滑动窗口协议实验报告

计 13　谭志鹏　2011011258

一、　实验目的

滑动窗口协议（Sliding Window Protocol）是计算机网络中为保证流控制和可靠传输而常用的一种协议。 窗口机制是重传、流控、拥塞控制的基本方法，它在发送方和接收方分别设定发送窗口和接收窗口，发送窗口和接收窗口按照某种规律不断地向前滑动。

本次要求能够实现滑动窗口协议中的 1 比特滑动窗口协议和退后 N 帧协议，更深刻地理解滑动窗口协议。

二、　实验原理

1、 1 比特滑动窗口协议

当发送窗口和接收窗口的大小固定为 1 时（即 1 比特滑动窗口协议），滑动窗口协议退化为停等协议（stop－and－wait）。该协议规定发送方每发送一帧后就要停下来，等待接收方已正确接收的确认（acknowledgement）返回后才能继续发送下一帧，信道利用率很低。由于接收方需要判断接收到的帧是新发的帧还是重复的帧，因此发送方要为每一个帧加一个序号。停等协议规定只有一帧完全发送成功后才能发送新的帧，因而只用 1 比特来编号就够了。

2、 退后 N 帧协议

在退后 N 帧协议中，发送方在发完一个数据帧后，不必停下来等待确认帧，而是连续发送若干个数据帧。发送方在每发送完一个数据帧时，都要对该帧设置计时器。若在所设置的超时时间内未收到该帧的确认帧，则该帧就被判为出错或丢失，发送方就必须重新发送该帧及其后的所有帧。

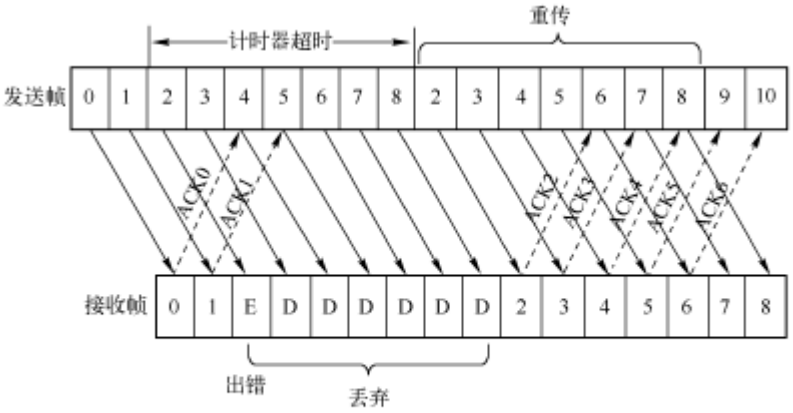

图 3-3　退后 N 帧协议

三、 实验内容

1、 实验分析

本实验需要实现的函数如下：

1）1 比特滑动窗口协议测试函数 stud_slide_window_stop_and_wait()；

2）退后 N 帧协议测试函数 stud_slide_window_back_n_frame()。

测试函数分三种情况：

1）当发送方需要发送帧时，调用学生测试函数，并置参数 messageType 为 MSG_TYPE_SEND，测试函数应该将该帧缓存，存入发送队列中。若发送窗口未满，则打开一个窗口，并调用 SendFRAMEPacket 函数发送该帧。若发送窗口已满，则直接返回，进入等待状态。

2）当发送方收到接收方的 ACK 后，调用学生测试函数，并置参数 messageType 为 MSG_TYPE_RECEIVE，测试函数应该首先检查 ACK 值，再将该 ACK 对应的窗口关闭。由于关闭了等待应答的已发送数据的窗口，等待发送的新帧就可以打开新的窗口并发送。

3）发送方每发送一个帧，系统都会为该帧创建一个计时器，当成功收到 ACK 帧后，计时器会被取消。若某个帧在计时器超时后仍未收到 ACK，系统则会调用测试函数，并置参数 messageType 为 MSG_TYPE_TIMEOUT，告知测试函数该帧超时，测试函数根据帧序号将该帧及后面发送过的帧重新发送。

2、 实验程序

（1） 程序主要结构：

程序中发送的帧 frame 包括帧头 frame_head 和帧大小 size 两部分，帧头包括帧类型，帧序号和确认需要以及帧数据。 Buffer 则用于缓存待发送的帧。

```
typedef enum{data , ack , nak} frame_kind;

typedef struct frame_head   //桢头
{
    frame_kind kind;
    unsigned int seq;
    unsigned int ack;
    unsigned char data[100];
};

typedef struct frame    //帧
{
    frame_head head;
    unsigned int size;
};

typedef struct buffer    //缓存
{
    struct frame *pframe;
    uint32_t size;
};
```

（2） 1 比特滑动窗口协议程序：

程序依据 messageType 的三种情况分别处理：

1、**发送超时**：由于发送窗口大小为 1，因此直接重新发送 frames.front()， 其中 frames 为发送队列的缓存。

```
case MSG_TYPE_TIMEOUT:
    SendFRAMEPacket((unsigned char*)&frames.front() , sizes.front());  //重新发送
    break;
```

2、**发送请求**：首先将发送帧存入发送缓存队列，若发送窗口大小为 0，则发送该帧，否则暂不发送

```
case MSG_TYPE_SEND:
    frames.push(tframe);    //存入发送队列
    sizes.push(bufferSize);
    if(windowSize == 1)
    {
        return 0;        //若发送窗口大小为1，暂时不发送
    }
    else
    {
        SendFRAMEPacket((unsigned char* )&frames.front() , sizes.front());   //发送该帧
        windowSize = 1;
    }

    break;
```

3、**接收信号**：收到一个帧是判断该帧 ack 的值是否为期待的帧，若是则将窗口大小设置为 0，此时若发送队列不为空，则可以继续发送发送队列中的帧。

```
case MSG_TYPE_RECEIVE:
    if(tframe.head.ack == frames.front().head.seq)  //判断该帧的ack值
    {
        frames.pop();    //弹出该帧，关闭窗口
        sizes.pop();
        windowSize = 0;
        if(!frames.empty())   //发送等待的桢
        {
            SendFRAMEPacket((unsigned char* )&frames.front() , sizes.front());
            windowSize = 1;
        }
    }
    break;
```

（3）、退后 N 帧协议

若从滑动窗口的角度来分析 1 比特滑动窗口和退后 N 帧这两种协议，它们的差别仅在于各自窗口的大小不同而已。1 比特滑动窗口协议的发送窗口=1，接收窗口=1；退后 N 帧协议的发送窗口>1，接收窗口=1。程序同样依据 messageType 的三种情况分别处理：

1、 **发送超时：**Nbuffer 为存储缓存的队列，则需要重新发送发送队列中为得到确认的所有帧，同时保证发送的数量不超过最大窗口大小 WINDOW_SIZE_BACK_N_FRAME（实验中为 4）。

```
case MSG_TYPE_TIMEOUT:
    for (int i = 0 ; i < nwindowSize && i < WINDOW_SIZE_BACK_N_FRAME; i++)  //重新发送发送队列里的帧
    {
        buffer buf = Nbuffer[i];
        SendFRAMEPacket((unsigned char *)(buf.pframe), buf.size);
    }
```

2、 **发送请求：**需要发送时，首先将待发送的帧存入缓存队列，之后判断当前窗口大小是否大于最大窗口大小，如果可以发送则发送 Nbuffer 中待发送的下一帧，同时将发送窗口大小加 1。

```
case MSG_TYPE_SEND:
    buffer buf;
    tframe = new frame;
    *tframe = *(frame *)pBuffer;
    buf.pframe = tframe;
    buf.size = bufferSize;
    Nbuffer.push_back(buf);     //将要发送的帧存入已经发送但是还未确认的缓存区队列

    if(nwindowSize < WINDOW_SIZE_BACK_N_FRAME)      //当窗口大小小于最大窗口大小时，发送该帧
    {
        buf = Nbuffer[nwindowSize];
        SendFRAMEPacket((unsigned char *)(buf.pframe) , buf.size);
        nwindowSize++;
    }
```

3、 **接收信号：**接收到一帧时通过判断确认序号，弹出已经被确认的帧，关闭这些窗口。同时窗口关闭时检查缓冲区是否有待发送的帧。

```
case MSG_TYPE_RECEIVE:
    ack = ntohl(((frame *)pBuffer)->head.ack);      //转换字节序
    exp_ack = ntohl(Nbuffer.front().pframe->head.seq);
    while (ack >= exp_ack)              //确认序号不小于期望序号，弹出已经确认的帧
    {
        Nbuffer.pop_front();
        exp_ack++;
        nwindowSize--;
        if (nwindowSize < WINDOW_SIZE_BACK_N_FRAME && !Nbuffer.empty())   //同时继续发送缓冲区的帧
        {
            buffer buf = Nbuffer[nwindowSize];
            SendFRAMEPacket((unsigned char *)(buf.pframe) , buf.size);
            nwindowSize++;
        }
    }
```

四、 思考题

1）退后 N 帧协议与 1bit 滑动窗口协议相比有何优点？

退后 N 帧协议中，由于发送窗口大小为 N, 因为可以连续发送数据帧，提高了发送效率，而不用每次发送后都停下来等待。所以在高带宽，高延迟的情况下，退后 N 帧协议的处理效率远高于 1bit 滑动窗口协议。

2）退后 N 帧协议有什么缺点，如何改进？

在某帧超时重传时，必须把原来可能已正确传送过的数据帧进行重传，这样使得传输效率降低，如果运用选择性重传重新发送超时帧即可。

五、 实验总结

本次试验设计得两个算法并不是很复杂，不过由于是第一次做网络原理的实验，一开始对于实验系统不熟悉，一开始甚至不知道如何开始编写程序。不过通过和同学交流加上自己尝试，逐渐熟悉了实验系统。

实验中遇到一个问题是一开始在处理 MSG_TYPE_RECEIVE 时，我只是弹出了发送队列中的帧，关闭了发送窗口，而没有继续发送等待的帧。由于关闭一个窗口后发送窗口便有富余，应该随机发送缓存队列中的帧。

另外实验中的一个收获是合理使用的 queue 和 deque 两种数据结构，简化了编程。由于 1 比特滑动窗口协议每次只要处理队列头的帧，因此使用 queue 结构比较合适。而退后 N 帧协议中由于发送超时时要退回重传，queue 的结构不在适合，因此选用了 deque 结构，能够依据索引来引用队列中的值。

总体来说，通过这次实验加深了对于滑动窗口协议的理解，收获比较大。