## 第三次作业





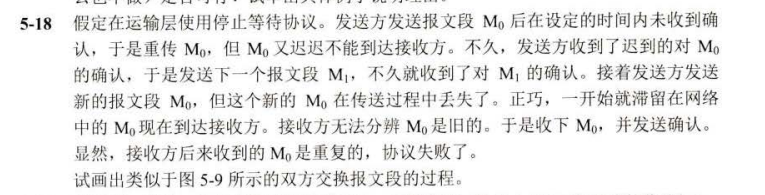
* 发送方的UDP对应用程序交下来的报文，在添加首部后就向下交付 IP 层。UDP 对应用层交下来的报文，既不合并，也不拆分，而是保留这些报文的边界。接收方 UDP 对 IP 层交上来的 UDP 用户数据报，在去除首部后就原封不动地交付上层的应用进程，一次交付一个完整的报文。
* TCP的流指的是流入到进程或从进程流出的字节序列，虽然应用程序和TCP的交互是一次一个数据块，但TCP把应用程序交下来的数据仅仅看成是一连串的无结构的字节流

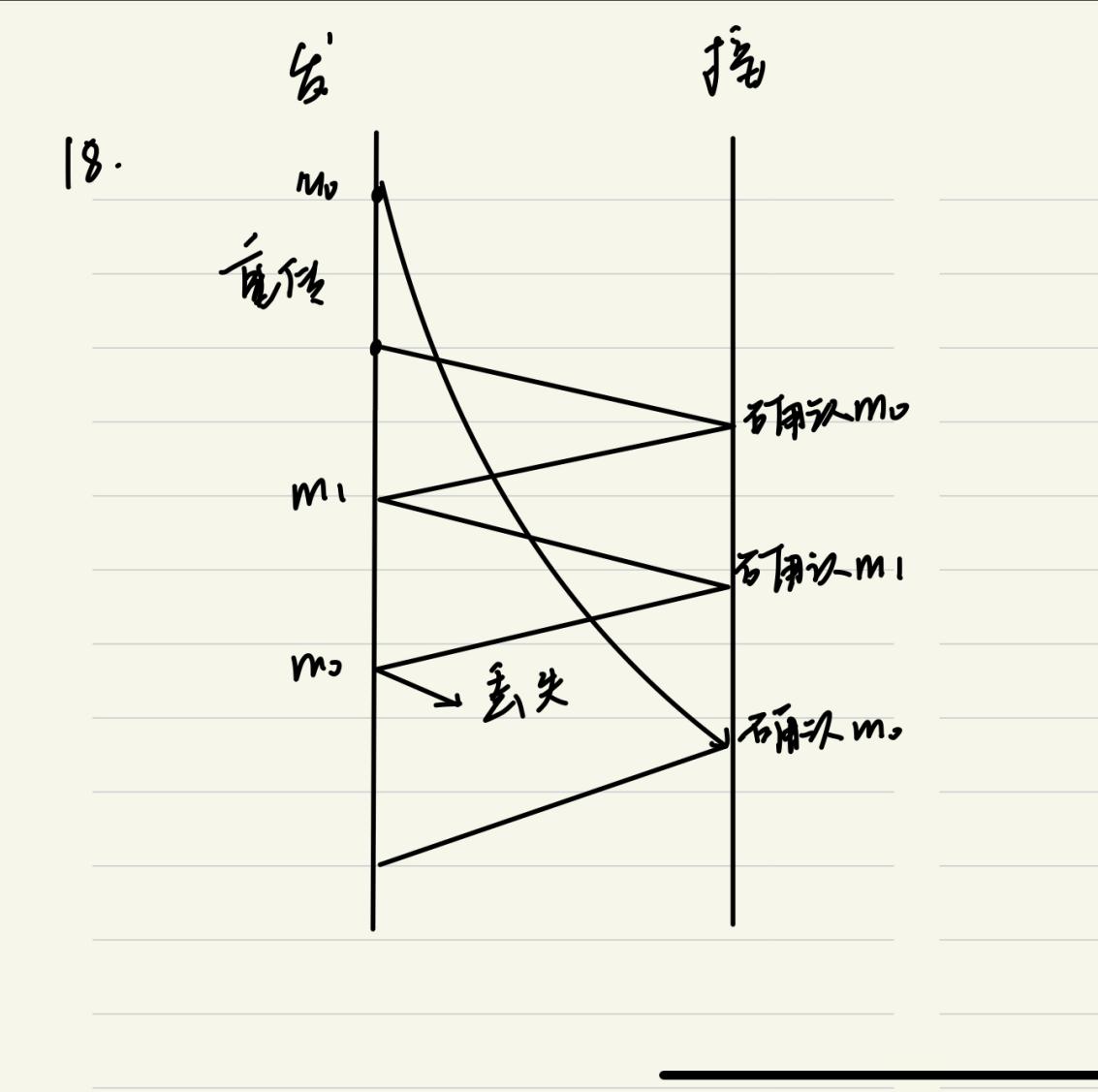




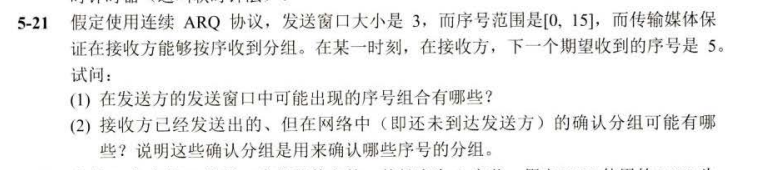
不可行，如果回复确认的信息丢失，会导致发送方发重复的数据，而接收方只是不予理睬，发送方会认为信息丢失，导致一直重传





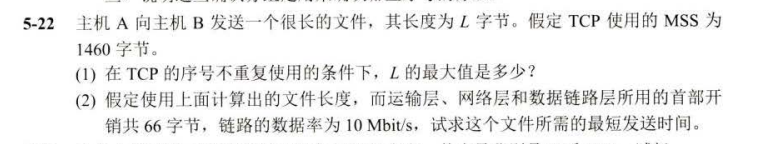






* + 若5号之前的包都按序收到，发送方收到了2-4的确认，则窗口中为5-7
  + 若都没收到确认，则窗口中为2-4
  + 若没有收到3的确认，则3-5
  + 若没有收到4的确认，则4-6
* 因为期望收到5，且窗口为3，说明1的确认被收到了，则在网络中的确认分组可能还有3，4，5，分别对2，3，4号分组进行确认



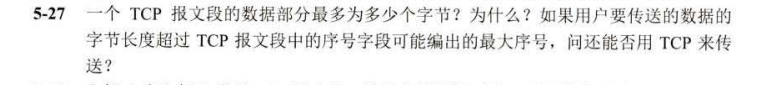


* TCP序号字段共32位，每一位代表一个字节段，所以L的最大值位4GB
* 4GB共需要2^32 / 1460 = 2941759个TCP报文

总字节数为2941759 \* 66 + 2^32 = 4489123390个字节

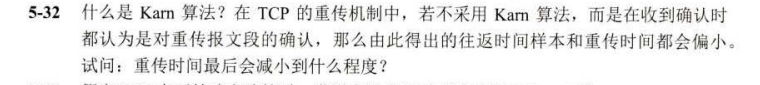
发送时间为4489123390 \* 8 / (10^7) = 3591.3s 约一个小时





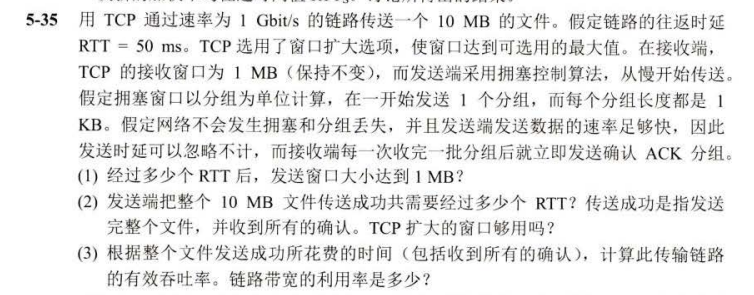
* 由于IP数据报的最大长度为65535，减去IP首部和TCP首部的至少40字节，TCP数据部分最多为66495字节
* 可以，因为TCP是通过循环使用序号，只要能保证后一个要用的序号没被占用即可





* 在计算加权平均RTTs时，只要报文段重传了，就不采用其往返时延样本。这样得出的加权平均RTTs和RTO就较准确
* 若不采用karn算法，收到的确认被当成是对重传报文段的确认，则计算出的RTTs和RTO都会偏小。导致报文段过多重传，RTO越来越短，最终趋近于1/2RTO





* 根据慢开始特性，每经过一个RTT，发送窗口×2，从1kB到1MB要经历10个RTT，即0.5s
* 网络不会发生拥塞，发送窗口达到1MB是，经过了10个RTT，且发送了接近1MB的信息。这时继续扩大窗口，再经历4个RTT，发送信息超过10MB，，共经历14个RTT。

选用扩展窗口最大值后，窗口大小为，1GB-1B，够用

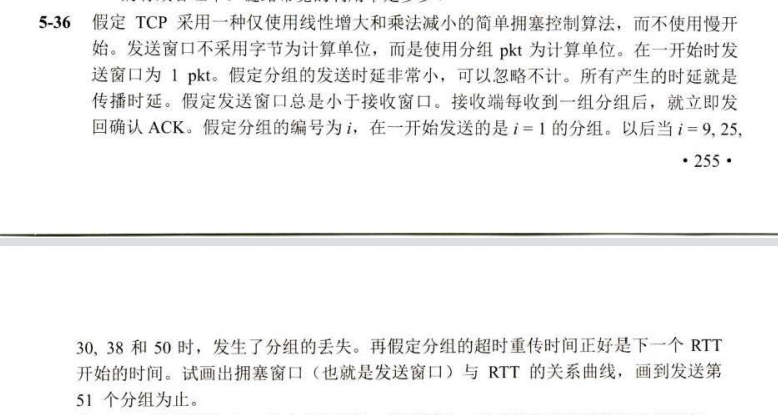
* 时间为：14\*50ms = 0.7s

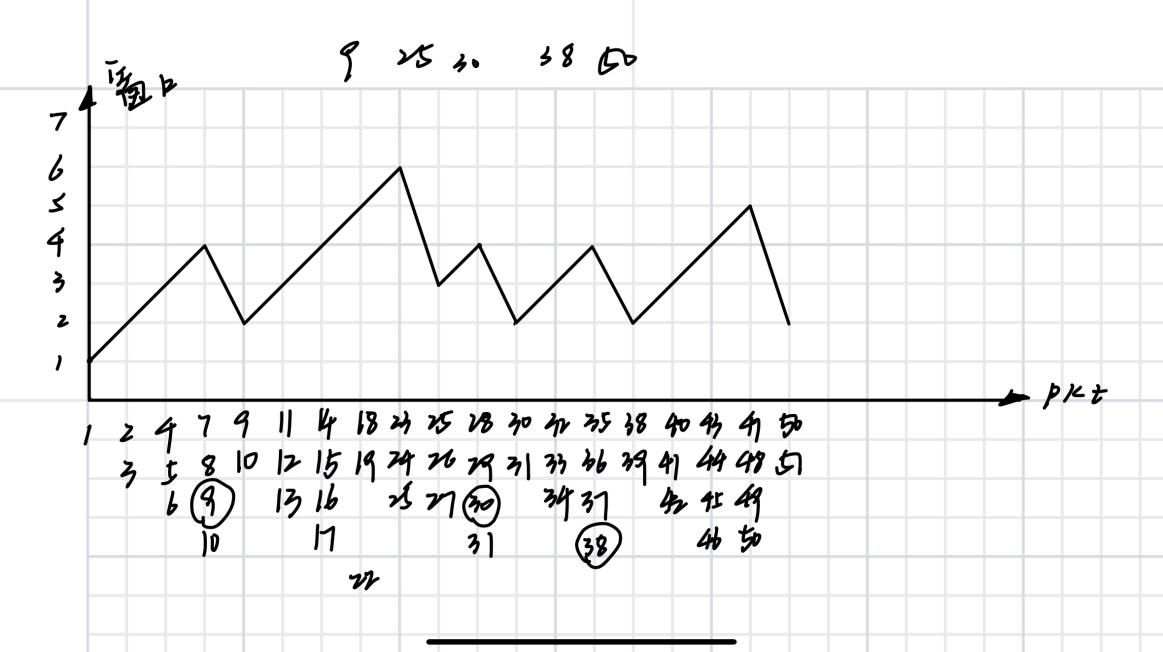
有效数据量为：10MB

有效吞吐率为：10MB/0.7s = 119.8Mbit/s

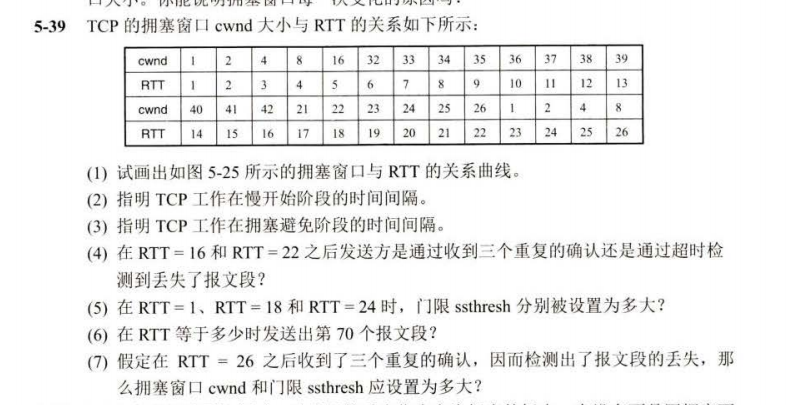
带宽利用率为：119.8Mbit/s / 1Gbit/s = 11.98%

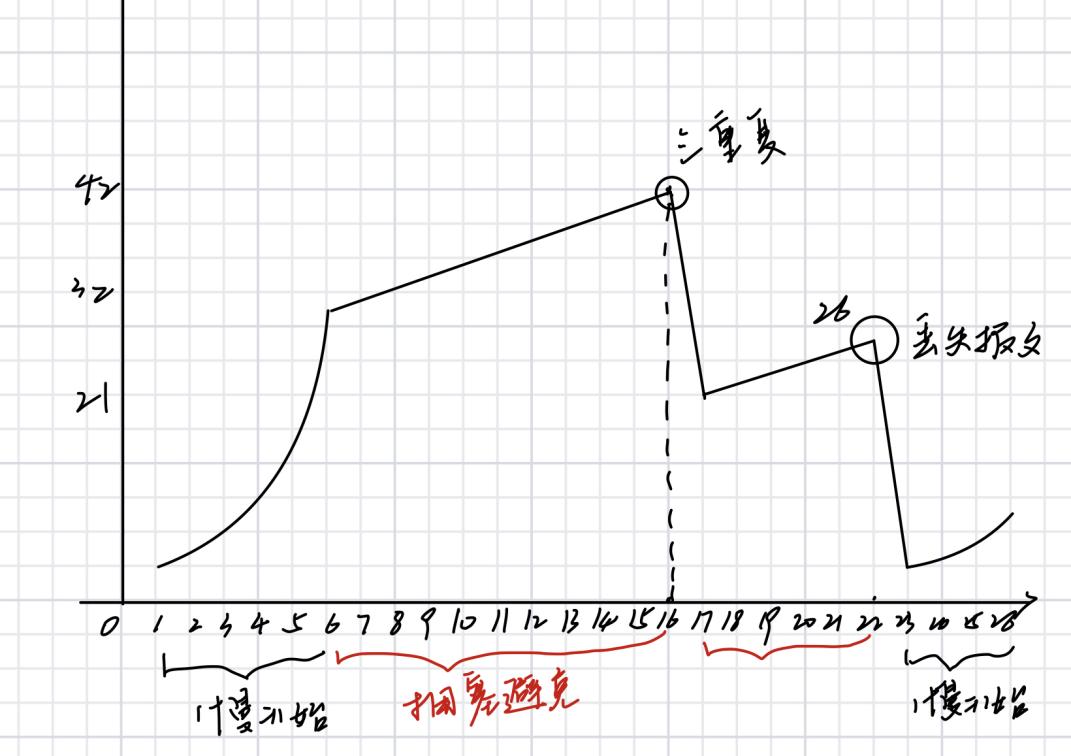






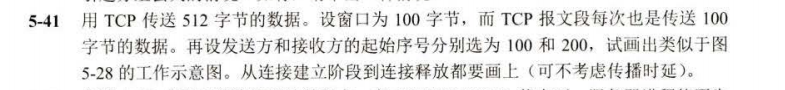


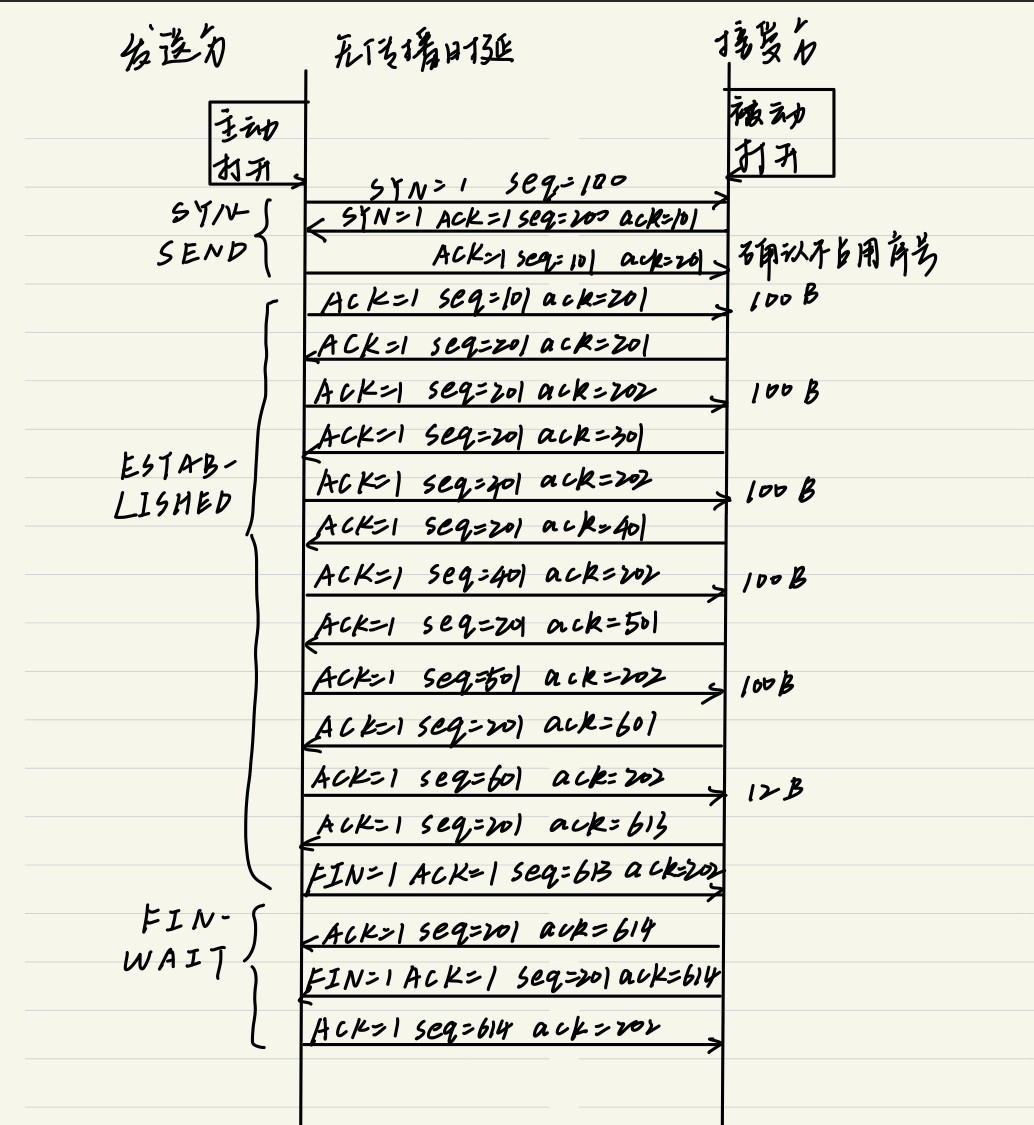




* 慢开始：1-6，23-16
* 拥塞避免：6-16，17-22
* RTT = 16为三重复，因为拥塞窗口减半，RTT=22为丢失，拥塞窗口降为1
* RTT = 1 ---32 RTT = 18 ---21 RTT = 24 ---13
* RTT = 7时
* 门限设为4，拥塞窗口设为4





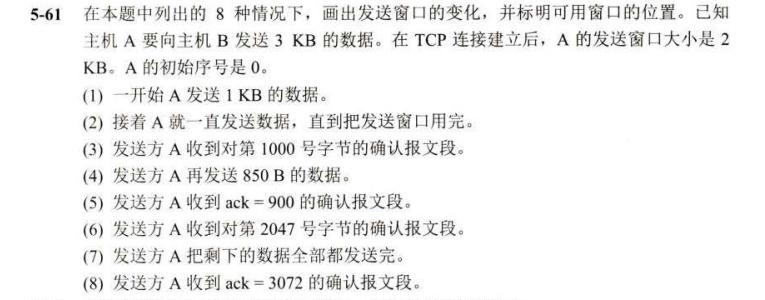


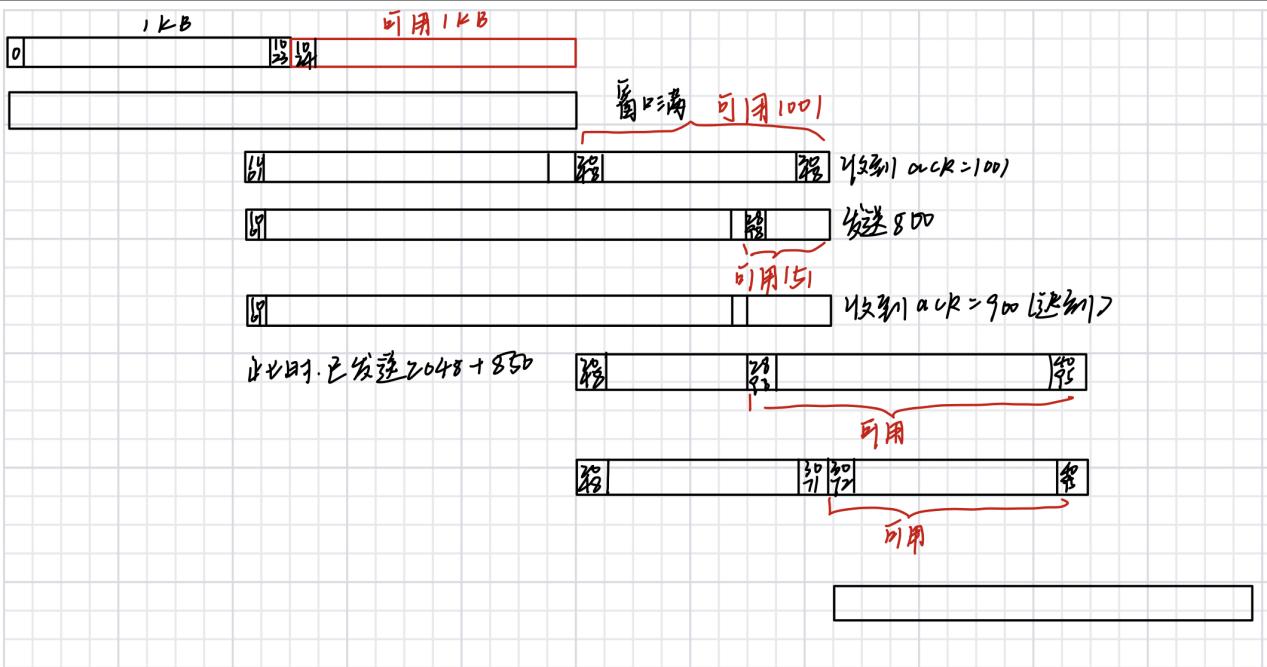




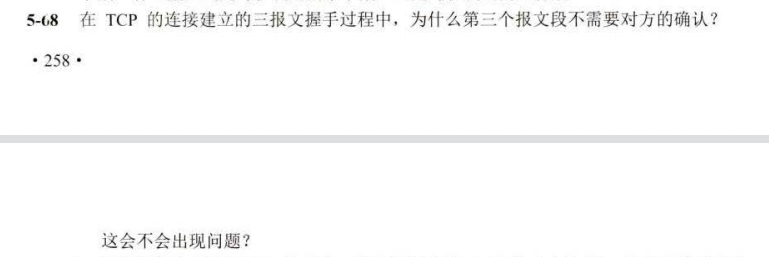
如果不进行第三次握手，比如A向B发送连接请求，B向A发送确认后，就建立了连接，但如果连接确认在传输中丢失，A不知道连接是否建立，这时A只能忽略 B 发来的任何数据分组，只等待连接确认应答分组。











如果需要对方确认，其实也就是为了保证发送方的第三次握手在中途没有丢失，到达了接受方，建立了连接。

但在第三个报文不需要对方的确认情况下，如果第三次握手丢失了，而发送方并不知道丢失，并且认为已经成功建立了连接，接下来就会发送数据，而该数据报文的seq，ack和第三次握手发送的确认报文一致，只是携带了数据部分，本质上也是对第二次握手的确认。这样，尽管第三次握手的确认没有传到接收方，但数据部分到达的时候，也再次对第二次握手的确认，不会造成没有连接的情况。