No. Date

第5周 9

R1 a. 在发送端，该协议接收应用程序要发送的数据、目的地址、端口号，然后协议会添加4字节头部信息，即端口号。将这1200字节的报文段连同目的地址交付给网络层。
在接收端，协议提取端口和数据，将数据发送给端口所标志的程序。

b. 在头部信息中增4字节的源端口号，将数据减少为1192 Bytes

c. No.

R2 a. 寄信时，家庭成员必须将信件本身、目的地地址和收件人的姓名交给代表。委托将收件人的姓名清楚地写在信函的顶部。然后委托将信放入信封中，并在信封上写入目标住宅的地址。代表将信交给邮件服务部门接收方，委托收到来自邮件服务部门的信，从信封中取出信件，并记下信件顶部写的收件人姓名。然后，代表将这封信交给具有此名称的家庭成员。

b. 不用 只需检查信封上的地址。

R3. 源 y 目的 x.

R4 TCP的拥塞控制会降低某些应用的发送速率，且该应用不需要可靠数据传输。

R5. 大多数防火墙会拦截UDP.

R6. 在应用添加差错检测机制.

~~R7~~

No. Date

R10. 处理丢包事件. 如果丢包可以重传.

R11. RTT 固定加的处在，发送方可以准确判断 ACK 是否丢失，但它仍需一个时间固定的定时器.

No. Date

第5周 10

R12 a. 接收方丢弃全部分组，之后发送方重传五个分组。

b. GBN使用累积确认，因此没有触发重传

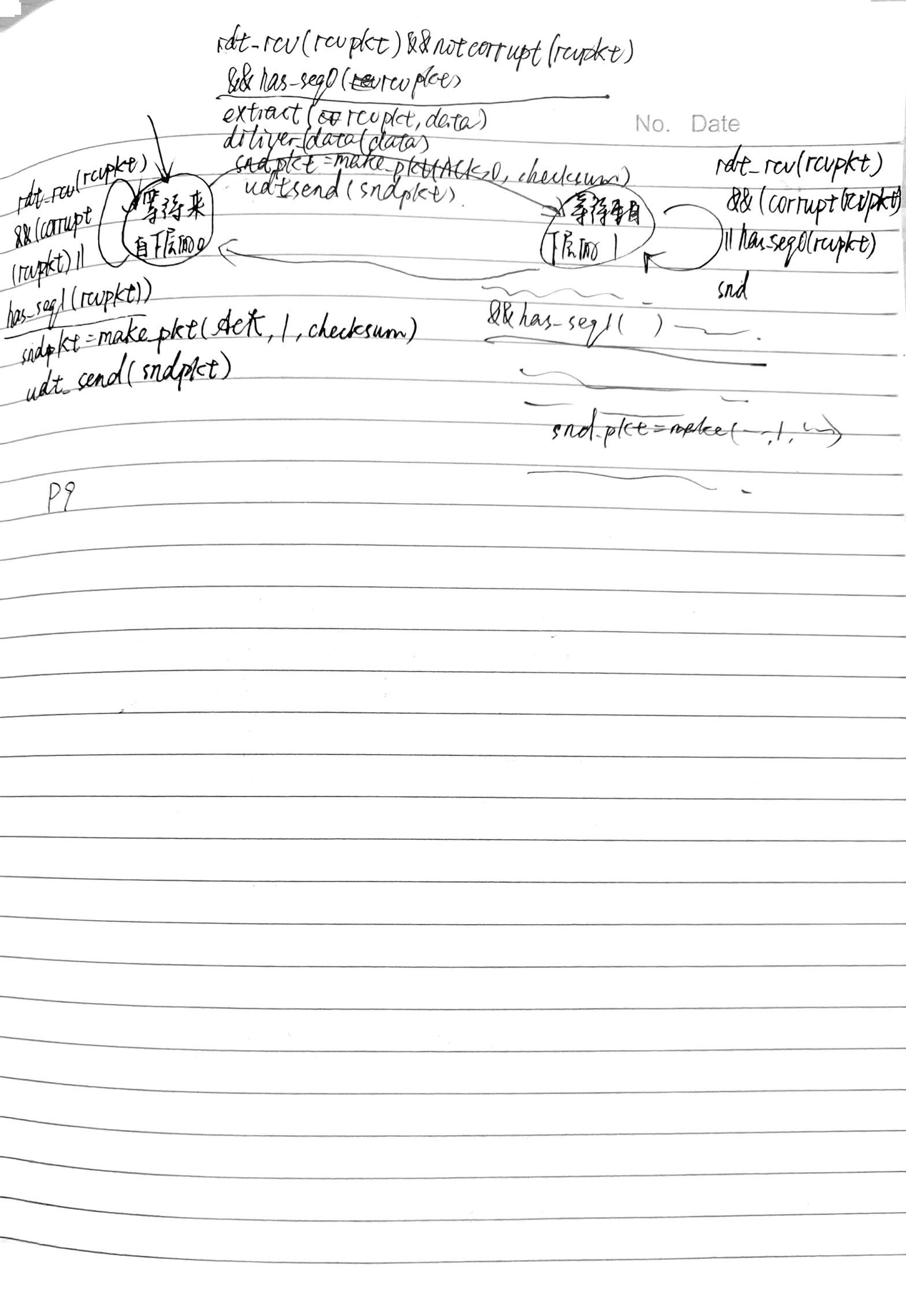
c. 只能发送五个，因为窗口大小就是5.

P5 不，接收方不能绝对确定是否没有发生任何错误。这是因为计算数据包校验和的方式。如果数据包中两个16位字的对应位(将加在一起)分别为0和1，那么即使它们各自翻转为1和0，您和仍然保持不变。因此，接收方计算的反码也将相同。这意味着即使存在传输错误，校验和也将确认。

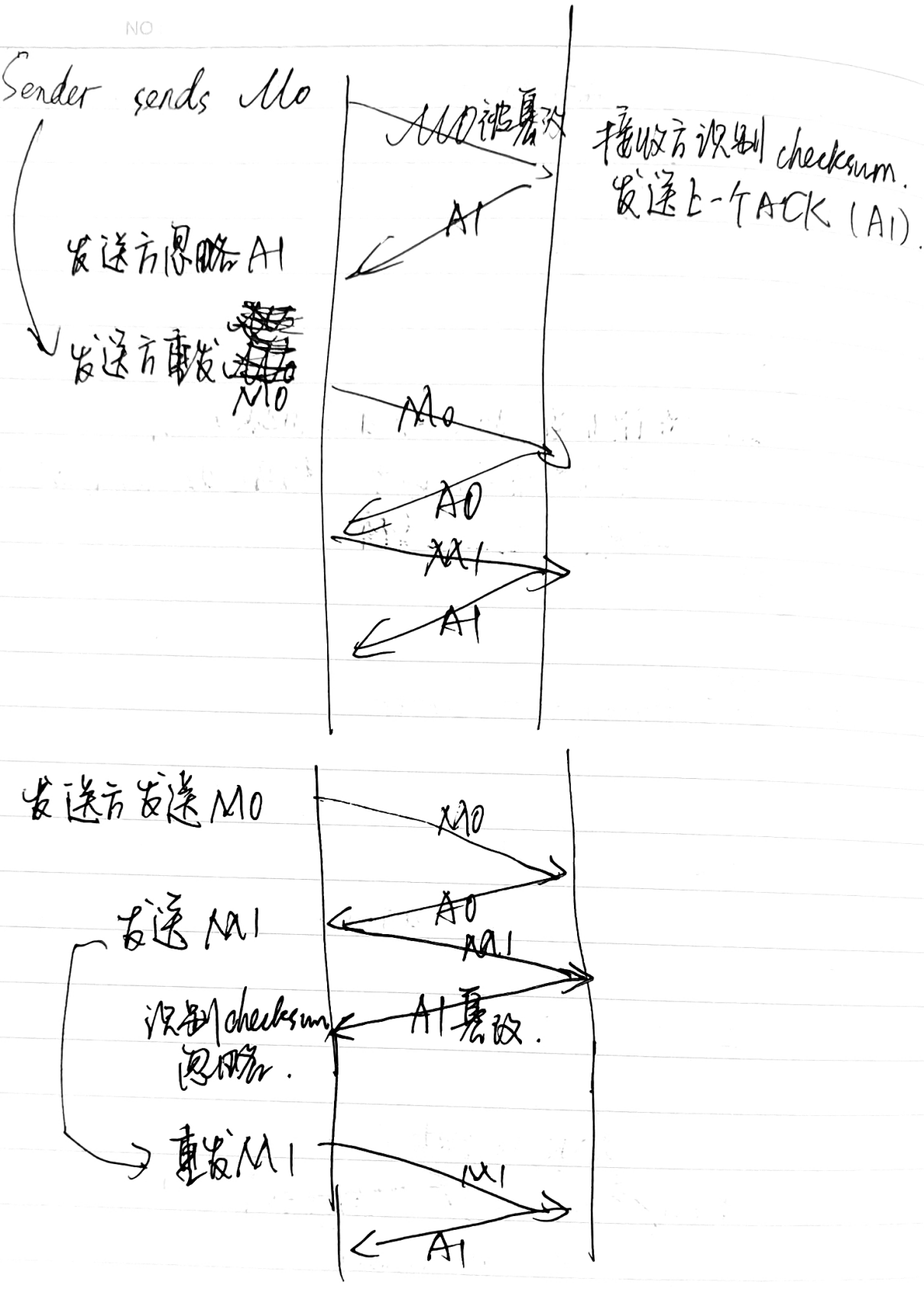
P6. 假设发送者处于"等待上面的调用"状态，而接收者处于"等待下面的"状态。发送者发送序列号为1的数据包，~~发送一个~~并转换为"等待ACK或NAK 1"，等待ACK或NAK。假设现在接收方正确地接收序列号为1的数据包，发送一个ACK，并转换为状态"从下面等待0"，等待序列号为0的数据包。但是ACK已损坏。当RDT 2.1发送器收到损坏的ACK时，它用序列号1重新发送数据包。但是接收方正在等待序列号为0的数据包，并且它是在没有序列号为0的数据包时发送一个NAK。因此，发送方将开始发送序列号为1的数据包，而接收方将继续对每个接收到的数据包进行NAK操作。

P8.

```
rdt_rcv(rcvpkt) && notcorrupt(rcvpkt)
&& has_seq0(rcvpkt)
extract(rcvpkt, data)
deliver_data(data)
sndpkt = make_pkt(ACK, 0, checksum)
udt_send(sndpkt)
```


rdt_rcv(rcvpkt) && notcorrupt(rcvpkt)
&& has_seq0(rcvpkt)
extract(rcvpkt, data)
deliver_data(data)
sndpkt = make_pkt(ACK=0, checksum)
udt_send(sndpkt)
rdt_rcv(rcvpkt)
&& (corrupt(rcvpkt)
|| has_seq1(rcvpkt))
snd
No. Date
rdt_rcv(rcvpkt)
&& corrupt(rcvpkt)
sndpkt = make_pkt(NAK, checksum)
udt_send(sndpkt)
rdt_rcv(rcvpkt)
&& notcorrupt(rcvpkt)
&& has_seq1(rcvpkt)
extract(rcvpkt, data)
deliver_data(data)
sndpkt = make_pkt(ACK=1, checksum)
udt_send(sndpkt)
rdt_rcv(rcvpkt)
&& (corrupt(rcvpkt)
|| has_seq0(rcvpkt))
snd
&& has_seq1( )
snd_pkt = make_(-, 1,)
P9

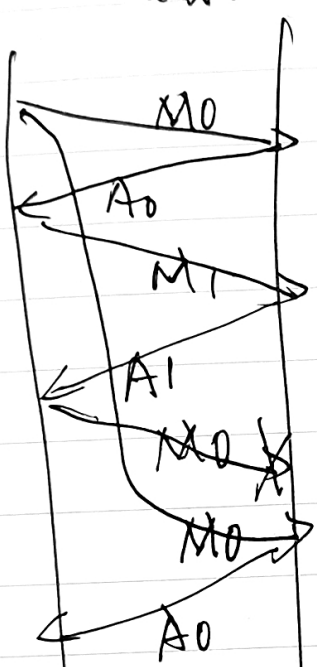
P9 Sender sends M0



P10 添加计时器，值大于RTT，将超时事件添加到"等待ACK或NAK 0/1"状态。若超时则重传最近发送的包。
若传输中丢包，发送方超时重发；若返回ACK/NAK丢包，发送方超时重发，接收方回旧的ACK。

P11. 不能正确工作。
e.g. 发送方发pkt0→等待ACK0，但
接收方正等待pkt1但收到0/损坏，但又不发NAK。
或发、收方都在等待。

P13



P14 发送数据不频繁，使用ACK更好。因为要到下一个包被接收时，接收方才能发现该包丢失。时延会很高。
发送较多数据时适合用~~ACK~~ NAK。因为只有出错才回复NAK，减少流量。

DATE NO

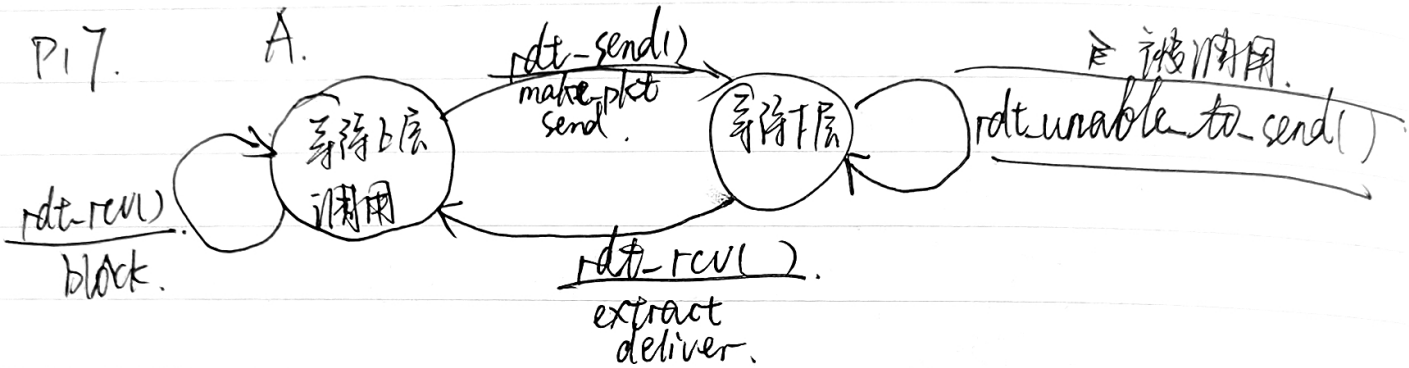
P15 $R = 1Gbps$ $L = 1500 bytes = 12000 bits = 12kb$

$$U_{sender} = \frac{L/R \times N}{RTT + L/R} = \frac{0.012ms \times N}{30ms + 0.012ms} = 0.98$$

$\Rightarrow N = 2451$

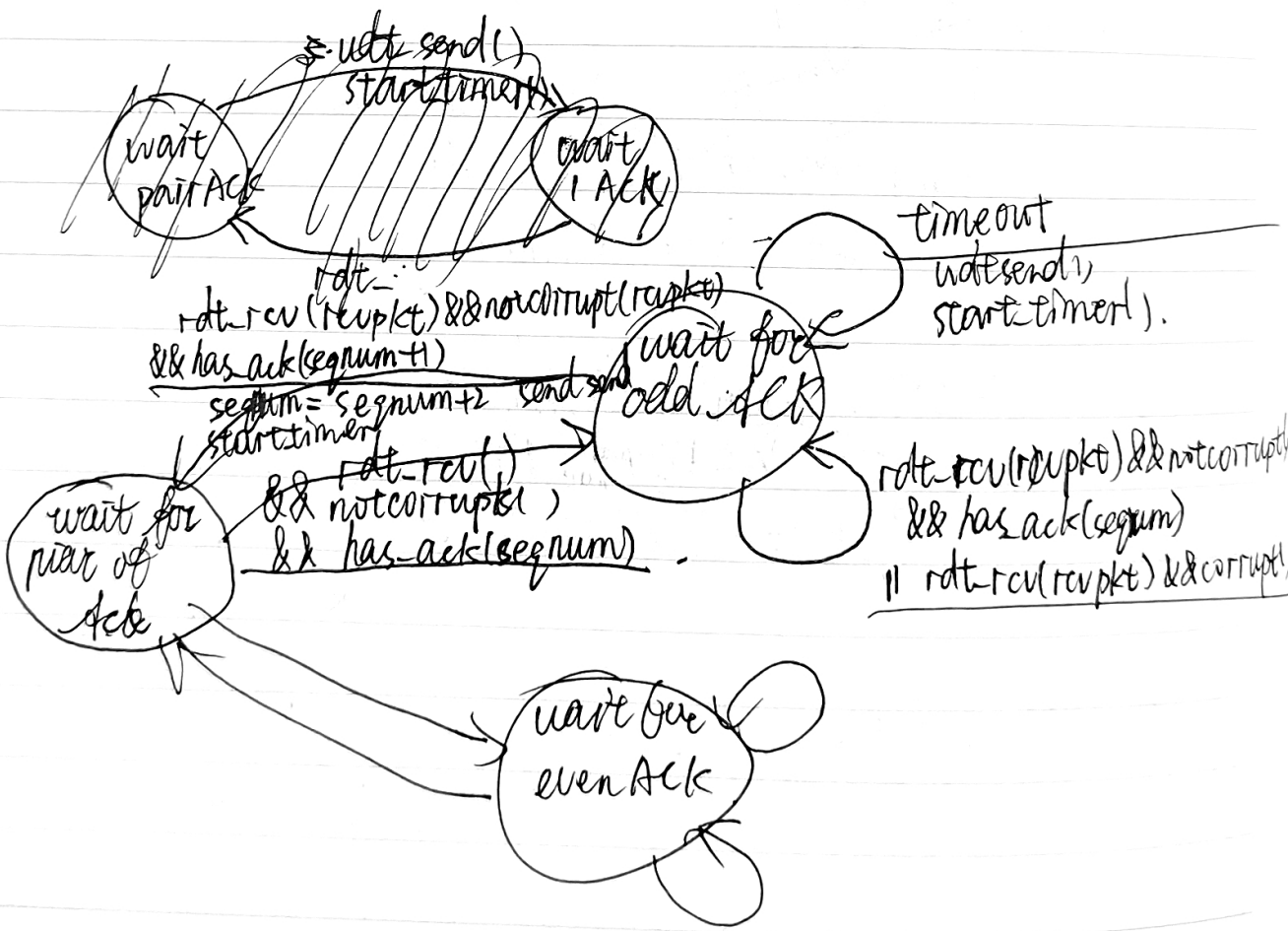
P16 Yes. Problem: 连续丢两个包 ACK 无法识别.

P17. A.



B 同A.

P18.



Week 6 Assignment 11

R8 不同由同一个 socket 处理.
对于每一个链接，都会创建 socket. 每个 socket 由
(源IP (=服务器IP), 源端口 (=80), 目的IP, 目的Port) 唯一标识.

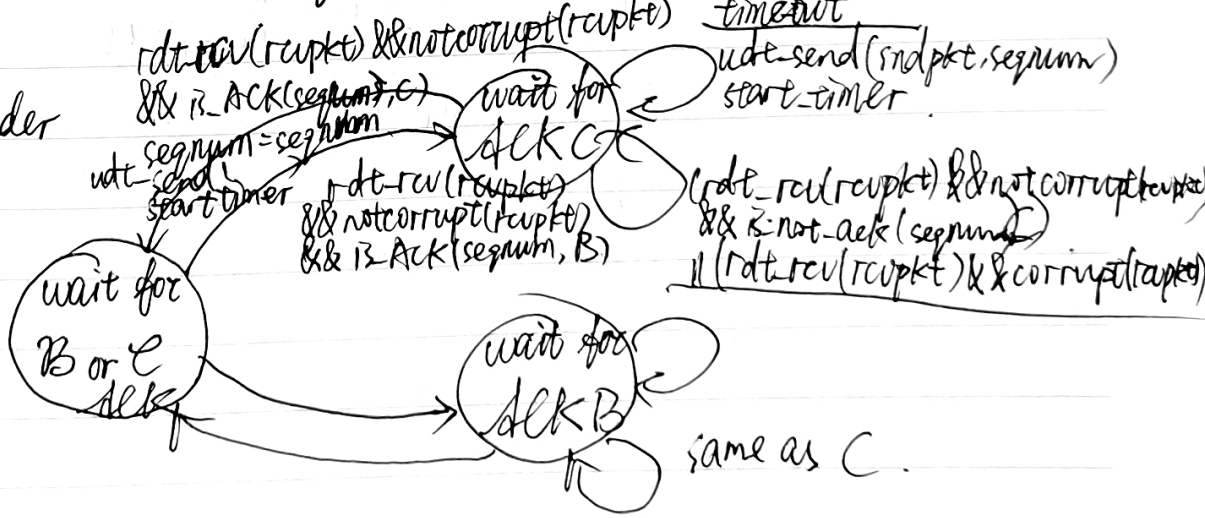
Week 7 Assignment 12

R14 a. False b. False c. True. d. False. e True
f. False g. False

R15 a. 20 bytes b. 90

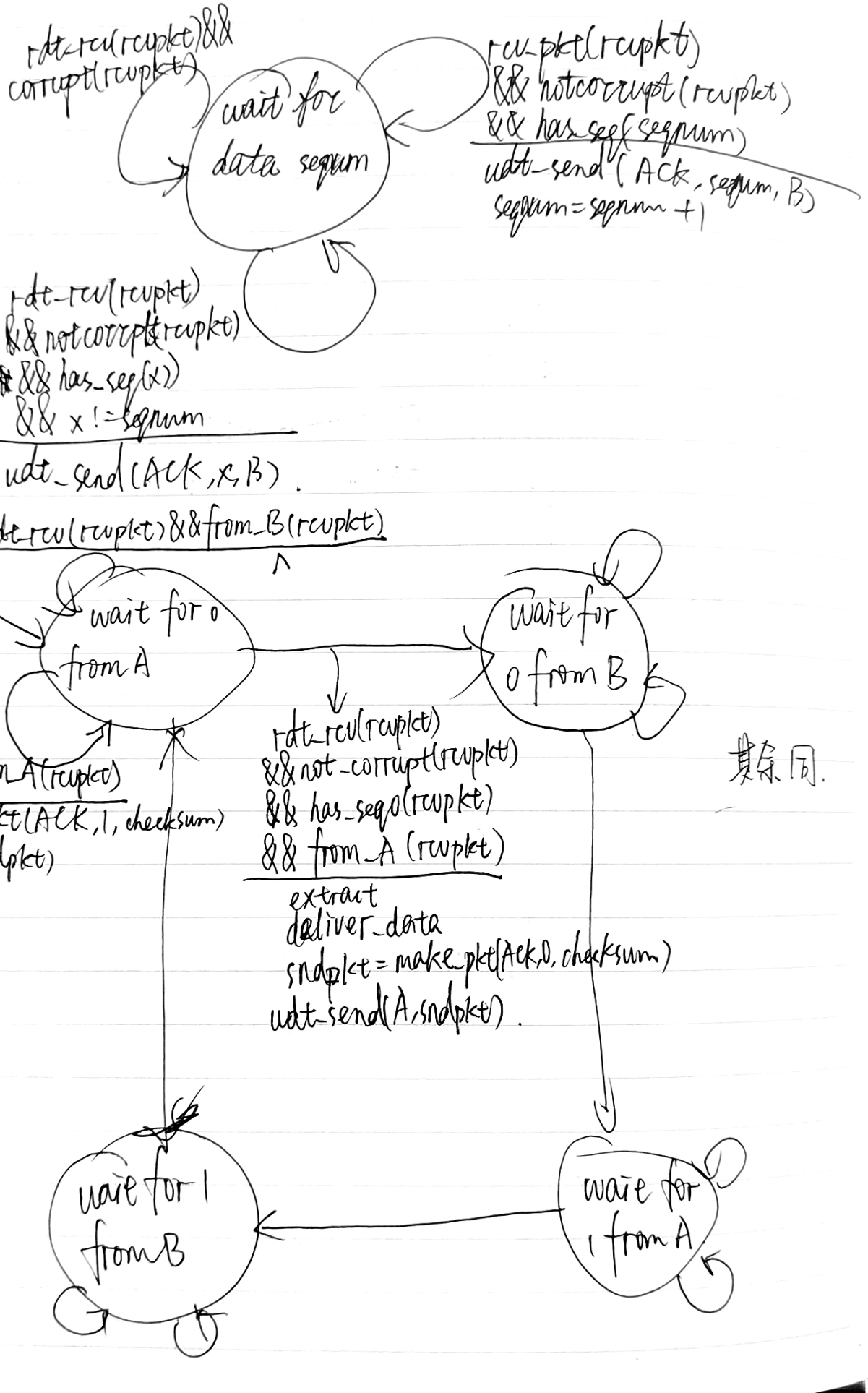
R16. 3 segments 1. seq=43, ack=80.
2. seq=80, ack=44
3. seq=44, ack=81.

P19 Sender



DATE NO

Receiver B



其余同.

P21. A: ① 等待上层的请求0：发送请求R0给B，开始计时
② 等待D0回复：若计时器未放，重发.
③ 等待上层的请求1.
④ 等待D1回复.

B. ① 发D0
② 发D1.

P22 ~~a [k-N, k] [k-3, k], [k-2, k] [k-1, k] [k, k]~~
~~b~~
a. 两种情况 ① 所有ACK都被接收 [k, k+N-1]
② 没有ACK被接收 [k-N, k-1].
综上，窗口base可能在[k-N, k].
b. [k-N-1, k-1].

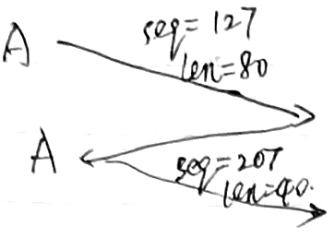
P23 $w \leq \frac{k}{2}$

P24 a) True
b) True.
c) True
d) True

P26 a) 2^{32}
b) number of segments: $\lceil \frac{2^{32}}{536} \rceil = 8012999$.

加入TCP头部信息总量: $66 \times 8012999 = 528857934$
总量 $2^{32} + 528857934 = 4.824 \times 10^9$ bytes.
除以 155 Mbps = 249 s.

P27

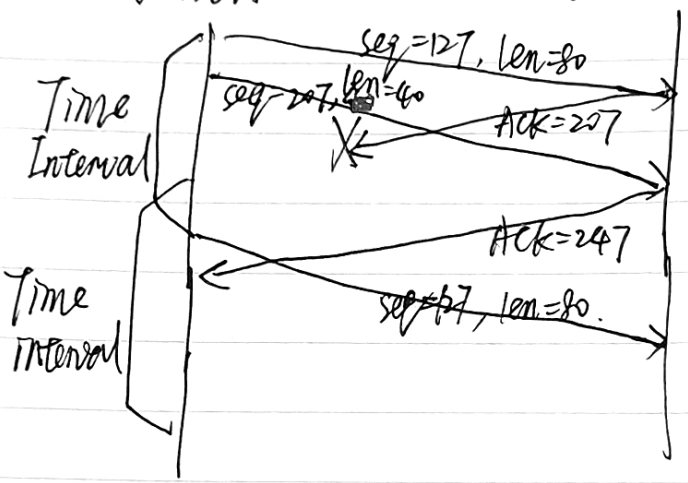


a) 207 source portnum = 302
destin portnum = 80.

b). 207 ; 80 ; 302

c) 127 B在等待127.

d).



P32 a) $EstimatedRTT^{(4)} = x\,SampleRTT_1 + (1-x)[x\,SampleRTT_2 + (1-x)[x\,SampleRTT_3 + (1-x)SampleRTT_4]]$

$= x\,SampleRTT_1 + (1-x)x\,SampleRTT_2 + (1-x)^2 x\,SampleRTT_3 + (1-x)^3 SampleRTT_4$

b) $EstimatedRTT^{(n)} = x\sum_{j=1}^{n-1}(1-x)^{j-1}SampleRTT_j + (1-x)^{n-1}SampleRTT_n.$

c) $EstimatedRTT^{(\infty)} = \frac{x}{1-x}\sum_{j=1}^{\infty}(1-x)^j SampleRTT_j$

$= \frac{1}{9}\sum_{j=1}^{\infty}0.9^j\,SampleRTT_j$

P36 收到重复的ACK只是代表存在一个包未发送成功. 未收到 ~~未发送成功~~ 的包可能未丢失，只是未传输到. 若重复一次或n重传可能导致过为冗余包.

P37 ~~a) GBN: 1,2,3,4,5, 2,3,4,5~~

~~b) SR~~

a) GBN: A发9个包. 1, 2, 3, 4, 5 ; 2, 3, 4, 5.
B发8个包. 1, 1, 1, 1 ; 2, 3, 4, 5.

~~b)~~ SR: A发6个包: 1, 2, 3, 4, 5; 2
B发5个包: 1, 3, 4, 5 ; 2

TCP: A发6个包: 1, 2, 3, 4, 5; 2
B发5个包: 2, 2, 2, 2; 6

b.) TCP.

P38. Yes. 总是大约为 cwnd/RTT

P39 NO ; Yes max 7out is R14

P40 a) [1, 6] & [23, 26]
b) [6, 16] & [17, 22]
c) triple duplicate ACK
d) segment loss
e) 32
f) 21
g) 14
h) 7th
i) 4 7
j) 21 1
k) 17 - 1
18 - 2
19 - 4
20 - 8
21 - 16
22 - 21
} 52