Lab 3 - Congestion Control (Tahoe)

Introduction to Computer Networks

Kuan-Wei Huang(黃冠維), Pei-Chieh Wu (吳沛潔), Cheng-Yuan Jian (簡呈原), Hsiang-Ting Huang (黃湘庭), Pham Ngoc Hoa (范玉花)

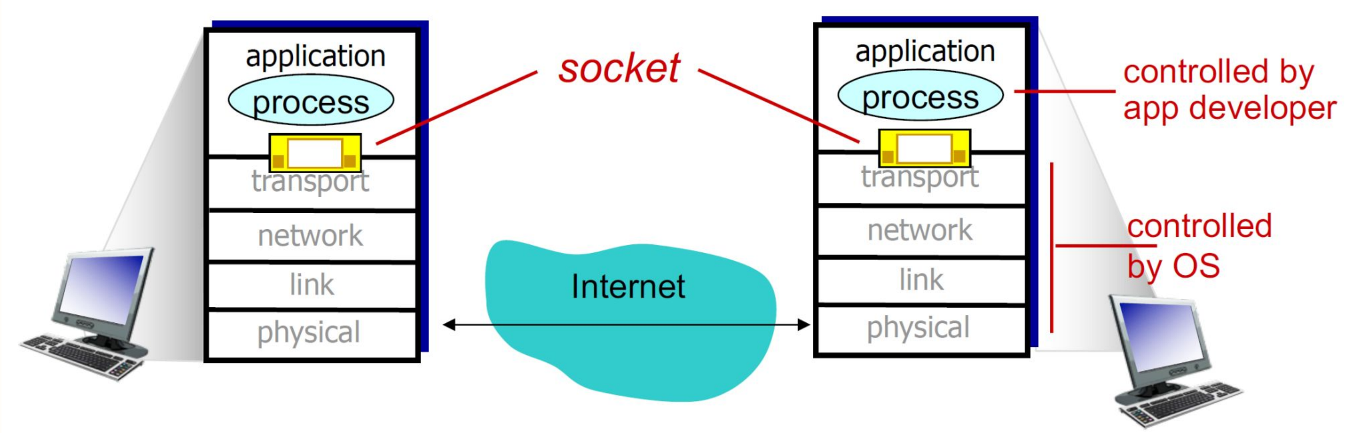
Purpose

- Learn what is Congestion Control
- Implement TCP congestion control with socket programming

What is Socket

Socket

- Socket: Door between application process and end-end-transport protocol
- Socket types for two transport services:
  - UDP: unreliable datagram
  - TCP: reliable, byte stream-oriented



What is Congestion?

Congestion

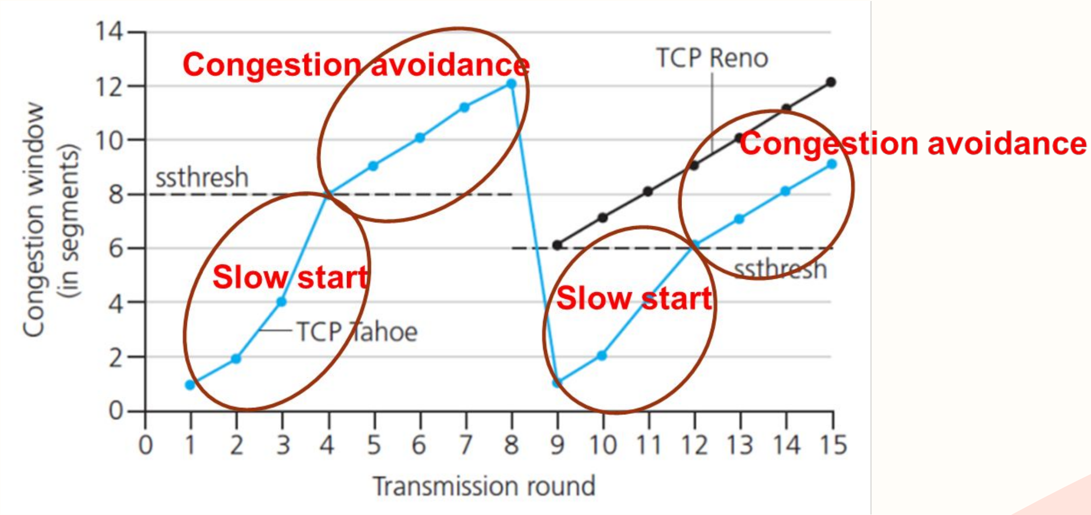
- When too many packets are sent simultaneously, the network can't handle them, resulting in congestion.
- Why can’t network handle them?
  - lost packets (buffer overflow at routers)

    (Our programming only considers packet loss.)
  - long delays (queueing in router buffers)

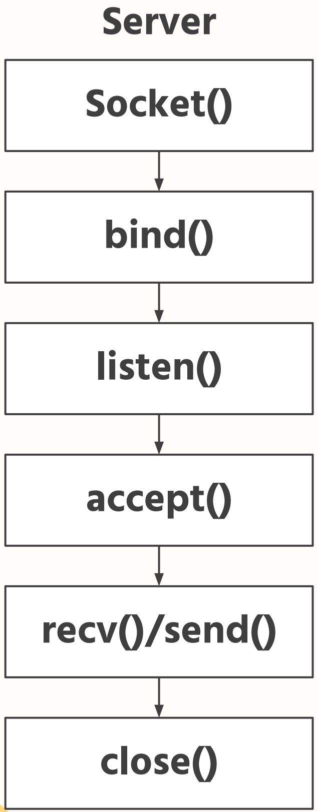
How Congestion Control works?

TCP Congestion Control

- TCP Tahoe (we will use Tahoe in this lab)
- TCP Reno

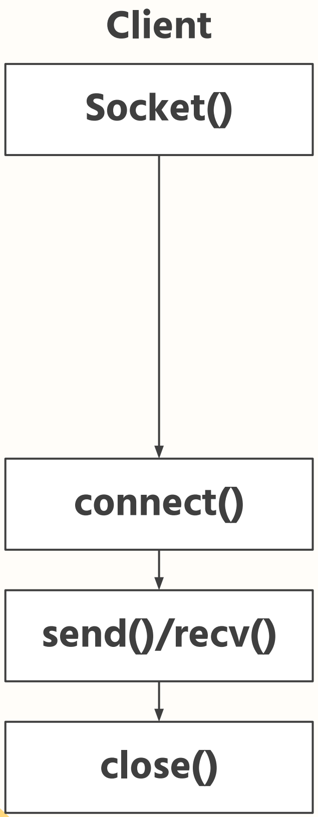


TCP flow



```
//Create TCP socket
int socket_fd = socket(PF_INET , SOCK_STREAM , 0);

//Bind socket to the address.
int bind(int sockfd, const struct sockaddr *addr, socklen_t addrlen);

//Listening the socket.
int listen(int sockfd, int backlog);

//Accept the connect request.
int accept(int sockfd, struct sockaddr *restrict addr, socklen_t *restrict addrlen);

//Send message to client
ssize_t send(int sockfd, const void *buf, size_t len, int flags);

//Close the connection
int close(int fd);
```

TCP flow



Client

Socket()

connect()

send()/recv()

close()

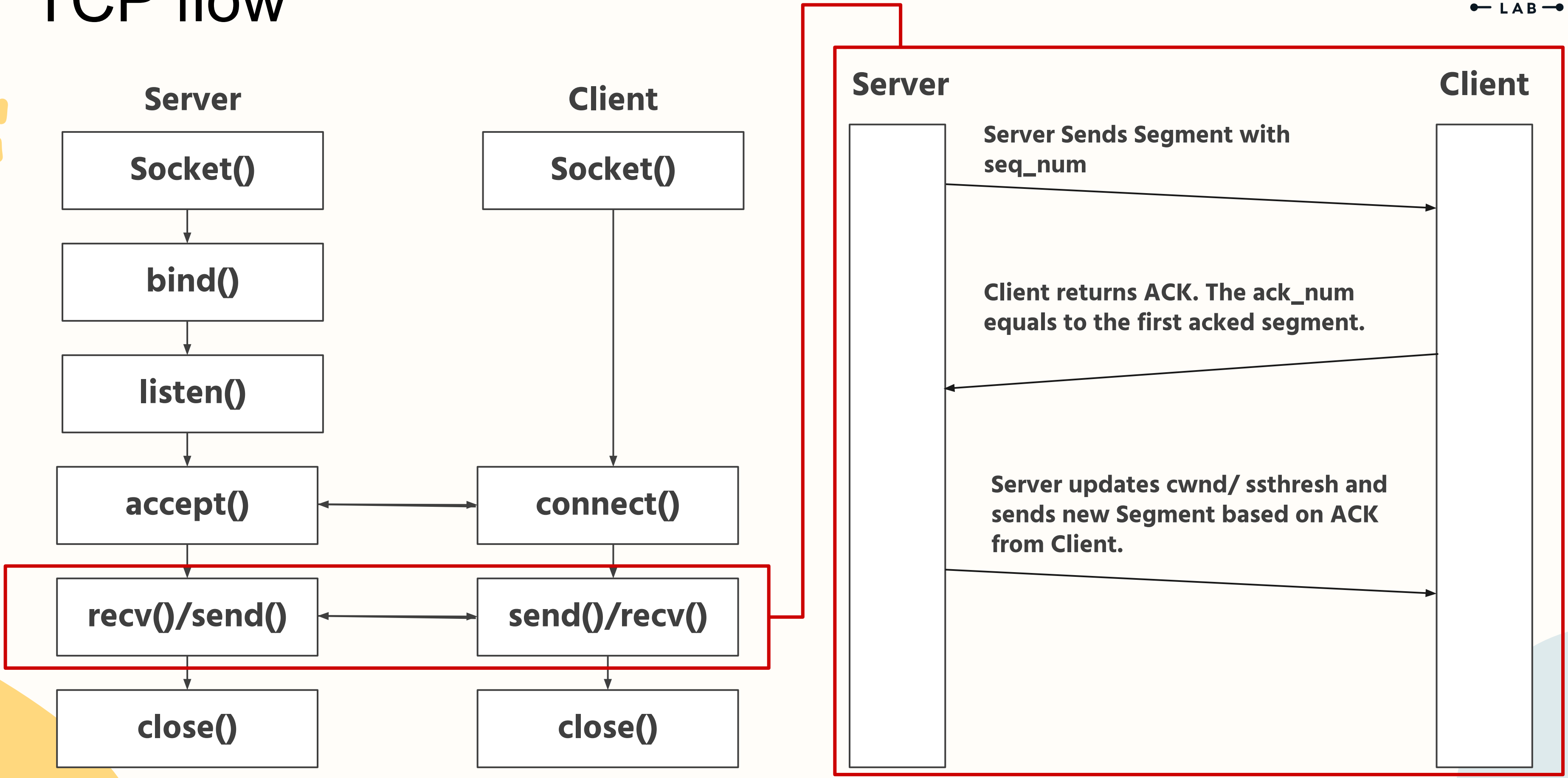
```
//Create TCP socket
int socket_fd = socket(PF_INET , SOCK_STREAM , 0);
```

```
//Accept the connect request.
int connect(int sockfd, const struct sockaddr *addr, socklen_t addrlen);
```

```
//Receive message from server
ssize_t recv(int sockfd, void *buf, size_t len, int flags);
```

```
//Close the connection
int close(int fd);
```

TCP flow



Server

Socket()

bind()

listen()

accept()

recv()/send()

close()

Client

Socket()

connect()

send()/recv()

close()

Server

Client

Server Sends Segment with seq_num

Client returns ACK. The ack_num equals to the first acked segment.

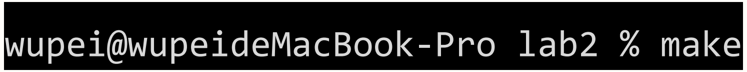
Server updates cwnd/ ssthresh and sends new Segment based on ACK from Client.

Assignment

Assignment

- In lab 3, you will each get a zip file containing :
  1. client.c
  2. server.c
  3. header.h
  4. header.c
  5. makefile

Makefile

- Same as lab2, you can compile your code using the command "make" under the lab2 folder
  - 

    ```
    wupei@wupeideMacBook-Pro lab2 % make
    ```

    - `make` #run Makefile to compile
    - `./server {sample_input.txt}` #run the server
      - E.g. `./server sample_input.txt` `: Use the sample_input to run the server`
    - `./client` #run the client
    - CTRL+C #exit server

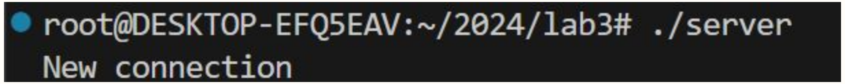
Assignment

- Implementation (70%)
  - Connect Server and Client with TCP socket and successfully send a message (15%)
  - Server sends a packet to client, and Client successfully receives the data. Client then returns an ACK to server. (15%)
  - Simulate packet loss and 3-duplicate ACK detection (20%)
  - Update cwnd / ssthresh and the state depend (20%)
- Report (30%)

Implementation (70%)

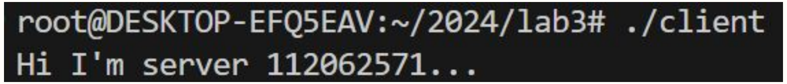
- Connect server and client with TCP socket and successfully send a message.(15%)

- Server Side



```
root@DESKTOP-EFQ5EAV:~/2024/lab3# ./server
New connection
```

- Client Side

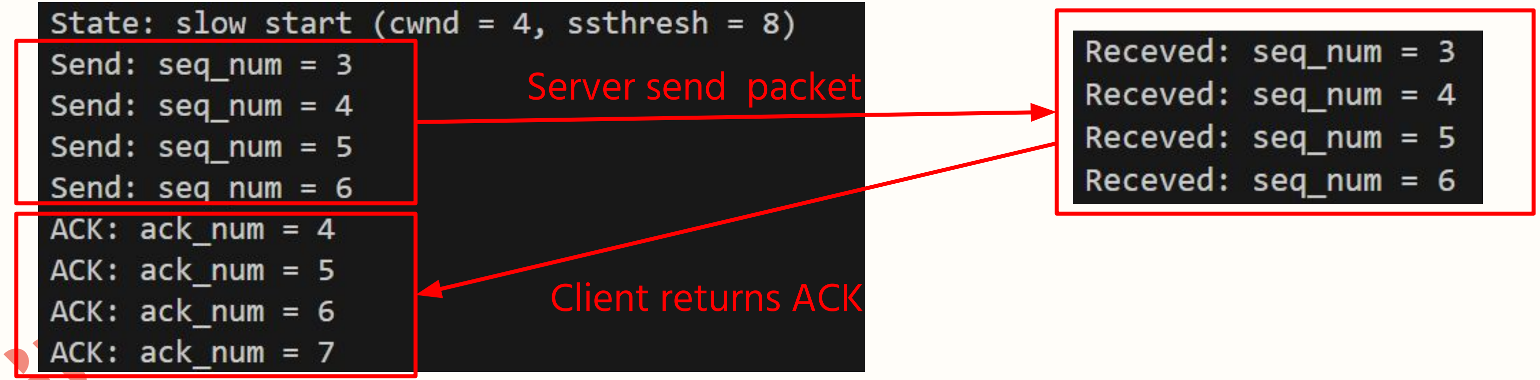


```
root@DESKTOP-EFQ5EAV:~/2024/lab3# ./client
Hi I'm server 112062571...
```

Implementation (70%)

- The server sends a packet to the client, and the client successfully receives the data. The client then returns an ACK to the server. (15%)

- Server Side
- Client Side



```
State: slow start (cwnd = 4, ssthresh = 8)
Send: seq_num = 3
Send: seq_num = 4
Send: seq_num = 5
Send: seq_num = 6
ACK: ack_num = 4
ACK: ack_num = 5
ACK: ack_num = 6
ACK: ack_num = 7
```

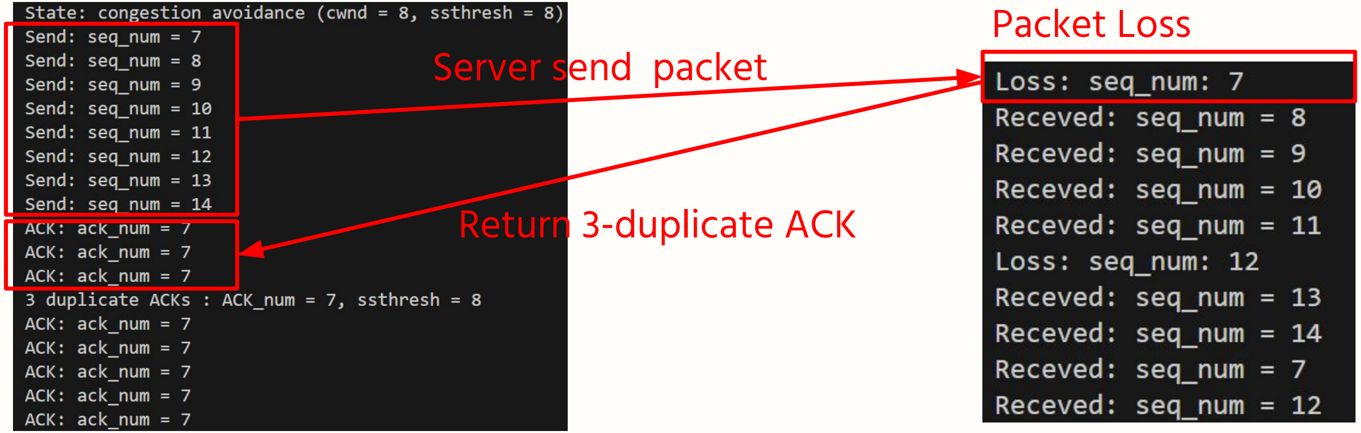
```
Receved: seq_num = 3
Receved: seq_num = 4
Receved: seq_num = 5
Receved: seq_num = 6
```

Implementation (70%)

- Simulate packet loss and 3-duplicate ACK detection (20%)

- Server Side

- Client Side



```
State: congestion avoidance (cwnd = 8, ssthresh = 8)
Send: seq_num = 7
Send: seq_num = 8
Send: seq_num = 9
Send: seq_num = 10
Send: seq_num = 11
Send: seq_num = 12
Send: seq_num = 13
Send: seq_num = 14
ACK: ack_num = 7
ACK: ack_num = 7
ACK: ack_num = 7
3 duplicate ACKs : ACK_num = 7, ssthresh = 8
ACK: ack_num = 7
ACK: ack_num = 7
ACK: ack_num = 7
ACK: ack_num = 7
ACK: ack_num = 7
```

Server send packet

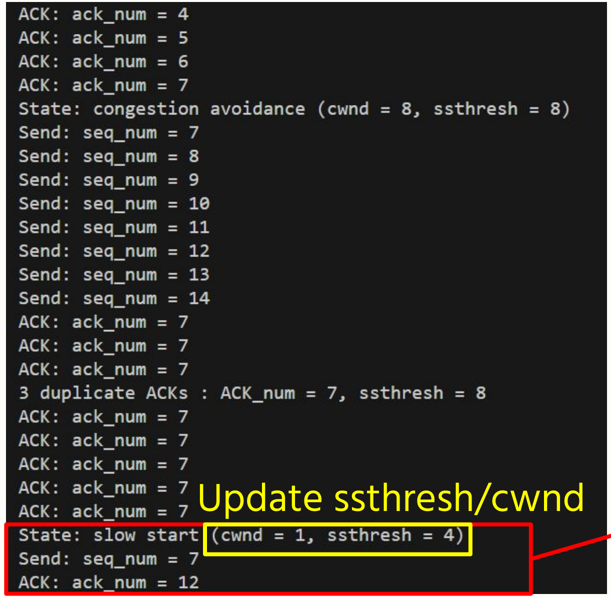
Return 3-duplicate ACK

Packet Loss

```
Loss: seq_num: 7
Receved: seq_num = 8
Receved: seq_num = 9
Receved: seq_num = 10
Receved: seq_num = 11
Loss: seq_num: 12
Receved: seq_num = 13
Receved: seq_num = 14
Receved: seq_num = 7
Receved: seq_num = 12
```

Implementation (70%)

- Update ssthresh/ cwnd and retransmit packet (20%)
- Server Side

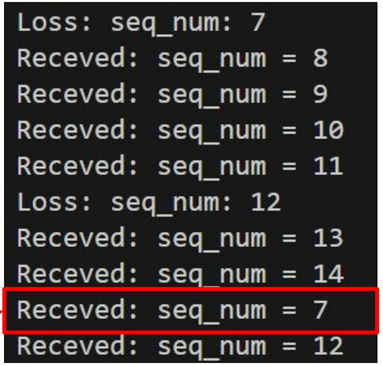


```
ACK: ack_num = 4
ACK: ack_num = 5
ACK: ack_num = 6
ACK: ack_num = 7
State: congestion avoidance (cwnd = 8, ssthresh = 8)
Send: seq_num = 7
Send: seq_num = 8
Send: seq_num = 9
Send: seq_num = 10
Send: seq_num = 11
Send: seq_num = 12
Send: seq_num = 13
Send: seq_num = 14
ACK: ack_num = 7
ACK: ack_num = 7
ACK: ack_num = 7
3 duplicate ACKs : ACK_num = 7, ssthresh = 8
ACK: ack_num = 7
ACK: ack_num = 7
ACK: ack_num = 7
ACK: ack_num = 7
ACK: ack_num = 7
State: slow start (cwnd = 1, ssthresh = 4)
Send: seq_num = 7
ACK: ack_num = 12
```

Update ssthresh/cwnd

Retransmit loss packet

- Client Side



```
Loss: seq_num: 7
Receved: seq_num = 8
Receved: seq_num = 9
Receved: seq_num = 10
Receved: seq_num = 11
Loss: seq_num: 12
Receved: seq_num = 13
Receved: seq_num = 14
Receved: seq_num = 7
Receved: seq_num = 12
```

Receive retransmitted packet

Report (30%)

- Briefly explain your code. Please do not copy and paste the code directly.
  - How does the server send the packet with the correct sequence number to client?
  - How to simulate packet loss
  - How to detect 3-duplicate ACKs
  - How to update cwnd and ssthresh
- Screenshot compiled results and explain where packet loss and retransmission occurred.

Name the report file as: report.pdf

Requirement

- Compress all files into one.
    - client.c
    - server.c
    - header.h
    - header.c
    - makefile
    - report.pdf
- Name the file Lab3_{studentID}.zip
    - (e.g. Lab3_112062571.zip)
- Upload to eeclass before **June 13th**.

- We will run your file
    - `make`
    - `./server`
    - `./client`

Penalty

- **Plagiarism will get 0 point**
- Late submission before **June 20th** only get **70%** of the original score.
- Late submission after **June 20th** will **not** be accepted