

**《计算机网络》课程实验报告**

**实验3-1**

****

学 院 网络空间安全学院

专 业 信息安全

学 号 2112060

姓 名 孙蕗

班 级 信息安全1班

1. **实验要求**
2. 作业要求：利用数据报套接字在用户空间实现面向连接的可靠数

据传输，功能包括：建立连接、差错检测、接收确认、超时重传

等。流量控制采用停等机制，完成给定测试文件的传输。

➢ 数据报套接字：UDP

➢ 协议设计：数据包格式，发送端和接收端交互，详细完整

➢ 建立连接、断开连接：类似TCP的握手、挥手功能

➢ 差错检验：校验和

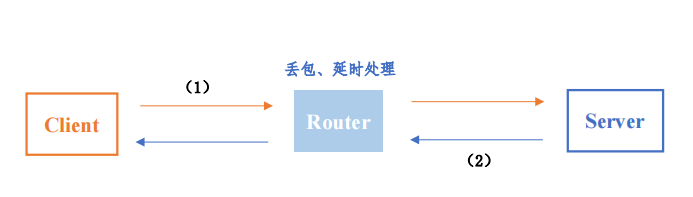
➢ 接收确认、超时重传：rdt2.0、rdt2.1、rdt2.2、rtd3.0等，亦可自行设计协议

➢ 单向传输：发送端、接收端

➢ 日志输出：收到/发送数据包的序号、ACK、校验和等，传输时间与吞吐率

➢ 测试文件：必须使用助教发的测试文件（1.jpg、2.jpg、3.jpg、helloworld.txt）

1. 路由器程序（该程序类似于一个代理服务器）主体为一个转发线程，线程不停获取发向路由器的数据包，通过 IP 地址和端口号判断是客户端还是服务器端发来的包，若为客户端发来的包，则进行丢包、延时处理后发向服务器端；若为服务器端发来的包，则不进行处理，直接转发给客户端。



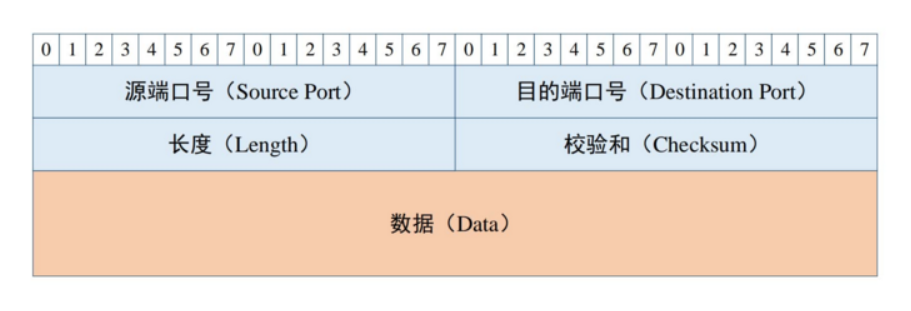
1. 客户端发出的包，目的 IP，端口号请设置为路由器的 IP 和端口号（即在程序界面中设置的）；
2. 服务器端发出的包，目的 IP，端口号请设置为路由器的 IP 和端口号（即在程序界面中设置的）；

丢包率为 double 类型，延时为 int 类型，单位为 ms，设置好丢包率后，日志中会打印出参数 misscount，“miscount：n”意味着每过 n 个包丢一个包，每次丢包或延时均会打印日志。

1. **程序设计**
2. **数据报套接字**

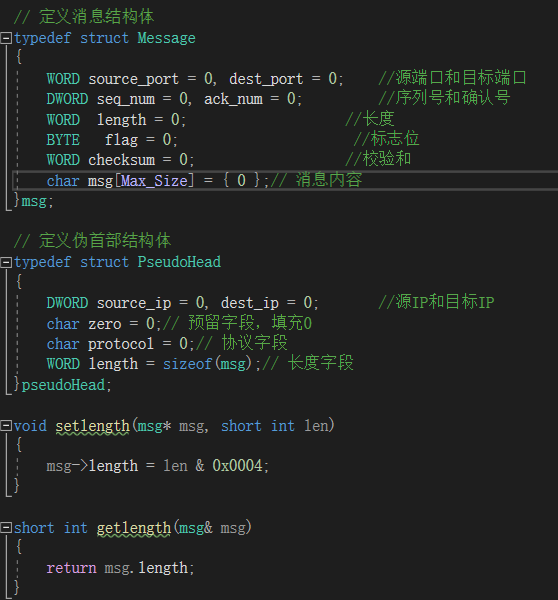
根据实验要求数据报套接字采用UDP。

UDP是无连接、不可靠、面向数据报的传输层通信协议。不需要建立连接，发送端和接收端不需要维护连接状态。无拥塞控制，数据报头部短，额外开销小，具有简单、传输效率较高的特点。



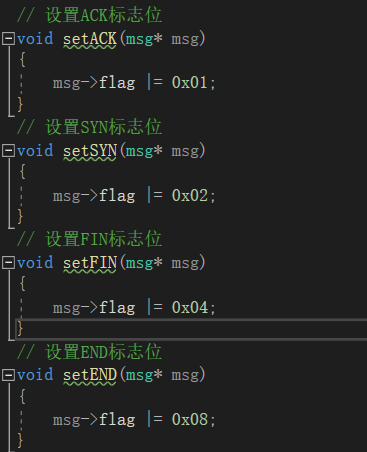
UDP的首部有4个字段，每个字段由两个字节构成。其中长度字段主要是记录UDP报文段中的字节数（首部+数据段），检验和字段的作用是接收方收到UDP报文后检查该报文段中是否出现了差错。

1. **协议设计**

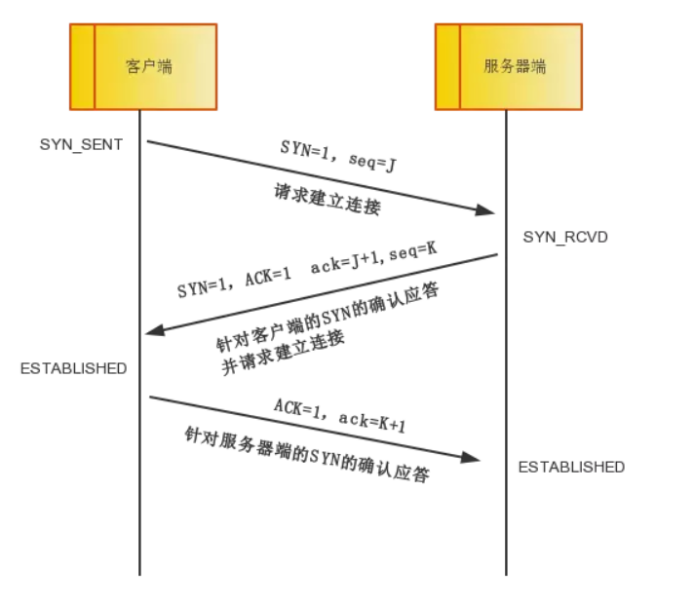


数据包格式：含有2字节的源端口，2字节的目标端口，4字节的seq，4字节的ack，2字节的长度，2字节的校验和，1字节的标志，若干字节数据。长度为传输报文时，记录当前报文所传输数据的有效字节数。

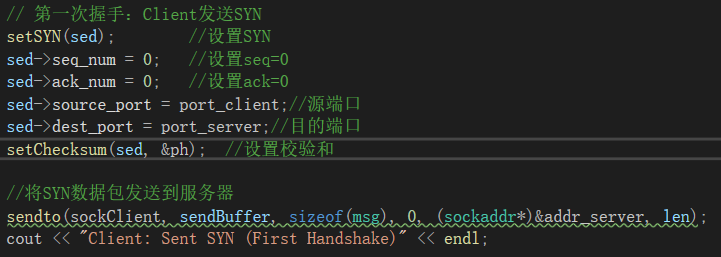
标志位flag：从低到高分别是ACK、SYN、FIN、END。END为传输文件结束后发送报文的标志位。ACK=0x01, SYN=0x02, FIN=0x04, END=0x08。



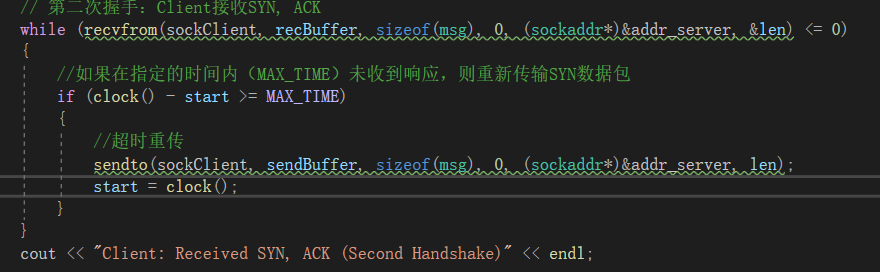
1. **建立连接断开连接（类似TCP的握手和挥手）**
2. **三次握手：**

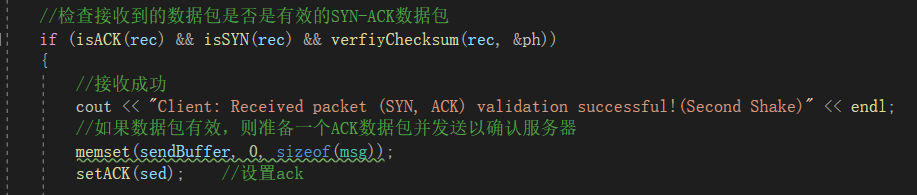


1. **客户端：**
2. 客户进程创建传输控制块TCB，发送第一次握手同步报文，标记位为SYN=1,seq=0,ack=0，客户端进程进入了 SYN\_SENT 同步已发送状态。客户端向服务器端发送报文，证明客户端的发送能力正常



1. 开始计时，接收服务器第二次握手报文，若超时则重传同步报文。

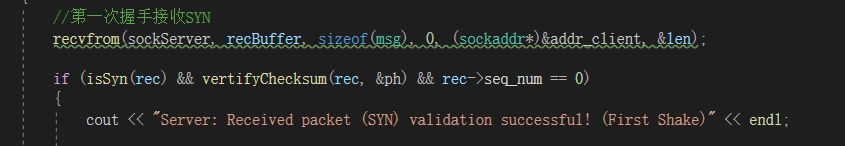


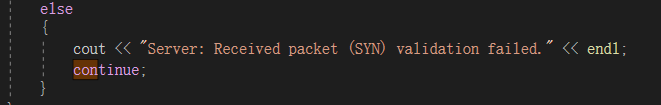


1. 判断接收的报文是否为标志位（SYN，ACK），seq=0，ack=1。若是,发送第三次握手的报文，向服务器确认：标志位ACK=1，seq=1，ack=1，客户端进入ESTABLISHED已建立连接状态，否则返回退出。第三次握手中客户端向服务器发送报文，服务器确认客户端的接收能力正常。

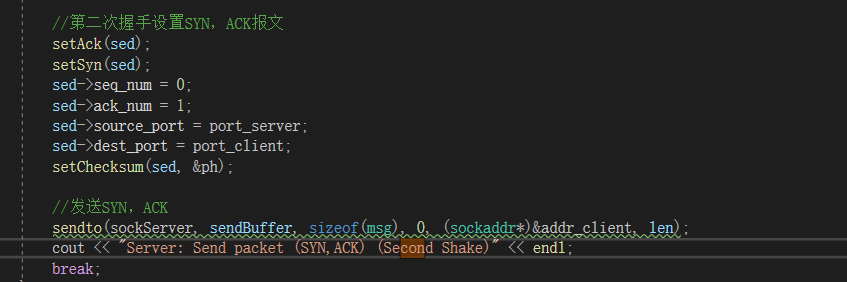


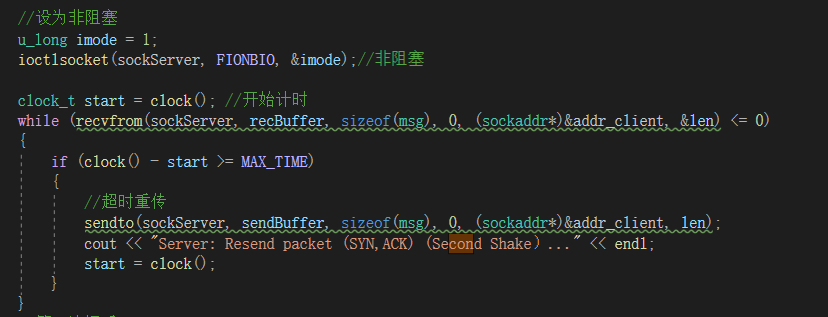
1. **服务器端：**
2. 阻塞，接收客户端第一次握手报文，如果是SYN=1,seq=0,ack=0。则发送报文:标志位（SYN=1，ACK=1），seq=0，ack=1；否则循环继续等待接收报文。



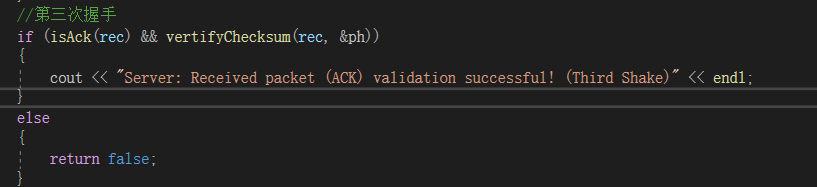


1. 发送第二次握手报文。非阻塞，开始计时，接收客户端第三次握手报文，若超时则重传第二次握手报文，服务器进程进入了 SYN\_RCVD 同步收到状态。第二次握手中，服务器端接收到报文并向客户端发送报文，证明服务器端的接收能力、发送能力正常，客户端得出客户端发送接收能力正常，但此时服务器不能确认客户端的接收能力有没有问题。

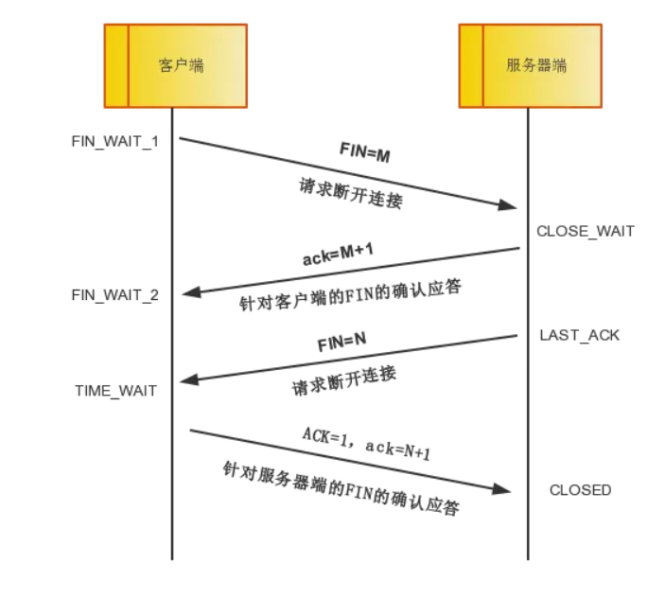




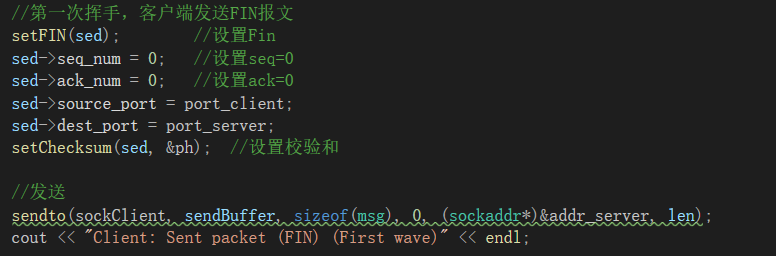
1. 判断接收的客户端第三次握手报文是否为：标志位（ACK=1），seq=1，ack=1。若是，连接成功。否则返回退出。



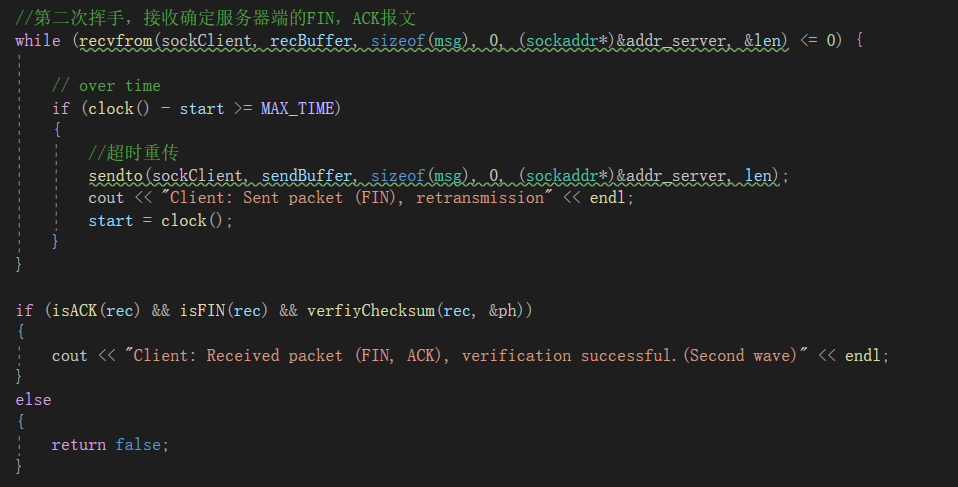
1. **四次挥手**



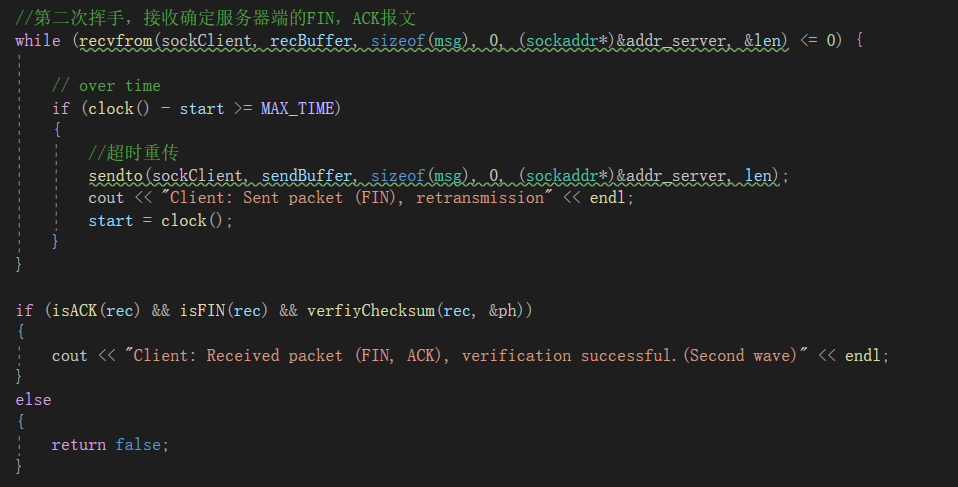
1. **客户端:**
2. 发送结束报文，第一次挥手标记位为FIN，客户端进入FIN\_WAIT\_1（终止等待1）状态。



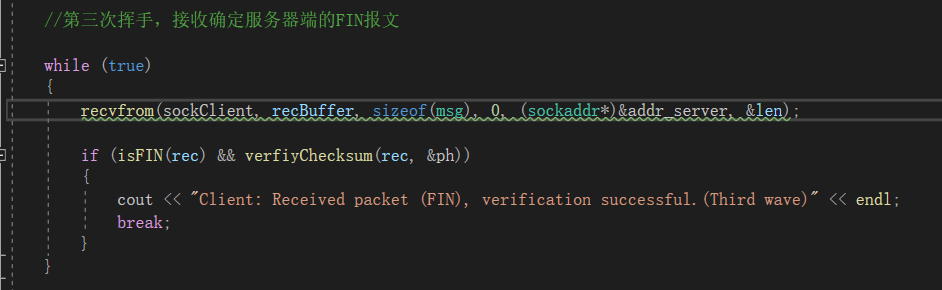
1. 开始计时，接收服务器第二次挥手报文，若超时则重传第一次挥手报文。

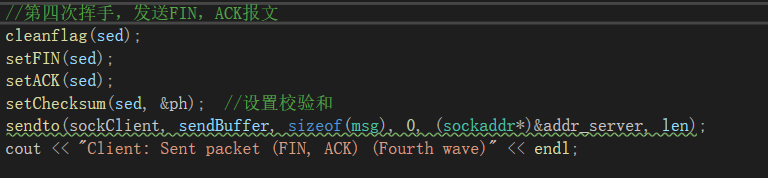


1. 判断接收的服务器端的第二次挥手报文是否为：标志位（FIN，ACK）。否则继续接收。客户端接收到服务器端的确认请求ACK=1后，客户端就会进入FIN\_WAIT\_2（终止等待2）状态，等待服务器发送连接释放报文。

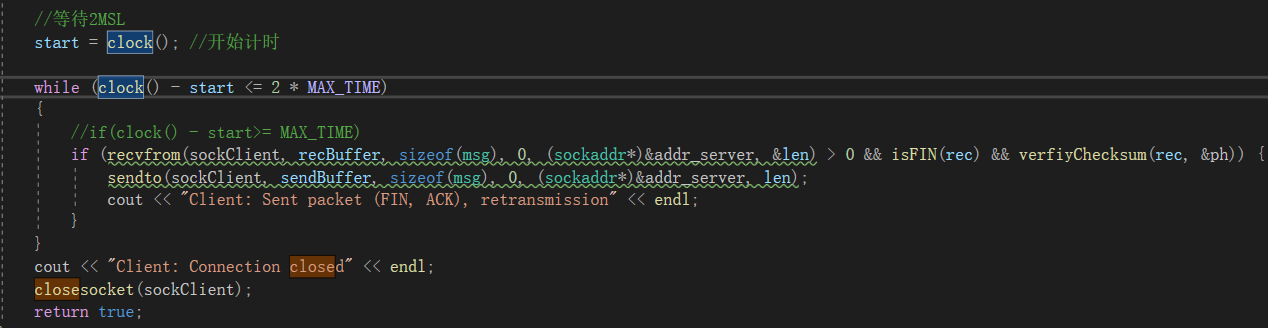


1. 阻塞，接收第三次挥手报文，判断是否为：标志位（FIN）。若是，发送第四次挥手报文，标志位（FIN，ACK）。否则继续接收。

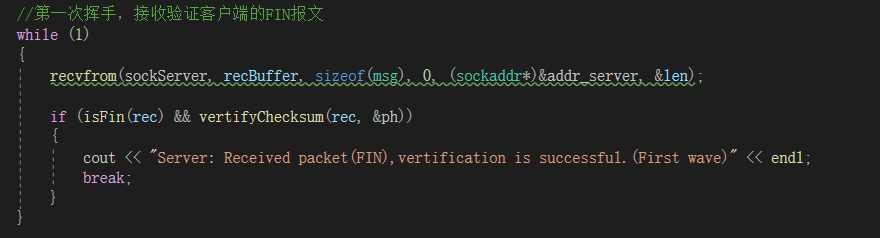


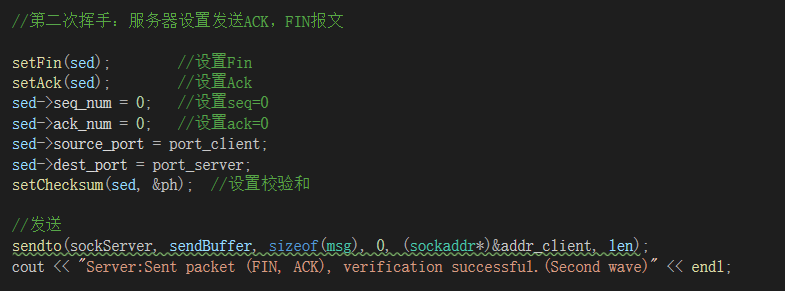


1. 客户端就进入了TIME\_WAIT（时间等待）状态，但此时TCP连接还未终止，经过2MSL后（最长报文寿命），当客户端撤销相应的TCB后，客户端才会进入CLOSED关闭状态。等待2MSL，如果在2MSL中收到服务器的FIN报文，重传第四次挥手确定报文。

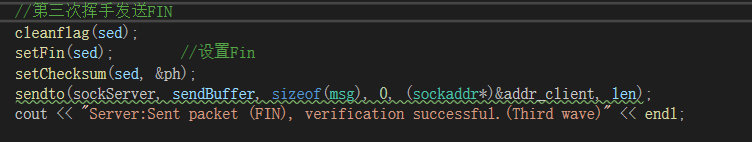


1. 返回退出。
2. **服务器:**
3. 阻塞，接收客户端第一次挥手报文，如果是FIN，发送第二次挥手报文标志位（FIN，ACK），服务端就进入了CLOSE\_WAIT 关闭等待状态。若服务器端还有数据要发送给客户端，客户端还会接受，服务器端会持续一段时间。否则循环继续等待接收第一次挥手报文。

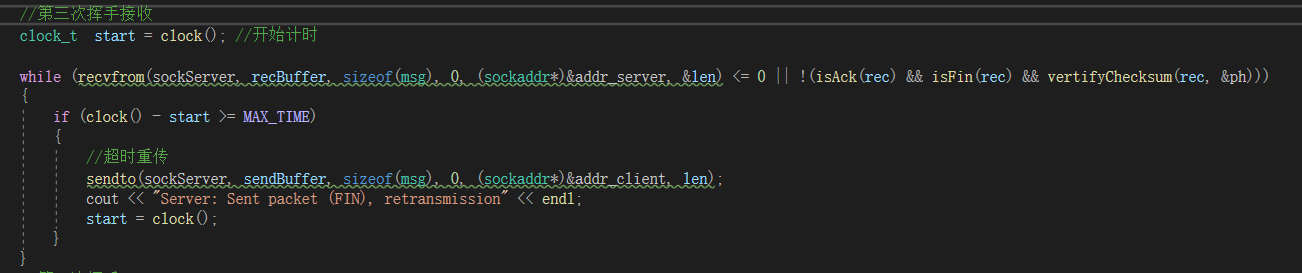


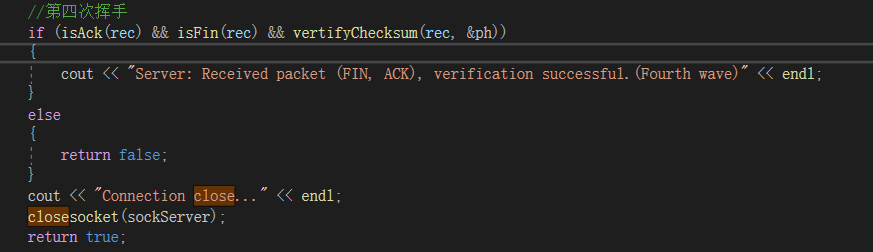


1. 若无要传输的数据，发送第三次挥手FIN报文。服务器将最后的数据发送完毕后，就向客户端发送连接释放报文FIN=1，服务器就进入了LAST\_ACK（最后确认）状态。



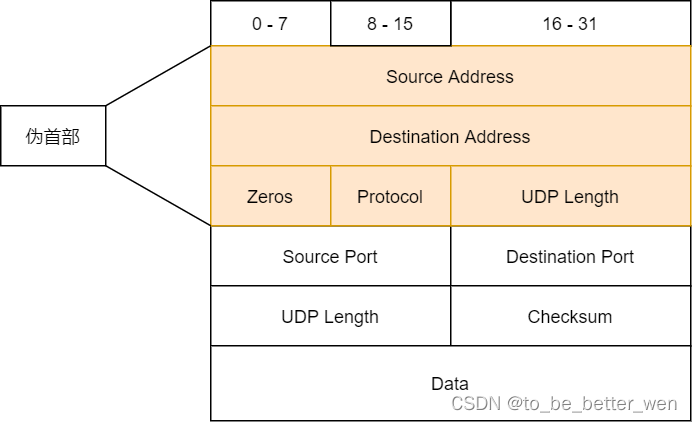
1. 非阻塞，超时重传第三次挥手FIN报文。判断接收的第四次挥手报文是否为(FIN,ACK)，如是则马上断开连接，立即进入CLOSED关闭状态。



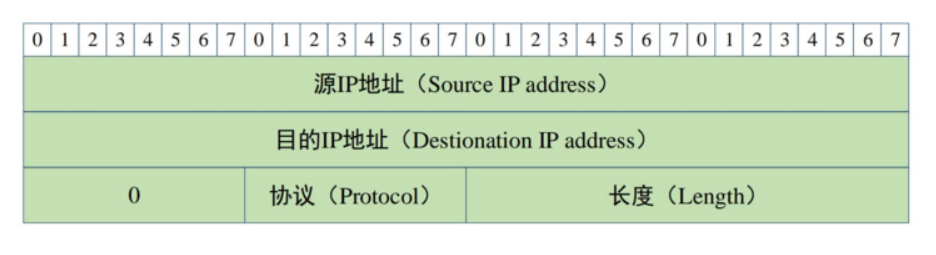


1. **差错检验：校验和**

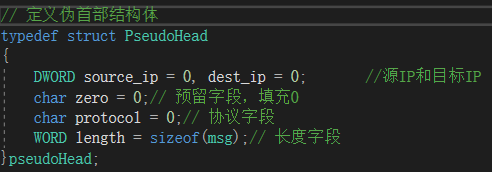
UDP数据校验和和TCP数据的校验和计算的方法是一致的，包括：UDP伪首部，UDP头部和UDP数据。

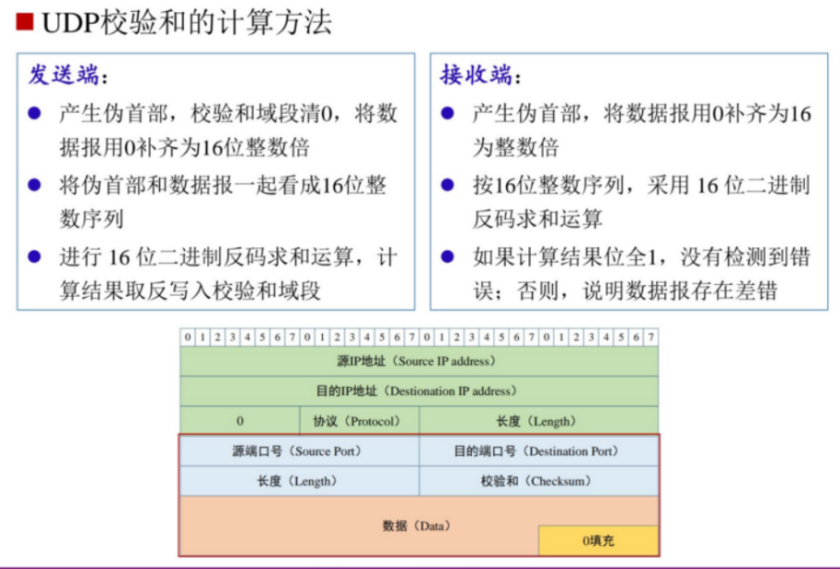


UDP的伪首部：



自定义的伪首部包含2字节的源端口，2字节的目标端口，2字节长度，版本号和填充0。

****

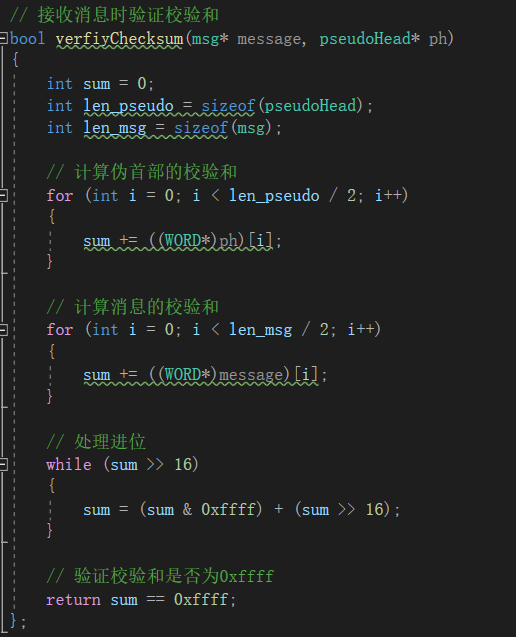




在发送数据时，发送端利用自己产生的伪首部和发送的UDP数据报计算校验和，按如下步骤计算校验和：

1. 把校验和字段设置为0；
2. 伪首部和消息的每两个字节作为16位字进行累加。如果有进位，就将进位部分加回到sum的低16位中。

③ 把累加结果取反存入校验和字段中



在接收数据时，接收端利用自己产生的伪首部和接收的UDP数据报计算校验和，按如下步骤计算检验和：

1. 伪首部和消息的每两个字节作为16位字相加，如果累加的结果有进位，则将进位部分加到低16位中，直到没有进位
2. 验证最终的16位累加结果是否为0xFFFF。如果是，说明数据在传输过程中没有发生错误，校验和有效；否则，校验和无效，数据可能已经损坏。
3. **可靠数据传输协议rdt**
4. rdt1.0

完全可靠的信道，可以保证从应用层的一侧到另一侧数据不丢失，因此此时的发送方和接收方只会有一个状态。

发送方： 等待应用程序下发调用指令，发送数据

① 应用进程调用 rdt\_send（data），将数据推送至传输层

② 传输层调用 make\_pkt，将源自于应用程序的报文分组打包成报文段

③ 传输层调用udt\_send方法，将报文段推送至信道

接收方： 等待来自下层的调用指令，接受数据并缓存

① 下层调用ret\_rev，将数据推送到传输层

② 传输层调用 extract，从报文段中提取出数据

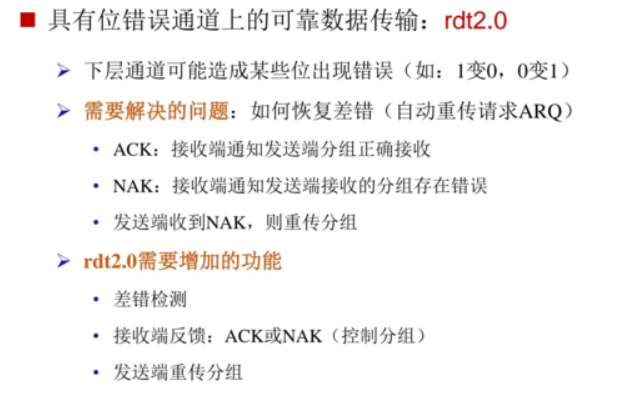
③ 传输层调用 deliver\_data将数据推送至应用层



1. rdt2.0

在rdt 2.0考虑比特差错出现的情况，比特差错通常会出现在可能受损的物理部件之中，因此引入比特差错校验。在考虑出现比特差错的 rdt 2.0 中，需要加入肯定确认（ACK positive acknowledgement）、否定确认（NAK negative acknowledgement）的情况。

对于否定确认的报文，需要提示发送方重新发送该数据。基于这种重传机制的可靠数据传输协议称为自动重传协议（Automatic Repeat reQuest，ARQ）。Sender需要增加一个状态，用于等待 ACK 或 NAK。



发送方：

① 应用层调用rdt\_send方法， 将数据推送至应用层

② 应用层调用make\_pdt将数据打包成报文段，并在报文段中封装进一个校验和

③ 应用层调用udt\_send方法将打包完成的报文段推送至信道

④ 发送端此时状态迁移为等待ACK应答或者NAK应答状态，处于阻塞状态

接收方：

① 下层通过rdt\_rcv方法，将数据推送到传输层

② 传输层接收到报文段，对报文段数据进行校验处理，校验成功则执行第4步，校验失败则跳转到第3步

③ 发送NAK指令， 继续等待下层的调用

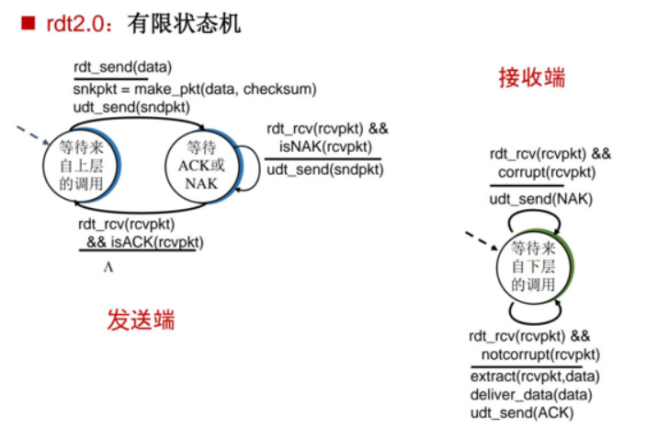
④ 发送ACK指令， 继续等待下层的调用

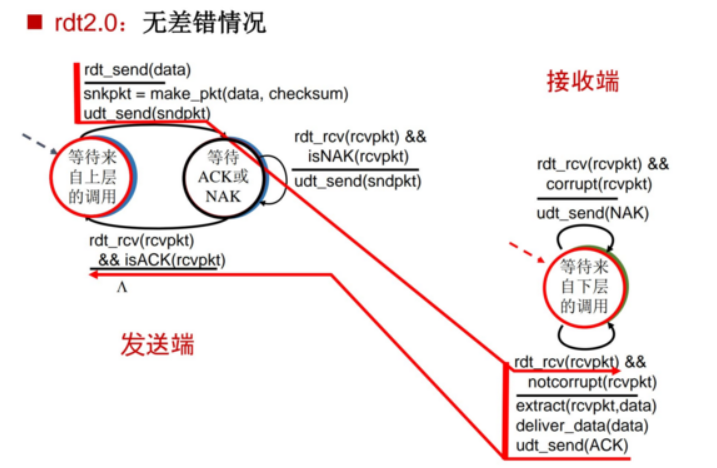
发送方：

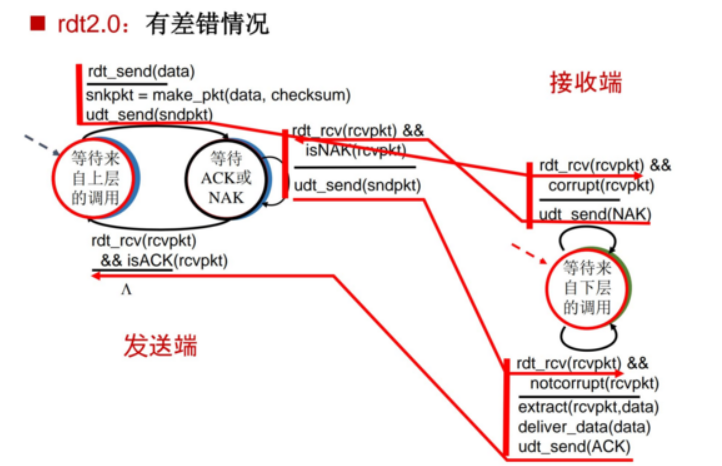
① 接收到NAK应答指令执行第2步， 接收到ACK执行第3步

② 接收到NAK，发送方直接将打包好的数据再一次通过udt\_send方法推送到信道, 保持等待ACK或NAK指令状态

③ 接受到ACK，发送方不再阻塞，可以发送新的数据，状态迁移为等待上层调用状态





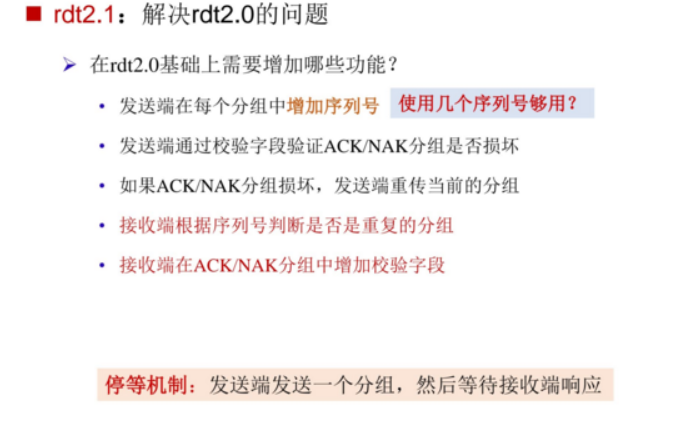




1. rdt2.1

ACK/NAK可能存在受损的可能性，无法保证接收端发送的ACK/NAK是否存在比特差错的情况。当发送方收到含糊不清的ACK和NAK分组时，只需重传当前数据分组即可。这种方法在发送方到接收方的信道中引入了冗余分组。但是接收方不知道它上次所发送的ACK或NAK是否被正确的接收到。

为了解决冗余分组的问题，在数据分组中添加一个新的字段，让发送方对其数据分组编号，即将发送数据分组的序号。



发送方：

① 应用层调用rdt\_send方法， 将数据推送至应用层

② 应用层调用make\_pdt将数据打包成报文段，并在报文段中封装进一个 校验码和一个值为0或1的序号

③ 应用层调用udt\_send方法将打包完成的报文段推送至信道。

④ 发送端此时状态迁移为等待序号为 0 或 1的报文段 ACK应答或 NAK应答状态。

接收端：

① 下层通过rdt\_rcv方法， 将数据推送到传输层

② 传输层接收到报文段，对报文段数据进行校验处理，校验成功则执行第4步，校验失败则执行第3步

③ 发送NAK指令，继续等待下层调用

④ 检测数据序号，如果是冗余数据，直接丢弃数据，发送对缓存栈中数据的ACK指令。对于非冗余数据，则将数据置换到缓存栈之中，发送一个确认对本次数据的ACK指令，继续等待来自下层的调用。

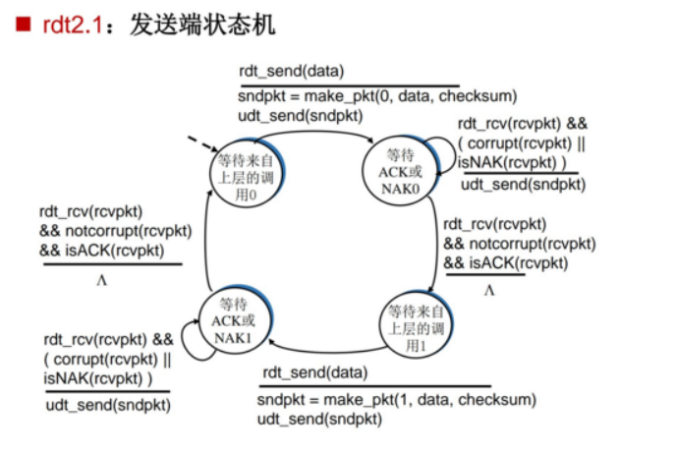
发送端：

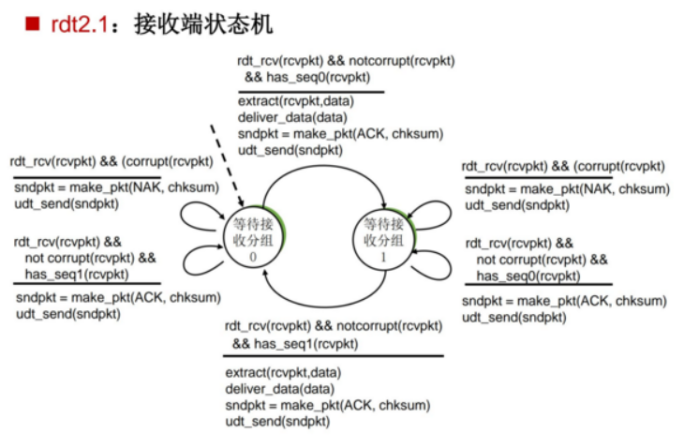
① 接收到应答指令后，进行数据校验处理，如果数据校验错误，直接重传上次数据，如果数据校验正确，则执行第2步。

② 判断接受到的应答指令，如果指令为ACK执行第3步， 如果指令为 NAK 执行第4步

③ 接收到ACK指令，发送端不再阻塞， 可以发送新的数据 1 或 0， 状态迁移到等待来自上层调用。

④ 接收到NAK指令，重传上次数据，状态迁移到等待ACK或NAK状态。

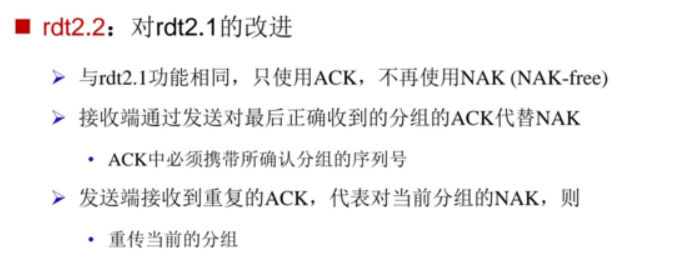




1. rdt2.2

有了序号seq，接收端不在需要用NAK指令去表示收到的数据产生了比特错误。对于比特受损的数据，接收端直接丢弃，并发送一个对缓存区数据的确认ACK。

发送端需要维护一个缓存，用于记录上一次发送的数据，当接受到的ACK与缓存序列号相同，那么就表示发送的数据发生了比特差错，此时重新发送一次缓存区中的数据即可。



发送端

① 应用层调用rdt\_send方法，将数据推送至传输层

② 应用层调用make\_pdt将数据打包成报文段，并在报文段中封装进一个校验和，和一个值为0 或 1的序号，并将其存储至缓存区

③ 应用层调用udt\_send方法将打包完成的报文段推送至信道。

④ 发送端此时状态迁移为 等待序号为 0 或 1 的报文段 的ACK或NAK应答状态

接收端

① 下层通过rdt\_rcv方法，将数据推送到传输层

② 传输层接收到报文段，对报文段数据进行校验处理，校验成功则执行第4步，校验失败则执行第3步

③ 发送缓存区数据序号的确认ACK指令，保持等待下层调用的状态

④ 检测本次数据序号，如果是冗余数据，直接丢弃数据，发送缓存栈中数据序号的ACK指令。对于非冗余数据，将数据置换到缓存区之中，并发送一个对本次数据序号的确认ACK指令

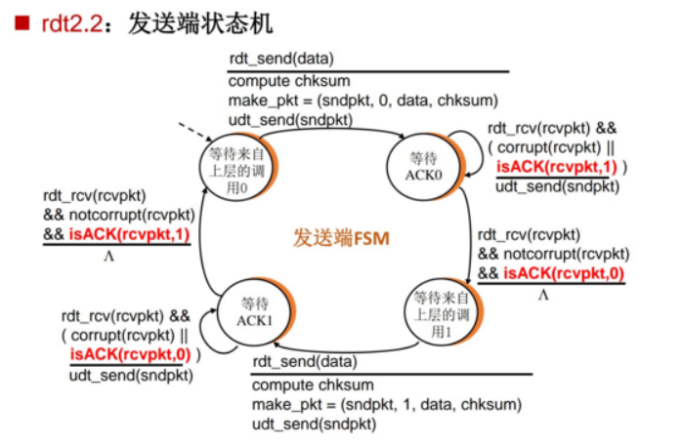
发送端：

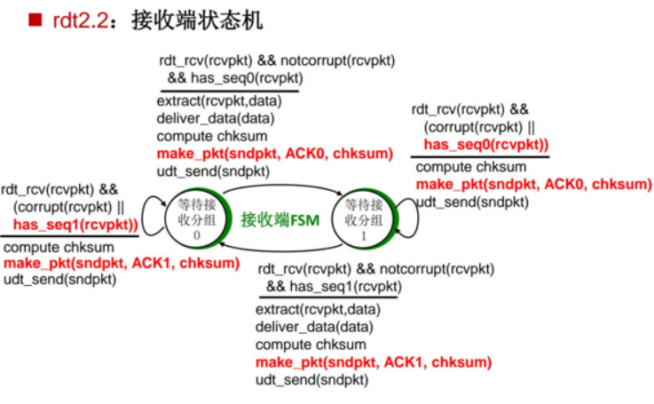
① 接收到应答指令后进行数据校验处理，如果数据校验错误， 直接重新发送上次数据，如果正确则执行第 2 步。

② 判断接受到的应答指令，如果指令为ACK中的序号等于缓存区的序号，执行第 3 步，如果ACK指令序号不等于缓存区序号则执行第4 步

③ 重新发送上次数据，状态迁移到 等待ACK应答指令

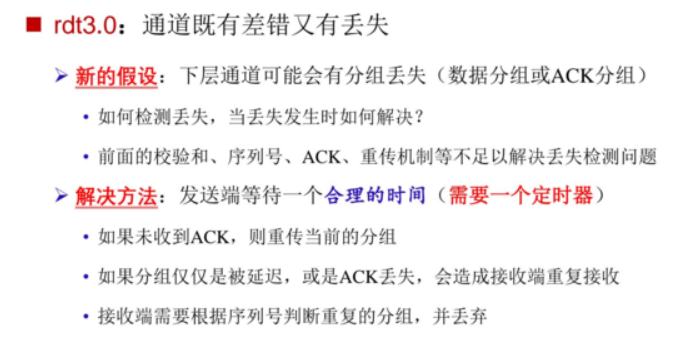
④ 发送端不再阻塞， 可以发送新的序号为 1 或者 0 的数据，状态迁移到等待来自上层调用





1. rdt3.0

不仅有比特差错，还有丢包异常，即发送方或者接受方由于网络阻塞等状况，并没有收到来自于对方的应答数据。加入一个定时器来处理丢包现象， 当发送一个报文段的时候，就开启一个定时器，在定时器结束期间，如果没有收到对应数据的应答报文，则重传数据。



发送端：

① 应用层调用rdt\_send方法， 将数据推送至应用层

② 应用层调用make\_pdt将数据打包成报文段，并在报文段中封装进一个校验和，和一个值为0 或 1的序号，并将其存储至缓存区

③ 应用层调用udt\_send方法将打包完成的报文段推送至信道，并启动一个定时器事件。

④ 发送端此时状态迁移为等待序号为 0 或 1报文段的ACK应答状态

⑤ 倘若在定时器等待时间内，没有收到响应，则重新执行第3步

接收端：

① 下层通过rdt\_rcv方法，将数据推送到传输层

② 传输层接收到报文段，对报文段数据进行校验处理，校验成功则执行第 4 步，校验失败则执行第 3 步

③ 发送缓存区数据序号的确认ACK指令, 同时开启一个定时器， 保持等待下层调用的状态。

④ 检测数据序号，如果是冗余数据，直接丢弃数据，发送缓存区中数据序号的ACK指令。对于非冗余数据，将数据置换到缓存区中，发送一个对该数据序号的确认ACK指令，同时开启一个定时器。

⑤ 倘若在定时器等待时间内，没有收到响应，则重新执行第3或第4步

发送端：

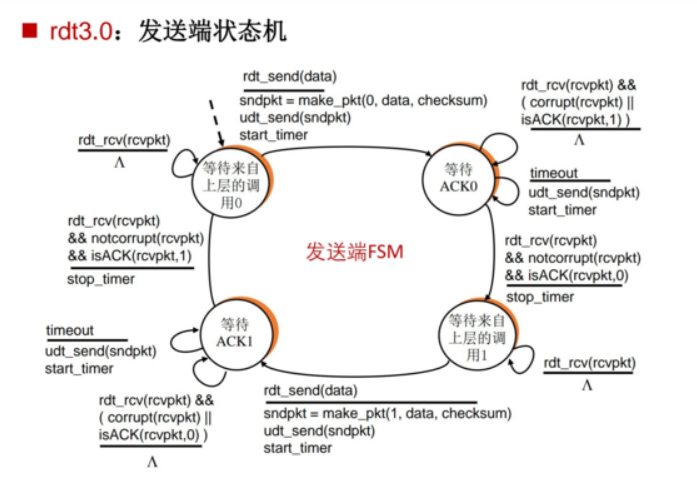
① 接收到应答指令后进行数据校验处理，如果数据校验错误，直接重新发送上次数据，如果正确则执行步第 2 步

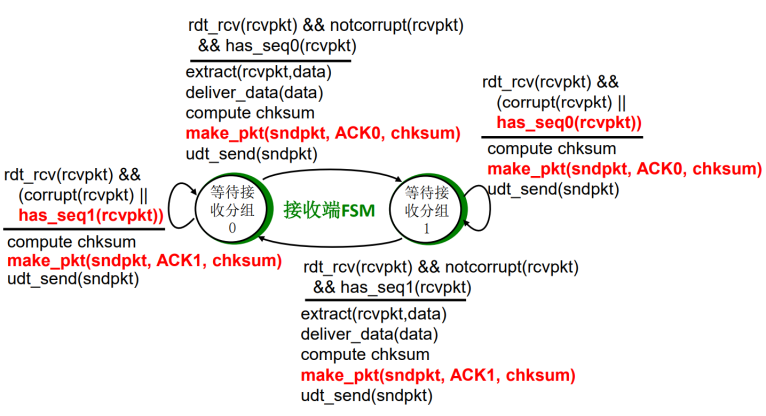
② 判断接受到的应答指令，如果指令为ACK中的序号等于缓存区的序号，执行第3步，如果ACK指令序号不等于缓存区序号则执行4

③ 重新发送上次数据，状态迁移到等待ACK，同时开启一个定时器

④ 发送端不再阻塞，可以发送新的序号值为 1 或 0 的数据，状态迁移到等待来自上层调用

⑤ 若在定时器等待时间内，没有收到响应，则重新执行第3或第4步。





1. 本次实验的可靠数据传输协议

本次实验使用的可靠数据传输协议基于rdt3.0。

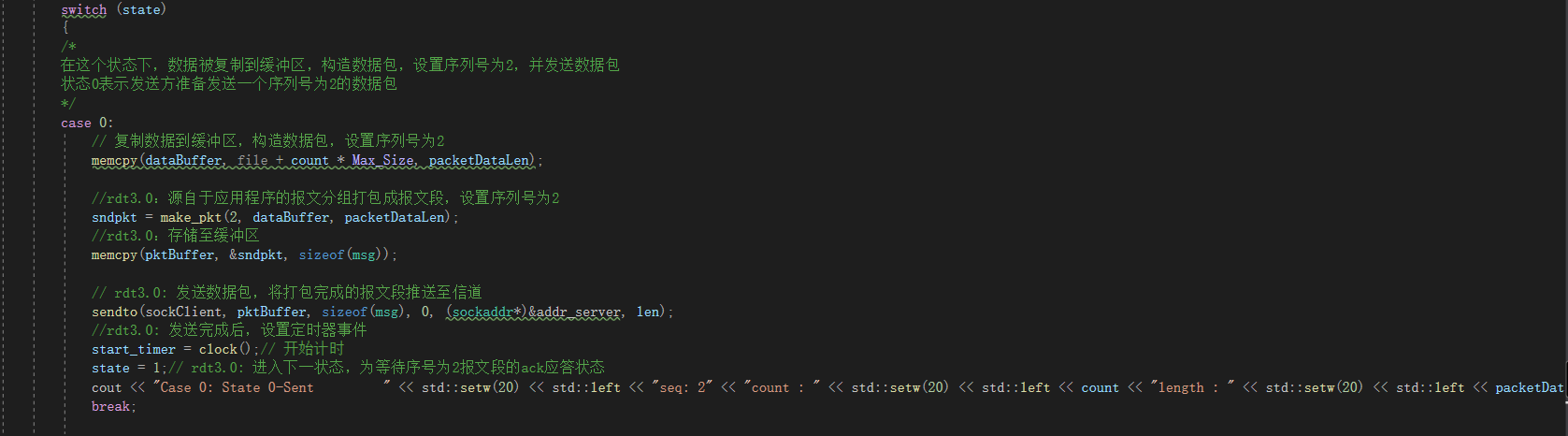
在本次实验中，由于三次握手的时候使用了0和1.所以将此处的两个seq号设为2和3。

客户端在发送完一个seq号的数据后进入该seq号的确定状态。如果超时则重发数据包，收到另一个seq的确定后维持状态不变。

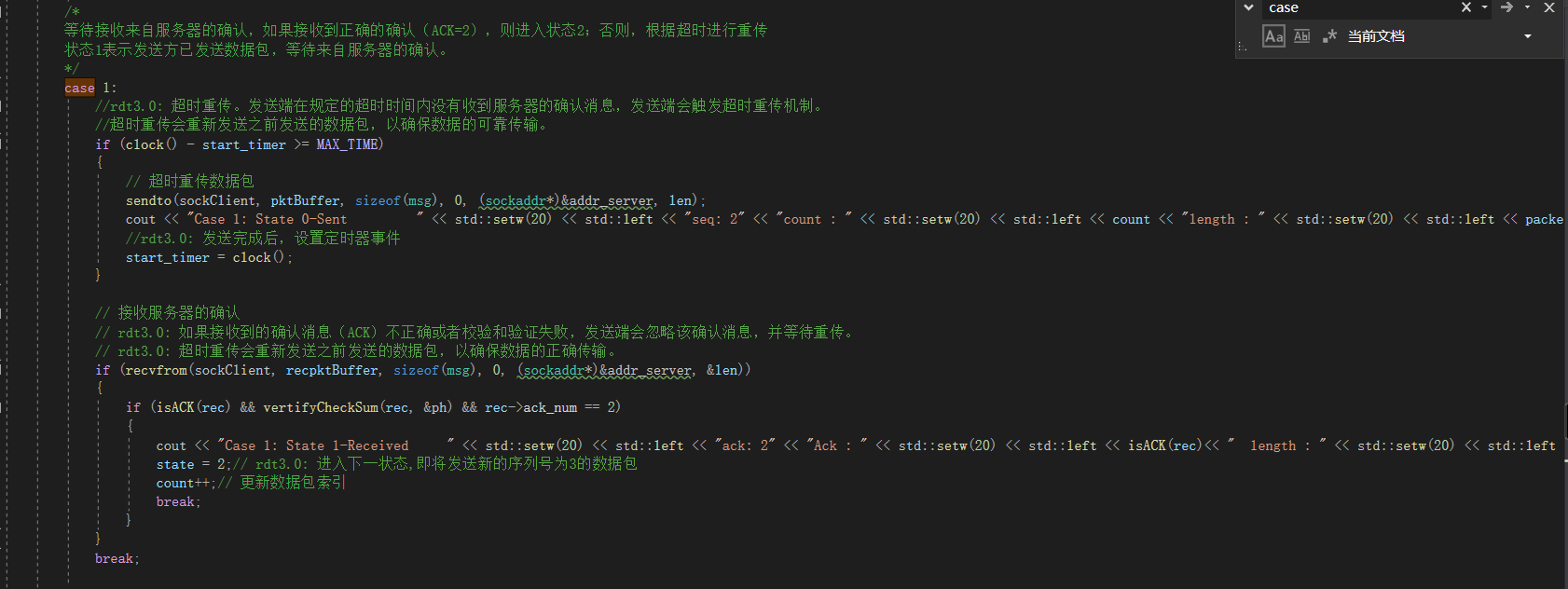
而服务器端在收到当前状态的正确seq号的时候会发送该seq号的ack，转换状态。若收到的seq号为另一状态或损坏，则发送其状态的ack。以希望让客户端进入发送下一预期seq的状态。

① 发送端的4个状态：

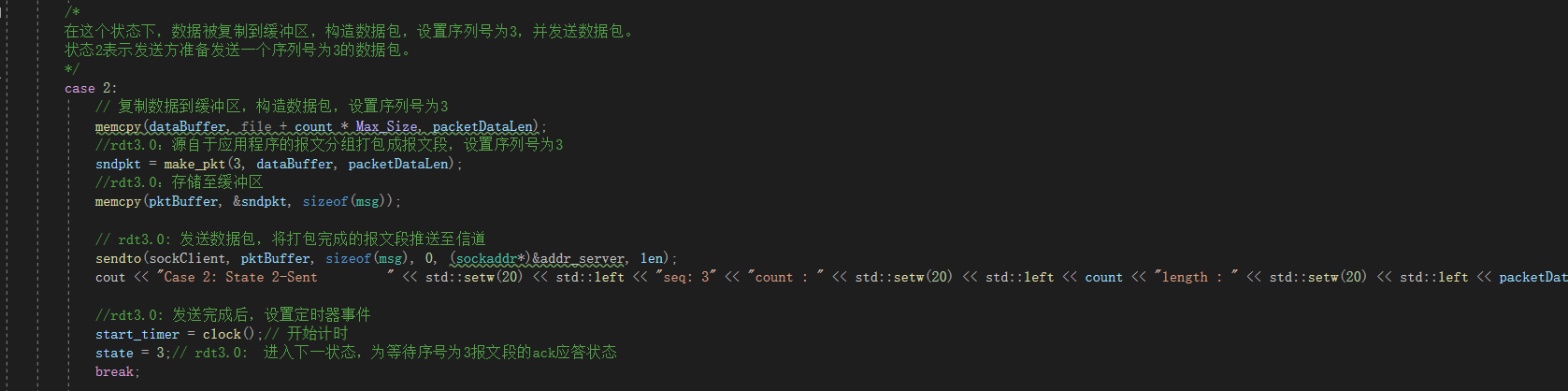
State 0表示发送方准备发送一个seq序列号为2的数据包,这个状态下，数据被复制到缓冲区，构造数据包，设置seq为2，并发送数据包。



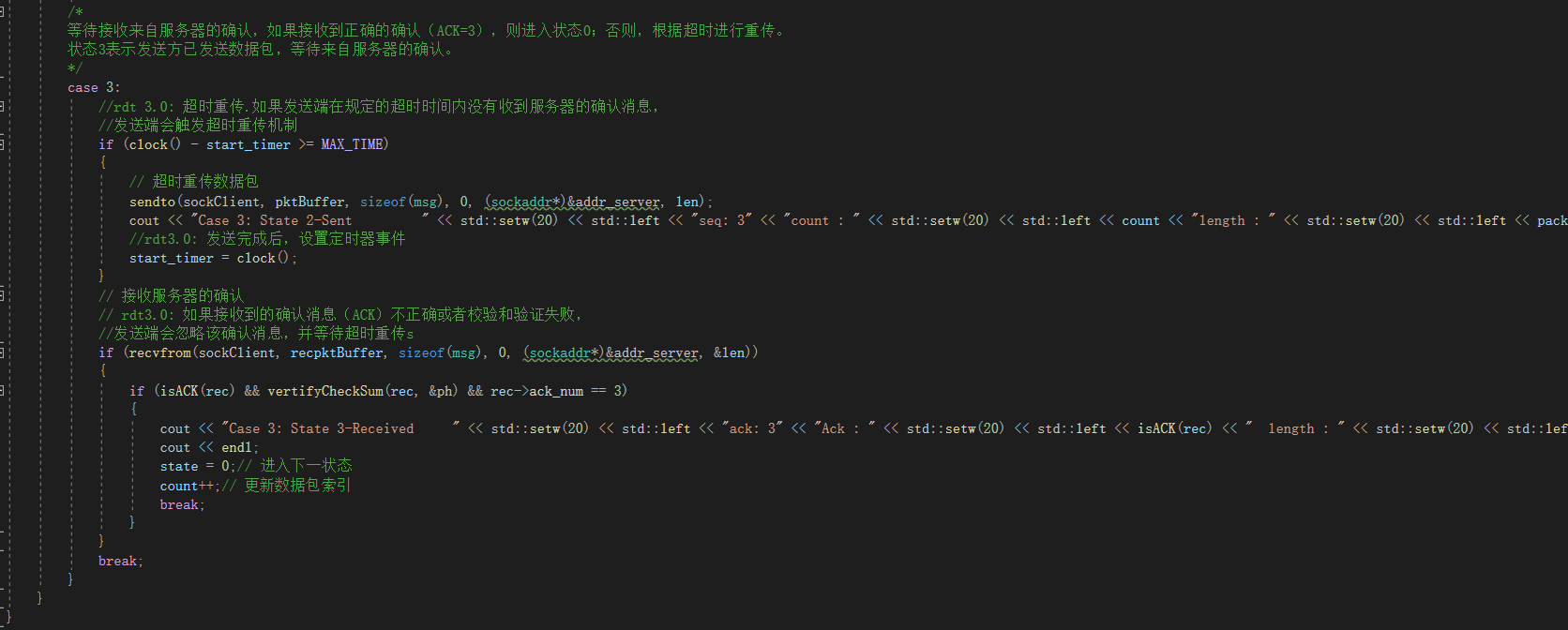
State 1表示发送端等待接收来自接收端的确认，如果接收到正确的确认（ack=2），则进入State 2；否则，根据超时进行重传State 0的数据包。State 1表示发送方已发送数据包，等待来自服务器的确认。



State 2表示发送方准备发送一个seq为3的数据包，这个状态下，数据被复制到缓冲区，构造数据包，设置seq为3，并发送数据包。



State 3表示发送端等待接收来自接收端的确认，如果接收到正确的确认（ack=3），则进入State 0；否则，根据超时进行重传State 2的数据包。State 3表示发送方已发送数据包，等待来自服务器的确认。

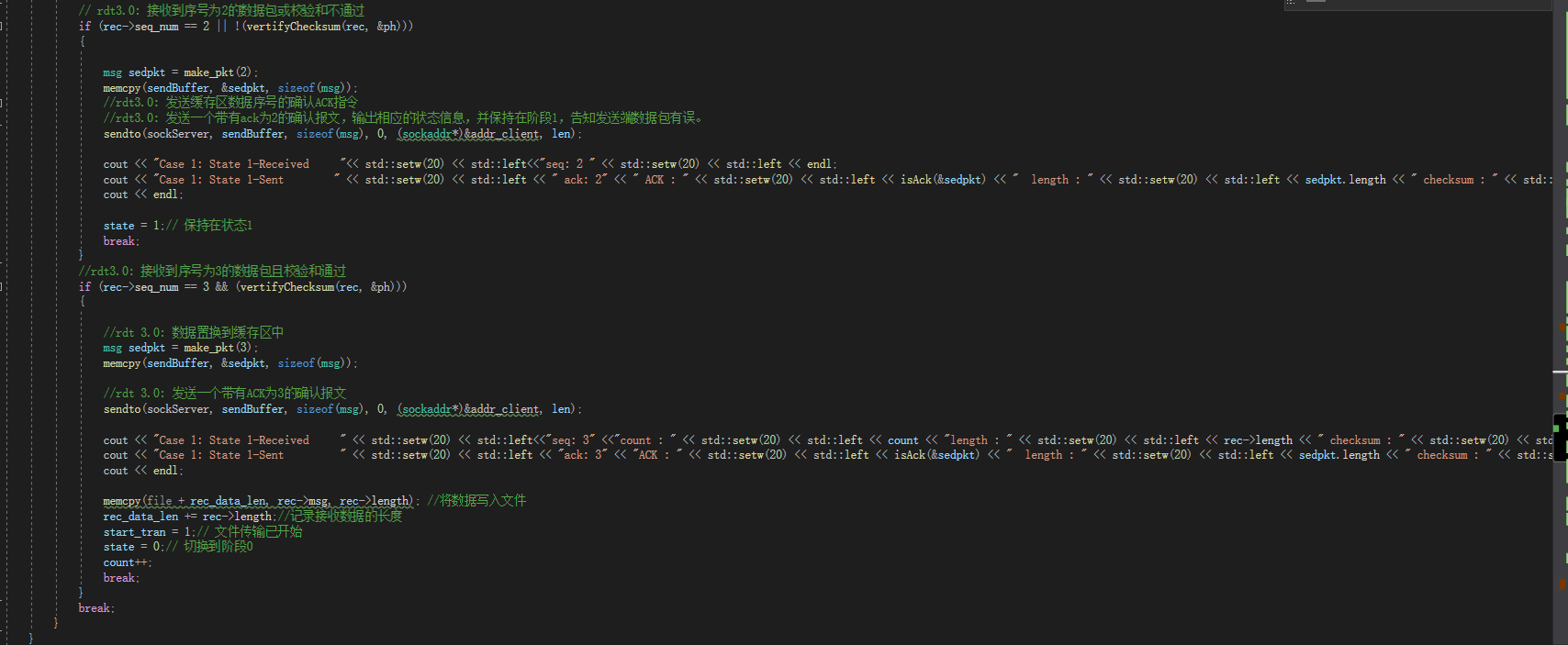


② 接收端的两个状态：

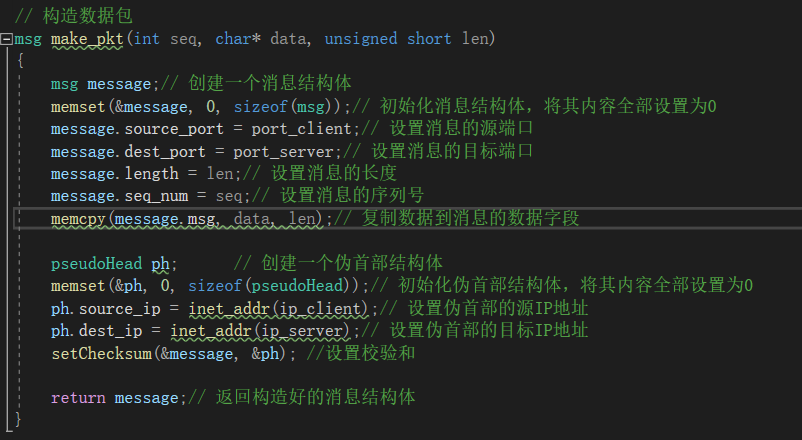
State 0:接收来自客户端的数据包，State 0预期收到的数据包的seq=2。如果接收到seq为2的数据包且校验和正确，发送ack为2的确认，表示正确确认，进入到State 1；如果接收到seq为3的数据包或校验和不通过，发送ack为3的确认,并保持在State 0，告知发送端数据包有误。



State 1: 接收来自客户端的数据包,State 1预期收到的数据包的seq=3。如果接收到seq为3的数据包且校验和通过，发送ack为3的确认，表示正确确认，进入到State 0；如果接收到seq为2的数据包或校验和不通过，发送ack为2的确认,并保持在State 1，告知发送端数据包有误。



1. **发送端和接收端的交互过程**
2. 发送端交互过程
3. 发送端构造数据包



发送端通过调用 make\_pkt 函数构造一个数据包，并填充数据字段。

1. 发送端设置校验和

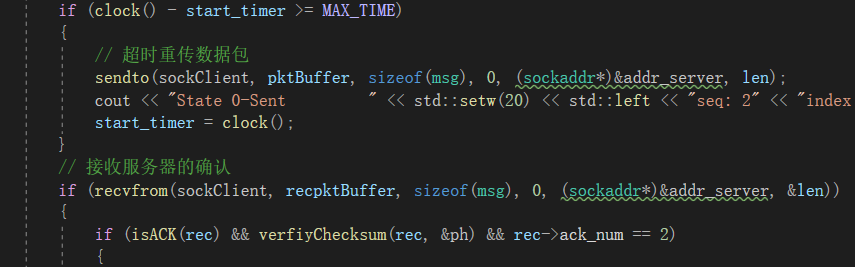
客户端使用 setChecksum 函数计算并设置校验和。

1. 发送端发送数据包

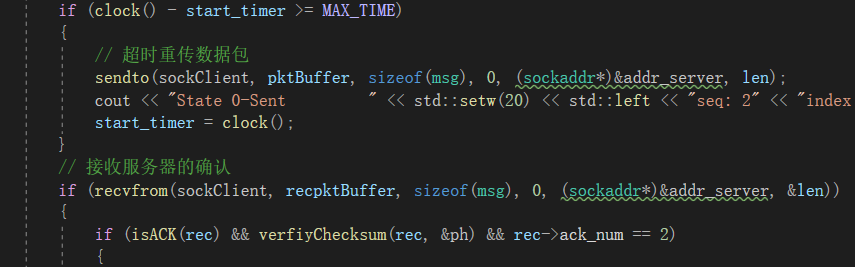


使用 sendto 函数将数据包发送到目的地址。

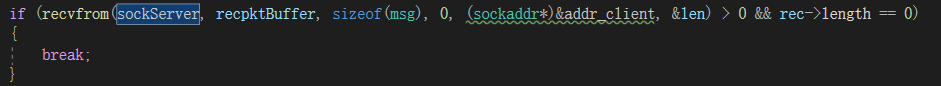
1. 等待确认： 发送端等待接收到来自接收端的确认。



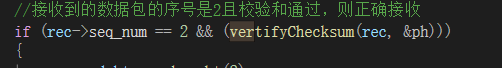
1. 处理超时： 如果在规定的时间内未收到确认，发送端可能会触发超时重传机制，重新发送数据包。



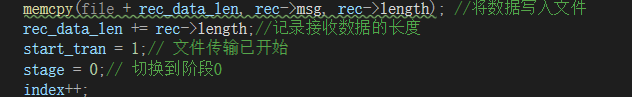
1. 接收端的交互过程
2. 接收数据包： 接收端通过 recvfrom 函数接收从发送端发送过来的数据包。



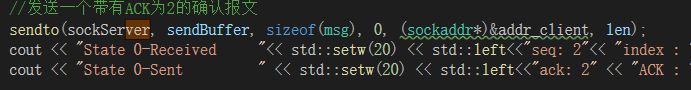
1. 验证校验和： 接收端使用 vertifyChecksum 函数验证接收到的数据包的校验和是否正确。



1. 处理数据包： 如果校验和正确，接收端根据数据包的内容执行相应的操作。

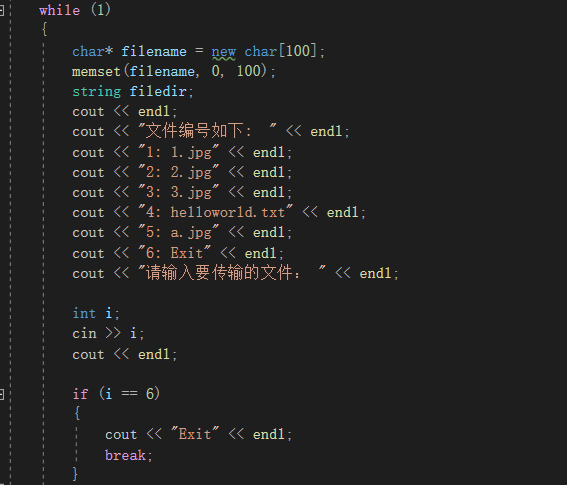


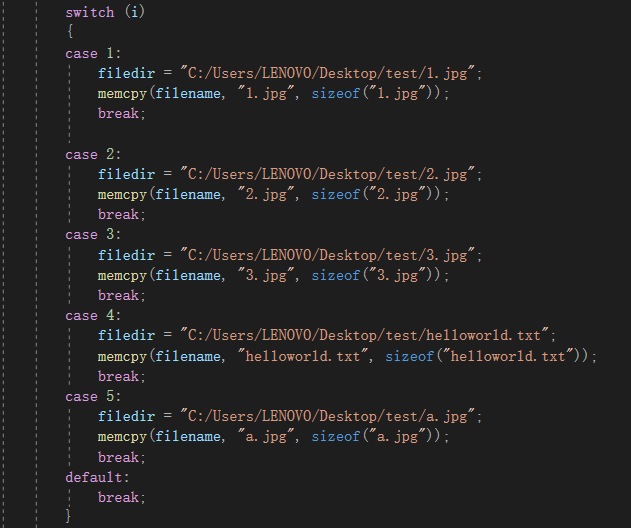
1. 构造确认包： 如果需要，接收端可以构造一个确认数据包，设置确认号，并发送给发送端。

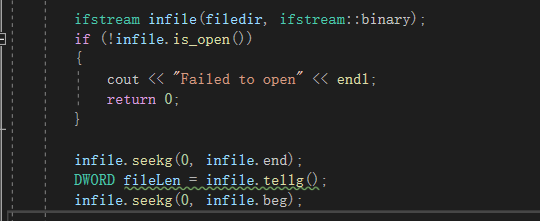


1. **文件读写**

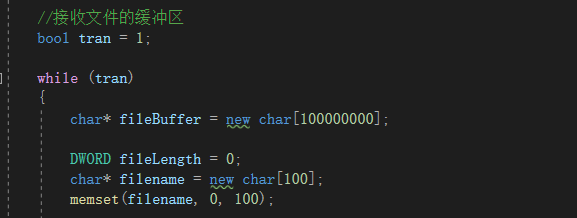
发送端：选择发送的文件，文件内容读入缓冲区，获得文件名，文件大小

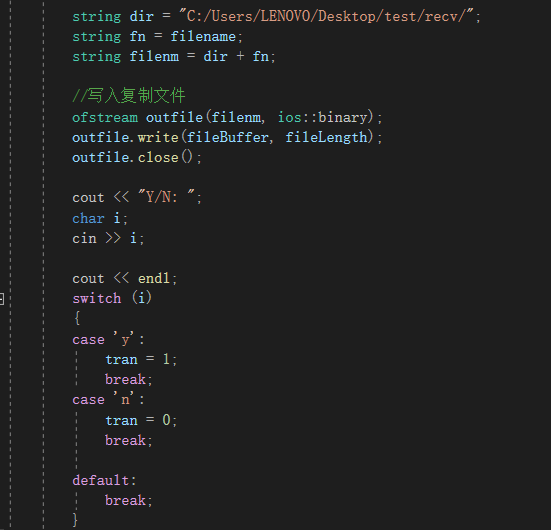






接收端：获得文件内容和文件名，在指定路径下写入收到的文件

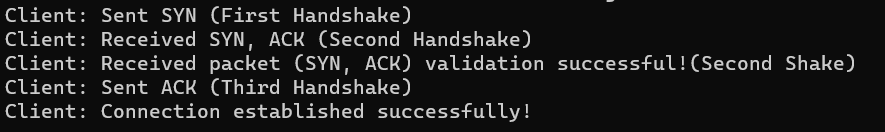




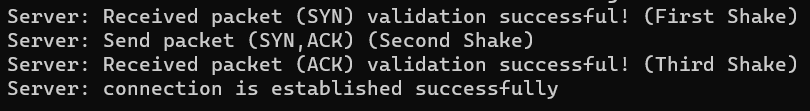
1. **程序运行**
2. **设置路由**



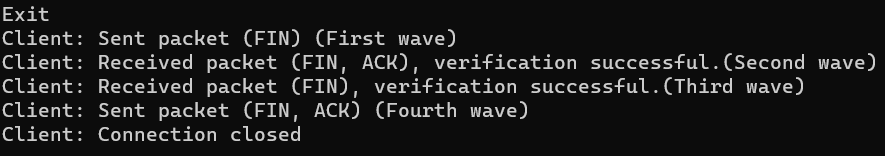
1. **三次握手**
2. **客户端**



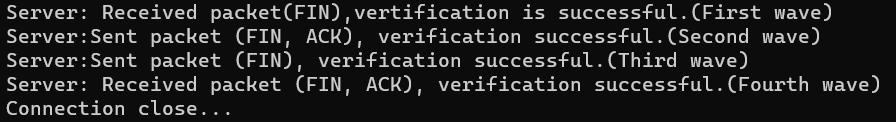
1. **服务器端**



1. **四次挥手**
2. **客户端**



1. **服务器端**

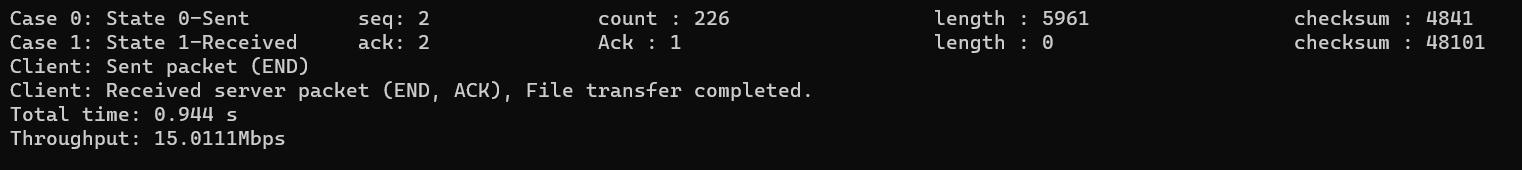


1. **正常传输**

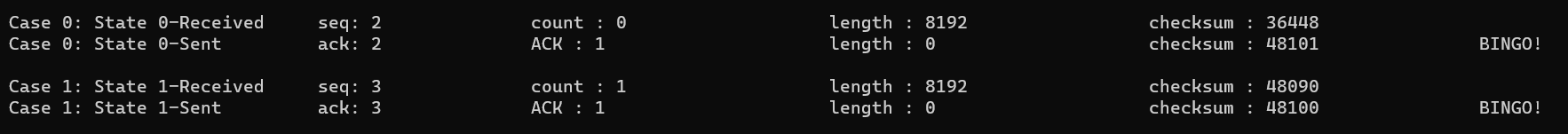
设置丢包率为0%，延时为0ms。

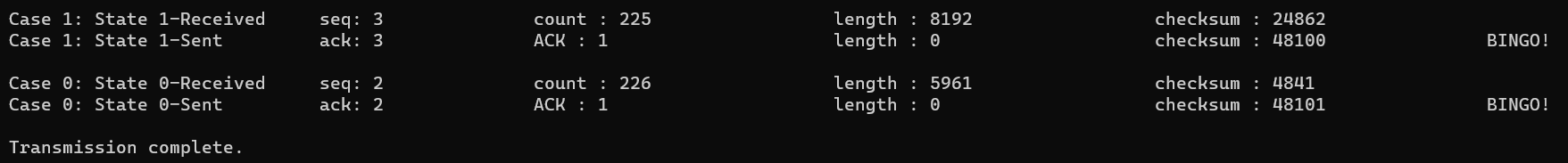
发送端：





接收端：





1. 发送端的4个状态：

State 0表示发送方准备发送一个序列号为2的数据包,这个状态下，数据被复制到缓冲区，构造数据包，设置序列号为2，并发送数据包。

State 1表示发送端等待接收来自接收端的确认，如果接收到正确的确认（ACK=2），则进入State 2；否则，根据超时进行重传State 0的数据包。State 1表示发送方已发送数据包，等待来自服务器的确认。

State 2表示发送方准备发送一个序列号为3的数据包，这个状态下，数据被复制到缓冲区，构造数据包，设置序列号为3，并发送数据包。

State 3表示发送端等待接收来自接收端的确认，如果接收到正确的确认（ACK=3），则进入State 0；否则，根据超时进行重传State 2的数据包。State 3表示发送方已发送数据包，等待来自服务器的确认。

1. 接收端的两个状态：

State 0:接收来自客户端的数据包，State 0预期收到的数据包的seq=2。如果接收到序号为2的数据包且校验和通过，发送ACK为2的确认，表示正确确认，进入到状态1；如果接收到序号为3的数据包或校验和不通过，发送ACK为3的确认,并保持在阶段0，告知发送端数据包有误。

State 1: 接收来自客户端的数据包,State 1预期收到的数据包的seq=3。如果接收到序号为3的数据包且校验和通过，发送ACK为3的确认，表示正确确认，进入到状态0；如果接收到序号为2的数据包或校验和不通过，发送ACK为2的确认,并保持在阶段1，告知发送端数据包有误。

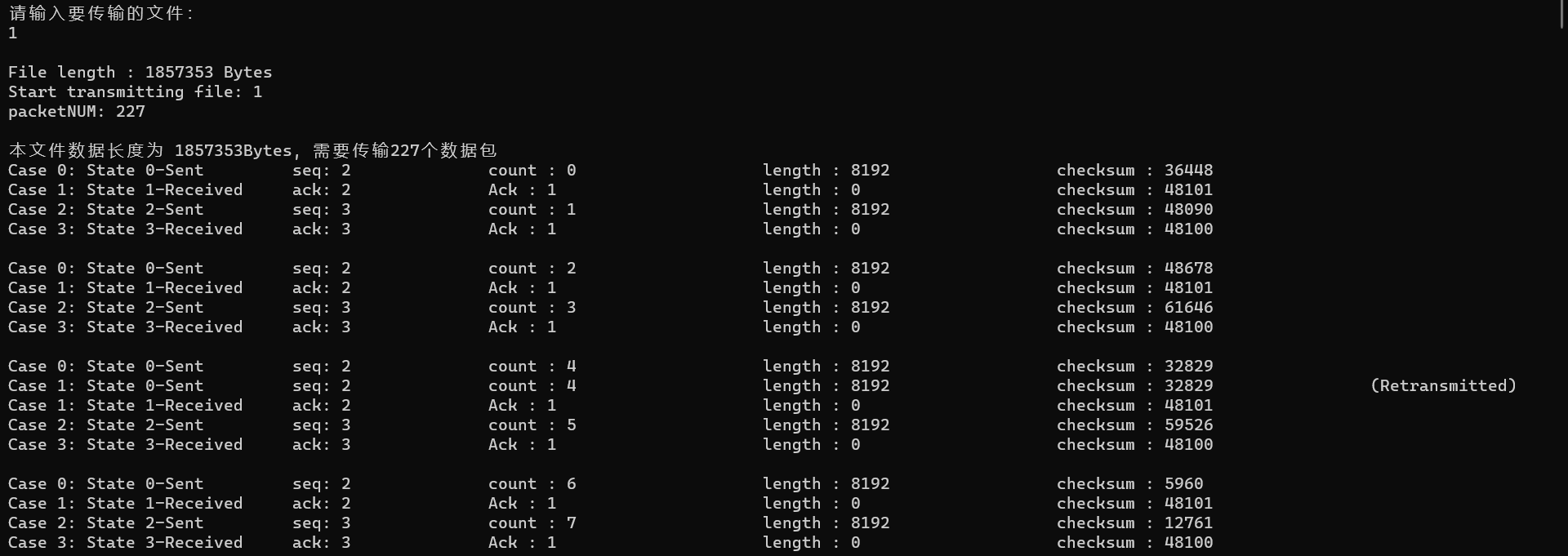
1. **丢包**

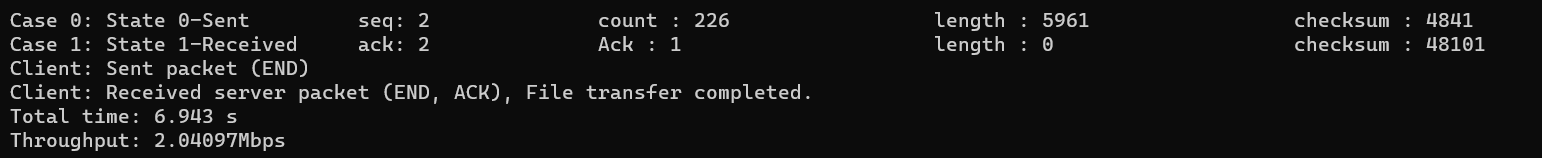
设置丢包率为20%（每5个包丢一个），延时为0ms



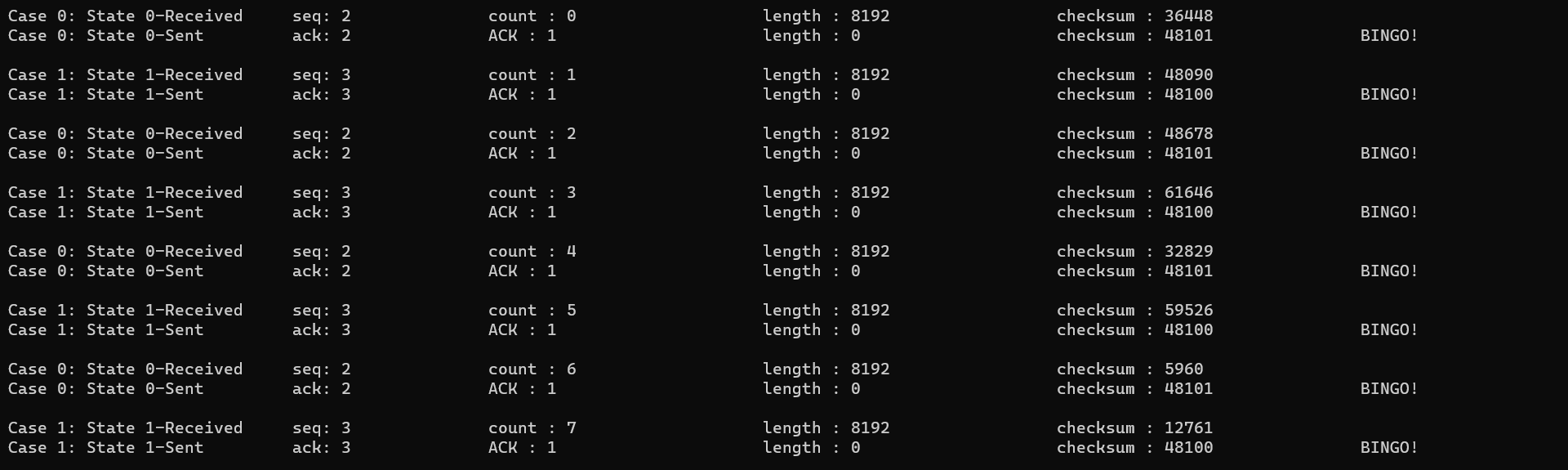


发送端：





接收端：



每5个包丢一个，接收端无法发送这个包的ACK确认，在发送端触发超时重传。如果发送端处于State 1状态，则重传State 0的数据包，如果处于State3的状态，则重传State 2的数据包。

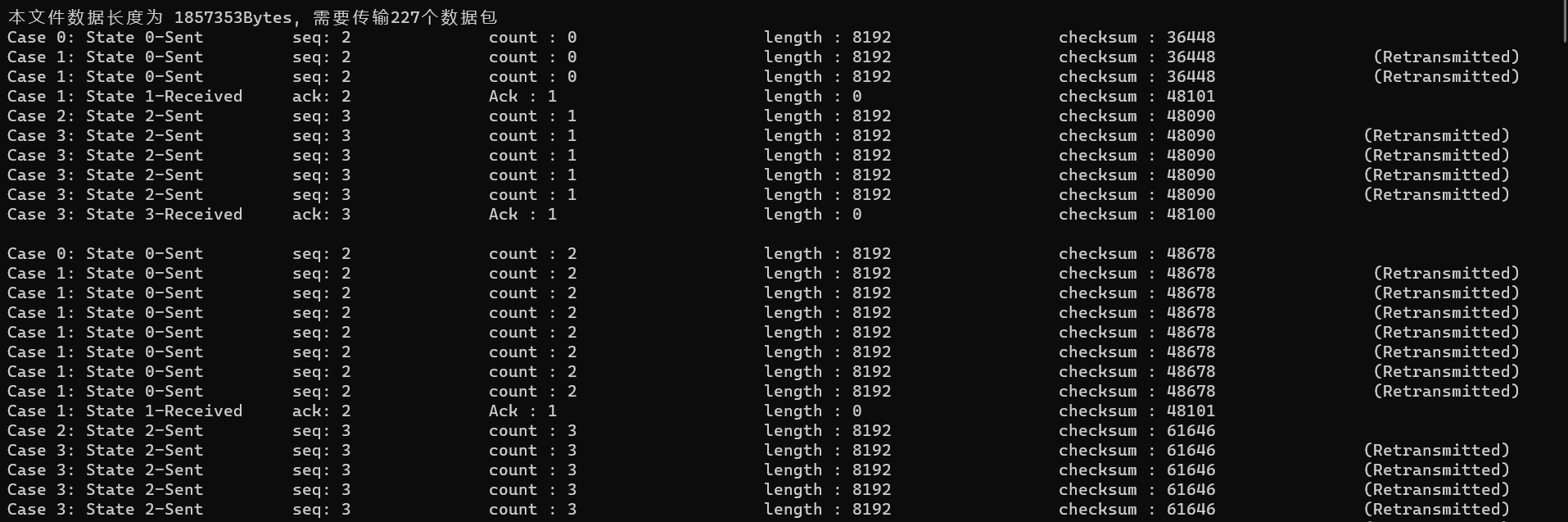
1. **超时重传**

设置丢包率为0%，延时为110ms（程序中设置的MAX\_TIME=100）。

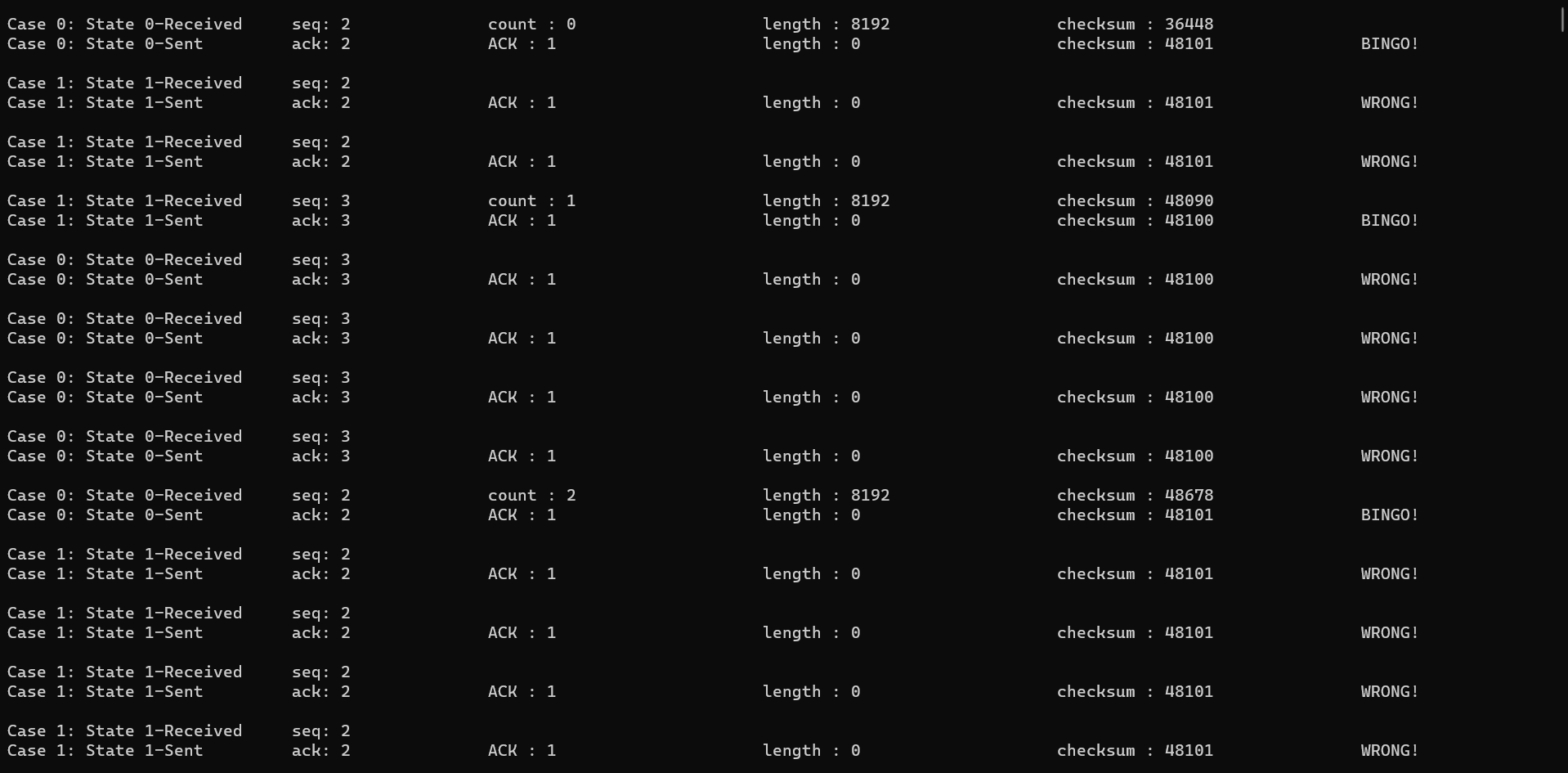




发送端：



接收端：



发送端State 1状态（State 3状态）发送端没能收到ACK，超时重传，重发State 0（State 2状态）的数据包，但并不知道是因为数据包丢失还是延时导致的没收到ACK。因为本次设置的是延时，超时重传会导致冗余分组，rdt2.1为解决这个问题引入了seq，rdt2.2取消了NAK。

接收端检测数据序号，如果是冗余数据，告诉发送端收到的信息是错误的（接收端State 0接收seq=3的数据包，State 1接收seq=2的数据包就是错误的信息），丢弃数据。

1. **在接收文件夹中查看传输结果**

