北 京 大 学

信息科学技术学院

本科生实验报告

实验课程：计算机网络概论

实验名称：滑动窗口协议

学生姓名：季凯航

学　　号：1400012727

提交时间：2016 年 11 月 27 日

目录

一、实验目的

计算机网络的数据链路层协议保证通信双方在有差错的通信线路上进行无差错的数据传输，是计算机网络各层协议中通信控制功能最典型的一种协议。本实验实现一个数据链路层协议的数据传送部分，目的在于使学生更好地理解数据链路层协议中的“滑动窗口”技术的基本工作原理，掌握计算机网络协议的基本实现技术。

二、实验要求

在一个数据链路层的模拟实现环境中，用 C 语言实现三个数据链路层协议：1 比特滑动窗口协议、回退 N 帧滑动窗口协议、选择性重传协议。(实际实现时使用了 C++ STL 和其他 C++特性)

三、实验内容

充分理解滑动窗口协议，根据滑动窗口协议，模拟滑动窗口协议中发送端的功能，对系统发送的帧进行缓存并加入窗口等待确认，并在超时或者错误时对部分帧进行重传。

编写停等及退回 N 滑动窗口协议函数，响应系统的发送请求、接收帧消息以及超时消息，并根据滑动窗口协议进行相应处理。

编写选择性重传协议函数，响应系统的发送请求、接受帧消息以及错误消息，并根据滑动窗口协议进行相应处理。

四、数据结构说明

frame_head

帧头。包含帧类型 type，序号 seq_num，确认码 ack，数据 data。

frame

帧。包含帧头 head，数据长度 size。

frame_to_send_queue

待发送序列。

backup_buffer_queue

已发送序列（用于重传）。

sent_buf_num

已发送缓存的帧数。

五、实验过程

1. 停等协议

停等式协议的滑动窗口大小为 1，即最多只能有 1 个已发送并缓存的帧。

若为 MSG_TYPE_SEND，即发送方将发送一个帧。此时将新帧加入待发送队列。若 backup_buffer_queue 中无缓存帧，则发送该帧并将其移入 backup_buffer_queue。否则该帧将在 backup_buffer_queue 为空时再发送和缓存。

若为 MSG_TYPE_RECEIVE，即发送方接收到确认帧。此时将缓存帧从 backup_buffer_queue 中删除，并检查待发送队列中是否有其他待发送帧。如果有，则发送并缓存。

若为 MSG_TYPE_TIMEOUT，表示接收方未正确接收帧。此时将 backup_buffer_queue 中的缓存帧重新发送。

2. 回退 N 协议

停等式协议的滑动窗口大小为 N，与窗口大小相关的操作需要修改。

若为 MSG_TYPE_SEND，即发送方将发送一个帧。此时将新帧加入待发送队列。若当前 backup_buffer_queue 中缓存帧不足 N，则发送帧并移入 backup_buffer_queue。

若为 MSG_TYPE_RECEIVE，即发送方接收到确认帧。此时检查所接收帧的帧头的 type 字段，若其类型非 ACK 则简单抛弃，否则获取其帧的序号。此时，backup_buffer_queue 中序号小于该序号的帧皆已确认接收，可将缓存帧从 backup_buffer_queue 中删除。若此后 backup_buffer_queue 中缓存帧不足 N，则循环发送帧并缓存。

若为 MSG_TYPE_TIMEOUT，即接收方未正确接收帧。此时将所有缓存帧重新发送。

3. 选择重传协议

在选择重传协议中，未成功接收的帧会返回 NAK 信息，故不再存在超时情

况。

若为 MSG_TYPE_SEND，操作与前一协议相同。

若为 MSG_TYPE_RECEIVE，则需检查所接收帧的帧头的 type 字段，以区分 ACK 与 NAK 信息。若为 ACK，操作与回退 N 协议相同。若为 NAK，则获取其帧的序号。此时，backup_buffer_queue 中序号小于该序号的帧皆已确认接收，可将缓存帧从 backup_buffer_queue 中删除。重传序号为 N 的帧；若此后 backup_buffer_queue 中缓存帧不足 N，则循环发送帧并缓存。

六、附录——代码

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

extern void SendFRAMEPacket(unsigned char* pData, unsigned int len);
#define WINDOW_SIZE_STOP_WAIT 1
#define WINDOW_SIZE_BACK_N_FRAME 4

#define DATA_BUFFER_SCALE 255

#ifndef  FRAME_TYPE
#define FRAME_TYPE
#define DATA 0
#define ACK 1
#define NAK 2
#endif

#define uint_t unsigned int
#define uchar_t unsigned char

typedef struct frame_head {
     uint_t type;
     uint_t seq_num;
     uint_t ack;
     uchar_t data[DATA_BUFFER_SCALE];
};

typedef struct frame {
     frame_head head;
```

```
    uint_t size;
};

queue<frame> frame_to_send_queue;
queue<frame> backup_buffer_queue;
frame frame_to_send;
uint_t sent_buf_num = 0;

inline uint_t reverse(const uint_t data) {
    // Reverse the word order
    return(((data & 0xff000000) >> 24) | ((data & 0xff0000) >> 8) | ((data & 0xff00) << 8) | ((data
& 0xff) << 24));
}


/*
* 停等协议测试函数
*/
int stud_slide_window_stop_and_wait(char* pBuffer, int bufferSize, UINT8 messageType) {
    switch (messageType) {
    case MSG_TYPE_TIMEOUT:
        frame_to_send = backup_buffer_queue.front();
        SendFRAMEPacket((unsigned char *)&frame_to_send, frame_to_send.size);
        return 0;
    case MSG_TYPE_SEND:
        memcpy(&frame_to_send, pBuffer, sizeof(frame));
        frame_to_send.size = bufferSize;
        frame_to_send_queue.push(frame_to_send);
        if (sent_buf_num < WINDOW_SIZE_STOP_WAIT) {
            frame_to_send = frame_to_send_queue.front();
            frame_to_send_queue.pop();
            backup_buffer_queue.push(frame_to_send);
            SendFRAMEPacket((unsigned char *)&frame_to_send, frame_to_send.size);
            sent_buf_num++;
        }
        return 0;
    case MSG_TYPE_RECEIVE:
        backup_buffer_queue.pop();
        sent_buf_num -= 1;
        if (!frame_to_send_queue.empty()) {
            frame_to_send = frame_to_send_queue.front();
            frame_to_send_queue.pop();
            backup_buffer_queue.push(frame_to_send);
            SendFRAMEPacket((unsigned char *)&frame_to_send, frame_to_send.size);
```

```
                sent_buf_num++;
            }
            return 0;
        DEFAULT:
            cerr << "Undefined Message Type." << endl;
            return -1;
        }
}

/*
* 回退帧测试函数n
*/
int stud_slide_window_back_n_frame(char* pBuffer, int bufferSize, UINT8 messageType) {
        switch (messageType) {
        case MSG_TYPE_TIMEOUT:
            for (int i = 0; i < sent_buf_num; i++) {
                frame_to_send = backup_buffer_queue.front();
                backup_buffer_queue.pop();
                backup_buffer_queue.push(frame_to_send);
                SendFRAMEPacket((unsigned char *)&frame_to_send, frame_to_send.size);
            }
            return 0;
        case MSG_TYPE_SEND:
            memcpy(&frame_to_send, pBuffer, sizeof(frame));
            frame_to_send.size = bufferSize;
            frame_to_send_queue.push(frame_to_send);
            while (sent_buf_num < WINDOW_SIZE_BACK_N_FRAME && !frame_to_send_queue.empty()) {
                frame_to_send = frame_to_send_queue.front();
                frame_to_send_queue.pop();
                backup_buffer_queue.push(frame_to_send);
                SendFRAMEPacket((unsigned char *)&frame_to_send, frame_to_send.size);
                sent_buf_num++;
            }
            return 0;
        case MSG_TYPE_RECEIVE:
            int stop_ack = reverse((((frame*)pBuffer)->head).ack);
            int type = reverse((((frame*)pBuffer)->head).type);
            if (type != ACK) {
                // do nothing
                return 0;
            }

            while (reverse((backup_buffer_queue.front()).head.seq_num) <= stop_ack) {
                backup_buffer_queue.pop();
```

```
                sent_buf_num--;
                if (sent_buf_num < WINDOW_SIZE_BACK_N_FRAME && !frame_to_send_queue.empty()) {
                    frame_to_send = frame_to_send_queue.front();
                    frame_to_send_queue.pop();
                    backup_buffer_queue.push(frame_to_send);
                    SendFRAMEPacket((unsigned char *)&frame_to_send, frame_to_send.size);
                    sent_buf_num++;
                }
            }
            return 0;
        DEFAULT:
            cerr << "Undefined Message Type." << endl;
            return -1;
        }
}

/*
* 选择性重传测试函数
*/
int stud_slide_window_choice_frame_resend(char* pBuffer, int bufferSize, UINT8 messageType) {
    switch (messageType) {
    case MSG_TYPE_SEND:
        memcpy(&frame_to_send, pBuffer, sizeof(frame));
        frame_to_send.size = bufferSize;
        frame_to_send_queue.push(frame_to_send);
        while (sent_buf_num < WINDOW_SIZE_BACK_N_FRAME && !frame_to_send_queue.empty()) {
            frame_to_send = frame_to_send_queue.front();
            frame_to_send_queue.pop();
            backup_buffer_queue.push(frame_to_send);
            SendFRAMEPacket((unsigned char *)&frame_to_send, frame_to_send.size);
            sent_buf_num++;
        }
        return 0;
    case MSG_TYPE_RECEIVE:
        int stop_ack = reverse((((frame*)pBuffer)->head).ack);
        int type = reverse((((frame*)pBuffer)->head).type);
        if (type == ACK) {
            while (reverse((backup_buffer_queue.front()).head.seq_num) <= stop_ack) {
                backup_buffer_queue.pop();
                sent_buf_num--;
                if (sent_buf_num < WINDOW_SIZE_BACK_N_FRAME && !frame_to_send_queue.empty()) {
                    frame_to_send = frame_to_send_queue.front();
                    frame_to_send_queue.pop();
                    backup_buffer_queue.push(frame_to_send);
```

```
                    SendFRAMEPacket((unsigned char *)&frame_to_send, frame_to_send.size);
                    sent_buf_num++;
                }
            }
        }
        else if (type == NAK) {
            while (reverse((backup_buffer_queue.front()).head.seq_num) < stop_ack) {
                backup_buffer_queue.pop();
                sent_buf_num--;
            }
            frame_to_send = backup_buffer_queue.front();
            SendFRAMEPacket((unsigned char *)&frame_to_send, frame_to_send.size);
            while (sent_buf_num < WINDOW_SIZE_BACK_N_FRAME && !frame_to_send_queue.
                empty()) {
                frame_to_send = frame_to_send_queue.front();
                frame_to_send_queue.pop();
                backup_buffer_queue.push(frame_to_send);
                SendFRAMEPacket((unsigned char *)&frame_to_send, frame_to_send.size);
                sent_buf_num++;
            }
        }
        return 0;
    DEFAULT:
        cerr << "Undefined Message Type." << endl;
        return -1;
    }
}
```