Project 2 overview

The best way to approach this project is in incremental steps. Do not try to implement all of the functionality at once.

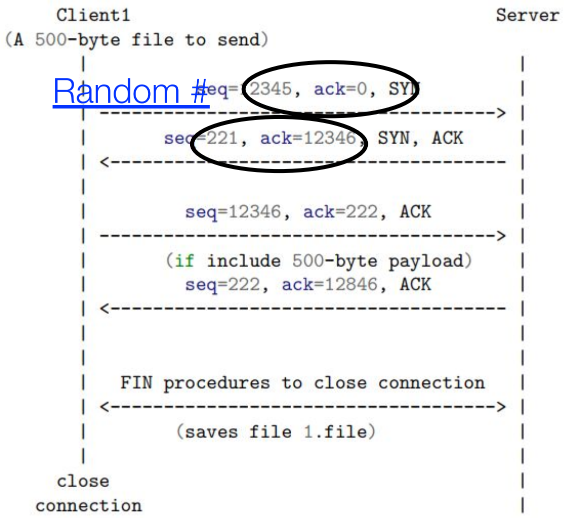
- First, assume there is no packet loss, implement the header fields and connection control functions (initialization with 3-way handshake and termination). Just have the client initiate the connection with 3-way handshake, send a small file (200 Bytes) as a packet, and the server respond with an ACK, and then the server use FIN procedure to close the connection.
- Second, introduce a large file transmission and pipe-lining. This means you must divide the file into multiple packets and transmit the packets based on the specified window size.
- Third, introduce packet loss. Now you have to add a timer for last sent packet (Go-Back-N) or several timers for each unacked packets (Selective repeat). If a timer times out, the corresponding (lost) packet should be retransmitted for the successful file transmission.

Stage 0：Small file transmission

- Small file transmission
  - A client initiate file transmission
  - A server accept connection requests, receive the file and save it with x.file
    - X indicated the counter of connection (starts with 1)
- Test
  - ./server 5000
  - ./client localhost 5000 testfile
  - In the server folder, check whether 1.file is saved and compare two files with diff command.

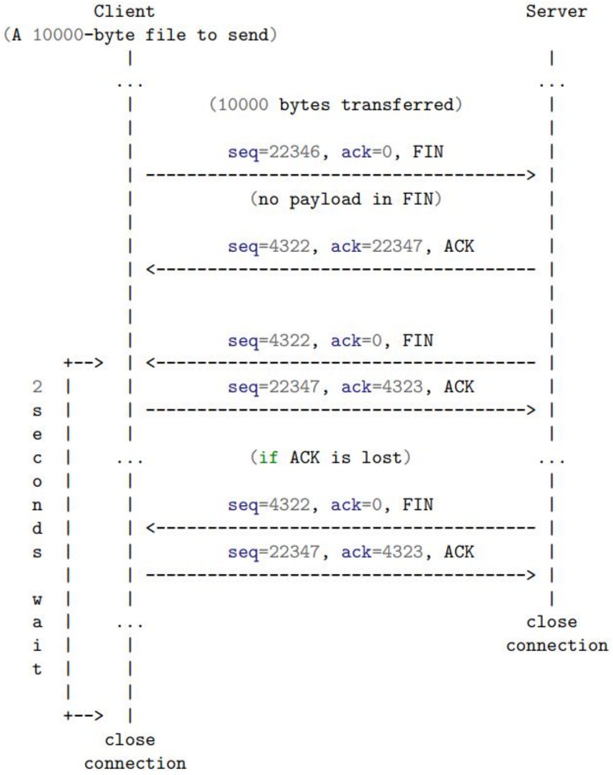
Stage 1: connection management

- Connection management



Random #

```
     Client1                                        Server
(A 500-byte file to send)
        |                                             |
        |        seq=12345, ack=0, SYN                |
        | ------------------------------------------> |
        |        seq=221, ack=12346, SYN, ACK         |
        | <------------------------------------------ |
        |                                             |
        |        seq=12346, ack=222, ACK              |
        | ------------------------------------------> |
        |      (if include 500-byte payload)          |
        |        seq=222, ack=12846, ACK              |
        | <------------------------------------------ |
        |                                             |
        |                                             |
        |    FIN procedures to close connection       |
        | <-----------------------------------------> |
        |          (saves file 1.file)                |
        |                                             |
      close                                           |
    connection                                        |
```



```
          Client                                        Server
 (A 10000-byte file to send)
              |                                            |
             ...                                          ...
              |        (10000 bytes transferred)           |
              |                                            |
              |        seq=22346, ack=0, FIN               |
              | -----------------------------------------> |
              |          (no payload in FIN)               |
              |                                            |
              |        seq=4322, ack=22347, ACK            |
              | <----------------------------------------- |
              |                                            |
              |                                            |
              |        seq=4322, ack=0, FIN                |
        +-->  | <----------------------------------------- |
   2    |     |        seq=22347, ack=4323, ACK            |
   s    |     | -----------------------------------------> |
   e    |     |                                            |
   c    |    ...         (if ACK is lost)                 ...
   o    |     |                                            |
   n    |     |        seq=4322, ack=0, FIN                |
   d    |     | <----------------------------------------- |
   s    |     |        seq=22347, ack=4323, ACK            |
        |     | -----------------------------------------> |
   w    |     |                                            |
   a    |    ...                                         close
   i    |     |                                        connection
   t    |     |
        |     |
        +-->  |
            close
          connection
```

Stage 1: connection management

- Packet header struct (12 bytes)
  - Needed fields: a Sequence Number field, an Acknowledgment Number field, and ACK, SYN , and FIN flags.
  - Example：
    - uint16_t to represent each field.
    - In total, 5*2 bytes are used. Then pad 2 byte of zeros.
  - Functions: printPacket(), htonHeader(), ntohHeader().

Stage 1: connection management

- Packet header struct (12 bytes)
- Example to construct a SYN packet
    - Header h1, then memset the struct
    - Set sequence number fields of h1 with a random number
    - Set SYN flag
    - Print header “SEND 12345 0 SYN”
- Example to parse a packet
    - Print header “RECV 4321 12346 SYN ACK”

Stage 1: connection management

- Client side logic
  - Send a packet with SYN to initiate the connection.
  - After receive packet with ACK, start send packets with data.
  - After transmitting the entire file, send FIN packet and wait for ACK.
  - After receive server FIN, send ACK and wait for 2 seconds to close the connection.

Note: always need to print out the header

Stage 1: connection management

- Server side logic
  - If a SYN packet, reply with packet with SYN flag and ACK flag, set ACK number field and sequence number field
  - If a data packet, write data field to file
  - If a FIN packet, reply with packet with ACK flag. Then send a packet with FIN flag. After receive ACK from client, close the connection.

Note: always need to print out the header

Stage 2: large file transmission and pipelining

- Pipelining
  - For client side, send 10 packets at the same time.
    - For every received ACK, send a new packet out. Keep the window at 10.
  - For server side, no much difference
- Large file transmission
  - Pay attention to sequence number (max = 25600)

Stage 3: reliable data transfer with packet loss

- Go-back-N is recommended
- For client side
    - Keep a timer, restart the timer for every sent packet.
    - If timeout, resend all packets in the window.
- For server side
    - Keep expected sequence number
    - Every time a data packet is received, check whether the sequence number is expected. If expected, write data field, otherwise drop it.

Sample output during packet loss

At sender side:

```
SEND 100 0
RECV 123 200 ACK
SEND 200 0
SEND 300 0
SEND 400 0
TIMEOUT 200
RESEND 200 0
RESEND 300 0
RESEND 400 0
```

Timer implementation

- Logic of the timer
  - setTimer(time period): Record the time out time point
  - Bool <- isTimeout(): Compare the current time with the time out time point. If smaller, return false; if equal or larger, return true
- How to get current time?
  - C++11 or higher
    - auto now =std::chrono::high_resolution_clock::now();
  - C
    - gettimeofday(&timeval, NULL)

Timer implementation

- How to compare time points
  - C++ 11 or higher: (**std::chrono::time_point** type)
    - if (now > timeout)
  - C: (**timeval** type)
    - double start = (double) s.tv_sec + (double) s.tv_usec/1000000;
    - double end = (double) e.tv_sec + (double) e.tv_usec/1000000;
    - If ((end - start) < 0.0)
- How to modify a time point
  - C++ 11 or higher: check +=,+,-,-= function of std::chrono::time_point
  - C: directly +=,+,-,-= on the timeval.tv_sec and timeval.tv_usec

Timer implementation

- You can use many other types from C or C++
    - E.g., you can simply use a large unsigned int uint32_t to keep POSIX timestamp to represent time
    - E.g., you can define your own time class/struct
- Tips
    - Time granularity is important (e.g., you shouldn’t use second granularity to measure timeout)
    - Keep checking time out in your loop

Example of check timer to transmit SYN

Pseudo code:

```
While (1) {
    sendSYN();
    setTimer(); // set a 0.5 second timer
    if (sent) {printHeader(resend=true)};
    else {printHeader(resend=false)};
    sent=true;
    while (1) {
        // receive ack packet and break the loop if received.
        // if timer expires, break the loop.
        if (isTimeout()) break;
    }
    // If ack is received, break the loop.
}
```

Hint: how to make socket non-blocking

We set a flag on a socket which marks that socket as non-blocking. This means that, when performing calls on that socket, if the call cannot complete, then instead it will fail with an error like EWOULDBLOCK or EAGAIN.

- fcntl(): change the mode of file descriptor
  - Example: fcntl(sockfd, F_SETFL, O_NONBLOCK);
  - Note: #include <fcntl.h>

Tc command

Example of set loss and check it:

```
➜  ~ sudo tc qdisc add dev lo root netem loss 10%
➜  ~ sudo tc qdisc show dev lo
qdisc netem 8001: root refcnt 2 limit 1000 loss 10%
➜  ~ ping localhost
PING localhost (127.0.0.1) 56(84) bytes of data.
64 bytes from localhost (127.0.0.1): icmp_seq=1 ttl=64 time=0.104 ms
64 bytes from localhost (127.0.0.1): icmp_seq=2 ttl=64 time=0.025 ms
..
64 bytes from localhost (127.0.0.1): icmp_seq=11 ttl=64 time=0.037 ms
64 bytes from localhost (127.0.0.1): icmp_seq=13 ttl=64 time=0.023 ms
...
^C
--- localhost ping statistics ---
22 packets transmitted, 19 received, 13% packet loss, time 20997ms
rtt min/avg/max/mdev = 0.018/0.031/0.104/0.020 ms
```