Name_____ Student ID_____ Department/Year_____

3rd Examination

Introduction to Computer Networks (Hybrid)
Class#: EE 4020, Class-ID: 901E31110
Spring 2021

13:10-14:00 Thursday
May 13, 2021

Cautions

1. There are in total 100 points to earn. You have 50 minutes to answer the questions. Skim through all questions and start from the questions you are more confident with.
2. Use only English to answer the questions. Misspelling and grammar errors will be tolerated, but you want to make sure with these errors your answers will still make sense.

1. (ch31, 10pt) Recall the functionalities supported by the transport layer. For each of the following statements, tell whether it is true or false?
   (1) Delay guarantee is one of the supported functionalities. (1pt)
   (2) Multiplexing and demultiplexing is one of the supported functionalities. (1pt)
   (3) Error detection is one of the supported functionalities. (1pt)
   (4) Reliable data transfer is one of the supported functionalities. (1pt)
   (5) Congestion control is one of the supported functionalities. (1pt)
   (6) Delay guarantee is a functionality provided only by UDP. (1pt)
   (7) Multiplexing and demultiplexing is a functionality provided only by UDP. (1pt)
   (8) Error detection is a functionality provided only by UDP. (1pt)
   (9) Reliable data transfer is a functionality provided only by TCP. (1pt)
   (10) Congestion control is a functionality provided only by TCP. (1pt)

Sample Solution:

(1) False

(2)-(5) True

(6)-(8) False

(9)-(10) True

2. (ch33, 6pt) Internet checksum is based on computing the 1's complement sum. Let's denote the function of 1's complement sum as F. Assume the simpler case of the Internet checksum applying F to two 4-bit numbers, S1 and S2.
   (1) The data sender will write the checksum value in the packet header before transmitting the packet. Tell the checksum F(S1, S2) for S1 = (1 1 1 1) and S2 = (0 0 0 0). (1pt)
   (2) Suppose there is 1 bit error during the transmission, such that the receiver sees S1 = (1 1 1 1) and S2 = (0 0 0 1). Will the receiver detect the error? (1pt)
   (3) Suppose there is 1 bit error. One in S1 and one in S2, such that the receiver sees S1 = (1 1 1 1) and S2 = (0 0 1 0). Will the receiver detect the error? (1pt)
   (4) Try a few more cases of having just 1 bit error. Will the receiver be able to detect all 1 bit error cases, yes or no? (1pt) And why or why not? (2pt)

Sample Solution:

(1) (0 0 0 0)
(2) Yes
(3) Yes
(4) Yes. There are 8 possibilities. In each case, the sum (S1+S2) will change. If the sum is different, the complement is of course also different. The Internet checksum will always detect cases with only 1 bit error.

3. (ch33, 12pt) Continue from Problem Set 2 and consider the cases of having 2 bit errors in either one or both of S1 and S2.
   (1) Suppose both bit errors are in S1, such that the receiver sees S1 = (1 1 0 0) and S2 = (0 0 0 0). Will the receiver detect the error? (1pt)
   (2) Suppose both bit errors are in S1, such that the receiver sees S1 = (1 1 1 1) and S2 = (0 0 1 1). Will the receiver detect the error? (1pt)
   (3) Suppose 1 bit error is in S1 and the other in S2, such that the receiver sees S1 = (0 1 1 1) and S2 = (0 0 0 1). Will the receiver detect the error? (1pt)
   (4) Suppose 1 bit error is in S1 and the other in S2, such that the receiver sees S1 = (1 1 1 0) and S2 = (0 0 0 1). Will the receiver detect the error? (1pt)
   (5) Among all 2 bit error patterns, i.e., $\binom{8}{2}$, how many will not be detected by the Internet checksum? (2pt) And why? (2pt)
   (6) Among all bit error patterns, how many will not be detected by the Internet checksum? (2pt) And why? (2pt)

Sample Solution:

(1) Yes
(2) Yes
(3) Yes
(4) No
(5) 4. When the bit error positions are the same, the receiver will not be able to detect the error. That is 4/28 chance the 2 bit error cases will slip.

    For the particular data set S1=(1 1 1 1) and S2=(0 0 0 0), flipping the bits at the same position in S1 and S2 is equivalent of subtracting an amount to S1 and adding the same amount to S2. The sum will remain the same.

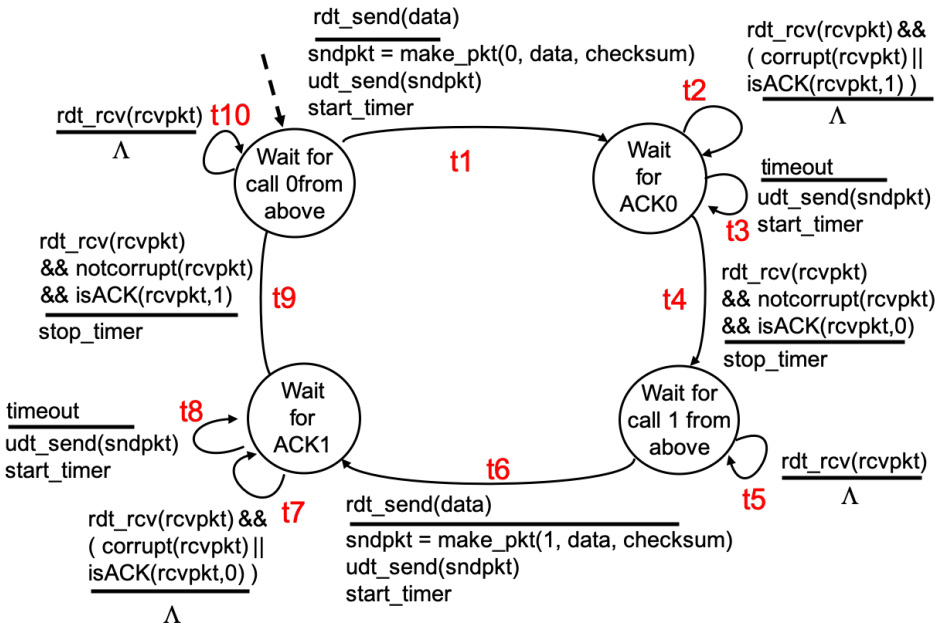
(6) 15. The # of patterns of S1 and S2 are 2^8. Thinking along the line that S1 and S2 are binary numbers of 4 bits. The # of patterns of S1 and S2 are 16^2, too. Bit flips are essentially S1 or S2 being added (modulo 16) by an amount. The amount of the patterns being different from the original is 16^2-1. Among these combinations, 15 of them will slip. They are the cases where S1 is added (modulo 16) by an amount, and S2 happens to be subtracted (modulo 16) by the same amount. The sum will be the same as the original S1+S2.

    Note though the "probability of bit error not being caught by the Internet checksum" is not quite 15/(16^2-1) as the probability of being added/subtracted by 1 vs. 2, 3…. are all different. Think about having 1 bit error vs. 2 bit errors. The chance

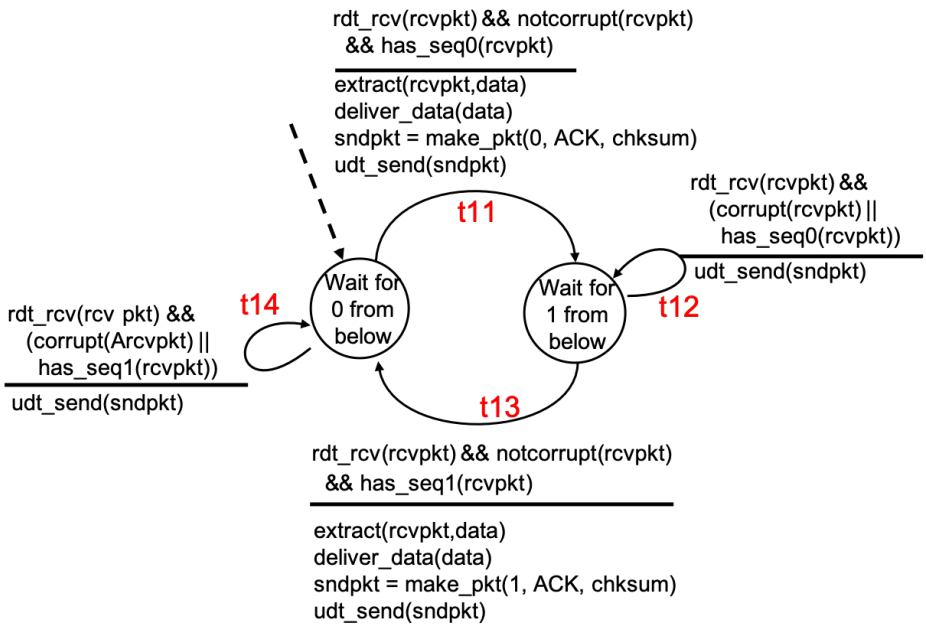
of latter is lower. But you see the light deriving the “probability of bit error not being caught by the Internet checksum”.

4. (ch34, 6pt) Provided below are the FSMs of rdt 3.0 sender and receiver. Indicate the order of the transitions (in terms of t1, t2, …, t14) taking place until the sender and receiver stabilize for the following scenarios.

rdt 3.0 sender:



rdt 3.0 receiver:



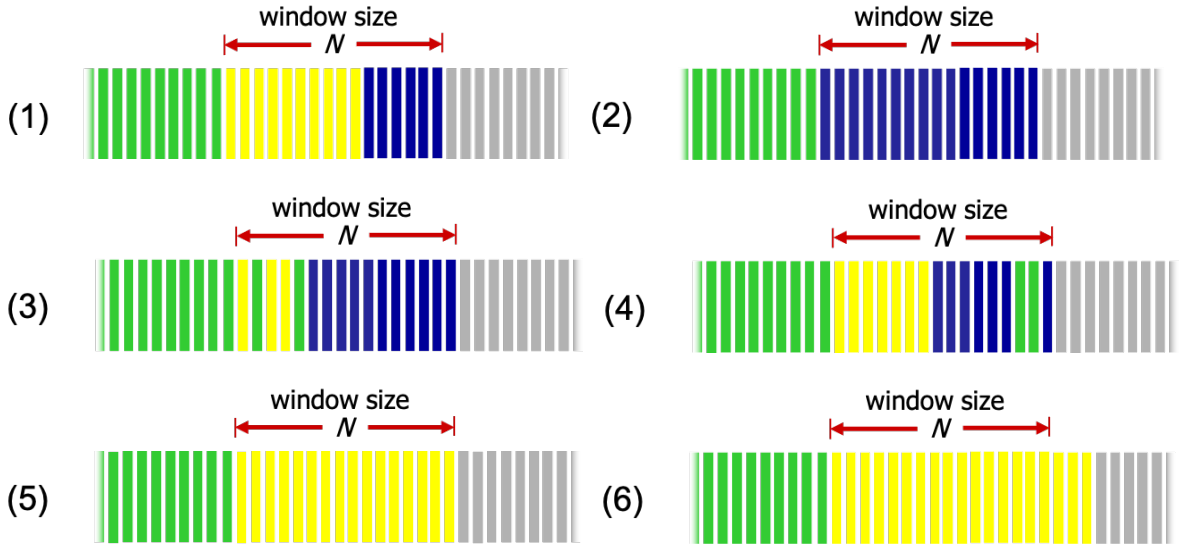
(1) Scenario 1: Both the sender and receiver start from the initial state. The sender gets a call from above to send just 1 data packet. There is a bit error in the ACK 0 packet, but all subsequent packet transmissions are fine, i.e., no bit error, no packet loss afterwards. (1%)
(2) Scenario 2: Continue from Scenario 1. The sender gets another call from above to send just 1 data packet. There is a bit error in the data packet going to the receiver but all subsequent packet transmissions are fine. (1%)
(3) Scenario 3: Both sender and receiver start from the initial state. The sender gets another call from above to send just 1 data packet. The ACK 0 packet does not arrive back at the sender while all other packet transmissions are fine. (1%)
(4) Scenario 4: Continue from Scenario 3. The sender gets a call from above to send just 1 data packet. The data packet does not arrive at the receiver but all subsequent packet transmissions are fine. (1%)
(5) One can extend the t2 in rdt 3.0 sender such that when the ACK packet is corrupted or is a duplicate (i.e., a NAK), the sender retransmits the data packet (instead of doing nothing). Discuss the benefits and drawbacks of the extension. (2%)

Sample Solution:

(1) t1, t11, t2, t3, t12, t4
(2) t6, t12, t7, t8, t13, t9
(3) t1, t11, t3, t12, t4
(4) t6, t8, t13, t9
(5) benefit – lower retransmission delay and therefore higher data throughput
drawback – multiple copies of the packet transmitted throughout the rest of the connection. Therefore, high network bandwidth consumption.

5. (ch34, 6pt) Recall the color scheme of the packets at the sender side of Go-Back-N (GBN) and Selective Repeat (SR) protocols.

Tell whether each of the 6 sender-side packet lineups below is from (a) only a GBN sender, (b) only an SR sender, (c) possibly both, or (d) neither of the two. (1pt each)

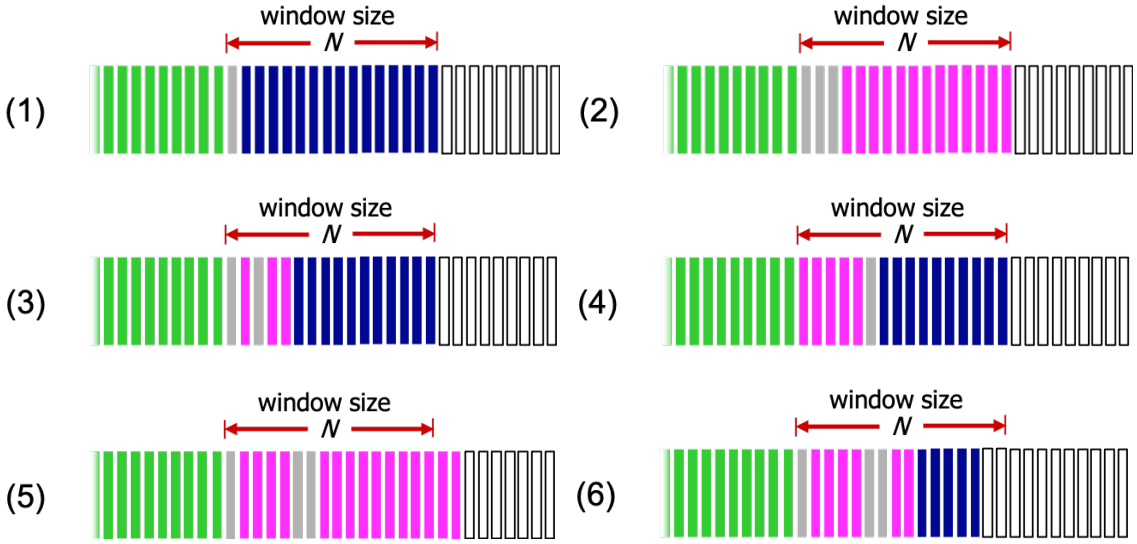


Sample Solution:

(1) (c)
(2) (c)
(3) (b)
(4) (d)
(5) (c)
(6) (d)

6. (ch34, 6pt) Recall the color scheme of the packets at the receiver side of Go-Back-N (GBN) and Selective Repeat (SR) protocols.

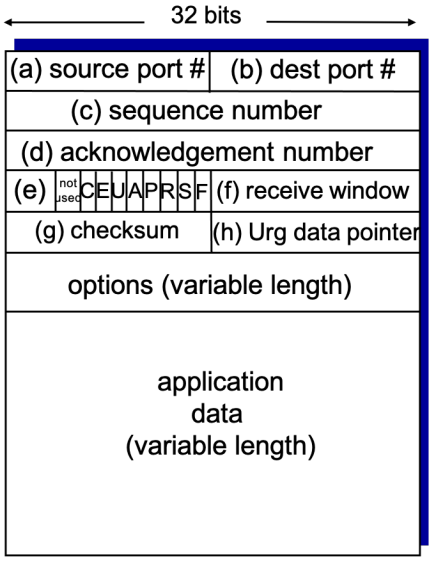
Tell whether each of the 6 receiver-side packet lineups below is from (a) only a GBN sender, (b) only a SR sender, (c) possibly both, or (d) neither of the two. (1pt each)



Sample Solution:

(1) (c)
(2) (b)
(3) (b)
(4) (d)
(5) (d)
(6) (d)

7. (ch35, 5pt) Shown below is the TCP packet format. Tell which of the fields ((a), (b), …, (h)) and the special bits (C, E, …, F) are used for each of the 5 functionalities below.

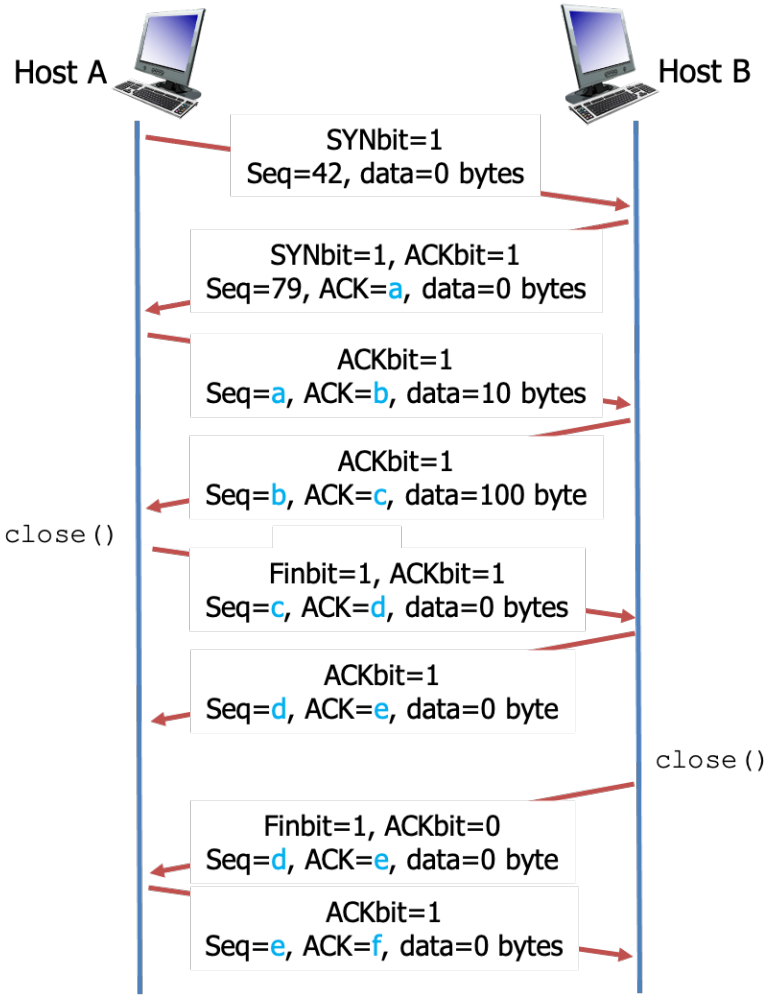


(1) Flow control. (1pt)
(2) 3-way handshake. (1pt)
(3) Demultiplexing. (1pt)
(4) Bit error detection. (1pt)
(5) Explicit congestion notification. (1pt)

Sample Solution:

(1) (f) receive window
(2) S and A bit (S bit only also OK)
(3) (a) Source and (b) destination port number. (and source and destination IP addresses but these are not in the transport layer header).
(4) (g) checksum
(5) C and E bit

8. (ch35, 5pt) Here is a short TCP connection from the beginning to the end. First, we see the 3-way handshake. In the last ACK of the 3-way handshake, the client (Host A) sends 10 bytes of data. In the ACK packet back to the client, the server (Host B) sends 100 bytes of data. The client, after receiving the ACK for the 10-byte data and the 100-byte data, closes the connection in the meantime sending the ACK for the 100-byte data. After a little while, the server closes the connection as well. Tell the value of a, b, c, e, f. (1pt each)



Sample Solution:

a=43, b=80, c=53, e=54, f=181

9. (ch35, 5pt) In TCP, to find the proper timeout interval for retransmission, we estimate the average round-trip time (RTT) and the average RTT deviation using the following formula. The formula basically computes a new average (A_{n+1}) by taking the weighted (α) sum of the current average (A_n) and the current sample (S_n), where $0 < \alpha < 1$.

$$A_{n+1} = (1-\alpha)\, A_n + (\alpha)\, S_n$$

One can expand A_n in terms of A_{n-1} and S_{n-1}.

$$A_n = (1-\alpha)\, A_{n-1} + (\alpha)\, S_{n-1}$$

Substitute A_n in the original formula of A_{n+1}.

$$\begin{aligned} A_{n+1} &= (1-\alpha)\, \{(1-\alpha)\, A_{n-1} + (\alpha)\, S_{n-1}\} + (\alpha)\, S_n \\ &= (1-\alpha)^2\, A_{n-1} + (1-\alpha)\, (\alpha)\, S_{n-1} + (\alpha)\, S_n \end{aligned}$$

One can repeat the process by expanding A_{n-1}. Assume $A_1 = 0$. The full expansion is as follows:

$$A_{n+1} = (1-\alpha)^{n-1}(\alpha)S_1 + (1-\alpha)^{n-2}(\alpha)S_2 + \ldots + (1-\alpha)^2(\alpha)S_{n-2} + (1-\alpha)(\alpha)S_{n-1} + (\alpha)S_n$$

One can see that an older RTT sample S_i is weighted off exponentially fast $(1-\alpha)^{(n-i)}$. The method is therefore called exponentially weighted moving average. Let's be cautious though. In order to call A_{n+1} a weighted average of S_i, i=1…n, the weights need to add up to 1. Show the weights of S_i, i=1…n will add to 1, when n approaches ∞.

Sample Solution:

$(1-\alpha)^{n-1}(\alpha) + (1-\alpha)^{n-2}(\alpha) + \ldots + (1-\alpha)^2(\alpha) + (1-\alpha)(\alpha) + (\alpha)$

$= \alpha \{ \underline{1+(1-\alpha)+ (1-\alpha)^2+\ldots+(1-\alpha)^{n-1}} \}$ <- the term in {} is a geometric series

$= \alpha \{[1-(1-\alpha)^n] / [1-(1-\alpha)]\}$ <- sum of a geometric series

$= \alpha\, (1/\alpha) = 1$ <- when n -> ∞

10. (ch37, 17pt) Below is the pseudo-code describing the reliable data transfer part of the TCP sender. The mechanism to adjust the congestion window size (cwnd) is missing. Please identify the code block where the missing functionalities shall be added and modify the code block accordingly.

```
1   NextSeqNum = InitialSeqNum; SendBase = InitialSeqNum
2   cwnd = InitialCwnd; ssthresh = InitialSsthresh
3   loop (forever) {
4       switch(event)
5       event: data received from application above
6           If (NextSeqNum + length(data) <= SendBase + cwnd) {
7               create TCP segment with sequence number NextSeqNum
8               pass segment to IP
9               NextSeqNum = NextSeqNum + length(data)
10              if (timer currently not running)
11                  start timer
12          } else
13              refuse data
14      event: timer timeout
15          retransmit from the smallest, not-yet-acknowledged segment
16          start timer
17      event: ACK received, with ACK field value of y
18          if (y > SendBase) {
19              SendBase = y
20              if (there are currently not-yet-acknowledged segments)
21                  (re)start timer
22              else
23                  cancel timer
24          } else {
25              increment count of dup ACKs received for y
26              if (count of dup ACKs received for y == 3) {
27                  resend segment with sequence number y
28                  count of dup ACKs received for y = 0
29              }
30          } /* end of if (y > SendBase)*/
31  }   /* end of loop forever */
```

A: lines 7–11
B: line 13
C: lines 15–16
D: lines 19–23
E: lines 25–29

(1) Where to add the code to reduce the congestion window size to 1? (1pt) And how should the code block(s) be modified? (1pt)
(2) Where to add the code to reduce the window size to half? (1pt) And how should the code block(s) be modified? (1pt)
(3) Where to add the code to set the ssthresh? (1pt) And how should the code block(s) be modified? (2pt)
(4) Where to add the code to increase the cwnd? (1pt) And how should the code block(s) be modified? (2pt)
(5) Where to add the code to implement fast recovery? (2pt) And how should the code block be modified? (5pt)

Sample Solution:

(1) C

```
        cwnd=1
15      retransmit from the smallest, not-yet-acknowledged segment
16      start timer
```

(2) E

```
25      increment count of dup ACKs received for y
26      if (count of dup ACKs received for y == 3) {
                cwnd=cwnd/2
27              resend segment with sequence number y
28              count of dup ACKs received for y = 0
29      }
```

(3) C and E

C block

```
        ssthresh=cwnd/2
        cwnd=1
15      retransmit from the smallest, not-yet-acknowledged segment
16      start timer
```

E block

```
25      increment count of dup ACKs received for y
26      if (count of dup ACKs received for y == 3) {
                ssthresh=cwnd/2
                cwnd=ssthresh
```

```
27              resend segment with sequence number y
28              count of dup ACKs received for y = 0
29      }
```

(4) D

```
19      SendBase = y
        If (cwnd <= ssthresh) {
                cwnd = cwnd+MSS
        } else {
                cwnd = cwnd+MSS(MSS/cwnd)
        }
20      if (there are currently not-yet-acknowledged segments)
21              (re)start timer
22      else
23              cancel timer
```

(5) C, D and E

C block

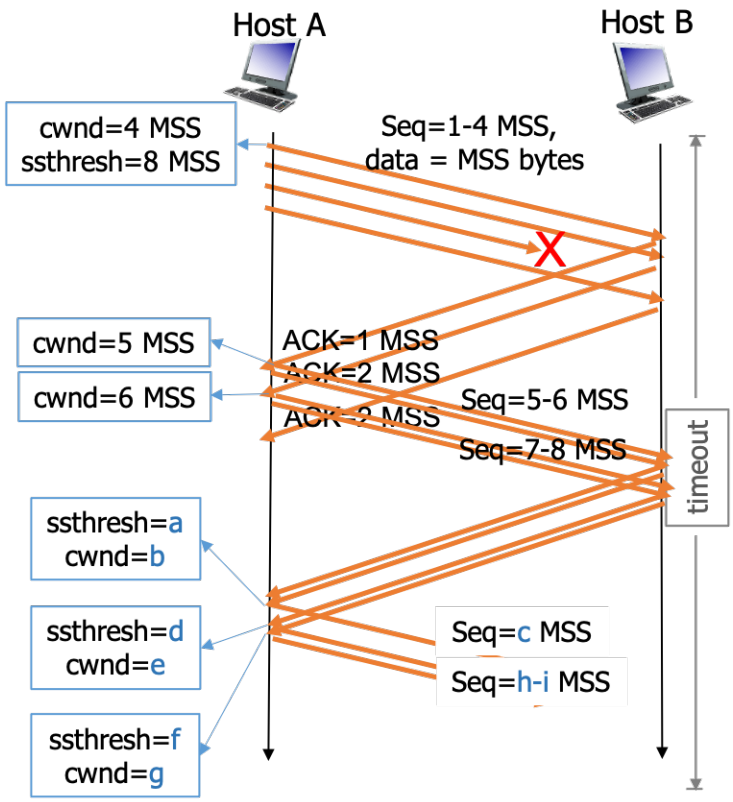
```
15              retransmit from the smallest, not-yet-acknowledged segment
16              start timer
-->             count of dup Acks received for the oldest unack pkt =0
```

D and E block

```
if (y > SendBase) {
        SendBase = y
        If (count of dup ACKs received for y >= 3) {
                cwnd=ssthresh
                count of dup ACKs received for y = 0
        } else {
                If (cwnd <= ssthresh) {
                        cwnd = cwnd+MSS
                 } else {
                        cwnd = cwnd+MSS(MSS/cwnd)
                 }
        }
        if (there are currently not-yet-acknowledged segments)
                (re)start timer
        else
```

```
            cancel timer
    } else {
        increment count of dup ACKs received for y
        if (count of dup ACKs received for y == 3) {
            ssthresh=cwnd/2
            cwnd=ssthresh + 3MSS
            resend segment with sequence number y
        }
        If (count of dup ACKs received for y > 3) {
            cwnd=cwnd + MSS
            transmit segments as allowed
        }
    } /* end of if (y > SendBase)*/
```

11. (ch37, 5pt) This is TCP with fast recovery. The scenario is as follows. Assume every packet is strickly 1 MSS large. Tell the value of a, b, c, e, and h. (1pt each)



Sample Solution:

a=3 MSS, ssthresh=cwnd/2

b=6 MSS, cwnd=ssthresh+3MSS

c=3, retransmit packet seq 3MSS

e=7 MSS, cwnd=cwnd+MSS (in fast recovery state and receiving another duplicate ack)

h=9, transmit packet seq 9MSS now the cwnd is increased by 1 MSS

12. (PA, 7pt) Consider the following Go program: server-exam3.go. Execute the server-exam3.go first. Then start two extra terminals. Execute client-102.go on the two terminals back-to-back.

server-exam3.go

```
package main

import "fmt"
import "bufio"
import "net"
import "time"

func check(e error) {
    if e != nil {
        panic(e)
    }
}

func handleConnection (c net.Conn, i int) {
    reader := bufio.NewReader(c)
    message, errr := reader.ReadString('\n')
    check(errr)
    fmt.Printf("%s", message)
    j := 0
    for (j < 10) {
        time.Sleep(1 * time.Second)
        fmt.Printf("client #%d processing: %d%% \n", i, 10*j)
        j++
    }
    time.Sleep(10 * time.Second)

    writer := bufio.NewWriter(c)
    newline := fmt.Sprintf("%d bytes received\n", len(message))
    _, errw := writer.WriteString(newline)
    check(errw)
    writer.Flush()
    c.Close()
```

```
}

func main() {
   fmt.Println("Launching server...")
   ln, _ := net.Listen("tcp", ":<your port#>")
   defer ln.Close()

   i := 1
   for {
      conn, _ := ln.Accept()

      fmt.Printf("%d ", i)
      handleConnection(conn, i)
      i++
   }
}
```

client-102.go

```
package main

import "fmt"
import "bufio"
import "net"


func check(e error) {
   if e != nil {
      panic(e)
   }
}

func main() {
   conn, errc := net.Dial("tcp", "127.0.0.1:<your port#>")
   check(errc)
   defer conn.Close()

   writer := bufio.NewWriter(conn)
```

```
    len, errw := writer.WriteString("Hello World!\n")
    check(errw)
    fmt.Printf("Send a string of %d bytes\n", len)
    writer.Flush()

    reader := bufio.NewReader(conn)
    message, errr := reader.ReadString('\n')
    check(errr)
    fmt.Printf("Server replies: %s", message)
}
```

(1) Tell the output on screen of server-exam3.go. (2%)

(2) Modify server-exam3.go so that `handleConnection(conn, i)` in `main()` is a goroutine. I.e., the line becomes `go handleConnection(conn, i)`. Tell the output on screen running the modified server-exam3.go. (2%).

(3) Are the two outputs the same? (1%)

(4) Continue from (3). Explain why the outputs are the same or different. (2%).

Sample Solution:

(1) Launching server...

1 Hello World!
client #1 processing: 0%
client #1 processing: 10%
client #1 processing: 20%
client #1 processing: 30%
client #1 processing: 40%
client #1 processing: 50%
client #1 processing: 60%
client #1 processing: 70%
client #1 processing: 80%
client #1 processing: 90%
2 Hello World!
client #2 processing: 0%
client #2 processing: 10%
client #2 processing: 20%
client #2 processing: 30%
client #2 processing: 40%
client #2 processing: 50%

client #2 processing: 60%
client #2 processing: 70%
client #2 processing: 80%
client #2 processing: 90%

(2) Something like the following with client #1 and #2's requests processed overlapping each other.

Launching server...
1 Hello World!
client #1 processing: 0%
client #1 processing: 10%
client #1 processing: 20%
client #1 processing: 30%
client #1 processing: 40%
2 Hello World!
client #1 processing: 50%
client #2 processing: 0%
client #1 processing: 60%
client #2 processing: 10%
client #1 processing: 70%
client #2 processing: 20%
client #1 processing: 80%
client #2 processing: 30%
client #1 processing: 90%
client #2 processing: 40%
client #2 processing: 50%
client #2 processing: 60%
client #2 processing: 70%
client #2 processing: 80%
client #2 processing: 90%

(3) Yes

(4) The former (non-goroutine) will complete client #1's request before starting client #2's. The latter (goroutine) allows the two requests to progress concurrently.

13. (PA, 10pt) Polly is running many servers on port 20001 to 21000 of the PA server. Among all these servers, one is special, called the bingo server. The end goal is to scan the ports between 20001 and 21000 and find the port number of the bingo server as quick as possible.

    Each of these servers takes a connection request and returns a simple string "`low\n`", "`bingo\n`", or "`high\n`". The bingo server returns "`bingo\n`". Those running on port numbers lower than the bingo server returns "`low\n`". Those running on port numbers higher than the bingo server returns "`high\n`".

    Now go on to the PA server and log in with the team's username and password. Create a subdirectory `exam3-<student ID>`. Go to the subdirectory and work on the following.

    (1) Create client-e3-1.go such that it is able to probe the server running on port 20001 and print the message from the server. (3pt)
    (2) Create client-e3-2.go such that it is able to probe through the servers on port 20001 to 21000 sequentially, stop at the bingo server, and tell the port number of the bingo server. (3pt)
    (3) Create client-e3-3.go such that it is able to binary search, find the bingo server within $\log_2 1000$ probes (i.e., 10 probes max), and tell the number of probes attempted. (4pt)

Sample Solution:

Whatever works