计算机网络实验三　RDT 通信程序设计

浮焕然 PB22061345

1.思考题

1.1 RDT 底层是 UDP，为什么程序中可以用 recv/send 而不是 recvfrom/sendto 收发数据？

```
int sockfd = Init_Socket_Sender();
```

在选择重传(RDT)协议中，底层使用的是 UDP 协议。在 UDP 编程中，通常使用 recvfrom 和 sendto 函数来收发数据，这两个函数需要指定对方的地址信息。然而，在 RDT 协议的实现中，由于通信双方已经建立了连接，即已经知道对方的地址信息，因此可以使用 recv 和 send 函数来简化操作。

1.2 停等发送端程序中是如何实现超时重传的？

停等协议中在一个 while(1)中一直使用

```
RDT_Send(sockfd, send_rdt_packet, send_packet_length, 0);
```

来一直发送同一个 RDT 数据包，在发送一次之后使用

```
struct pollfd pollfd = {sockfd, POLLIN};
int state = poll(&pollfd, 1, RDT_STOP_WAIT_TIME_OUT);
```

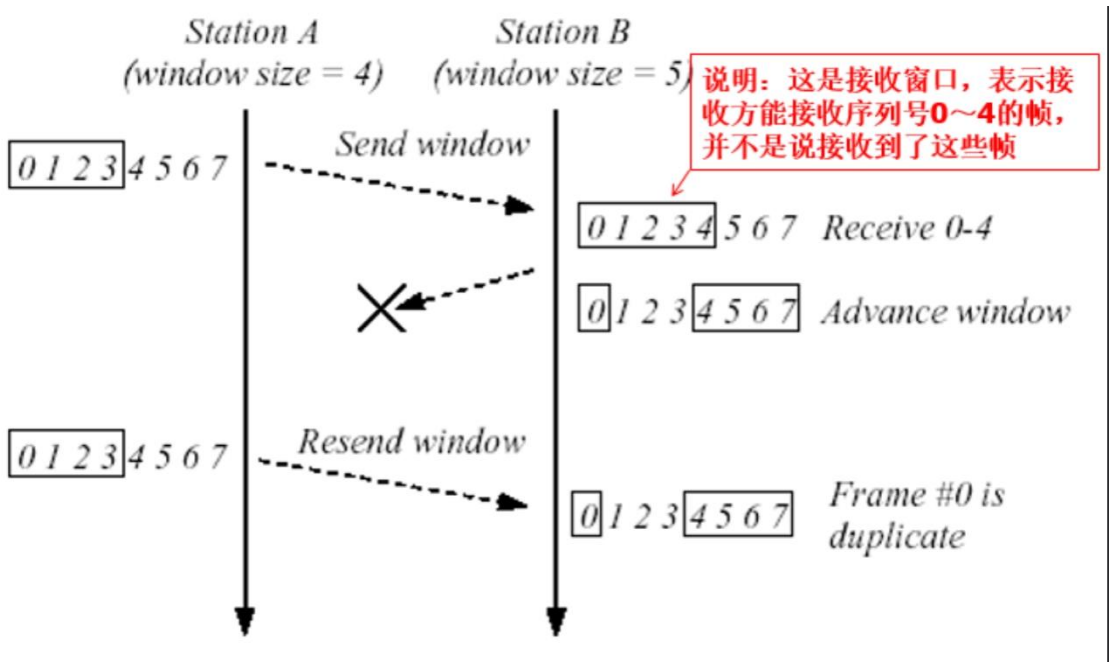
检查 poll 的状态，如果 state == 0，则说明超时，此时需要 continue 操作即可进入下个 while 循环，如果 state == 1，说明数据发送成功，此时使用 break 结束发送这个 RDT 数据包的 while 循环，即可进入下一个 RDT 数据包的发送。

1.3 参考两军对垒问题，在有发送和接收失败的情况下，怎么保证双方正确地结束通信？

超时重传机制：在通信过程中，如果发送方发送了一个信号后，等待接收方的确认信号（ACK），如果在一定时间内没有收到确认，发送方会认为消息丢失或失败，然后重新发送信号。

1.4 在选择重传协议中，为何窗口大小必须小于或等于序列号空间大小的一半？

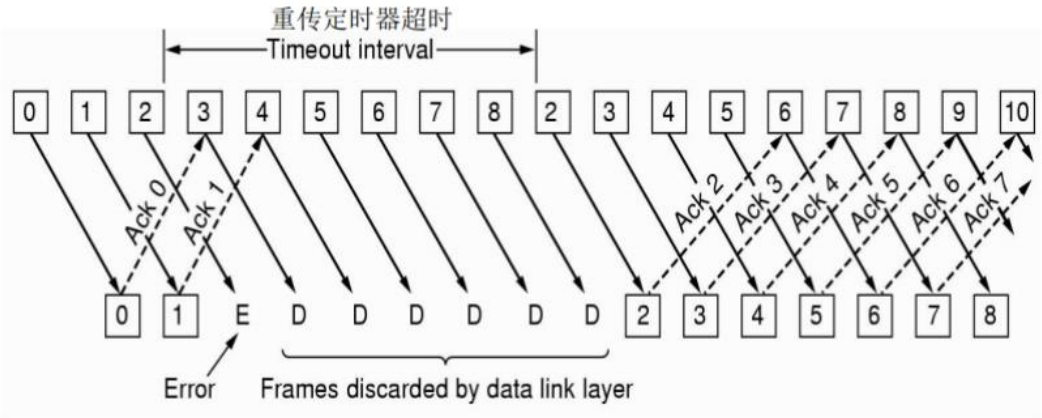
如果串口大小太多，会导致序列号空间有限，不断递增出现回绕，序列号重叠的现象。在滑动窗口协议中，发送方可以连续发送多个数据帧而不需要等待前一个数据帧的确认。如果窗口大小大于序列号空间的一半，那么在序列号循环回到 0 之前，发送方可能已经发送了超过一半的序列号范围的数据帧。当序列号循环时，接收方可能会混淆新发送的数据帧和之前发送的数据帧，因为它们的序列号相同。



2.实验总结

本实验要求在 UDP 的基础上实现简化版的可靠数据传输协议（Reliable data transfer protocols, RDT），并利用 RDT 传输一份文件，通过本次实验我学到了：

1. 通信的发送接收端口中是实要现超时重传与超时结束，防止程序因为意外卡住。
2. 停等协议发送端口发送一个包后通过阻塞等待接收端口回传 ACK/NACK/超时，这种方法简单但是效率低。
3. 停等协议相当于发送接收窗口均为 1；回退 N 协议相当于发送窗口不为 1 但接收窗口为 1；选择重传协议相当于发送接收窗口均不为 1。
4. 回退 n 协议的弊端：由于接收端进行累计确认，发送端无法获知这之后的包是否正确收到了，因此这种方法效率较低. 如果丢失了中间一个包而后续所有包都正确到达了，那么发送端将浪费大量资源进行重传。



5. 选择重传协议相对于回退 n 协议, 发送端只重新发送超时的包或者收到 NACK 的包, 因此大大降低了重传的开销。