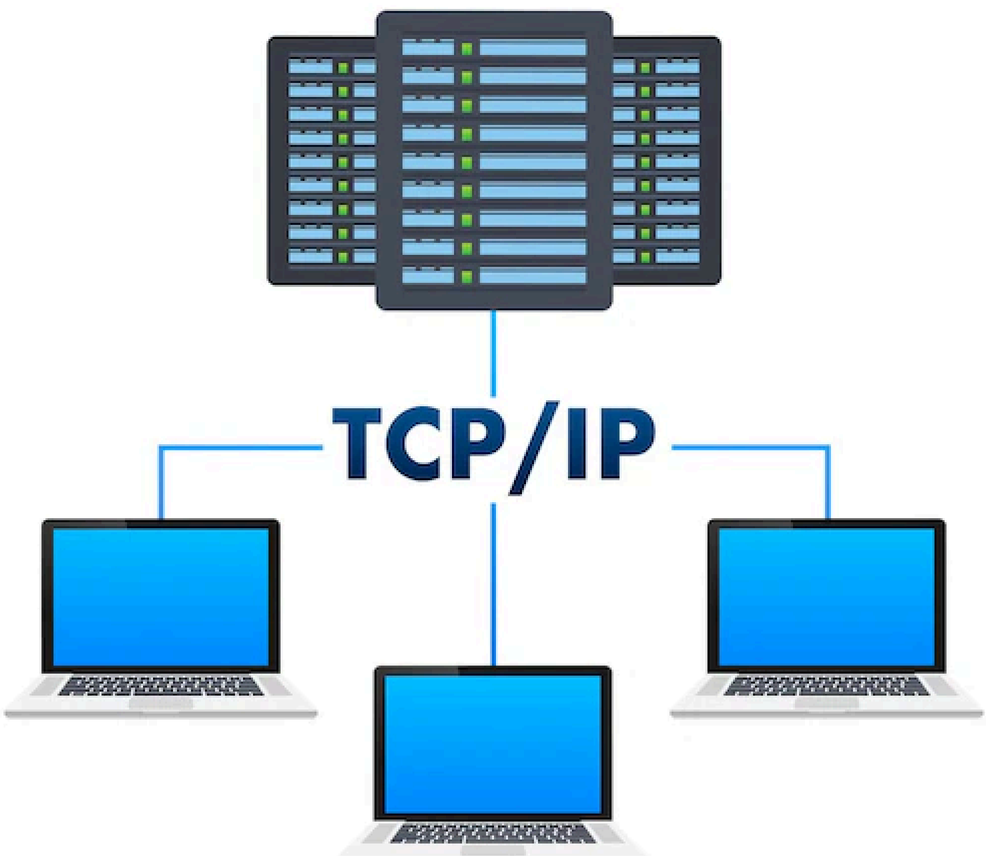
CS-335a

Project - Phase B Report

MicroTCP -a lightweight implementation of TCP



Introduction

MicroTCP project is fully implemented. It contains complete functionality as the core TCP protocol. For example, it provides features like reliable connection on a heavily used network, flow & congestion control and correct usage of information contained in a microTCP header. You can run it by creating a ‘build’ folder,entering it and executing “cmake .. ” & “make all”.

Implementation

Error Checking

First and foremost, we are using the ‘Checksum’ field of the header , so we can ensure that the packet was received successfully.

```
packet.checksum=0;
memcpy(recv_buf,&packet,sizeof(microtcp_header_t));
 if(checksum != crc32((const uint8_t*)recv_buf,size)){
    perror("checksum error 9");
    continue;      You, 1 second ago • Uncommitted changes
}
```

We consider that the packet was never received.

Packet Receiving in Correct Order

The purpose of our protocol is the reliability, so throughout the 2-way connection the receiving packet sequence is checked.

```
if(packet.seq_number!=socket->ack_number+packet.data_len){
    //send DUP ACK
    packet=create_header(socket->seq_number,ACK,0,socket->ack_number,socket->init_win_size-socket->buf_fill_level);
    packet.checksum=htonl(crc32((uint8_t*)&packet,sizeof(microtcp_header_t)));
    #ifdef  DEBUG
    printf("Sending ACK packet with ack number: %lu\n",socket->ack_number);
    #endif
    if(sendto(socket->sd,(void*)&packet,sizeof(microtcp_header_t),0,(struct sockaddr*)socket->address,socket->address_len)==-1){
        perror("sending ACK packet");
        exit(EXIT_FAILURE);
    }
    return EXIT_FAILURE;
}
```

Retransmissions

There are two reasons for a retransmission to occur;

- Timeout
- 3 - Duplicate ACK

Firstly if a timeout occurs we simply retransmit the last packet.

```c
timeout.tv_sec = 0;
timeout.tv_usec = MICROTCP_ACK_TIMEOUT_US;
if (setsockopt(socket->sd, SOL_SOCKET, SO_RCVTIMEO, & timeout ,sizeof( struct timeval)) < 0){
    perror("setsockopt");
  }
if(socket->state==CLOSING_BY_PEER){
  return -1;
}
while(1){
  status=recvfrom(socket->sd,recv_buf,(MAX_PAYLOAD_SIZE)+sizeof(microtcp_header_t),flags,(struct sockaddr*)socket->address,&socket->address_len);
  if(status==-1){
      perror("receiving packet");
      return -EXIT_FAILURE;
  }
```

On the other hand if we receive 3 duplicate ACK's we retransmit the packet with the correct sequence number.

```c
if(count==3){
  //fast retransmit
  data_sent=data_sent+(header.ack_number-starting_seq);
  remaining=remaining-(header.ack_number-starting_seq);
  socket->seq_number=header.ack_number;
  ack_check=0;
  socket->ssthresh=socket->cwnd/2;
  socket->cwnd/2+1;
  break;
}
```

Flow Control

Our protocol contains Flow-Control to avoid packet loss, due to differences in hardware specifications of host and client.

1. First packet has X size which is equal or less than MSS.

```c
while( data_sent < length){
    starting_seq=socket->seq_number;
    bytes_to_send = min( socket->curr_win_size , socket->cwnd ,remaining);
    chunks = bytes_to_send / MAX_PAYLOAD_SIZE;
    for(i = 0; i < chunks;i++){
        header=create_header(socket->seq_number+MAX_PAYLOAD_SIZE,ACK,MAX_PAYLOAD_SIZE,socket->ack_number,socket->init_win_size-socket->buf_fill_level);
        send_buf=calloc(1,MICROTCP_MSS);
        memcpy(send_buf,&header,sizeof(microtcp_header_t));
        memcpy(send_buf+sizeof(microtcp_header_t),buffer+data_sent+i*MAX_PAYLOAD_SIZE,MAX_PAYLOAD_SIZE);
        header.checksum=htonl(crc32((uint8_t*)send_buf,MICROTCP_MSS));
        memcpy(send_buf,&header,sizeof(microtcp_header_t));
        //print
        #ifdef  DEBUG
        printf("Sending packet with sequence number: %lu, i: %d, chunks: %d\n",socket->seq_number+MAX_PAYLOAD_SIZE,i,chunks);
        #endif
        if(sendto(socket->sd,send_buf,MICROTCP_MSS,flags,(struct sockaddr*)socket->address,socket->address_len)==-1){
            perror("sending packet");
            exit(EXIT_FAILURE);
        }
        socket->seq_number+=MAX_PAYLOAD_SIZE;
        free(send_buf);
    }
```

Firstly the curr_win_size variable is agreed through 3-way-handshake to be MICROTCP_MSS (1400B) size.

2. Receiver accepts the packet and stores it inside receive buffer. Then he sends back the ACK and informs the sender how many bytes he is willing to accept in the next packet.

```c
//copy recvbuf to socket->recv_buf
memcpy(socket->recvbuf+socket->buf_fill_level,recv_buf+sizeof(microtcp_header_t),packet.data_len);
socket->buf_fill_level+=packet.data_len;
socket->ack_number=packet.seq_number;
packet=create_header(socket->seq_number,ACK,0,socket->ack_number,socket->init_win_size-socket->buf_fill_level);   Current_Window_Size
packet.checksum=htonl(crc32((uint8_t*)&packet,sizeof(microtcp_header_t)));
#ifdef  DEBUG
printf("Sending ACK packet with ack number: %lu\n",socket->ack_number);
#endif
if(sendto(socket->sd,(void*)&packet,sizeof(microtcp_header_t),0,(struct sockaddr*)socket->address,socket->address_len)==-1){
    perror("sending ACK packet");
}
```

3. Sender can send maximum the window size bytes,that are mentioned in the 'Window' field of an ACK Header.

```c
while( data_sent < length){
    starting_seq=socket->seq_number;
    bytes_to_send = min( socket->curr_win_size , socket->cwnd ,remaining);
```

4. By forwarding bytes to the user, the receiver increases the window size by Y bytes.

```c
if( socket->buf_fill_level+(MAX_PAYLOAD_SIZE)>length){
  memcpy(buffer,socket->recvbuf+start_buf_level,socket->buf_fill_level-start_buf_level);
  //print the return
  int retval=socket->buf_fill_level-start_buf_level;
  socket->buf_fill_level=start_buf_level;
  return retval;
}
if( socket->buf_fill_level+(MAX_PAYLOAD_SIZE)>MICROTCP_RECVBUF_LEN){
  memcpy(buffer,socket->recvbuf+start_buf_level,socket->buf_fill_level-start_buf_level);
  //print the return
  int retval=socket->buf_fill_level-start_buf_level;
  socket->buf_fill_level=0;
  return retval;
}
```

5. In case of 0 window size, sender is sending a packet without payload repeatedly until he receives an ACK packet with non zero size window.

```c
if(socket->curr_win_size==0){
  //send a packet without payload
  header=create_header(socket->seq_number,ACK,0,socket->ack_number,socket->init_win_size-socket->buf_fill_level);
  header.checksum=htonl(crc32((uint8_t*)&header,sizeof(microtcp_header_t)));
  #ifdef  DEBUG
  printf("Sending ACK packet with ack number: %lu\n",socket->ack_number);
  #endif
  if(sendto(socket->sd,(void*)&header,sizeof(microtcp_header_t),0,(struct sockaddr*)socket->address,socket->address_len)==-1){
      perror("sending ACK packet");
      exit(EXIT_FAILURE);
  }

  while(recvfrom(socket->sd,(void*)recv_buf,MICROTCP_RECVBUF_LEN,flags,(struct sockaddr*)socket->address,&socket->address_len)==-1){
      header=create_header(socket->seq_number,ACK,0,socket->ack_number,socket->init_win_size-socket->buf_fill_level);
      header.checksum=htonl(crc32((uint8_t*)&header,sizeof(microtcp_header_t)));
      #ifdef  DEBUG
      printf("Sending ACK packet with ack number: %lu\n",socket->ack_number);
      #endif
      if(sendto(socket->sd,(void*)&header,sizeof(microtcp_header_t),0,(struct sockaddr*)socket->address,socket->address_len)==-1){
          perror("sending ACK packet");
          exit(EXIT_FAILURE);
      }
  }
```

Congestion Control

1. Slow Start

For every valid ACK received, the congestion window increases by the size of the Maximum Segment Size (MSS) in bytes. This implies that during each round-trip time (RTT), where x packets were sent and x ACKs were received, the congestion window doubles.

```
//slow start
if(socket->cwnd<=socket->ssthresh){
  cwnd_inc=1;
  socket->cwnd+=(MAX_PAYLOAD_SIZE);
}
```

2. Congestion Avoidance

During congestion avoidance in TCP, the congestion window increases by the size of the Maximum Segment Size (MSS) in bytes for each round-trip time (RTT), where x packets were sent and x ACKs were received. The congestion window continues to grow until packet loss occurs.

If 3 duplicate ACKs are received, the following changes are made:

ssthresh (slow start threshold) is set to half of the current congestion window (cwnd/2).

cwnd is set to half of its current value plus 1 (cwnd/2 + 1).

```
if(count==3){
  //fast retransmit
  data_sent=data_sent+(header.ack_number-starting_seq);
  remaining=remaining-(header.ack_number-starting_seq);
  socket->seq_number=header.ack_number;
  ack_check=0;
  socket->ssthresh=socket->cwnd/2;
  socket->cwnd/2+1;
  break;
}
```

In the event of a timeout while waiting for an ACK, the following adjustments are made:

ssthresh is set to half of the current congestion window (cwnd/2).
cwnd is set to the minimum of the Maximum Segment Size (MSS) and the slow start threshold (ssthresh).

```
status=recvfrom(socket->sd,(void*)recv_buf,MICROTCP_RECVBUF_LEN,flags,(struct sockaddr*)socket->address,&socket->address_len);
if(status==-1){
    socket->ssthresh=socket->cwnd/2;
    socket->cwnd=(MAX_PAYLOAD_SIZE<socket->ssthresh)?MAX_PAYLOAD_SIZE:socket->ssthresh;
```

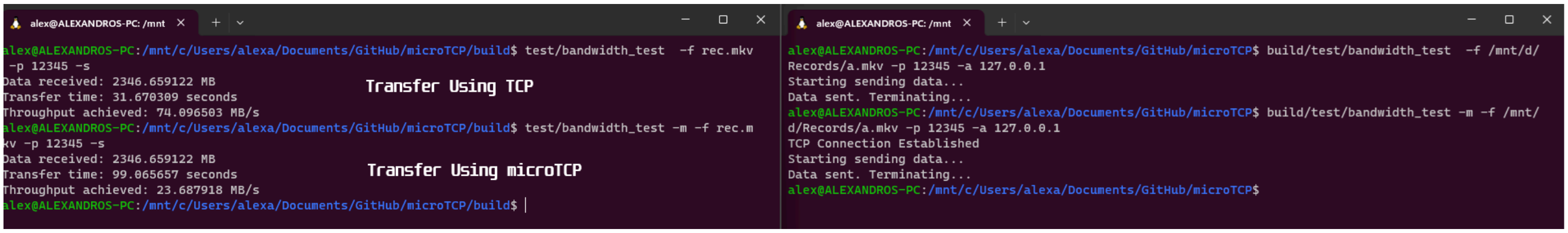
3. Fast Retransmit

Fast retransmit is a mechanism in TCP that resends a missing packet when three duplicate acknowledgments (ACKs) are received, indicating potential packet loss, without waiting for a timeout. So retransmission of the packets occurs.

```
if(count==3){
    //fast retransmit
    data_sent=data_sent+(header.ack_number-starting_seq);
    remaining=remaining-(header.ack_number-starting_seq);
    socket->seq_number=header.ack_number;
```

Performance

TCP Throughput = 74.10 MB/s
microTCP Throughput = 23.69 MB/s



Performance = $\frac{TCP\ Throughput}{microTCP\ Throughput} = \frac{74.10}{23.69}$ = 3.13

So the performance of TCP protocol is 213% faster than microTCP.

Team Members:

Dimitrios Vidalis csd4559
Alexandros Markodimitrakis csd4337