第三章-2

P27. 主机 A 和 B 经一条 TCP 连接通信，并且主机 B 已经收到了来自 A 的最长为 126 字节的所有字节。假定主机 A 随后向主机 B 发送两个紧接着的报文段。第一个和第二个报文段分别包含了 80 字节和 40 字节的数据。在第一个报文段中，序号是 127，源端口号是 302，目的地端口号是 80。无论何时主机 B 接收到来自主机 A 的报文段，它都会发送确认。

a. 在从主机 A 发往 B 的第二个报文段中，序号、源端口号和目的端口号各是什么？

b. 如果第一个报文段在第二个报文段之前到达，在第一个到达报文段的确认中，确认号、源端口号和目的端口号各是什么？

c. 如果第二个报文段在第一个报文段之前到达，在第一个到达报文段的确认中，确认号是什么？

d. 假定由 A 发送的两个报文段按序到达 B。第一个确认丢失了而第二个确认在第一个超时间隔之后到达。画出时序图，显示这些报文段和发送的所有其他报文段和确认。（假设没有其他分组丢失。）对于图上每个报文段，标出序号和数据的字节数量；对于你增加的每个应答，标出确认号。

a. 序号 207 源端口号 302 目的端口号 80
b. 确认号 207 源端口号 80 目的端口号 302
c. 确认号 127
d. tbc
   a -1> b
   a -2> b
   a -1> b
   a <2- b
   a <1- b

P32. 考虑 TCP 估计 RTT 的过程。假设 $\alpha = 0.1$，令 SampleRT_{T1} 设置为最新样本 RTT，令 SampleRT_{T2} 设置为下一个最新样本 RTT，等等。

a. 对于一个给定的 TCP 连接，假定 4 个确认报文相继到达，带有 4 个对应的 RTT 值：SampleRT_{T4}、SampleRT_{T3}、SampleRT_{T2} 和 SampleRT_{T1}。根据这 4 个样本 RTT 表示 EstimatedRTT。

b. 将你得到的公式一般化到 n 个 RTT 样本的情况。

c. 对于在（b）中得到的公式，令 n 趋于无穷。试说明为什么这个平均过程被称为指数移动平均。

我们有公式：$EstimatedRTT_{i+1} = (1-\alpha) \cdot EstimatedRTT_i + \alpha \cdot SampleRTT$

a.

$EstimatedRTT_1 = SampleRTT_{T_4}$

$EstimatedRTT_2$
$= 0.9 \cdot EstimatedRTT_1 + 0.1 \cdot SampleRTT_{T_3}$
$= 0.9 \cdot SampleRTT_{T_4} + 0.1 \cdot SampleRTT_{T_3}$

$EstimatedRTT_3$
$= 0.9 \cdot EstimatedRTT_2 + 0.1 \cdot SampleRTT_{T_2}$
$= 0.81 \cdot SampleRTT_{T_4} + 0.09 \cdot SampleRTT_{T_3} + 0.1 \cdot SampleRTT_{T_2}$

$EstimatedRTT_4$
$= 0.9 \cdot EstimatedRTT_3 + 0.1 \cdot SampleRTT_{T_1}$
$= 0.729 \cdot SampleRTT_{T_4} + 0.081 \cdot SampleRTT_{T_3} + 0.009 \cdot SampleRTT_{T_2} + 0.1 \cdot SampleRTT_{T_1}$

b.

假设采样是从Tn → T1

$EstimatedRTT_n = 0.1 \cdot \Sigma_{i=n-1}^{1}(9^{i-1} SampleRTT_{T_i}) + 0.9^{n-1} SampleRTT_{T_n}$

c.

$\lim_{n\to\infty}(1 \cdot \Sigma_{i=n-1}^{1}(9^{i-1} SampleRTT_{T_i}) + 0.9^{n-1} SampleRTT_{T_n})$
$= \frac{1}{9} \cdot \Sigma_{i=1}^{\infty}(9^i SampleRTT_{T_i})$

i越大代表离现在越远，且权重是按照指数衰减的，所以叫指数移动平均

P40. 考虑图 3-61 假设 TCP Reno 是一个经历如上所示行为的协议，回答下列问题。在各种情况中，简要地论证你的回答。

a. 指出 TCP 慢启动运行时的时间间隔。
b. 指出 TCP 拥塞避免运行时的时间间隔。
c. 在第 16 个传输轮回之后，报文段的丢失是根据 3 个冗余 ACK 还是根据超时检测出来的？
d. 在第 22 个传输轮回之后，报文段的丢失是根据 3 个冗余 ACK 还是根据超时检测出来的？

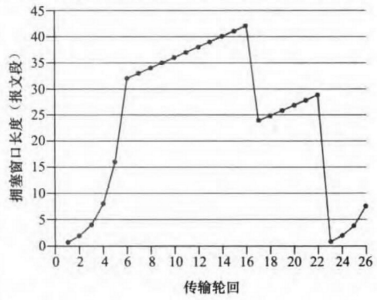

图 3-61 TCP 窗口长度作为时间的函数

e. 在第 1 个传输轮回里，ssthresh 的初始值设置为多少？
f. 在第 18 个传输轮回里，ssthresh 的值设置为多少？
g. 在第 24 个传输轮回里，ssthresh 的值设置为多少？
h. 在哪个传输轮回内发送第 70 个报文段？
i. 假定在第 26 个传输轮回后，通过收到 3 个冗余 ACK 检测出有分组丢失，拥塞的窗口长度和 ssthresh 的值应当是多少？
j. 假定使用 TCP Tahoe（而不是 TCP Reno），并假定在第 16 个传输轮回收到 3 个冗余 ACK。在第 19 个传输轮回，ssthresh 和拥塞窗口长度是什么？
k. 再次假设使用 TCP Tahoe，在第 22 个传输轮回有一个超时事件。从第 17 个传输轮回到第 22 个传输轮回（包括这两个传输轮回），一共发送了多少分组？

a. [0,6], [23,26]
b. [6,16],[17,22]
c. 是根据3个冗余ack
d. 是根据超时
e. 32
f. (32 + (16 - 6)) / 2 = 21
g. ((21 + 3) + (22 - 17)) / 2 = 14.5 -> 14
h. cwnd :[0,1,2,4,8,16,32,33,34,35,36,37,38,39,40,41,42,24,25,26,27,28,29,1,2,4,8]
   前缀和：[0, 1, 3, 7, 15, 31, 63, 96, 130, 165, 201, 238, 276, 315, 355, 396, 438, 462, 487, 513, 540, 568, 597, 598, 600, 604, 612]
   在第7个传输轮回
i. cwnd = 8 / 2 + 3 = 7
   sstresh = 8 / 2 = 4
j. cwnd = 1; sstresh = 42 / 2 = 21
k. cwnd :[0,1,2,4,8,16,32,33,34,35,36,37,38,39,40,41,42,1,2,4,8,16,21....]
   所以17~22共有 1+2+4+8+16+21 = 52 个分组

P48 考虑仅有一条单一的 TCP（Reno）连接使用一条 10Mbps 链路，且该链路没有缓存任何数据。假设这条链路是发送主机和接收主机之间的唯一拥塞链路。假定某 TCP 发送方向接收方有一个大文件要发送，而接收方的接收缓存比拥塞窗口要大得多。我们也做下列假设：每个 TCP 报文段长度为 1500 字节；该连接的双向传播时延是 150ms；并且该 TCP 连接总是处于拥塞避免阶段，即忽略了慢启动。

a. 这条 TCP 连接能够取得的最大窗口长度（以报文段计）是多少？
b. 这条 TCP 连接的平均窗口长度（以报文段计）和平均吞吐量（以 bps 计）是多少？
c. 这条 TCP 连接在从丢包恢复后，再次到达其最大窗口要经历多长时间？

a. 10 * 10^6 * 150 * 10^(-3) / (8*1500) = 125
b. 因为拥塞避免状态，所以cwnd在 125 / 2 ~ 125 的整数上均匀分布
   所以，平均窗口长度为 (62 + 125) / 2 = 93.5
   平均吞吐量 93.5 * 1500 * 8 / (150*10^(-3)) = 7.48Mbps
c. (125 - 62) * 150 * 10^(-3) = 9.45 s

P52 考虑一种简化的 TCP 的 AIMD 算法，其中拥塞窗口长度用报文段的数量来度量，而不是用字节度量。在加性增中，每个 RTT 拥塞窗口长度增加一个报文段。在乘性减中，拥塞窗口长度减小一半（如果结果不是一个整数，向下取整到最近的整数）。假设两条 TCP 连接 C1 和 C2，它们共享一条速率为每秒 30 个报文段的单一拥塞链路。假设 C1 和 C2 均处于拥塞避免阶段。连接 C1 的 RTT 是 50ms，连接 C2 的 RTT 是 100ms。假设当链路中的数据速率超过了链路的速率时，所有 TCP 连接经受数据报文段丢失。

a. 如果在时刻 t_0，C1 和 C2 具有 10 个报文段的拥塞窗口，在 1000ms 后它们的拥塞窗口为多长？
b. 经长时间运行，这两条连接将取得共享该拥塞链路的相同的带宽吗？

P52 考虑一种简化的 TCP 的 AIMD 算法，其中拥塞窗口长度用报文段的数量来度量，而不是用字节度量。在加性增中，每个 RTT 拥塞窗口长度增加一个报文段。在乘性减中，拥塞窗口长度减小一半（如果结果不是一个整数，向下取整到最近的整数）。假设两条 TCP 连接 C1 和 C2，它们共享一条速率为每秒 30 个报文段的单一拥塞链路。假设 C1 和 C2 均处于拥塞避免阶段。连接 C1 的 RTT 是 50ms，连接 C2 的 RTT 是 100ms。假设当链路中的数据速率超过了链路的速率时，所有 TCP 连接经受数据报文段丢失。

a. 如果在时刻 t_0，C1 和 C2 具有 10 个报文段的拥塞窗口，在 1000ms 后它们的拥塞窗口为多长？

b. 经长时间运行，这两条连接将取得共享该拥塞链路的相同的带宽吗？

```python
from typing import List
from math import gcd
import sys
SLOW_START,CONG_AVOID,FAST_RECOVERY = 0,1,2
time = 0
STEP_TIME = 1
class Host: pass
class Channel: pass
class Message: pass
class Host:
    def __init__(self,name:str,cwnd:int,state:int,interval:int,channel: Channel):
        self.cwnd,self.state,self.interval = cwnd,state,interval
        self.channel:Channel = channel
        self.name = name
    def messageArrive(self,msg: Message):
        if(self.state != CONG_AVOID): return # TBC
        if(msg.is_dropped):
            self.cwnd = max(1,self.cwnd // 2)
        else:
            self.cwnd += 1
        self.sendMessage()
    def sendMessage(self):
        msg = Message(self.cwnd,self.interval,self)
        print("a message sent: ({})".format(msg),end="; ")
        channel.addMsg(msg)
class Message:
    def __init__(self, size:int, left_time: int, host: Host):
        self.size,self.left_time = size,left_time
        self.host:Host = host
        self.rate = 1000 * self.size / self.host.interval
        self.is_dropped = False
    def update(self):
        self.left_time -= STEP_TIME
    def Cong(self):
        self.is_dropped = True
    def __repr__(self) -> str:
        return "size:{},rate:{},host:{}".format(self.size,self.rate,self.host.name)
class Channel:
    def __init__(self,max_rate:float,curr_msg:List[Message]=[]):
        self.max_rate = max_rate
        self.curr_msgs:List[Message] = curr_msg
    def update(self):
        if self.isCong():
            for msg in self.curr_msgs: msg.Cong()
        tmp = self.curr_msgs
        self.curr_msgs:List[Message] = []
        for msg in tmp:
            msg.update()
            if(msg.left_time <= 0): msg.host.messageArrive(msg)
            else: self.trasportMsg(msg)
    def isCong(self)->bool:
        return sum(msg.rate for msg in self.curr_msgs) > self.max_rate
    def trasportMsg(self,msg: Message):
        self.curr_msgs.append(msg)
    def addMsg(self,msg:Message):
        self.curr_msgs.append(msg)
if __name__ == "__main__":
    f = open("output.txt", "w")
    sys.stdout = f
    channel = Channel(30)
    c1,c2 = Host("C1",10,CONG_AVOID,50,channel),Host("C2",10,CONG_AVOID,100,channel)
    STEP_TIME = gcd(c1.interval,c2.interval)
    print("------------time {}-----------".format(time))
    c1.sendMessage()
    c2.sendMessage()
    print("")
    time += STEP_TIME
    while time <= 1000:
        print("------------time {}-----------".format(time))
        channel.update()
```

```
        time += STEP_TIME
        print("")
    f.close()
```

output:

```
------------time 0-----------
a message sent: (size:10,rate:200.0,host:C1); a message sent: (size:10,rate:100.0,host:C2);
------------time 50-----------
a message sent: (size:5,rate:100.0,host:C1);
------------time 100-----------
a message sent: (size:2,rate:40.0,host:C1); a message sent: (size:5,rate:50.0,host:C2);
------------time 150-----------
a message sent: (size:1,rate:20.0,host:C1);
------------time 200-----------
a message sent: (size:1,rate:20.0,host:C1); a message sent: (size:2,rate:20.0,host:C2);
------------time 250-----------
a message sent: (size:1,rate:20.0,host:C1);
------------time 300-----------
a message sent: (size:1,rate:20.0,host:C1); a message sent: (size:1,rate:10.0,host:C2);
------------time 350-----------
a message sent: (size:2,rate:40.0,host:C1);
------------time 400-----------
a message sent: (size:1,rate:20.0,host:C1); a message sent: (size:1,rate:10.0,host:C2);
...... ...... ......
------------time 850-----------
a message sent: (size:2,rate:40.0,host:C1);
------------time 900-----------
a message sent: (size:1,rate:20.0,host:C1); a message sent: (size:1,rate:10.0,host:C2);
------------time 950-----------
a message sent: (size:2,rate:40.0,host:C1);
------------time 1000-----------
a message sent: (size:1,rate:20.0,host:C1); a message sent: (size:1,rate:10.0,host:C2);
```

a. 1000ms之后，两个主机的cwnd 均为 1

b. 不会有相同的带宽

P54 考虑修改 TCP 的拥塞控制算法。不使用加性增，使用乘性增。无论何时某 TCP 收到一个合法的 ACK，就将其窗口长度增加一个小正数 a（$0 < a < 1$）。求出丢包率 L 和最大拥塞窗口 W 之间的函数关系。论证：对于这种修正的 TCP，无论 TCP 的平均吞吐量如何，一条 TCP 连接将其拥塞窗口长度从 $W/2$ 增加到 W，总是需要相同的时间。

$$\left(\frac{W}{2}\right) * (1+a)^n = W$$

$$L = \frac{1}{\sum_{i=0}^{n} \frac{W}{2}(1+a)^i} = \frac{2}{W} * \frac{1}{\left(\frac{(1+a)^{n+1}-1}{a}\right)} = \frac{2}{W}\left(\frac{a}{2(1+a)-1}\right)$$

$$= \frac{2a}{W(2a+1)}$$

从$\frac{W}{2}$到W的次数为$n = (\log_{1+a} 2)_{向上取整}$

所以都需要$RTT*n$的时间 该时间与吞吐量无关