

实验2：可靠数据传输协议-停等协议的设计与实现

1.实验目的

理解可靠数据传输的基本原理；掌握停等协议的工作原理；掌握基于UDP 设计并实现一个停等协议的过程与技术。

2.实验环境

- ④接入Internet的实验主机；
- ④Windows xp、Windows 7或Windows 11；
- ④开发语言：C/C++（或Java）等。

3.实验内容

- 1) 基于UDP 设计一个简单的停等协议，实现单向可靠数据传输（服务器到客户的数据传输）。
- 2) 模拟引入数据包的丢失，验证所设计协议的有效性。
- 3) 改进所设计的停等协议，支持双向数据传输；
- 4) 基于所设计的停等协议，实现一个C/S 结构的文件传输应用。

4.实验方式

每位同学独立上机编程实验，实验指导教师现场指导。

5.实验要点

- 1) 基于UDP 实现的停等协议，可以不进行差错检测，可以利用UDP

协议差错检测;

2) 为了验证所设计协议是否可以处理数据丢失, 可以考虑在数据接收端或发送端引入数据丢失。

3) 在开发停等协议之前, 需要先设计协议数据分组格式以及确认分组格式。

4) 计时器实现方法: 对于阻塞的socket 可用`int setsockopt(int socket, int level, int option_name, const void* option_value, size_t option_len)`函数设置套接字发送与接收超时时间; 对于非阻塞socket 可以使用累加sleep 时间的方法判断socket 接受数据是否超时(当时间累加量超过一定数值时则认为套接字接受数据超时)。

6.实验报告

在实验报告中要说明所设计停等协议数据分组格式、确认分组格式、各个域作用, 协议两端程序流程图, 协议典型交互过程, 数据分组丢失验证模拟方法, 程序实现的主要类 (或函数) 及其主要作用、 UDP 编程的主要特点、实验验证结果, 详细注释源程序等。

实验 3：可靠数据传输协议-GBN 协议的设计与实现

1.实验目的

理解滑动窗口协议的基本原理；掌握 GBN 的工作原理；掌握基于 UDP 设计并实现一个 GBN 协议的过程与技术。

2.实验环境

- ④接入 Internet 的实验主机；
- ④Windows xp、Windows 7 或 Windows 11；
- ④开发语言： C/C++（或Java）等。

3.实验内容

- 1) 基于UDP 设计一个简单的GBN 协议，实现单向可靠数据传输（服务器到客户的数据传输）。
- 2) 模拟引入数据包的丢失，验证所设计协议的有效性。
- 3) 改进所设计的GBN 协议，支持双向数据传输；
- 4) 将所设计的GBN 协议改进为SR协议。

4.实验方式

每位同学独立上机编程实验，实验指导教师现场指导。

5.实验要点

- 1) 基于 UDP 实现的 GBN 协议，可以不进行差错检测，可以利用

UDP 协议差错检测;

2) 自行设计数据帧的格式, 应至少包含序列号 Seq 和数据两部分;

3) 自行定义发送端序列号 Seq 比特数 L 以及发送窗口大小 W , 应满足条件 $W+1 \leq 2^L$ 。

4) 一种简单的服务器端计时器的实现办法: 设置套接字为非阻塞方式, 则服务器端在 `recvfrom` 方法上不会阻塞, 若正确接收到 ACK 消息, 则计时器清零, 若从客户端接收数据长度为 -1 (表示没有接收到任何数据), 则计时器+1, 对计时器进行判断, 若其超过阈值, 则判断为超时, 进行超时重传。(当然, 如果服务器选择阻塞模式, 可以用到 `select` 或 `epoll` 的阻塞选择函数, 详情见 MSDN)

5) 为了模拟 ACK 丢失, 一种简单的实现办法: 客户端对接收的数据帧进行计数, 然后对总数进行模 N 运算, 若规定求模运算结果为零则返回 ACK, 则每接收 N 个数据帧才返回 1 个 ACK。当 N 取值大于服务器端的超时阈值时, 则会出现服务器端超时现象。

6) 当设置服务器端发送窗口的大小为 1 时, GBN 协议就是停-等协议。

6.参考内容

作为只实现单向数据传输的 GBN 协议, 实质上就是实现为一个 C/S 应用。

服务器端: 使用 UDP 协议传输数据 (比如传输一个文件), 等待客户端的请求, 接收并处理来自客户端的消息 (如数据传输请求), 当客户端开始请求数据时进入“伪连接”状态 (并不是真正的连接, 只是一种类似连接的数据发送的状态), 将数据打包成数据报发送, 然后等待客户端的 ACK 信息, 同时启动计时器。当收到 ACK 时, 窗口滑动, 正常发送下一个数据报, 计时器重新计时; 若在计时器超时前没有收到 ACK, 则全部重传窗口内的所以已发送的数据报。

客户端: 使用 UDP 协议向服务器端请求数据, 接收服务器端发送的

数据报并返回确认信息ACK（注意GBN为累积确认，即若ACK=1和3，表示数据帧2已经正确接收），必须能够模拟ACK丢失直至服务器端超时重传的情况。

(1) 服务器端设计参考

1) 命令解析

为了测试客户端与服务器端的通信交互，方便操作，设置了此过程。首先，服务器接收客户端发来的请求数据，

“-time”表示客户端请求获取当前时间，服务器回复当前时间；

“-quit”表示客户端退出，服务器回复“Goodbye!”；

“-testgbn”表示客户端请求开始测试GBN协议，服务器开始进入GBN传输状态；

其他数据，则服务器直接回复原数据。

2) 数据传输数据帧格式定义

在以太网中，数据帧的MTU为1500字节，所以UDP数据报的数据部分应小于1472字节（除去IP头部20字节与UDP头的8字节），为

此，定义UDP数据报的数据部分格式为：

Seq	Data	0
-----	------	---

Seq为1个字节，取值为0~255，故序列号最多为256个）；

Data≤1024个字节，为传输的数据；

最后一个字节放入EOF0，表示结尾。

3) 源代码

```
#include "stdafx.h" //创建VS项目包含的预编译头文件
#include <stdlib.h>
#include <time.h>
#include <WinSock2.h>
#include <fstream>

#pragma comment(lib, "ws2_32.lib")
```

```

#define SERVER_PORT      12340      //端口号
#define SERVER_IP        "0.0.0.0" //IP 地址
constint BUFFER_LENGTH = 1026;      //缓冲区大小, 以太网中 UDP 的
数据
帧中包长度应小于 1480 字节)

constint SEND_WIND_SIZE = 10; //发送窗口大小为 10, GBN 中应满足  $W + 1 \leq N$ 
(W 为发送窗口大小, N 为序列号个数)

//本例取序列号 0...19 共 20 个
//如果将窗口大小设为 1, 则为停-等协议

constint SEQ_SIZE = 20; //序列号的个数, 从 0~19 共计 20 个
//由于发送数据第一个字节如果值为 0, 则数据会发送
失败

//因此接收端序列号为 1~20, 与发送端一一对应

BOOL ack[SEQ_SIZE]; //收到ack 情况, 对应 0~19 的ack
int curSeq; //当前数据包的 seq
int curAck; //当前等待确认的 ack
int totalSeq; //收到的包的总数
int totalPacket; //需要发送的包总数

//*****
// Method:      getCurTime
// FullName:    getCurTime
// Access:      public
// Returns:     void
// Qualifier:   获取当前系统时间, 结果存入ptime 中
// Parameter:  char * ptime
//*****
void getCurTime(char *ptime){
    char buffer[128];
    memset(buffer, 0, sizeof(buffer));
    time_t c_time;
    struct tm *p;
    time(&c_time);
    p = localtime(&c_time);

```

```
    sprintf_s(buffer, "%d/%d/%d %d:%d:%d",
        p->tm_year + 1900,
        p->tm_mon,
        p->tm_mday,
        p->tm_hour,
        p->tm_min,
        p->tm_sec);
    strcpy_s(pTIME, sizeof(buffer), buffer);
}

/*****
// Method:          seqIsAvailable
// FullName:        seqIsAvailable
// Access:          public
// Returns:         bool
// Qualifier:      当前序列号 curSeq 是否可用
*****/
bool seqIsAvailable(){
    int step;
    step = curSeq - curAck;
    step = step >= 0 ? step : step + SEQ_SIZE;
    //序列号是否在当前发送窗口之内
    if(step >= SEND_WIND_SIZE){
        return false;
    }
    if(ack[curSeq]){
        return true;
    }
    return false;
}

/*****
// Method:          timeoutHandler
// FullName:        timeoutHandler
// Access:          public
*****/
```

```

// Returns:      void
// Qualifier:  超时重传处理函数, 滑动窗口内的数据帧都要重传
//*****
void timeoutHandler(){
    printf("Timer out error.\n");
    int index;
    for(int i = 0; i < SEND_WIND_SIZE; ++i){
        index = (i + curAck) % SEQ_SIZE;
        ack[index] = TRUE;
    }
    totalSeq -= SEND_WIND_SIZE;
    curSeq = curAck;
}

//*****
// Method:      ackHandler
// FullName:    ackHandler
// Access:      public
// Returns:     void
// Qualifier:   收到ack, 累积确认, 取数据帧的第一个字节
//由于发送数据时, 第一个字节 (序列号) 为 0 (ASCII) 时发送失败, 因此加一了
, 此处需要减一还原
// Parameter:  char c
//*****
void ackHandler(char c){
    unsigned char index = (unsigned char)c - 1; //序列号减一
    printf("Recv a ack of %d\n", index);
    if(curAck <= index){
        for(int i = curAck; i <= index; ++i){
            ack[i] = TRUE;
        }
        curAck = (index + 1) % SEQ_SIZE;
    }else{
        //ack 超过了最大值, 回到了curAck 的左边
        for(int i = curAck; i < SEQ_SIZE; ++i){

```



```
        ack[i] = TRUE;
    }
    for(int i = 0; i <= index; ++i){
        ack[i] = TRUE;
    }
    curAck = index + 1;
}
}

//主函数
int main(int argc, char* argv[])
{
    //加载套接字库（必须）
    WORD wVersionRequested;
    WSADATA wsaData;
    //套接字加载时错误提示
    int err;
    //版本2.2
    wVersionRequested = MAKEWORD(2, 2);
    //加载dll 文件Socket 库
    err = WSAStartup(wVersionRequested, &wsaData);
    if(err != 0){
        //找不到winsock.dll
        printf("WSAStartup failed with error: %d\n", err);
        return -1;
    }
    if(LOBYTE(wsaData.wVersion) != 2 || HIBYTE(wsaData.wVersion) != 2)
    {
        printf("Could not find a usable version of Winsock.dll\n");
        WSACleanup();
    }else{
        printf("The Winsock 2.2 dll was found okay\n");
    }
    SOCKET sockServer = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);
    //设置套接字为非阻塞模式
```

```

intiMode = 1; //1: 非阻塞, 0: 阻塞
ioctlsocket(sockServer, FIONBIO, (u_long FAR*) &iMode); //非阻塞设置
SOCKADDR_IN addrServer; //服务器地址
//addrServer.sin_addr.S_un.S_addr = inet_addr(SERVER_IP);
addrServer.sin_addr.S_un.S_addr = htonl(INADDR_ANY); //两者均可
addrServer.sin_family = AF_INET;
addrServer.sin_port = htons(SERVER_PORT);
err = bind(sockServer, (SOCKADDR*)&addrServer, sizeof(SOCKADDR));
if(err){
    err = GetLastError();
    printf("Could not bind the port %d for
socket.Error code is %d\n", SERVER_PORT, err);
    WSACleanup();
    return -1;
}

SOCKADDR_IN addrClient; //客户端地址
int length = sizeof(SOCKADDR);
char buffer[BUFFER_LENGTH]; //数据发送接收缓冲区
ZeroMemory(buffer, sizeof(buffer));
//将测试数据读入内存
std::ifstream icin;
icin.open("../test.txt");
char data[1024 * 113];
ZeroMemory(data, sizeof(data));
icin.read(data, 1024 * 113);
icin.close();
totalPacket = sizeof(data) / 1024;
int recvSize ;
for(int i=0; i < SEQ_SIZE; ++i){
    ack[i] = TRUE;
}
while(true){
    //非阻塞接收, 若没有收到数据, 返回值为-1
    recvSize

```

```

recvfrom(sockServer,buffer,BUFFER_LENGTH,0,((SOCKADDR*)&addrClient),&length);
    if(recvSize < 0){
        Sleep(200);
        continue;
    }
    printf("recv from client: %s\n",buffer);
    if(strcmp(buffer,"-time") == 0){
        getCurTime(buffer);
    }else if(strcmp(buffer,"-quit") == 0){
        strcpy_s(buffer,strlen("Goodbye!") + 1,"Goodbye!");
    }else if(strcmp(buffer,"-testgbn") == 0){
        //进入 gbn 测试阶段
        //首先 server (server 处于0 状态) 向client 发送205 状态码 (server
进入 1 状态)
        //server 等待client 回复200 状态码, 如果收到 (server 进入2 状态),
则开始传输文件, 否则延时等待直至超时\
        //在文件传输阶段, server 发送窗口大小设为
ZeroMemory(buffer,sizeof(buffer));
        intrecvSize;
        intwaitCount = 0;
        printf("Begainto test GBN protocol,please don't abort the process\n");
        //加入了一个握手阶段
        //首先服务器向客户端发送一个 205 大小的状态码 (我自己定义的)
表示服务器准备好了, 可以发送数据
        //客户端收到 205 之后回复一个 200 大小的状态码, 表示客户端
准 备好了, 可以接收数据了
        //服务器收到200 状态码之后, 就开始使用GBN 发送数据了
        printf("Shake hands stage\n");
        int stage = 0;
        bool runFlag = true;
        while(runFlag){
            switch(stage){
                case 0://发送205 阶
                    段 buffer[0] =
                        205;
                        buffer,
                        strlen(buffer)+1,
                        0,
                        sendto(sockServer,

```

```

(SOCKADDR*)&addrClient, sizeof(SOCKADDR));
        Sleep(100);
        stage = 1;
        break;
    case 1://等待接收200 阶段，没有收到则计数器+1，超时则
    放弃此次“连接”，等待从第一步开始
        recvSize =
recvfrom(sockServer,buffer,BUFFER_LENGTH,0,((SOCKADDR*)&addrClient),&length);
        if(recvSize < 0){
            ++waitCount;
            if(waitCount > 20){
                runFlag = false;
                printf("Timeout error\n");
                break;
            }
            Sleep(500);
            continue;
        }else{
            if((unsigned char)buffer[0] == 200){
                printf("Begin a file transfer\n");
                printf("File size is %dB, each packet is 1024B
and packet total num is %d\n",sizeof(data),totalPacket);
                curSeq = 0;
                curAck = 0;
                totalSeq = 0;
                waitCount = 0;
                stage = 2;
            }
        }
        break;
    case 2://数据传输阶段
        if(seqIsAvailable()){
            //发送给客户端的序列号从 1 开始
            buffer[0] = curSeq + 1;
            ack[curSeq] = FALSE;

```

```
//数据发送的过程中应该判断是否传输完成
//为简化过程此处并未实现

memcpy(&buffer[1],data + 1024 * totalSeq,1024);
printf("send apacket with a seq of %d\n",curSeq);

sendto(sockServer, buffer, BUFFER_LENGTH,
0, (SOCKADDR*)&addrClient, sizeof(SOCKADDR));

++curSeq;
curSeq %= SEQ_SIZE;
++totalSeq;
Sleep(500);
}

//等待 Ack, 若没有收到, 则返回值为-1, 计数器+1
recvSize =
recvfrom(sockServer,buffer,BUFFER_LENGTH,0,((SOCKADDR*)&addrClient),&length);
if(recvSize < 0){
    waitCount++;
    //20 次等待ack 则超时重传
    if (waitCount > 20)
    {
        timeoutHandler();
        waitCount = 0;
    }
}else{
    //收到 ack
    ackHandler(buffer[0]);
    waitCount = 0;
}
Sleep(500);
break;
}
}

sendto(sockServer, buffer, strlen(buffer)+1, 0, (SOCKADDR*)&addrClient,
sizeof(SOCKADDR));
Sleep(500);
```

```
}  
//关闭套接字, 卸载库  
closesocket(sockServer);  
WSACleanup();  
return 0;  
}
```

(2) 客户端设计参考

1) ACK 数据帧定义



由于是从服务器端到客户端的单向数据传输, 因此ACK 数据帧不包含任何数据, 只需要将ACK 发送给服务器端即可。

ACK 字段为一个字节, 表示序列号数值;

末尾放入0, 表示数据结束。

2) 命令设置

客户端的命令和服务端端的解析命令向对应, 获取当前用户输入并发送给服务器并等待服务器返回数据, 如输入“-time”得到服务器的当前时间。

此处重点介绍 “-testgbn [X] [Y]” 命令, [X],[Y]均为[0,1]的小数, 其中:

[X]表示客户端的丢包率, 模拟网络中报文丢失;

[Y]表示客户端的ACK 的丢失率。(使用随机函数完成)。

如果用户不输入, 则默认丢失率均为 0.2。

3) 源代码

```
// GBN_client.cpp : 定义控制台应用程序的入口点。  
//  
#include "stdafx.h"  
#include <stdlib.h>  
#include <WinSock2.h>  
#include <time.h>
```

```

#pragma comment(lib,"ws2_32.lib")

#define SERVER_PORT    12340 //接收数据的端口号
#define SERVER_IP      " 127.0.0.1" // 服务器的IP 地址

const int BUFFER_LENGTH = 1026;
const int SEQ_SIZE = 20; //接收端序列号个数, 为 1~20

/*****
/*      -time 从服务器端获取当前时间
      -quit 退出客户端
      -testgbn [X] 测试GBN 协议实现可靠数据传输
              [X] [0,1] 模拟数据包丢失的概率
              [Y] [0,1] 模拟ACK 丢失的概率
*/
*****/

void printTips(){
    printf("*****\n");
    printf("|      -time to get current      |\n");
    printf("|      -quit to exit client      |\n");
    printf("|      -testgbn [X] [Y] to test the gbn      |\n");
    printf("*****\n");
}

/*****
// Method:    lossInLossRatio
// FullName:  lossInLossRatio
// Access:    public
// Returns:   BOOL
// Qualifier:  根据丢失率随机生成一个数字, 判断是否丢失, 丢失则
返回 TRUE, 否则返回FALSE
// Parameter: float lossRatio [0,1]
*****/
BOOL lossInLossRatio(float lossRatio){
    int lossBound = (int) (lossRatio * 100);

```

```
    intr = rand() % 101;
    if(r <= lossBound){
        return TRUE;
    }
    return FALSE;
}

int main(int argc, char* argv[])
{
    //加载套接字库 (必须)
    WORD wVersionRequested;
    WSADATA wsaData;
    //套接字加载时错误提示
    int err;
    //版本 2.2
    wVersionRequested = MAKEWORD(2, 2);
    //加载 dll 文件 Socket 库
    err = WSAStartup(wVersionRequested, &wsaData);
    if(err != 0){
        //找不到 winsock.dll
        printf("WSAStartup failed with error: %d\n", err);
        return 1;
    }
    if(LOBYTE(wsaData.wVersion) != 2 || HIBYTE(wsaData.wVersion) != 2)
    {
        printf("Could not find a usable version of Winsock.dll\n");
        WSACleanup();
    }else{
        printf("The Winsock 2.2 dll was found okay\n");
    }
    SOCKET socketClient = socket(AF_INET, SOCK_DGRAM, 0);
    SOCKADDR_IN addrServer;

    addrServer.sin_addr.S_un.S_addr = inet_addr(SERVER_IP);
    addrServer.sin_family = AF_INET;
    addrServer.sin_port = htons(SERVER_PORT);
```



```

//接收缓冲区
char buffer[BUFFER_LENGTH];
ZeroMemory(buffer,sizeof(buffer));
int len = sizeof(SOCKADDR);
//为了测试与服务器的连接,可以使用 -time 命令从服务器端获得当前
时间
//使用 -testgbn [X] [Y] 测试GBN 其中[X]表示数据包丢失概
率 //
[Y]表示ACK 丢包概率 printTips();

intret;

int interval = 1;//收到数据包之后返回ack 的间隔, 默认为1 表示每个都
返回ack, 0 或者负数均表示所有的都不返回ack

char cmd[128];
float packetLossRatio = 0.2; //默认包丢失率0.2
float ackLossRatio = 0.2; //默认ACK 丢失率0.2
//用时间作为随机种子, 放在循环的最外面
srand((unsigned)time(NULL));
while(true){
    gets_s(buffer);
    ret
    =
    sscanf(buffer,"%s%f%f",&cmd,&packetLossRatio,&ackLossRatio);
    //开始GBN 测试, 使用GBN 协议实现UDP 可靠文件传输
    if(!strcmp(cmd,"-testgbn")){
        printf("%s\n","Begin to test GBN protocol, please don't abort the
process");
        printf("The loss ratio of packet is %.2f,the loss
ratio of ack is %.2f\n",packetLossRatio,ackLossRatio);
        intwaitCount = 0;
        int stage = 0;
        BOOLb;
        unsigned charu_code;//状态码
        unsigned short seq;//包的序列号
        unsigned short recvSeq;//接收窗口大小为1, 已确认的序列号
        unsigned short waitSeq;//等待的序列号
        sendto(socketClient, "-testgbn",
        strlen("-testgbn")+1, 0,

```

```

(SOCKADDR*)&addrServer, sizeof(SOCKADDR));
    while (true)
    {
        //等待 server 回复设置UDP 为阻塞模式

        recvfrom(socketClient,buffer,BUFFER_LENGTH,0,(SOCKADDR*)&addrServer, &len);

        switch(stage){
        case 0://等待握手阶段
            u_code = (unsigned char)buffer[0];
            if((unsigned char)buffer[0] == 205)
            {
                printf("Ready for file transmission\n");
                buffer[0] = 200;
                buffer[1] = '\0';
                sendto(socketClient,                buffer,
                        2,                0,
(SOCKADDR*)&addrServer, sizeof(SOCKADDR));
                stage = 1;
                recvSeq = 0;
                waitSeq = 1;
            }
            break;
        case 1://等待接收数据阶段
            seq = (unsigned short)buffer[0];
            //随机法模拟包是否丢失
            b = lossInLossRatio(packetLossRatio);
            if(b){
                printf("The packet with a seq of %d loss\n",seq);
                continue;
            }
            printf("recv a packet with a seq of %d\n",seq);
            //如果是期待的包, 正确接收, 正常确认即可
            if(!(waitSeq - seq)){
                ++waitSeq;
                if(waitSeq == 21){

```

```

        waitSeq = 1;
    }
    //输出数据
    //printf("%s\n",&buffer[1]);
    buffer[0] = seq;
    recvSeq = seq;
    buffer[1] = '\0';
} else {
    //如果当前一个包都没有收到, 则等待 Seq 为 1 的数据包, 不是则不返回ACK (因为并没有上一个正确的ACK)
    if(!recvSeq){
        continue;
    }
    buffer[0] = recvSeq;
    buffer[1] = '\0';
}
b = lossInLossRatio(ackLossRatio);
if(b){
    printf("The    ack    of    %d\n", (unsigned char)buffer[0]);
    continue;
}
    sendto(socketClient,    buffer,
2,    0, (SOCKADDR*)&addrServer, sizeof(SOCKADDR));
    printf("send a ack of %d\n", (unsigned char)buffer[0]);
    break;
}
    Sleep(500);
}
}
    sendto(socketClient,    strlen(buffer)+1,    0,
(SOCKADDR*)&addrServer, sizeof(SOCKADDR));
    ret
recvfrom(socketClient, buffer, BUFFER_LENGTH, 0, (SOCKADDR*)&addrServer,
&len);

```

```
        printf("%s\n",buffer);
        if(!strcmp(buffer,"Goodbye!")){
            break;
        }
        printTips();
    }
    //关闭套接字
    closesocket(socketClient);
    WSACleanup();
    return 0;
}
```

7.实验报告

在实验报告中要说明所设计GBN 协议数据分组格式、确认分组格式、各个域作用，协议两端程序流程图，协议典型交互过程，数据分组丢失验证模拟方法，程序实现的主要类（或函数）及其主要作用、实验验证结果，详细注释源程序等。

