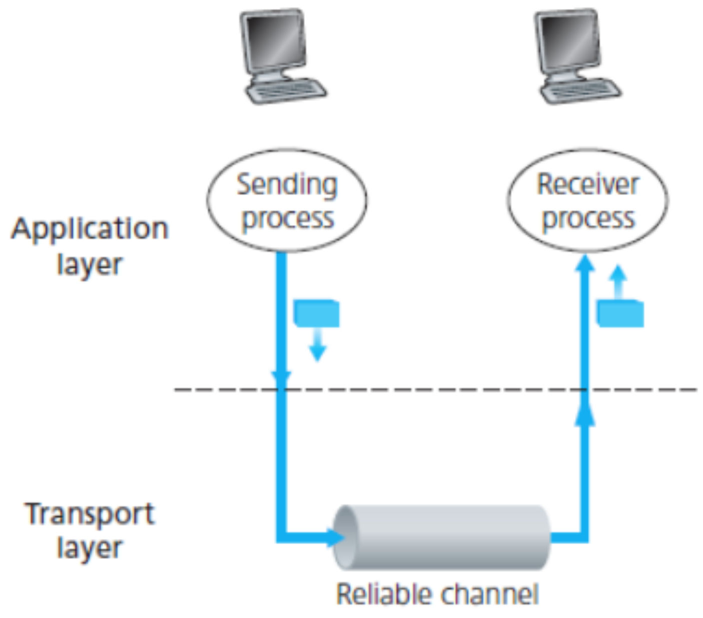
Lab3 – TCP Reliable Data Transfer

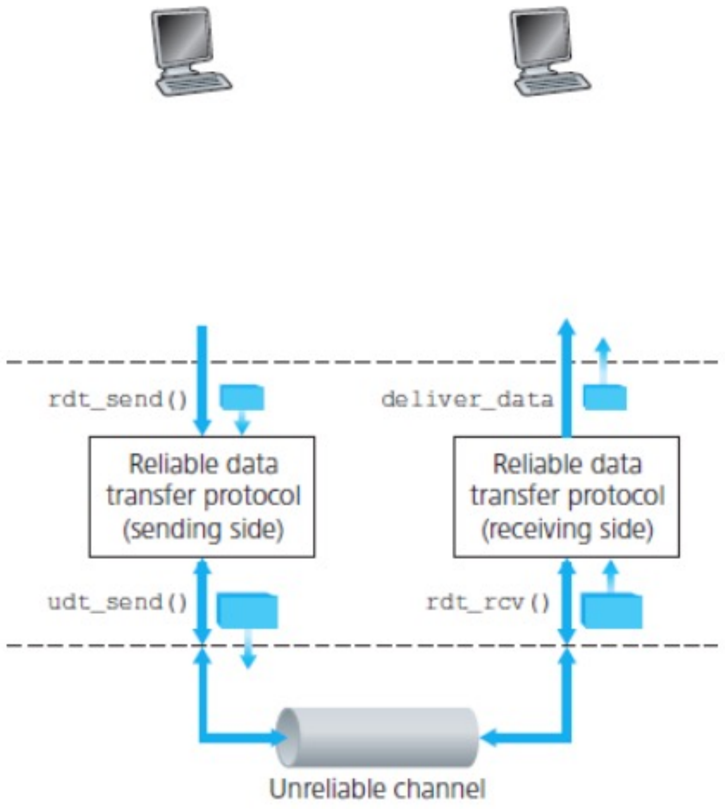
Introduction to Computer Networks

5/29/2023

Reliable Data Transfer

Why do we need to learn about RDT?



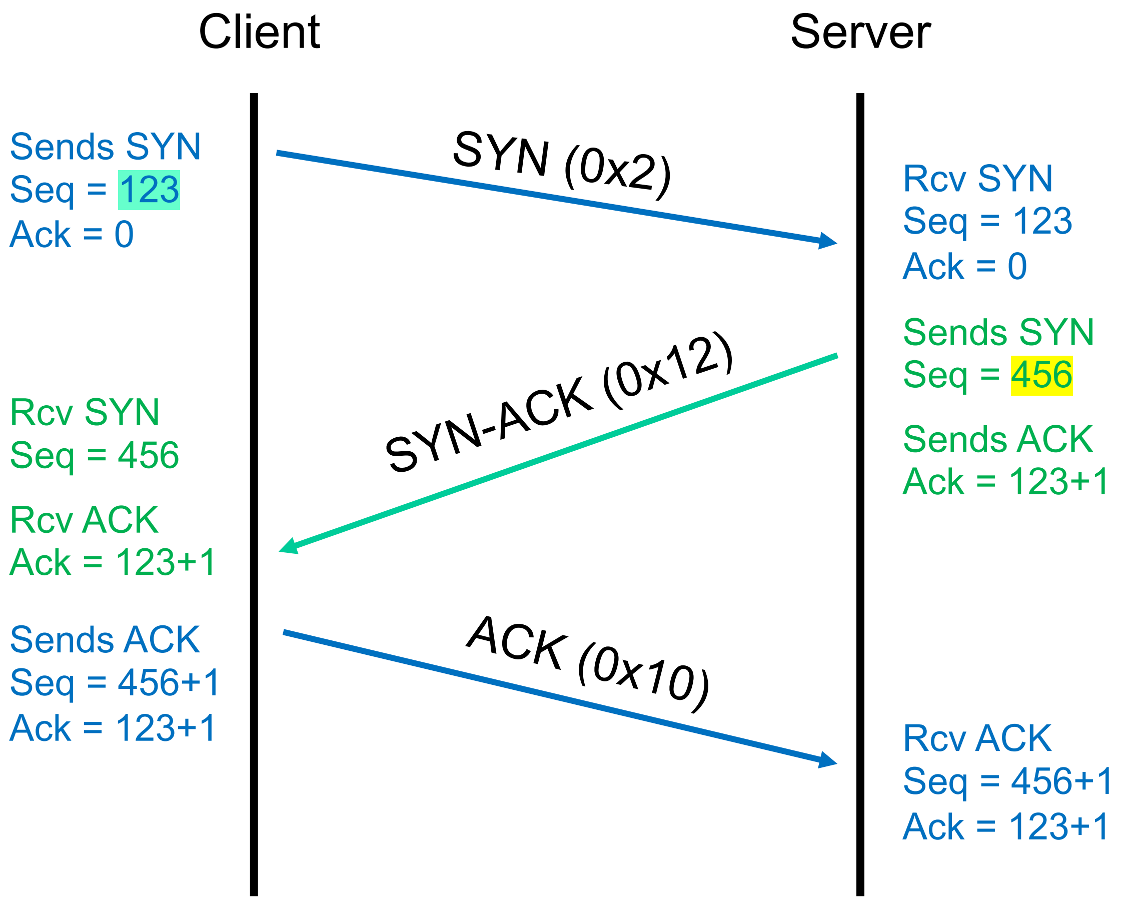


rdt_send()
deliver_data
Reliable data transfer protocol (sending side)
Reliable data transfer protocol (receiving side)
udt_send()
rdt_rcv()
Unreliable channel

Transmission Control Protocol (TCP)

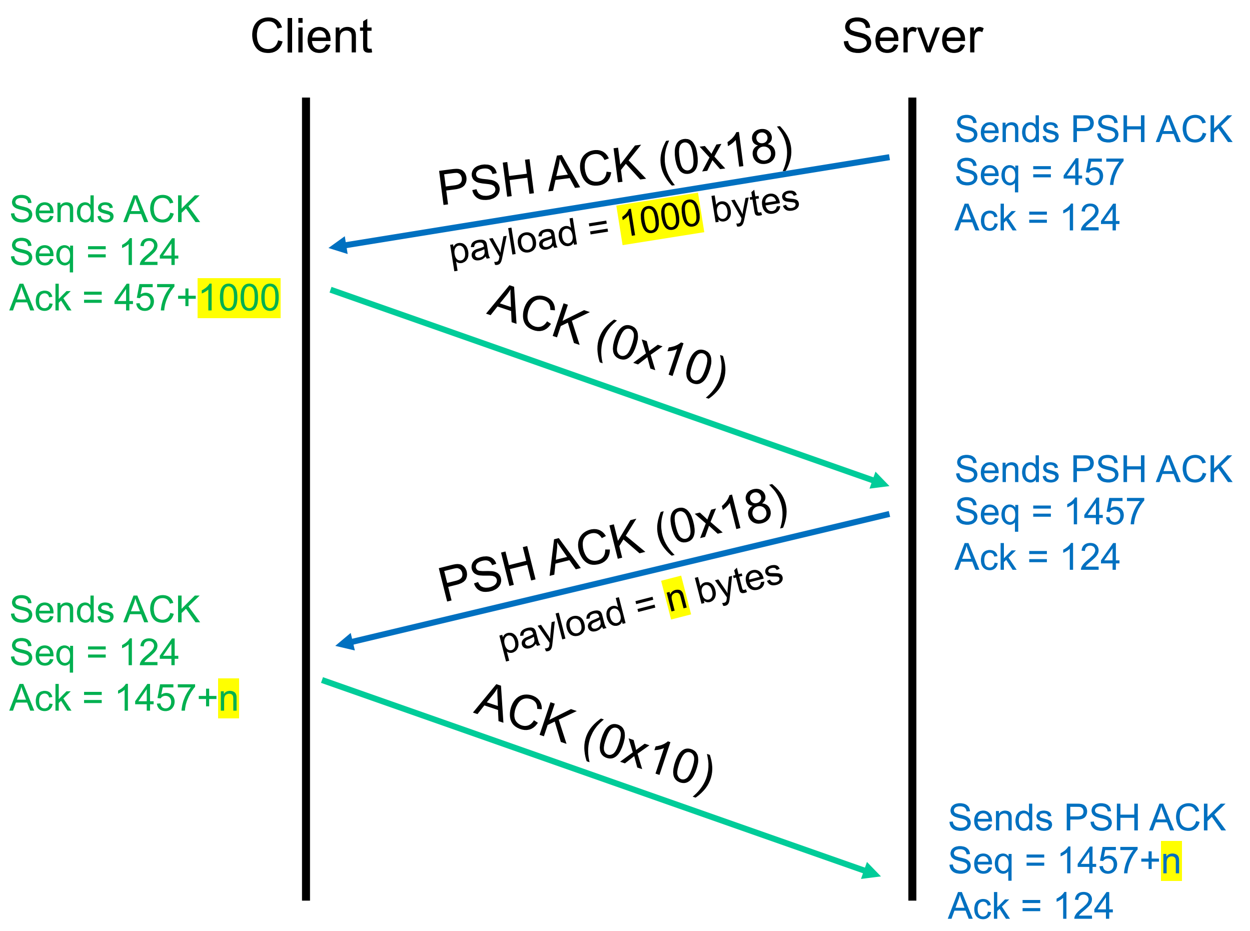
1. Three-way handshake
   - Set up a TCP/IP connection over an IP based network.
2. Reliable data transfer
   - Error detection
   - Sequence numbering
   - Feedback
   - Timers

3-Way Handshake



- **SYN**: SYNchronize
- **ACK**: ACKnowlegement
- **SYN-ACK**: SYNchronize-ACKnowledgement
- **c-isn**: Client Initial Sequence Number
- **s-isn**: Server Initial Sequence Number

Transmit Data



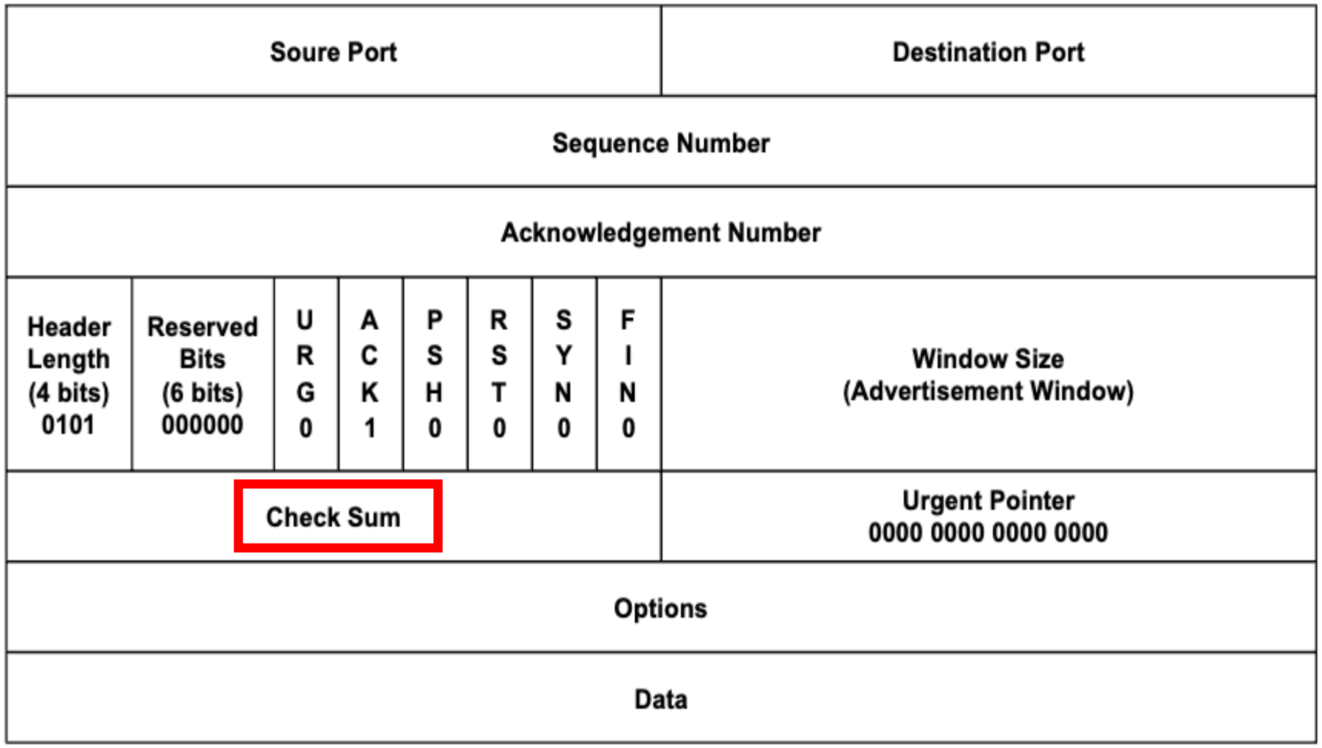
- **PSH**: PuSH
- **ACK**: ACKnowlegement
- **PSH-ACK**: PuSH-ACKnowledgement
- **Payload**: the carrying capacity of a packet or other transmission data unit

Reliable Data Transfer

- **Error detection**: to detect corrupt packets
- **Sequence numbering**: to make sure packets are in the correct order
- **Feedback**: to recover missing packets
- **Timers**: to handle packet loss

Error Detection

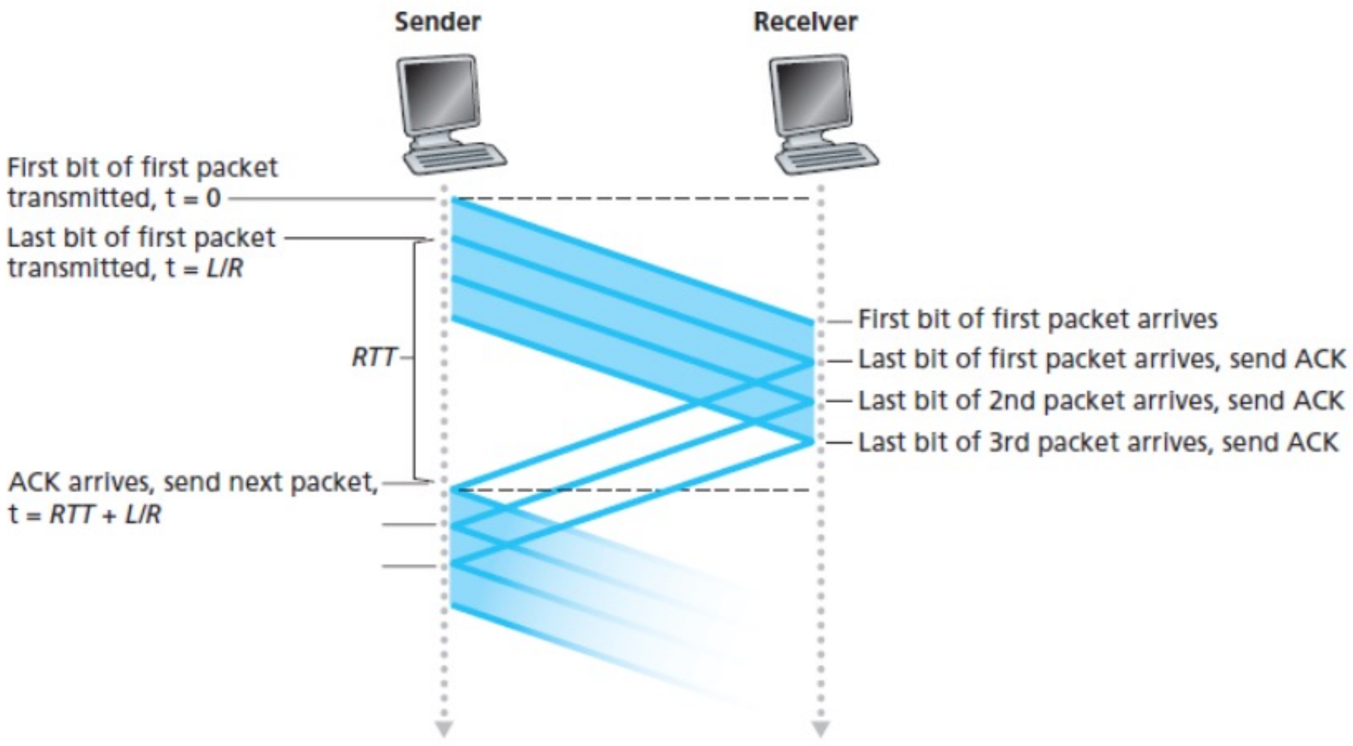
- Use checksum to detect whether the packet is corrupt or not.



Soure Port								Destination Port
Sequence Number								
Acknowledgement Number								
Header Length (4 bits) 0101	Reserved Bits (6 bits) 000000	URG 0	ACK 1	PSH 0	RST 0	SYN 0	FIN 0	Window Size (Advertisement Window)
Check Sum								Urgent Pointer 0000 0000 0000 0000
Options								
Data								

Sequence Numbering

- Packets in the network layer are routed individually. It’s possible that they are received in a different order than they are transmitted.

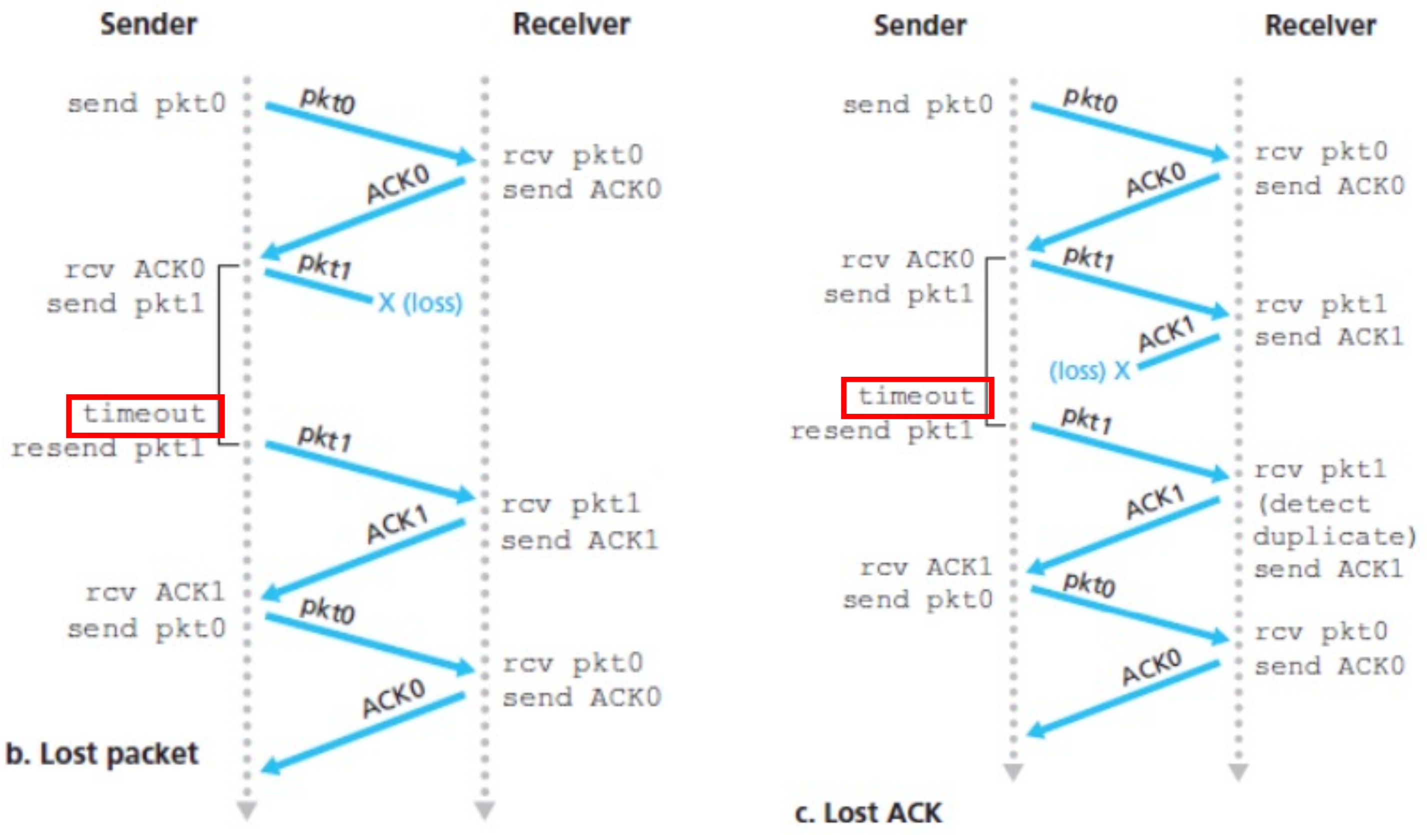

Sender
Receiver
First bit of first packet transmitted, t = 0
Last bit of first packet transmitted, t = L/R
RTT
ACK arrives, send next packet, t = RTT + L/R
First bit of first packet arrives
Last bit of first packet arrives, send ACK
Last bit of 2nd packet arrives, send ACK
Last bit of 3rd packet arrives, send ACK

Feedback

- **NAK** - "I did not receive the packet with sequence number n."
- **Individual ACK** - "I received the packet with sequence number n."
- **Cumulative ACK** - "I have received *all* packets with sequence numbers *up to but not including* n."

Timers

- Sender resends packet after the timer expires.



The Assignment

The Assignment

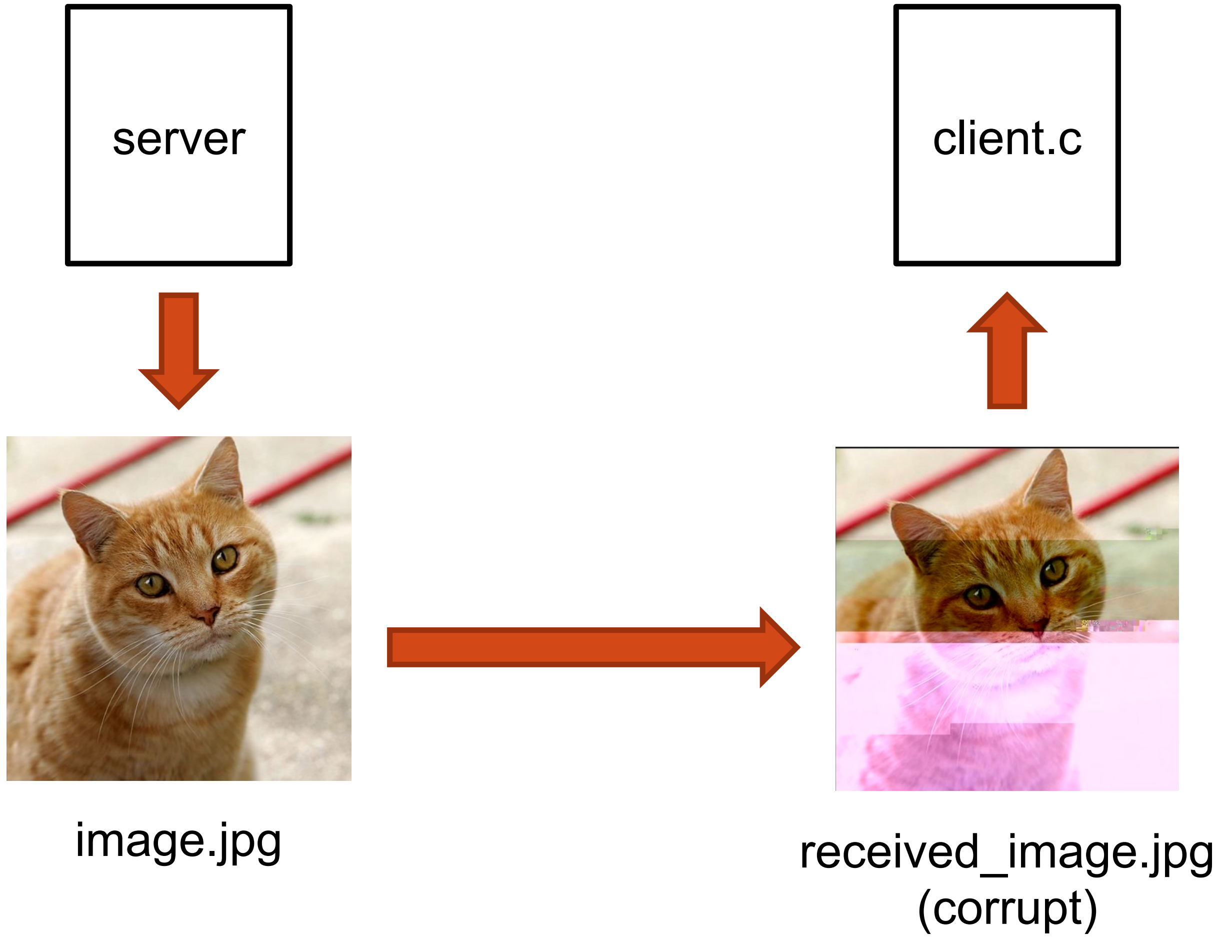
In lab 3, you will each get a zip file containing:

1. client.c
2. server
3. header.h
4. Makefile

In the same folder, you should have:

1. image.jpg (can be your own image)

server



client.c

For this lab, you only need to do client.c!

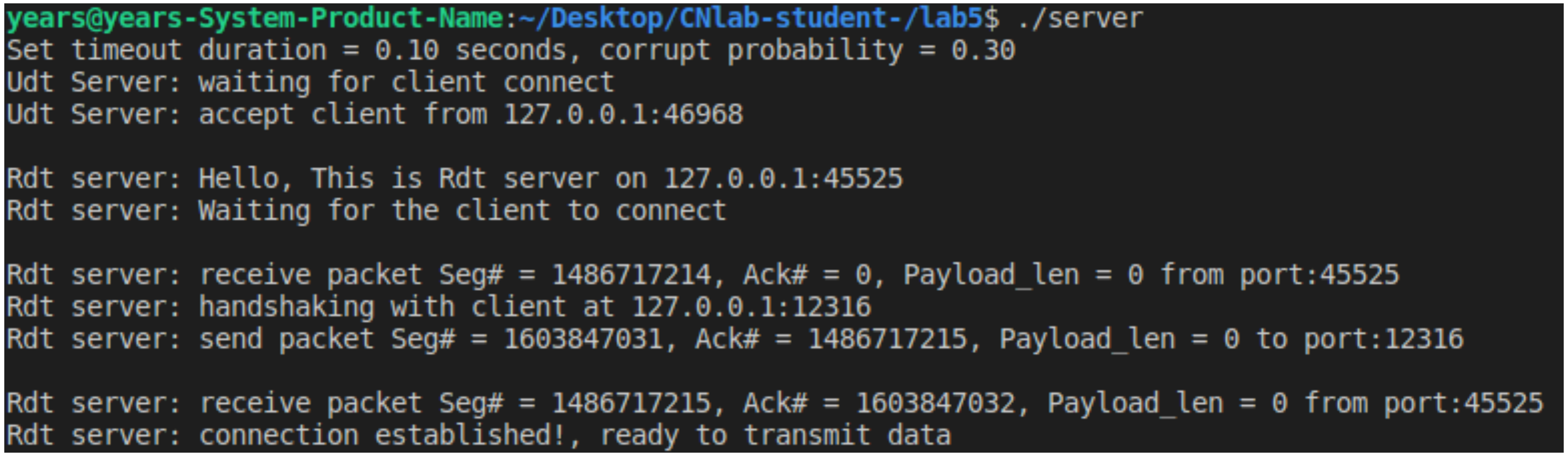
TASK 1: Establish 3-way handshake
TASK 2: Receive image file (may be corrupt)
IMPORTANT: you can use your own image, but rename it “image.jpg”.

TASK 3: Receive correct image file by checking the bit errors in the packet.

TASK 1

TASK 1: Establish 3-way handshake

Server output:



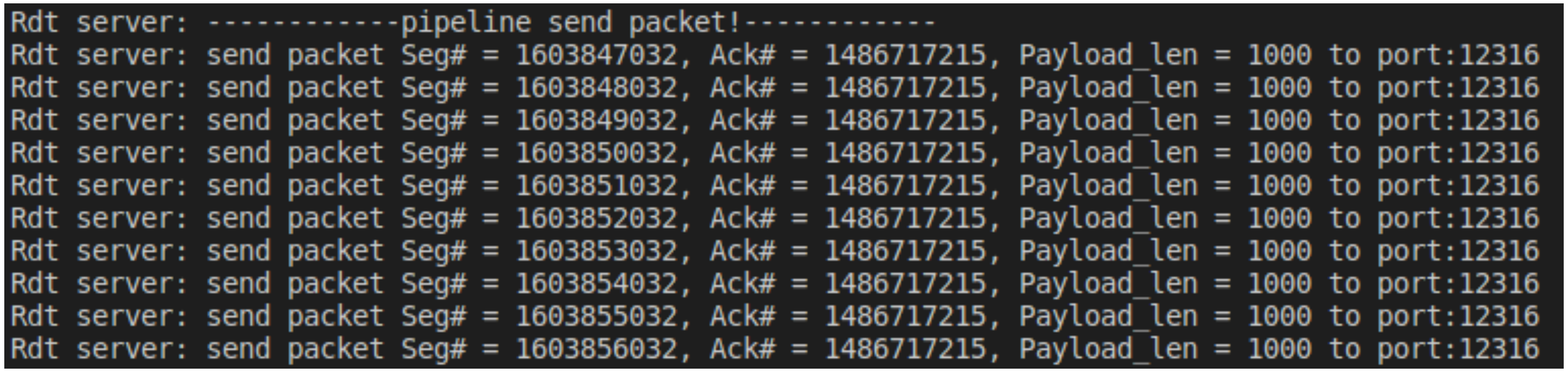
```
years@years-System-Product-Name:~/Desktop/CNlab-student-/lab5$ ./server
Set timeout duration = 0.10 seconds, corrupt probability = 0.30
Udt Server: waiting for client connect
Udt Server: accept client from 127.0.0.1:46968

Rdt server: Hello, This is Rdt server on 127.0.0.1:45525
Rdt server: Waiting for the client to connect

Rdt server: receive packet Seg# = 1486717214, Ack# = 0, Payload_len = 0 from port:45525
Rdt server: handshaking with client at 127.0.0.1:12316
Rdt server: send packet Seg# = 1603847031, Ack# = 1486717215, Payload_len = 0 to port:12316

Rdt server: receive packet Seg# = 1486717215, Ack# = 1603847032, Payload_len = 0 from port:45525
Rdt server: connection established!, ready to transmit data
```

Hint: use the socket programming from lab 2

TASK 2

TASK 2: Receive image file (may be corrupt)

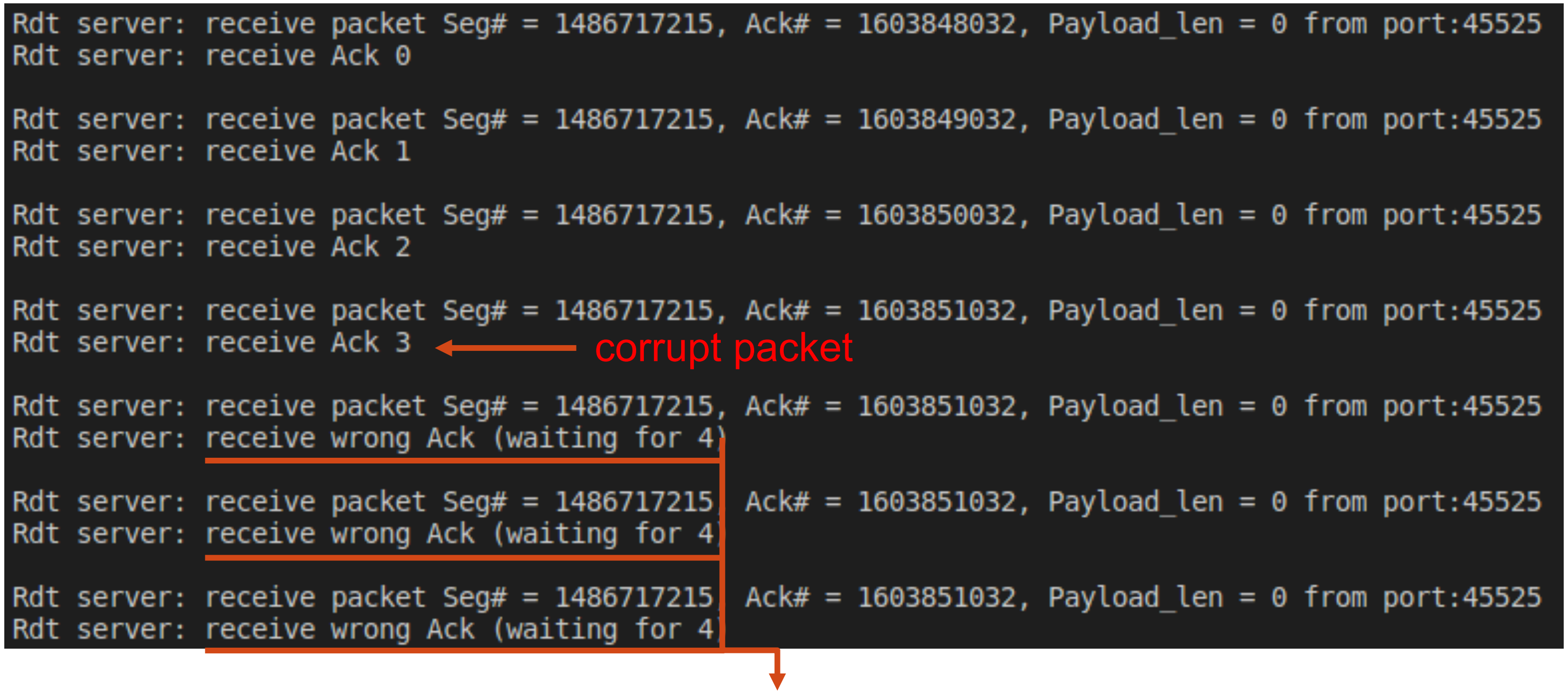
Server output:



```
Rdt server: ------------pipeline send packet!------------
Rdt server: send packet Seg# = 1603847032, Ack# = 1486717215, Payload_len = 1000 to port:12316
Rdt server: send packet Seg# = 1603848032, Ack# = 1486717215, Payload_len = 1000 to port:12316
Rdt server: send packet Seg# = 1603849032, Ack# = 1486717215, Payload_len = 1000 to port:12316
Rdt server: send packet Seg# = 1603850032, Ack# = 1486717215, Payload_len = 1000 to port:12316
Rdt server: send packet Seg# = 1603851032, Ack# = 1486717215, Payload_len = 1000 to port:12316
Rdt server: send packet Seg# = 1603852032, Ack# = 1486717215, Payload_len = 1000 to port:12316
Rdt server: send packet Seg# = 1603853032, Ack# = 1486717215, Payload_len = 1000 to port:12316
Rdt server: send packet Seg# = 1603854032, Ack# = 1486717215, Payload_len = 1000 to port:12316
Rdt server: send packet Seg# = 1603855032, Ack# = 1486717215, Payload_len = 1000 to port:12316
Rdt server: send packet Seg# = 1603856032, Ack# = 1486717215, Payload_len = 1000 to port:12316
```

TASK 3

TASK 3: Receive correct image file by checking the bit errors in the packet.

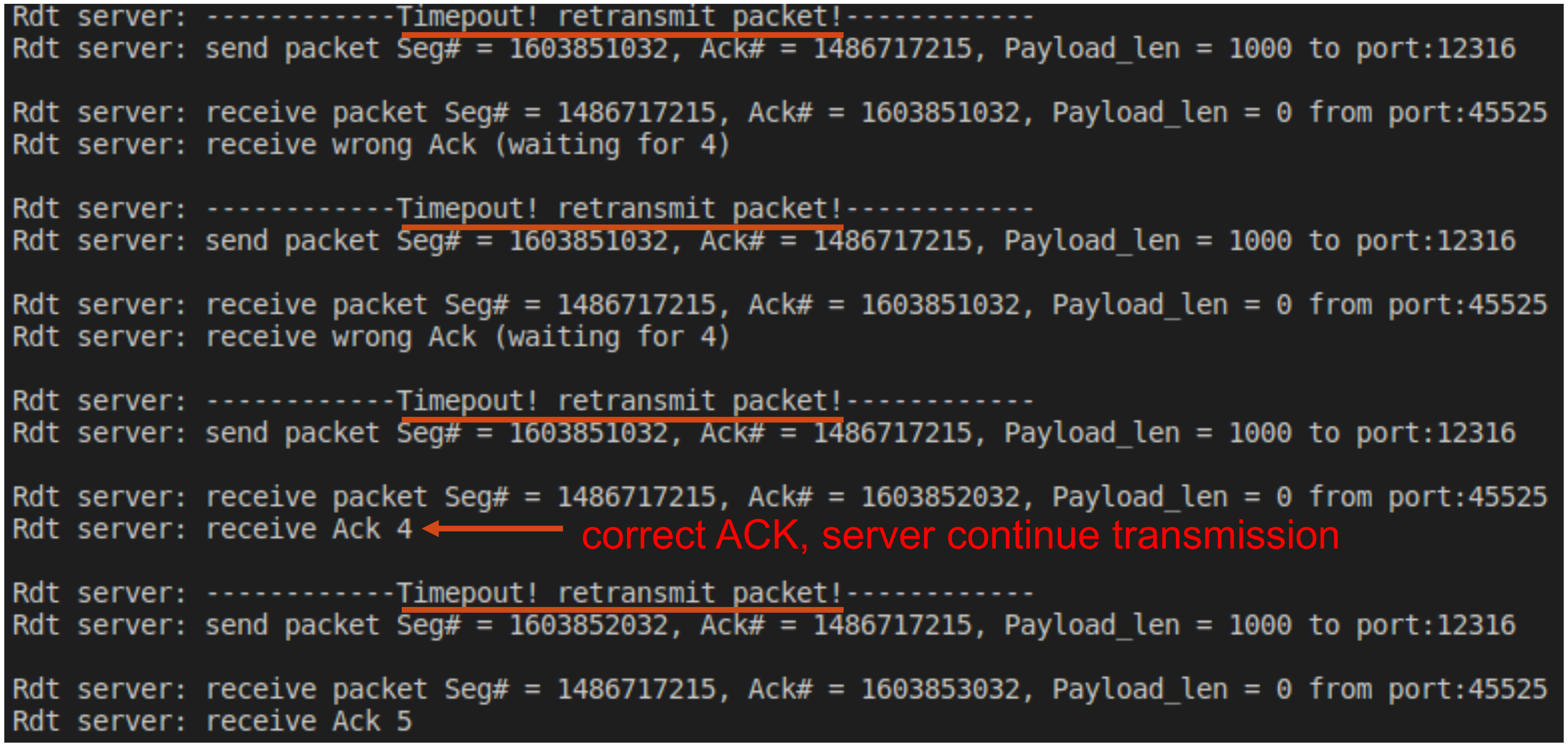


```
Rdt server: receive packet Seg# = 1486717215, Ack# = 1603848032, Payload_len = 0 from port:45525
Rdt server: receive Ack 0

Rdt server: receive packet Seg# = 1486717215, Ack# = 1603849032, Payload_len = 0 from port:45525
Rdt server: receive Ack 1

Rdt server: receive packet Seg# = 1486717215, Ack# = 1603850032, Payload_len = 0 from port:45525
Rdt server: receive Ack 2

Rdt server: receive packet Seg# = 1486717215, Ack# = 1603851032, Payload_len = 0 from port:45525
Rdt server: receive Ack 3

Rdt server: receive packet Seg# = 1486717215, Ack# = 1603851032, Payload_len = 0 from port:45525
Rdt server: receive wrong Ack (waiting for 4)

Rdt server: receive packet Seg# = 1486717215, Ack# = 1603851032, Payload_len = 0 from port:45525
Rdt server: receive wrong Ack (waiting for 4)

Rdt server: receive packet Seg# = 1486717215, Ack# = 1603851032, Payload_len = 0 from port:45525
Rdt server: receive wrong Ack (waiting for 4)
```

client.c should send ACK 3 (“wrong” ACK) to request a retransmission.

TASK 3

Server output when Timeout:

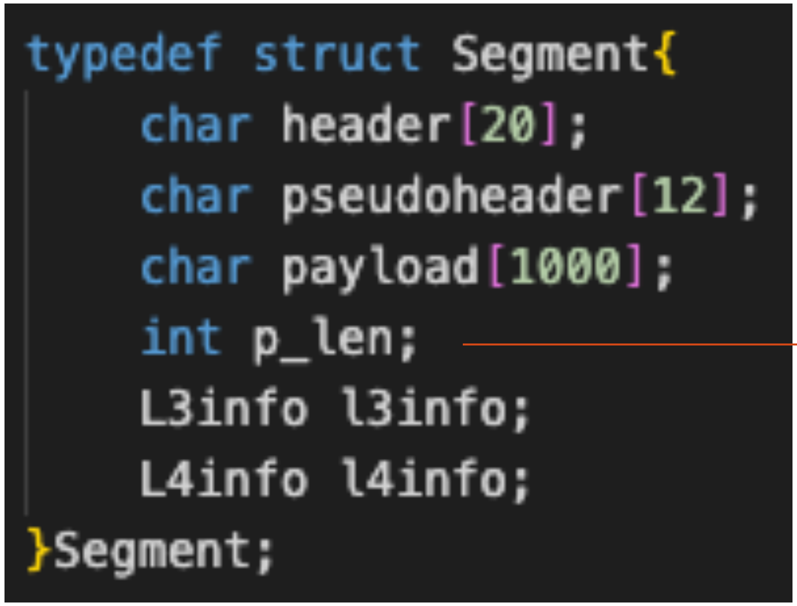


```
Rdt server: ------------Timepout! retransmit packet!------------
Rdt server: send packet Seg# = 1603851032, Ack# = 1486717215, Payload_len = 1000 to port:12316

Rdt server: receive packet Seg# = 1486717215, Ack# = 1603851032, Payload_len = 0 from port:45525
Rdt server: receive wrong Ack (waiting for 4)

Rdt server: ------------Timepout! retransmit packet!------------
Rdt server: send packet Seg# = 1603851032, Ack# = 1486717215, Payload_len = 1000 to port:12316

Rdt server: receive packet Seg# = 1486717215, Ack# = 1603851032, Payload_len = 0 from port:45525
Rdt server: receive wrong Ack (waiting for 4)

Rdt server: ------------Timepout! retransmit packet!------------
Rdt server: send packet Seg# = 1603851032, Ack# = 1486717215, Payload_len = 1000 to port:12316

Rdt server: receive packet Seg# = 1486717215, Ack# = 1603852032, Payload_len = 0 from port:45525
Rdt server: receive Ack 4

Rdt server: ------------Timepout! retransmit packet!------------
Rdt server: send packet Seg# = 1603852032, Ack# = 1486717215, Payload_len = 1000 to port:12316

Rdt server: receive packet Seg# = 1486717215, Ack# = 1603853032, Payload_len = 0 from port:45525
Rdt server: receive Ack 5
```

correct ACK, server continue transmission

header.h

Segment Structs

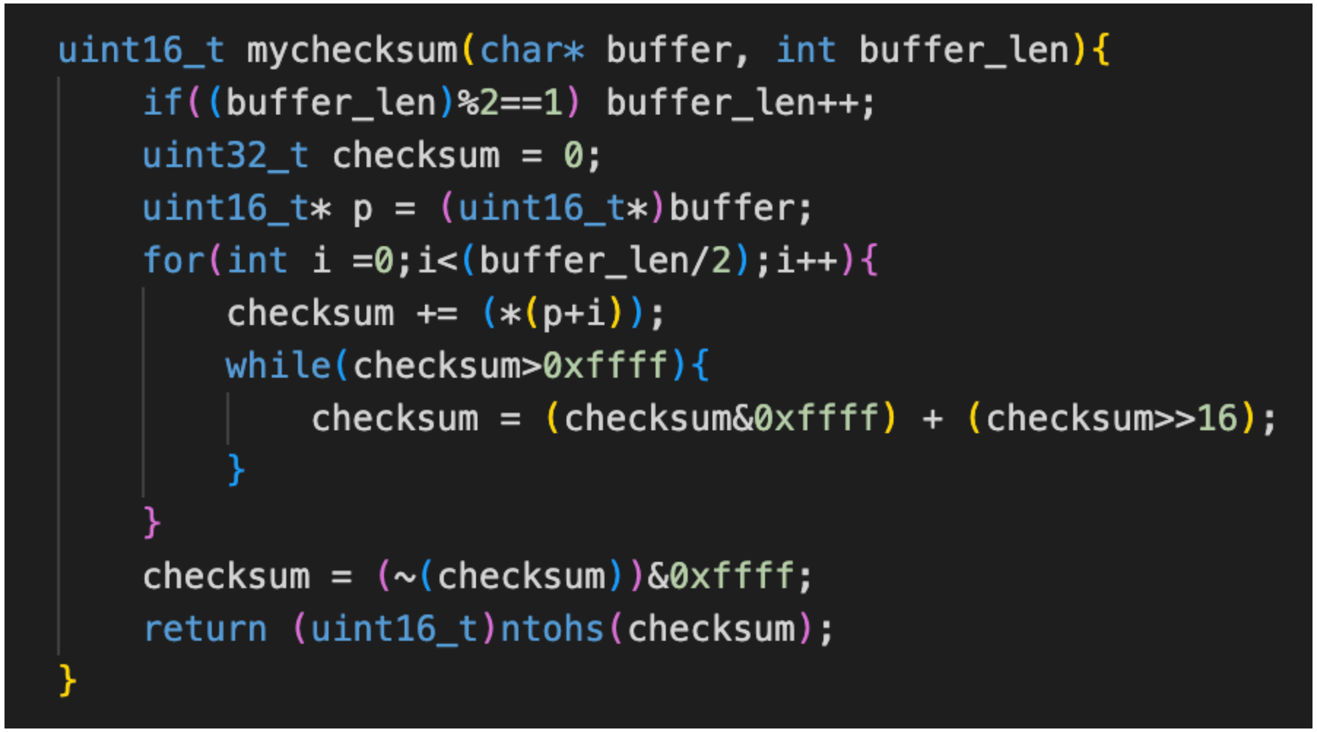


```
typedef struct Segment{
    char header[20];
    char pseudoheader[12];
    char payload[1000];
    int p_len;
    L3info l3info;
    L4info l4info;
}Segment;
```

payload length

header.h

Predefined function **mychecksum**(char* buffer, int buffer_len)



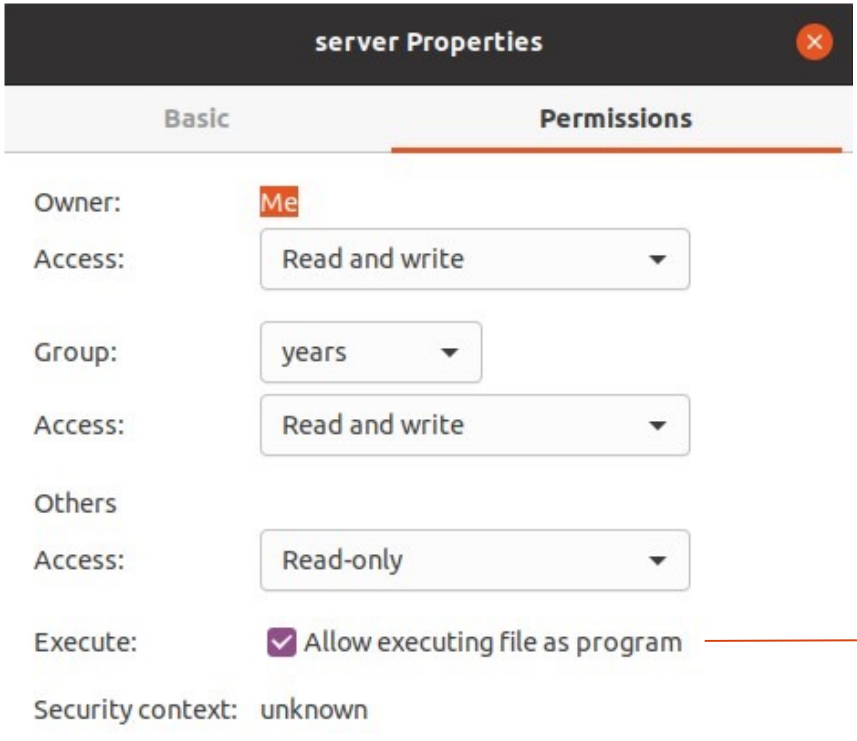
```
uint16_t mychecksum(char* buffer, int buffer_len){
    if((buffer_len)%2==1) buffer_len++;
    uint32_t checksum = 0;
    uint16_t* p = (uint16_t*)buffer;
    for(int i =0;i<(buffer_len/2);i++){
        checksum += (*(p+i));
        while(checksum>0xffff){
            checksum = (checksum&0xffff) + (checksum>>16);
        }
    }
    checksum = (~(checksum))&0xffff;
    return (uint16_t)ntohs(checksum);
}
```

buffer should contain header + pseudoheader + payload, then use it as the function’s input.

Important Notes

1. The SrcIP and DesIP are both 127.0.0.1,
   TCP header length will be 20 bytes, windowsize = 65535 bytes
2. The Handshake packets won't be corrupt.
3. The packet will only be corrupt, but not lost or be disordered.
4. Only the packets come from server may be corrupt.
   (Don't have to worry that the ack sent by client will corrupt.)
5. We offer mychecksum() for you to verify the checksum, and don't forget to verify the pseudoheader part.
6. Once server finish transmit the file, it will close the client socket.
7. You can adjust server by
   ./server {timeout duration} {corrupt probability}

Important Notes



server Properties

Basic | Permissions

Owner: Me

Access: Read and write

Group: years

Access: Read and write

Others

Access: Read-only

Execute: [✓] Allow executing file as program

Security context: unknown

If you encounter a permission problem, you should try checking this option.

Report

1. What is SYN, SYN-ACK, PSH-ACK, and ACK? (10%)
2. Why not just use 0 as the Initial Sequence Number (ISN)? (10%)
3. Why do we need Three-Way Handshake when establishing a connection, but 4-Way Say Goodbye when ending the connection? (10%)
4. What is Cumulative ACK? (10%)
5. Why do we need the pseudoheader? (10%)
6. We know the IP protocol has checksum, and so do the TCP protocol. What's different? And why do we need both? (20%)
7. Explain your code. (20%)
8. What did you learn in LAB 3? (10%)

Requirements

1. Implementation of TCP RDT (70%)

- Client.c:
  - TASK 1: Establish 3-way handshake (60%)
  - TASK 2: Receive JPG file (may be corrupt) (25%)
  - TASK 3: Receive correct JPG file (15%)

2. Report (30%)

Lab 3 percentage: 15%

Submission date: 6/22 (Thu) 11:59 PM

If you have any questions, please post it on eeclass!