计算机网络原理实验报告——滑动窗口协议

张钰晖 计55 2015011372 yuhui-zh15@mails.tsinghua.edu.cn

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，一般在传输层TCP中实现，有些情况下也在数据链路层实现。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受窗口，发送窗口和接受窗口按照某种规律不断的向前滑动，滑动窗口协议由此得名。

本实验要求能够实现滑动窗口协议中的1bit滑动窗口协议和退后N帧协议，更深刻的理解滑动窗口协议。

二、实验原理

（1）窗口机制简介

在滑动窗口协议中，发送方始终保持一个已发送但尚未确认的帧的序号表，称为发送窗口。**发送窗口的上界表示要发送的下一个帧的序号，下界表示未得到确认的帧的最小序号。**发送窗口大小=上界-下界，大小可变。发送方每发送一个帧，序号取上界值，上界加1；每接收到一个确认序号等于发送窗口下界的正确响应帧，下界加1；若确认序号落在发送窗口之内，则发送窗口下界连续加1，直到发送窗口下界=确认序号+1。

接收方有一个接收窗口，大小固定，但不一定与发送窗口相同。**接收窗口的上界表示允许接收的最大序号，下界表示希望接收的序号。**接收窗口容纳允许接收的信息帧，落在窗口外的帧均被丢弃。序号等于下界的帧被正确接收，并产生一个响应帧，上界、下界都加1，接收窗口大小保持不变。

下图很好的说明了窗口机制（假设发送窗口大小为2，接收窗口大小为1）。

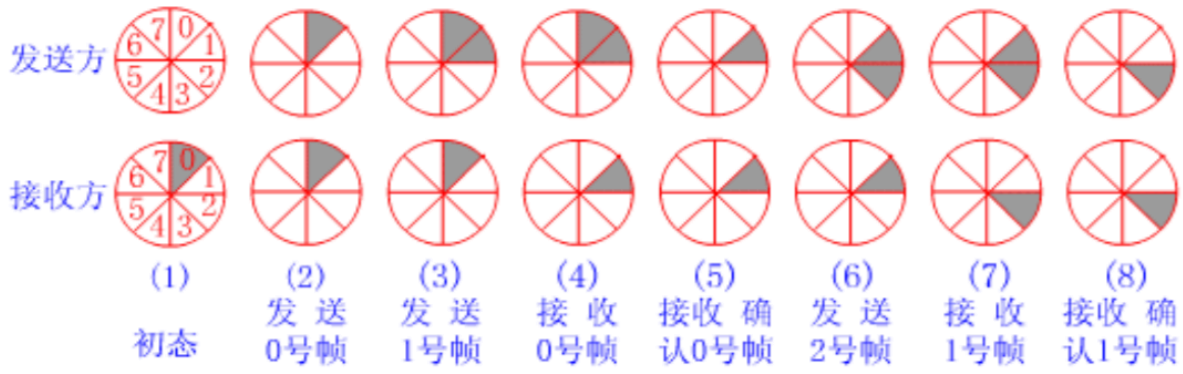
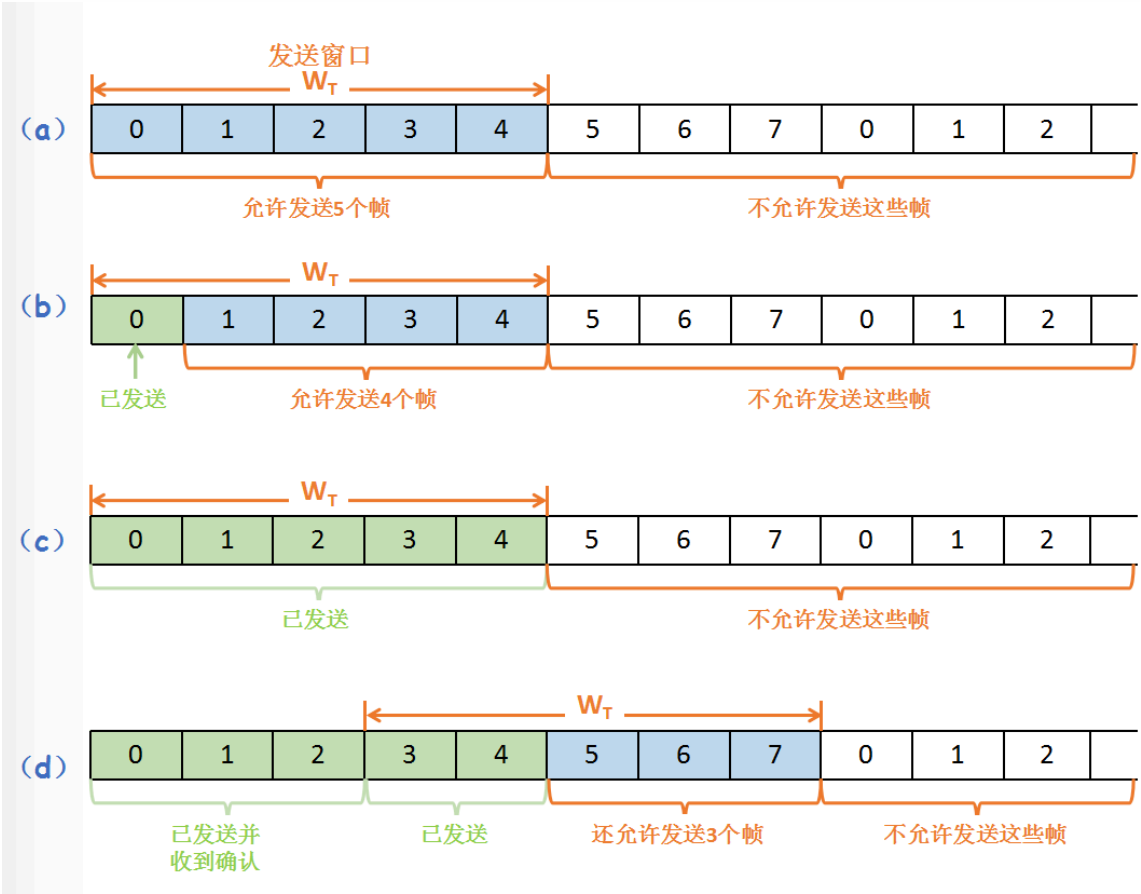


图 6.1　窗口机制示意图

下图也很好的说明了窗口机制（假设发送窗口大小为5）。



（2）1bit滑动窗口协议

当发送窗口和接收窗口的大小固定为1（即1bit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，因此只用1bit来编号就够了。

下图很好的说明了1bit滑动窗口协议的发送方和接收方运行流程。

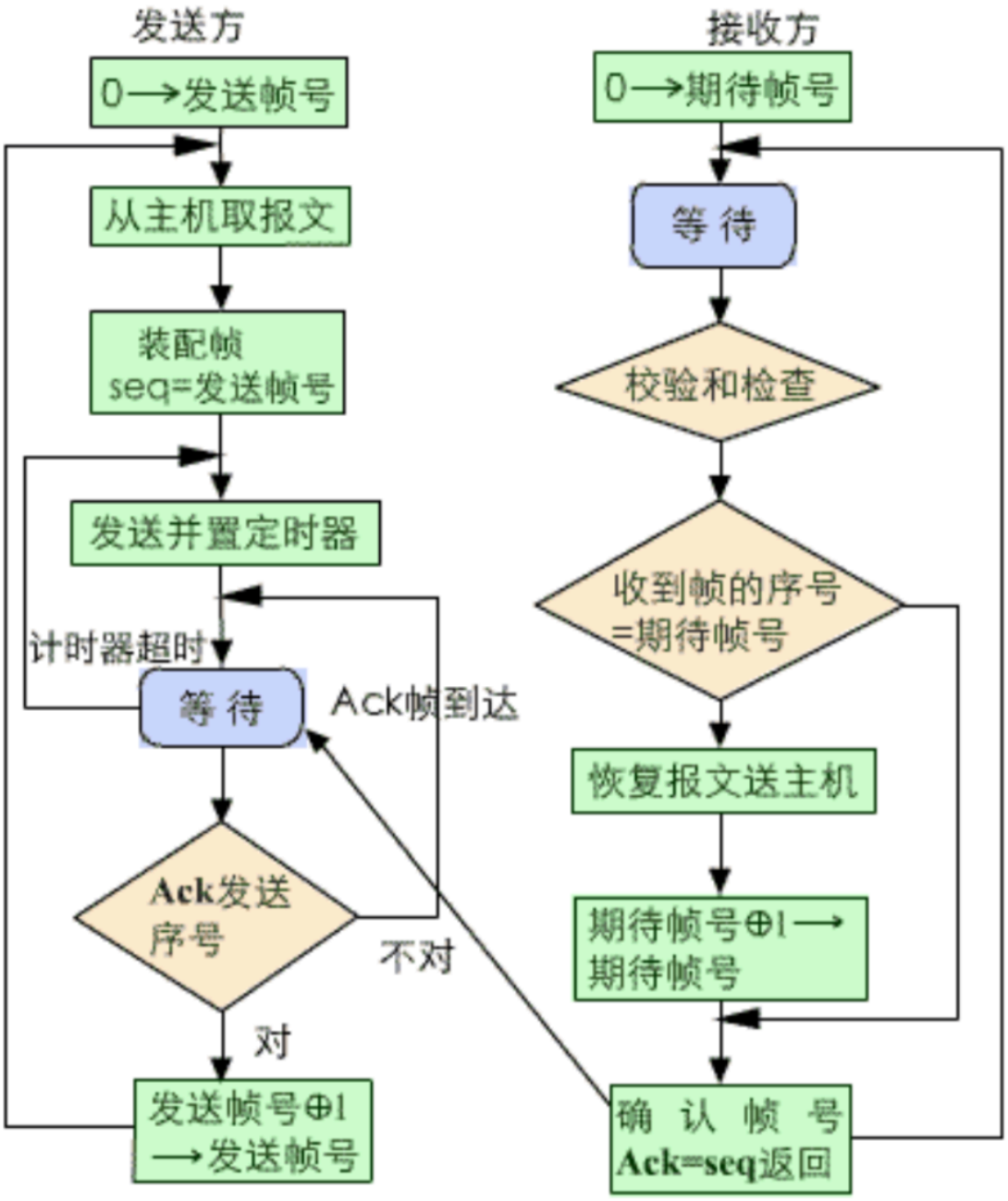
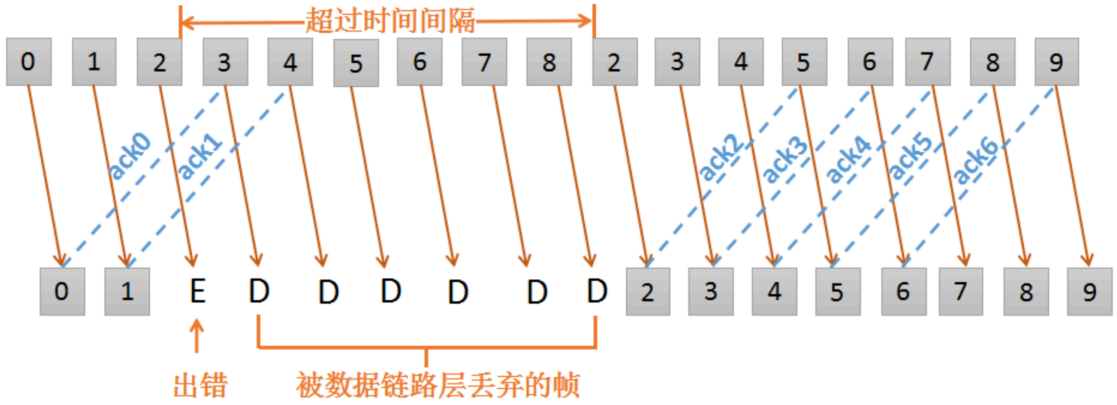


图 6.2　1 比特滑动窗口协议流程图

（3）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。

下图很好的说明了退后N帧协议机制。



退后N帧协议发送窗口较大，接收窗口仍是1。因连续发送数据帧，这种协议提高了效率；但在重传时，又必须把原来可能已正确传送过的数据帧进行重传，这种做法又使得传输效率降低。选择性重传可以弥补这一缺点，本次实验不涉及。

（4）实验处理流程

在两台主机通信环境中，网络拓扑可以简化为两台主机直接相连，中间的具体连接方式可以抽象为一条简单的链路。

在本次实验中，需要实现以下两个函数：

1. 1bit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()

在下列情况下，系统会主动调用实现的函数：

1. 当发送方需要发送帧时，调用函数，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。

三、实验内容

（1）实验要求

本实验要求在NetRiver实验系统环境中，用C语言实现1bit滑动窗口协议和退后N帧协议。根据滑动窗口协议原理，实现滑动窗口协议中发送方的功能，对发送方发出的帧进行缓存，等待确认，并在超时发生时对部分帧进行重传。

（2）接口函数说明

1. 1bit滑动窗口协议测试函数int stud_slide_window_stop_and_wait(char *pBuffer, int bufferSize, UINT8 messageType)

- pBuffer：指针，指向系统要发送或接收到的帧内容，或者指向超时消息中超时帧的序号内容。指向的数据为网络序，要转换成主机序，是从数据帧头开始的。
- bufferSize：pBuffer表示的内容长度（字节数）。
- messageType：传入的消息类型，可以分为以下三种：
  - MSG_TYPE_SEND：系统要发送一个帧
  - MSG_TYPE_RECEIVE：系统接收到一个帧的ACK
  - MSG_TYPE_TIMEOUT：某个帧超时

对于MSG_TYPE_TIMEOUT消息，pBuffer指向数据的前4字节为超时帧的序号，以UINT32类型存储，在与帧中的序号比较时，请注意字节序，并进行必要的转换。**（参数为主机序）**

对于MSG_TYPE_SEND和MSG_TYPE_RECEIVE类型消息，pBuffer指向数据的结构以如下代码中frame结构的定义。**（参数为网络序）**

```
typedef enum {data, ack, nak} frame_kind;
typedef struct frame_head {
    frame_kind kind;
    unsigned int seq;
    unsigned int ack;
    unsigned char data[100];
};
typedef struct frame {
    frame_head head;
    unsigned int size;
};
```

2. 退后N帧协议测试函数int stud_slide_window_back_n_frame(char *pBuffer, int bufferSize, UINT8 messageType)

   说明：该函数的参数定义和返回值，与上述1bit滑动窗口协议测试函数相同。

3. 发送帧函数void SendFRAMEPacket(unsigned char* pData, unsigned int len)
   - pData：指向要发送的帧的内容的指针，从帧头开始。数据需要以网络序存储。
   - len：要发送的帧的长度。

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?

   答：1bit滑动窗口协议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返回后才能继续发送下一帧，信道利用率很低。而退后N帧协议连续发送数据帧，这种协议提高了效率。

2. 退后N帧协议有什么缺点，如何改进?

   答：在重传时，必须把原来可能已正确传送过的数据帧进行重传，这种做法又使得传输效率降低。选择性重传可以弥补这一缺点。

五、问题及解决办法

1. 退后N帧协议TIMEOUT时，从队列中找到超时节点，并发送超时节点及其之后窗口内节点无法通过测试?

解决：按照实验原理，本应该这样处理，但因为服务器端数据处理有问题，导致接收方部分顺序错乱，改为重发窗口内所有节点即可通过测试。

2. 整数顺序有网络序和主机序，如何正确判断与转换?

   解决：在SEND和RECEIVE时接收到的数据为网络序，在TIMEOUT时接收到的顺序为主机序，这可以通过每次接受和发送数据时输出调试信息，并利用ntohl()函数完成网络序与主机序的转换，最终程序一致采用主机序。

3. 为什么程序整体采用双端队列实现?

   解决：考虑主要有以下两点：一是队列的顺序性(先进先出)充分模拟了滑动窗口，可以非常简单的实现，二是采用封装好的数据结构不必考虑内存回收问题(C++标准库已提供解决方案)，在连续发送大量数据时不会导致内存泄漏。

六、实验小结

本次实验中，我通过双端队列(deque)简洁的实现了经典的滑动窗口协议，原理在代码中进行了较为详尽的注释。

本次实验不算很复杂，实现过程中没有遇到特别严重的问题，大部分通过输出调试信息和尝试得到了解决，在这个过程中我对网络数据的传输也得到了更加深刻的理解。

尽管这次实验只是一次简单的尝试，由于时间原因，并没有来得及实现选择性重传协议，也没有实现更复杂的协议。但通过本次实验，我进入了协议世界的大门，在实验中充分理解了滑动窗口协议，并感受到滑动窗口协议简洁而高效稳定的传输特性。

希望今后能通过类似的编程实验更充分的理解计算机网络原理。

七、源代码

```
#include "sysinclude.h"
#include <iostream>
#include <deque>
using namespace std;

#define WINDOW_SIZE_STOP_WAIT 1 // 停等协议窗口大小
#define WINDOW_SIZE_BACK_N_FRAME 4 // 回退N帧协议窗口大小

extern void SendFRAMEPacket(unsigned char* pData, unsigned int len);
// 系统发送包函数

typedef enum {data, ack, nak} frame_kind;
typedef struct frame_head {
    frame_kind kind;
    unsigned int seq;
    unsigned int ack;
    unsigned char data[100];
};
typedef struct frame {
    frame_head head;
    unsigned int size;
```

```
};
typedef struct buffer {
    frame frm;
    int size;
};
/*
buffer 结构:
buffer+0    frame_kind  帧类型
buffer+4    seq         发送帧序号
buffer+8    ack         确认帧序号
buffer+12   data[100]   帧数据
buffer+112  size        帧数据大小
buffer+116  size        帧大小
*/

/*
* 停等协议测试函数
*/
deque<buffer> stop_and_wait_deque;
int stop_and_wait_window_size = 0;
int stud_slide_window_stop_and_wait(char *pBuffer, int bufferSize,
UINT8 messageType) {
    // 若类型为发送：构造帧，加入队列，若当前发送窗口未满，发送该帧
    if (messageType == MSG_TYPE_SEND) {
        buffer buf;
        buf.frm = *(frame*)pBuffer;
        buf.size = bufferSize;
        stop_and_wait_deque.push_back(buf);
        if (stop_and_wait_window_size < WINDOW_SIZE_STOP_WAIT) {
            stop_and_wait_window_size++;
            SendFRAMEPacket((unsigned char*)&buf.frm, buf.size);
        }
    }
    // 若类型为接收：若队列头序号等于确认号，弹出该元素，如果有等待发送的帧，继续发
送，如果没有，移动窗口下界
    else if (messageType == MSG_TYPE_RECEIVE) {
        unsigned int ack = ntohl(((frame*)pBuffer)->head.ack);
        if (!stop_and_wait_deque.empty()) {
            buffer buf = stop_and_wait_deque.front();
            printf("<DEBUG STOP_AND_WAIT RECEIVE>: buf.frm.head.seq =
%d, ack = %d\n", ntohl(buf.frm.head.seq), ack);
            if (ntohl(buf.frm.head.seq) == ack) {
                stop_and_wait_deque.pop_front();
                if (!stop_and_wait_deque.empty()) {
                    buf = stop_and_wait_deque.front();
                    SendFRAMEPacket((unsigned char*)&buf.frm,
buf.size);
                }
                else {
```

```
                    stop_and_wait_window_size--;
                }
            }
        }
    }
    // 若类型为超时：如果队列头序号等于超时号，重新发送该帧
    else if (messageType == MSG_TYPE_TIMEOUT) {
        unsigned int seq = *(unsigned int*)pBuffer;
        buffer buf = stop_and_wait_deque.front();
        printf("<DEBUG STOP_AND_WAIT TIMEOUT>: buf.frm.head.seq = %d,
seq = %d\n", ntohl(buf.frm.head.seq), seq);
        if (ntohl(buf.frm.head.seq) == seq) {
            SendFRAMEPacket((unsigned char*)&buf.frm, buf.size);
        }
    }
    return 0;
}

/*
* 回退n帧测试函数
*/
deque<buffer> back_n_frame_deque;
int back_n_frame_window_size = 0;
int stud_slide_window_back_n_frame(char *pBuffer, int bufferSize,
UINT8 messageType) {
    // 若类型为发送：构造帧，加入队列，若当前发送窗口未满，发送该帧
    if (messageType == MSG_TYPE_SEND) {
        buffer buf;
        buf.frm = *(frame*)pBuffer;
        buf.size = bufferSize;
        back_n_frame_deque.push_back(buf);
        if (back_n_frame_window_size < WINDOW_SIZE_BACK_N_FRAME) {
            back_n_frame_window_size++;
            SendFRAMEPacket((unsigned char*)&buf.frm, buf.size);
        }
    }
    // 若类型为接收：在队列中找到序号等于确认号的帧，弹出之前所有元素，移动窗口下
界，如果有等待发送的帧，继续发送至窗口已满，移动窗口上界
    else if (messageType == MSG_TYPE_RECEIVE) {
        unsigned int ack = ntohl(((frame*)pBuffer)->head.ack);
        int i, j;
        for (i = 0; i < back_n_frame_deque.size() && i <
back_n_frame_window_size; i++) {
            buffer buf = back_n_frame_deque[i];
            printf("<DEBUG BACK_N_FRAME RECEIVE>: buf.frm.head.seq =
%d, ack = %d\n", ntohl(buf.frm.head.seq), ack);
            if (ntohl(buf.frm.head.seq) == ack) break;
        }
```

```
109         if (i < back_n_frame_deque.size() && i < back_n_frame_window_size) {
110             for (j = 0; j <= i; j++) {
111                 back_n_frame_deque.pop_front();
112                 back_n_frame_window_size--;
113             }
114         }
115         for (i = back_n_frame_window_size; i < back_n_frame_deque.size(); i++) {
116             if (back_n_frame_window_size < WINDOW_SIZE_BACK_N_FRAME) {
117                 buffer buf = back_n_frame_deque[i];
118                 back_n_frame_window_size++;
119                 SendFRAMEPacket((unsigned char*)&buf.frm, buf.size);
120             }
121             else {
122                 break;
123             }
124         }
125     }
126     // 若类型为超时：在队列中找到序号等于超时号的帧，重发窗口内所有元素<见问题与解决方案>
127     else if (messageType == MSG_TYPE_TIMEOUT) {
128         unsigned int seq = *(unsigned int*)pBuffer;
129         int i, j;
130         for (i = 0; i < back_n_frame_deque.size() && i < back_n_frame_window_size; i++) {
131             buffer buf = back_n_frame_deque[i];
132             printf("<DEBUG BACK_N_FRAME TIMEOUT>: buf.frm.head.seq = %d, seq = %d\n", ntohl(buf.frm.head.seq), seq);
133             if (ntohl(buf.frm.head.seq) == seq) break;
134         }
135         if (i < back_n_frame_deque.size() && i < back_n_frame_window_size) {
136             for (j = 0; j < back_n_frame_deque.size() && j < back_n_frame_window_size; j++) {
137                 buffer buf = back_n_frame_deque[j];
138                 SendFRAMEPacket((unsigned char*)&buf.frm, buf.size);
139             }
140         }
141     }
142     return 0;
143 }
144
145 /*
146 * 选择性重传测试函数
147 */
148 int stud_slide_window_choice_frame_resend(char *pBuffer, int bufferSize, UINT8 messageType) {
149     return 0;
```

```
150 }
151
```