ME2012 高级计算机网络　第 3 章部分习题参考解答　2012-05 起

蓝色的题目是布置过的习题要求掌握

UDP

3.1，请说明 **UDP** 用户数据报报文段的基本格式。并指出用户数据报是如何实现传输层协议的基本功能的？

[解答]　32 位

源端口号	目的地端口号
长度	检验和
应用程序数据（报文）	

UDP 从发送主机的应用进程得到报文，附加上为复用/分解服务所需要的源和目的地端口号字段，以及另外两个字段（长度和校验和）； 然后把形成的报文段交给网络层。网络层把该报文段封装到一个 IP 数据报中（即把该报文段作为负荷数据），再以尽力而为的方式把 IP 数据报送交到接收主机（依 IP 地址）。该报文段到达接收主机后，UDP 使用"目的端口号"把数据递交到相应的应用进程。

3.2，为什么需要 UDP？为什么用户不能直接访问 IP？UDP 协议主要用在哪些场合？

[解答][1]与 TCP 相比，需要使用 UDP 的原因是：UDP 简单（小的段首部，在发送方和接收方无状态变量等）；快捷（无需连接建立的延迟时间，不考虑拥塞控制和发送速率管理等）。应用层还能更好地控制要发送的数据和发送的时间。

[2]仅仅使用 IP 分组对于应用进程来说是不够的。一个数据段要从源应用进程传送到目的地进程，必须规定目的地地址和应用进程的相应端口 PORT（见"套接字" Socket）。IP 分组仅包括目的地的地址，分组送到目的地机器后，网络控制程序不能确定把分组递交给哪一个进程。在 UDP 用户数据报中包含了一个目的地的端口，此信息是必须的，有了它，才能在目的地机器中把数据段递交到正确的应用进程。

[3]目前 UDP 在因特网中的主要应用有：远程文件服务（NFS）；流式多媒体（专用协议）；IP 电话（专用协议）；网络管理（SNMP）；路由选择协议（RIP）；和名址转换（DNS）等。其特征是要求快捷、可以不必考虑在传输中的可靠服务等场合。

3.3， **[UDP 和 TCP 的校验和问题]** **UDP 和 TCP 采用和的反码来计算 16 位字的校验和。**

（1） 本题为了计算简单起见，只采用 **8** 位来计算。假定有以下的三个 **8** 位字节：**01010101**，**01110000**，**01001100**，请你写出计算其校验和的全过程。

（2）UDP 为什么要采用和的反码，而不直接采用和本身？

（3）采用这类反码方案，接收方如何检测出差错？能够查出 **1** 位错？能否查出 **2** 位错？

[解答] 01010101　　{1} 左算式表明校验和为 **11101101**；

01110000 +　　(2) 采用反码的主要原因是：在数据即使是全零的情况下，

11000101　　校验和不再是全零，容易检测差错的发生。

11000101　　(3)接收方可以把收到的数据再次作检验和，与送来的校验和比较判

01001100 +　　别差错。

00010010[加进位 **1** 回绕后]　　，总能查出 1 位错；

11101101 [反码]　　还能查出大部分的 2 位错（存在不能查出的一些场合）。

RDT 可靠数据传输协议

3.4，请简要小结：在可靠数据传输协议 **rdt2.0**、 **rdt2.1**、**rdt2.2** 和 **rdt3.0** 中，分别采用了什么措施来解决相应的传输差错的。

[解答] 这几个版本都在《停止等待》前提下工作:

rdt2.0 对付位差错----采用《检查和》， 使用反馈 ACK/NAK，出错时发送方重传分组。

rdt2.1 和 2.2 对付 ACK/NAK 的位差错 --- 采用序号(0/1 号)机制。
rdt3.0 对付分组丢失 --- 采用《超时重传》机制。
3.5 画出并简要说明 rdt3.0 接收方的 FSM 有限状态机。
[解答]不妨认为有两个状态：状态 0 和状态 1。用动作 0—1 表示从状态 0 转向状态 1；用动作 0—0 表示从状态 0 仍回到状态 0；类似地可有动作 1—0 和动作 1—1。
状态 **0**：等待来自下层的 **0** 号分组
事件 0—1：调用 rdt3.0 接收分组,且收到的数据分组正确未受损并且是 **SEQ#0** 分组
rdt_rcv(rcvpkt) && notcorrupt(rcvpkt) && has_seg0(rcvpkt)
动作 0—1：从分组中提取出数据 extract(rcvpkt, data)
把数据递交到上层 deliver_data (data)
组成 ACK0 分组 sndpkt= make_pkt(ACK,0,checksum)
调用下层发送 ACK 反馈 udt_sent(sndpkt); 转向状态 **1** [等待 **1** 号分组]
事件 0—0：调用 rdt3.0 接收分组,且收到的数据分组受损或者是 SEQ#1 分组
rdt_rcv(rcvpkt) && (corrupt(rcvpkt)||has_seq1(rcvpkt))
动作 0—0：重复地发送已发送过的 ACK0： udt_sent(sndpkt);
序号不对，仍发 **ACK** [发送方不处理 **ACK**]； 仍处在状态 **0** 不变。
状态 **1**：等待来自下层的 **1** 号分组；
完全类似地有：事件 1—0，动作 1—0；和事件 1—1，动作 1—1。

3.6， 请参照 rdt2.0~rdt3.0 指出引起下列差错的可能原因：
（1）传输中受损的报文段，（2）丢失的报文段，（3）重复的报文段，（4）乱序的报文段，（5）受损的确认，（6）丢失的确认，（7）重复的确认。
[解答]为了加深理解起见，不妨参照 rdt2.0~rdt3.0 或 TCP 中的处理方案来回答。
[1]传输中受损的报文段：受损 Corrupt 指报文中的二进位发生差错（0→1 或者 1→0；个别位或者许多位）；原因是：通信信道上“噪声”（热噪声或冲击噪声）、干扰等导致物理信号的破坏或信号畸变等，从而引起的一类传输差错。
[2]丢失的报文段：丢失 Lost 的主要可能原因包括传输有关的软硬件故障，或者是在传输途中任何一站（如`路由器、主机等）或接收方来不及处理所造成的缓冲站队列溢出等。
[3]重复的报文段：原因之一是：如果某个报文段的发送过程被延迟较长时间但并未丢失，会引起“过早超时”现象，超时重传就会引起接收方收到重复的冗余报文段（多余的副本）；原因之二是：某一个分组已被接收方正确收到但是接收方发向发送方的确认 ACK 丢失（或者延迟了较长时间），发送方还是认为接收方尚未能收到，于是发送方启动超时重传，引起接收方收到了重复的报文段。
[4]乱序的报文段：在非停等待的流水化连续传送报文的情况下（不论是 GBN 还是 SR），某个报文段的丢失或者较长延迟，都会引起下一报文段被接收方先收到的现象，引起接收方收到乱序的报文段。
[5]丢失的确认：丢失原因相同于丢失的报文段。
[6]受损的确认：受损原因相同于受损的报文段。
[7]重复的确认报文段：分两种情况，一是“过早超时”的重传引起的重复确认（第一次确认来得迟了）；二是对于乱序的报文段，接收方采用最后一次正确确认来回答（通常会丢弃乱序报文）。

3.7，请简要说明对于在 **3.6** 题中所列出的各类差错，在 **TCP** 类的协议中分别采用的相应差

错控制方法。（提示：建议用 **rdt2.0** 到 **rdt3.0** 中所用的对策简要说明即可）。
[解答]为了加深理解起见，不妨参照 rdt2.0~rdt3.0 或 TCP 的处理方案来回答。
[1]传输中受损的报文段相对应的差错控制方法：接收方采用“检验和”来发现此类差错，与发送方配合用 ARQ 自动重传请求来试图纠正该类差错。[在 rdt3.0 中接收方校验后不发确认,由发送方超时重传。]
[2]丢失的报文段相对应的差错控制方法：采用确认机制，由发送方发现丢失并重传，发送方采用定时器机制，在超时还未能收到 ACK 确认时，发送方重传该丢失的报文段。（为了避免过早超时而引起多发重复的报文段，一般应采用序号机制加以辅助，用序号来判别报文段是否重复。）
[3]重复的报文段相对应的差错控制方法：采用序号机制，接收方检查报文段的序号，可发现是否收到了重复的报文段。一旦收到重复报文段，接收方可重发已确认过的 ACK，使发送方不产生误解。
[4]乱序的报文段相对应的差错控制方法：用序号和窗口机制来发现和控制。比如 GBN 场合接收方一般会忽略乱序的报文段；SR 场合则会确认接收正确的乱序报文段，在由接收进程真正接收之前交由缓冲区暂时存放，需加上收发窗口的辅助控制。
[5]丢失的确认相对应的差错控制方法：在 rdt3.0 中由发送方发现与处理。作为丢失报文一样地超时重传。
[6]受损的确认相对应的差错控制方法：在 rdt3.0 中由发送方发现与处理。作为丢失确认一样地超时重传。
[7]重复的确认报文段：作为重传的策略来实施。比如第一种场合下，发送方可不处理。第二种场合，发送方应按序重发。
3.8 请简要说明当发送方窗口与接收方窗口的长度都是 1 时，比特交替协议与 GBN 协议相同，同样也与 SR 协议相同。
[解答]注意到在窗口大小是 1 的场合，GBN、SR 与比特交替协议在功能上是等价的。在窗口大小为 1 时，排除了在窗口内有乱序分组的可能性；一个累积 ACK 在这个场合下也等同于一个常规的 ACK，因为它只可能指向在窗口内的单个分组。

3.9 举例说明在 SR 协议（与 GBR 协议）中，发送方有可能会收到落在其当前窗口之外的分组的 ACK。（为简单起见，不妨可令发送窗口大小为 3）
[解答]比如，假定发送方有窗口大小为 3，并在 t0 时刻发送了分组 1、2 和 3； 在 t1（t1>t0），接收方发 ACK1、2、3； 在 t2(t2>t1) 发送方超时并重新发送分组 1、2、3； 在 t3，接收方收到了重复的分组 1、2 和 3； 在 t4，发送方收到了接收方在 t1 发送的 ACK，并推进其窗口到 4、5、6； 在 t5，发送方收到了接收方在 t2 后发送的 ACK1、2、3，这些 ACK 处于其窗口之外了。

TCP
3.10，为什么一个 TCP 报文段所携带的用户数据最多为 65515B？为什么 TCP 头部中最多只能有 44B 的 TCP 选项？
[解答] 2^{16}=65536B， 65535B－20B（最小首部）= 65515B。
TCP 报文段中的“首部长度位”4bits，32 位=4B； $2^4 \times 4B = 64B$。
64B － 最小必要首部 20B = 44B。

3.11 考虑从主机 **A** 向主机 **B** 传输 **L** 字节的大文件,假定 **MSS** 等于 **1460** 字节.
[1]在 **TCP** 序号允许的范围内,文件长度 **L** 的最大允许值是多少**?[TCP** 的序号字段是 **4** 个字

节].

[2]请估算出该最大文件需要化多长的时间?假定传输层、网络层和数据链路层总共加在没个报文段首部上的长度是 66 个字节；传输分组的链路速率是 10Mbps；不采用流量控制和拥塞控制，因此主机 A 能够一个接一个地连续不断发送报文段。[注: 2^{32}= 4,294,967,296]

[解答] 2^{32}= 4,294,967,296 个可能的序号.

[1]在 TCP 中,序号不是随着每一段增加 1,而是需增加该段所发送的数据的字节数;因此序号与 MSS 无关. 从 A 到 B 可发送的最大的文件的大小简单地就是由 2^{32}= 4,294,967,296≈4. 19G 字节可表示出的字节数.

[2]MSS=1460, 报文段的个数是⌈2^{32}/1460⌉=2, 941, 758 个, 首部的 66 个字节要加到每一报文段上去, 即首部引起加上 66×2, 041, 758= 194, 156, 028 个字节. 从而,

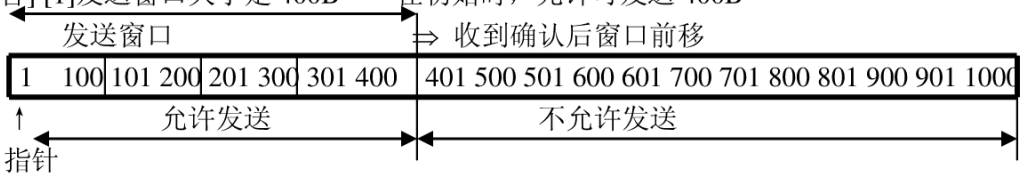
总的传输字节数是: 2^{32} +194, 156, 028 字节 =3591×10^7位.

因此, 采用 10Mbps 链路, 要花 3591 秒= 59 分钟来传输该文件.

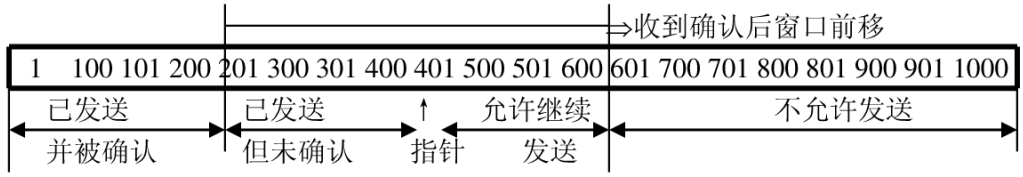
3.12，如果在建立 TCP 连接过程中收发双方协商的初始窗口大小是 400B，使用 1 到 1000 作为分析窗口的基本范围，请你画出发送窗口以下的 3 个演变场景：（1）在初始时，允许可发送 400B；（2）已发送了 400B，其中序号 1 到 200 的报文已收到了接收方的正确确认；（3）发送方已收到来自于接收方对序号为 201 到 400 的报文的确认，并且还收到接收方通知窗口扩大为 500B。

要求在这三种场景中都标出：[1]可发送的起始字节的指针箭头；[2]已发送并确认的范围；[3]已发送但未被确认的范围；[4]允许可继续发送的范围；[5]不可以发送的范围；[6]发送窗口范围；[7]收到确认后窗口将要前移的起点处的方向箭头。

[解答] [1]发送窗口大小是 400B　　在初始时，允许可发送 400B



[2]　已发送了 400B，其中序号 1 到 200 的报文已收到了接收方的正确确认



[3]收到的确认序列号为 401，窗口增大为 500，还可以发送 500B

3.13，假定序号空间大小为 2^K，请举例说明 GBN 方案的滑动窗口的大小 W 应不大于 2^K －1；而 SR 方案的滑动窗口的大小 W 应不大于 2^{K-1}。

[解答]本题只要求了解一下结论，不作详细推导要求。理解此结论可参阅讲稿 3.4 小节的片子中“GBN 中窗口大小”一页，和 SR 的最后一张片子（也可见有关的图 3.27）。

3.14，请说明为什么 TCP 协议要求对每一个 TCP 数据字节都要进行编号？顺序号与确认号各有什么用途？

[解答]TCP 的基本特征是采用可靠的按序发送的面向字节流的机制。

由于报文段到达可能为乱序，因此需要按发送顺序编号各个报文段。

TCP 按顺序编号 每一个 8 位二进位组[字节]，把报文段中第一个字节的编号作为该报文段的编号。

TCP 把要传输的整个报文（可能包括几个报文段）当作是一个个字节组成的字节流。即 TCP 采用字节流，允许无“报文段规定长度的边界标记”，提供了组成报文段的长度的很大灵活性。

由于可靠传输、流量控制、拥塞控制和连接管理等最好都用报文段序号机制辅助，因此，TCP 协议要求对每一个 TCP 数据字节都要进行编号。

在 TCP 报文段的头部，有发送序号字段：它占 4 字节，指出报文中数据在发送方的数据流中的位置。[用字节编号]

也有确认序号字段：它也占 4 字节，指出接收方希望下一次接收的字节序号。---期待接收的序号。可以非常方便地实现累积确认（TCP 的确认是对接收到的数据的最高序号，即收到的数据流中的最后一个序号，表示确认）。

发送顺序号和确认序号 主要用于可靠的数据传输。

3.15，请举例说明使用两次握手来建立 TCP 连接可能会出现的不正常情况。

[解答]考虑主机 R 和 S 之间的连接通信的例。假定主机 R 向 S 发出一个请求连接的分组，主机 S 收到后允许连接，并且 S 发送《确认 ACK 应答分组》，如果是《两次握手》协议，S 认为连接已成功，从而主机 S 也可以开始向 R 发送《数据分组》了。

但是，这时如果发生了 S 向 R 回应的《确认 ACK 应答分组》丢失的情况，主机 R 就不知道主机 S 是否已准备好接收或发送《数据分组》；也不知道 S 建议采用什么序列号用于 S 到 R 的数据通信，同样也不知道 S 是否同意（确认）由 R 所建议的 R 通信用初始序列号；R 甚至于怀疑 S 是否已收到自己发给它的《连接请求分组》？------ 在这种情况下：

主机 R 只能认为连接尚未建立成功，将会忽略 S 发来的任何《数据分组》；[只是继续耐心等待接收 S 同意连接的那个被丢失了的《确认 ACK 应答分组》]。

而主机 S 则认为连接已成功，并且 S 已发出《数据分组》，一旦发出的数据分组超时（因主机 R 认为连接未成功，从而忽略 S 发来的数据分组），S 再不断重复地发送同样的《数据分组》，这时，将会形成了《死锁》的局面。

3.16，考虑从美国西海岸到东海岸的端系统网络中发送数据分组，已知条件：带宽 R=1Gbps 的链路，15ms 端到端的传播时间，1KB 大小的分组（L）；如果要求信道利用率大于 90%，请大致估算出需要使窗口达到多大？

[解答]从已知条件： 15ms 端到端传播时间，即 RTT= 30ms；

带宽 R=1Gbps 的链路，L=1KB 分组大小； 这时，传输发送时间= L/R= 0.008ms ；

令 W 为窗口大小，

粗糙地看， 信道利用率 U = [W*(L/R)] / (RTT+L/R) >0.9

窗口大小 W> 0.9*30.008/(0.008)=3376。

3.17 考虑 TCP 估算 RTT 的过程。令 X=α=0.1,假定 SampleRTT1 是最新采样的 RTT, SampleRTT2 是下一个最新采样的 RTT,依此类推。

[1]对于一个给定的 TCP 连接，假定有 4 个确认报文相继到达，带有 4 个对应的 RTT 值：SampleRTT4、SampleRTT3、SampleRTT2 和 SampleRTT1。请用这 4 个采样 RTT 来表示出 EstimateRTT。

[2]把[1]中的公式推广到 n 个 RTT 样本的场合。

[3]在[2]得到的公式中，令 n→∞，说明这个过程可被称为指数移动平均。

[解答][1]令 EstimateRTT（n）表示第 n 次采样后估算的 RTT；

EstimateRTT（1）=SampleRTT1；

EstimateRTT（2）= X SampleRTT1+（1-X）SampleRTT2；

EstimateRTT（3）=X SampleRTT1+（1-X）[X SampleRTT2+（1-X）SampleRTT3]；

=X SampleRTT1+（1-X）X SampleRTT2+ （1-X）^{2} SampleRTT3；

EstimateRTT（4）= X SampleRTT1+（1-X）EstimateRTT（3）

= X SampleRTT1+（1-X）X SampleRTT2+ （1-X）^{2} X SampleRTT3+ （1-X）^{3} SampleRTT4；

[2] EstimateRTT（n）= X$\sum_{j=1}^{n-1}$ （1-X）$^{j-1}$ SampleRTTj +（1-X）$^{n-1}$ SampleRTTn。

[3]当 n→∞时，（1-X）$^{n-1}$→0；EstimateRTT（n）= X$\sum_{j=1}^{n-1}$ （1-X）$^{j-1}$ SampleRTTj

= X/（1-X）×$\sum_{j=1}^{n-1}$ （1-X）^{j} SampleRTTj。当前 X=0.1，

EstimateRTT（n）=1/9$\sum_{j=1}^{n-1}$ (0.9)^{j} SampleRTTj，表明对采样 RTT 的权重呈指数衰减。

3.18，如果一条 TCP 连接的当前往返时间 RTT=30ms，并且分别在相隔时间量 26ms、32ms 和 28ms 后收到确认，请你估算出新的 RTT 值。（假定α =0.1）。

[解答]Estimated RTT= （1－α ）×老的估算 RTT + α ×采样 RTT

Estimated RTT1= 0.9×30+ 0.1×26 = 27+2.6 = 29.6ms；

Estimated RTT2= 0.9×29.6+ 0.1×32=26.64+3.2 = 29.84ms；

因此，Estimated RTT= 0.9×29.84 + 0.1 ×28 =26.856 + 2.8 = 29.656ms。

3.19，如果一条 TCP 连接当前估算的往返时间 RTT=20ms，当前估算 RTT 的偏差值是 2 毫秒，此后，分别依次在相隔时间量 25ms、30ms 和 20ms 后又收到确认，请你估算出用于超时的时间量设置值。（假定α =0.1，β=0.2）。

[解答]先估算出新的 RTT，每次同时 RTT 估算偏差值。最后用估算公式估算用于超时的时间量设置值。

新的 Estimated RTT= （1－α ）×老的估算 RTT + α ×采样 RTT

新的 RTT 偏差值 DevRTT= （1－β ）×老的 DevRTT + β ×| Estimated RTT－采样 RTT|；

TimeoutInterval= EstimatedRTT + 4×DevRTT；

因此有，

新的 Estimated RTT1= （1－0.1）×20 + 0.1×25=18+2.5=20.5ms;

新的 RTT 偏差值 DevRTT1= （1－β ）×老的 DevRTT + β ×| Estimated RTT－采样 RTT|；

=(1-0.2)×2ms+0.2×(25-20.5)ms = 1.6+0.9 = 2.5ms;

新的 Estimated RTT2= （1－0.1）×估算 RTT1(20.5) + 0.1×30=18.45+3=21.45ms;

新的 RTT 偏差值 DevRTT2= （1－0.2）×DevRTT1(2.5ms) + 0.2×|21.45－30|=

= 2+1.71=3.71ms；

新的 Estimated RTT3= （1－0.1）×估算 RTT2 (21.45ms)+ 0.1×采样 RTT(20ms)

= 0.9*21.45+2=21.305ms;

新的 RTT 偏差值 DevRTT3= （1－0.2）×DevRTT 2(3.71ms)+ 0.2×|21.305－20|

= 2.968+0.261=3.229ms；

超时时间量可设为 TimeoutInterval= EstimatedRTT + 4×DevRTT

=21.305+4×3.229=21.305+12.916= 34.221ms。

3.20，假定在建立 TCP 连接时，主机 A 和主机 B 几乎同时发出连接请求，似乎会发生冲突，请画出如果主机 A 的请求略快一点点时连接过程发生的场景

[解答]从概念上看，允许有以下三类可能的理解：

（1）A 与 B 之间只需要建立一个双向的 TCP 连接，因此 A 或 B 可以拒绝多余的连接；

（2）A 和 B 分别请求与另一服务器 S 作 TCP 连接；这与一下的（3）类似。

（3）主机 A 和 B 确实分别有着向对方的不同 TCP 连接要求；以下将按（3）叙述：

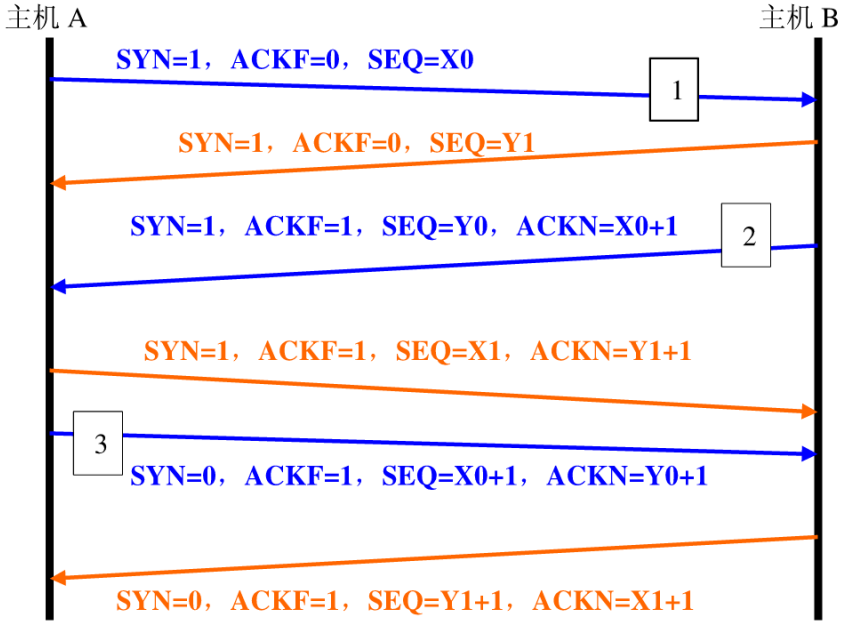
[1]允许两台主机中有多个应用进程分别建立 TCP 连接和交换数据。

[2]在建立连接期间允许分别采用不同的建议 SEQ 序号和相应的确认序号 ACKN（简记 AN）；为了明晰起见，把“**ACK** 标记位”记成 **ACKF**，确认序号记为 **ACKN**（仅当 **ACKF=1** 时，**ACKN** 确认序号才有意义）。

比如：主机 A 请求建立连接时第 1 次握手通告建议的 SEQ 序号是 X0[SYN=1、ACKF=0、SEQ=X0]，第 2 次握手主机 B 确认时通告的 SEQ 序号是 Y0、AN 确认序号=X0+1[SYN=1、ACKF=1 、SEQ=Y0 、AN=X0+1]，第 3 次握手则有主机 A 确认 SEQ=X0+1 和 AN=Y0+1[SYN=0、ACKF=1、SEQ=X0+1、AN=Y0+1]；

而另一个由主机 B 发出的请求建立连接其第 1 次握手通告的 SEQ 序号是 Y1[SYN=1、ACKF=0、SEQ=Y1]，第 2 次握手主机 A 确认时通告的 SEQ 序号是 X1、AN 确认序号=Y1+1[SYN=1、ACKF=1、SEQ=X1、AN=Y1+1]，第 3 次握手则有主机 B 确认 SEQ=Y1+1 和 AN=X1+1[SYN=0、ACKF=1、SEQ=Y1+1、AN=X1+1]。

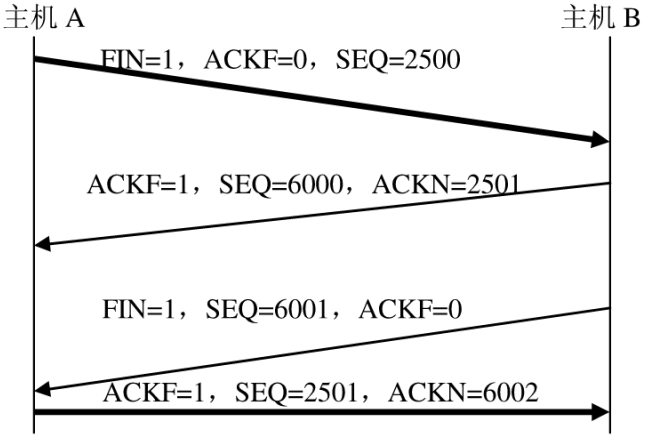
如下图所示：



3.21，假定主机 A 请求释放的报文段序号 SEQ=2500，主机 B 应答释放的报文段序号 SEQ=6000，请你详细画出在主机 A 和主机 B 之间的进行 TCP 传输时，连接释放的交互过程。其中也要列出所有的有关的标记位。

[解答] 为了明晰起见，把“**ACK** 标记位”记成 **ACKF**，确认序号记为 **ACKN**（仅当 **ACKF=1**

时，**ACKN** 确认序号才有意义）。



3.22 请回顾 TCP 吞吐能力的描述,假定在连接速率从 W/(2RTT)变化到 W/RTT 的期间内,只丢失了一个分组(在该期间的结束时),

[1]证明丢包率 L = 1/ [3/8 ×W^2 + 3/4 ×W];(提示先计算出在此期间发送的分组个数)

[2]如果一个连接的丢包率是 L,采用[1]中公式的近似表达式(认为 W 很大),请证明平均带宽大体上应为 (1.22×MSS)/[RTT×SQR(L)];

(此公式中指 L 开平方,提示采用平均吞吐率公式)

[3]假定针对 1500 字节的分组长度和一个 100 毫秒的 RTT,TCP 需要支持 10Gbps 的连接,所能容忍的丢包率是多少?[提示采用(2)中得到的公式]

[解答][1]丢包率 L 是分组丢失数与已发送的分组数之比；在此期间发送的分组个数是:

W/2+(W/2 +1)+(W/2 +2) + …+W = $\sum^{W/2}_{n=0}$ (W/2+ n)= (W/2+1)×(W/2)+$\sum^{W/2}_{n=0}$ n

=W^2/4+W/2+(W/2)×(W/2+1)/2 = 3W^2/8 + 3/4×W;

在此期间丢失 1 个分组，因此，丢包率 L = 1/[3/8×W^2+ 3/4×W].

[2]对于很大的窗口 W，3/8W^2>>3/4 W；L 近似于 8/(3W^2)； 即 W= SQR[8/(3L)](开平方)；

平均吞吐率 = 3/4×W×(MSS/RTT) = 3/4×SQR[8/(3L)]×[MSS/RTT]

= 1.22×MSS/[RTT×SQR(L)];

[3]采用(2)中的公式,10Gbps= 1.22×MSS/[RTT×SQR(L)]

= 1.22×(1500×8 位)/[0.1 秒×SQR(L)];

SQR(L)= 14640 位/10^9位； L= 2.14×10^{-10} .