

**计算机网络**

**课程实验报告**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 实验名称 | 可靠数据传输-停等的设计与实现 | | | | | |
| 姓名 | 常添 | | 院系 | 计算学部 | | |
| 班级 | 2203102 | | 学号 | 2022111259 | | |
| 任课教师 | 聂兰顺 | | 指导教师 | 聂兰顺 | | |
| 实验地点 | G001 | | 实验时间 | 2024.10.17 | | |
| 实验课表现 | 出勤、表现得分(10) |  | 实验报告  得分(40) |  | 实验总分 |  |
| 操作结果得分(50) |  |
| 教师评语 | | | | | | |
|  | | | | | | |

****

|  |
| --- |
| 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。 |
| 实验内容： |
| 必做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）。  必做2：模拟引入数据包的丢失，验证所设计协议的有效性。展示代码  选做1：改进所设计的 GBN 协议，支持双向数据传输；  选做2：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  选做3：将所设计的 GBN 协议改进为 SR 协议。展示SR和GBN代码 |
| 实验过程： |
| 本次实验的源代码如下：  GBN协议客户端代码：  lab2\_gbn\_client.cpp  #include <stdlib.h>  #include <WinSock2.h>  #include <time.h>  #include <stdio.h>  #include <fstream>  #include <sstream>  #include <cstdio>  #pragma comment(lib,"ws2\_32.lib")  #pragma warning(disable:4996)  #define SERVER\_PORT 12340  #define SERVER\_IP "127.0.0.1"  using namespace std;  const int BUFFER\_LENGTH = 1027;  const int SEQ\_SIZE = 20;  BOOL ack[SEQ\_SIZE];  int curSeq;  int curAck;  int totalSeq;  int totalPacket;  int waitSeq;  const int SEND\_WIND\_SIZE = 10;  void printTips() {  printf("| gbn + [rate1] + [rate2] + operand + filename |\n");  }  BOOL lossInLossRatio(float lossRatio) {  int lossBound = (int)(lossRatio \* 100);  int r = rand() % 100;  return r < lossBound ? TRUE : FALSE;  }  bool seqIsAvailable() {  int step;  step = curSeq - curAck;  step = step >= 0 ? step : step + SEQ\_SIZE;  if (step >= SEND\_WIND\_SIZE) {  return false;  }  return ack[curSeq] ? true : false;  }  void timeoutHandler() {  printf("Timer out error.\n");  int index;  for (int i = 0; i < (curSeq - curAck + SEQ\_SIZE) % SEQ\_SIZE; ++i) {  index = (i + curAck) % SEQ\_SIZE;  ack[index] = TRUE;  }  totalSeq -= ((curSeq - curAck + SEQ\_SIZE) % SEQ\_SIZE);  curSeq = curAck;  }  void ackHandler(char c) {  unsigned char index = (unsigned char)c - 1;  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 {  for (int i = curAck; i < SEQ\_SIZE; ++i) {  ack[i] = TRUE;  }  for (int i = 0; i <= index; ++i) {  ack[i] = TRUE;  }  curAck = index + 1;  }  }  int main() {  WORD wVersionRequested;  WSADATA wsaData;  int err;  wVersionRequested = MAKEWORD(2, 2);  err = WSAStartup(wVersionRequested, &wsaData);  if (err != 0) {  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);  char buffer[BUFFER\_LENGTH];  ZeroMemory(buffer, sizeof(buffer));  int len = sizeof(SOCKADDR);  int ret;  int interval = 1;  char cmd[128];  float packetLossRatio = 0.2;  float ackLossRatio = 0.2;  char operation[10];  char filename[100];  int sendack = 0;  int iMode = 0;  int loct = 0;  int waitCount = 0;  srand((unsigned)time(NULL));  while (true) {  printTips();  gets\_s(buffer);  ret = sscanf(buffer, "%s %f %f %s %s", &cmd, &packetLossRatio, &ackLossRatio, &operation, &filename);  if (!strcmp(cmd, "gbn")) {  printf("%s\n", "Begin 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);  int stage = 0;  BOOL b;  unsigned char u\_code;  unsigned short seq;  unsigned short recvSeq;  unsigned short waitSeq;  unsigned short recvPacket;  sendto(socketClient, buffer, BUFFER\_LENGTH, 0, (SOCKADDR\*)&addrServer, sizeof(SOCKADDR));  if (!strcmp(operation, "download")) {  char data[1024 \* 113];  loct = 0;  iMode = 0;  int flg = 1;  ioctlsocket(socketClient, FIONBIO, (u\_long FAR\*) & iMode);  while (true) {  recvfrom(socketClient, buffer, BUFFER\_LENGTH, 0, (SOCKADDR\*)&addrServer, &len);  switch (stage) {  case 0:  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;  loct = 0;  }  break;  case 1:  seq = (unsigned short)buffer[0];  b = lossInLossRatio(packetLossRatio);  if (b) {  printf("The packet with a seq of %d loss\n", seq - 1);  continue;  }  printf("recv a packet with a seq of %d\n", seq - 1);  if (!(waitSeq - seq)) {  memcpy(data + 1024 \* loct, buffer + 2, 1024);  if (buffer[1] == '0') flg = 0;  ++loct;  ++waitSeq;  if (waitSeq == 21) {  waitSeq = 1;  }  buffer[0] = seq;  recvSeq = seq;  recvPacket = (unsigned short)buffer[1];  buffer[2] = '\0';  }  else {  if (!recvSeq) {  continue;  }  buffer[0] = recvSeq;  buffer[1] = recvPacket;  buffer[2] = '\0';  }  b = lossInLossRatio(ackLossRatio);  if (b) {  printf("The ack of %d loss\n", (unsigned char)buffer[0] - 1);  continue;  }  sendto(socketClient, buffer, 3, 0, (SOCKADDR\*)&addrServer, sizeof(SOCKADDR));  printf("send a ack of %d\n", (unsigned char)buffer[0] - 1);  break;  }  if (flg == 0) {  printf("File transfer complete\n");  break;  }  Sleep(20);  }  ofstream ofs;  ofs.open(filename, ios::out);  char buff[1300];  printf("%d", loct);  for (int i = 0; i < loct; ++i) {  memcpy(buff, data + 1024 \* i, 1024);  ofs << buff << endl;  }  ofs.close();  if (flg == 0) {  ZeroMemory(buffer, sizeof(buffer));  continue;  }  }  else if (!strcmp(operation, "upload")) {  std::ifstream fin;  fin.open(filename, ios\_base::in);  if (!fin.is\_open()) {  printf("Unable to open file");  continue;  }  iMode = 1;  ioctlsocket(socketClient, FIONBIO, (u\_long FAR\*) & iMode);  char buff[1024] = { 0 };  char data[1024 \* 113];  loct = 0;  int flg = 1;  while (fin.getline(buff, sizeof(buff))) {  if (buff[0] == '0') break;  memcpy(data + 1024 \* loct, buff, 1024);  ++loct;  }  fin.close();  totalPacket = loct;  ZeroMemory(buffer, sizeof(buffer));  int waitCount = 0;  printf("Begin to test GBN protocol, please don't abort the process\n");  printf("Shake hands stage\n");  stage = 0;  bool runFlag = true;  int recvSize;  while (runFlag) {  switch (stage) {  case 0:  buffer[0] = 205;  sendto(socketClient, buffer, strlen(buffer) + 1, 0, (SOCKADDR\*)&addrServer, sizeof(SOCKADDR));  Sleep(100);  stage = 1;  break;  case 1:  recvSize = recvfrom(socketClient, buffer, BUFFER\_LENGTH, 0, ((SOCKADDR\*)&addrServer), &len);  if (recvSize < 0) {  ++waitCount;  if (waitCount > 20) {  runFlag = false;  printf("Timeout error\n");  break;  }  Sleep(20);  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", totalPacket \* 1024, totalPacket);  curSeq = 0;  curAck = 0;  totalSeq = 0;  waitCount = 0;  stage = 2;  for (int i = 0; i < SEQ\_SIZE; ++i) {  ack[i] = TRUE;  }  }  }  break;  case 2:  if (seqIsAvailable() && totalSeq < loct) {  buffer[0] = curSeq + 1;  if (totalSeq == loct - 1) buffer[1] = '0';  else buffer[1] = '1';  ack[curSeq] = FALSE;  memcpy(&buffer[2], data + 1024 \* totalSeq, 1024);  printf("send a packet with a seq of %d\n", curSeq);  sendto(socketClient, buffer, BUFFER\_LENGTH, 0, (SOCKADDR\*)&addrServer, sizeof(SOCKADDR));  ++curSeq;  curSeq %= SEQ\_SIZE;  ++totalSeq;  Sleep(20);  }  recvSize = recvfrom(socketClient, buffer, BUFFER\_LENGTH, 0, ((SOCKADDR\*)&addrServer), &len);  if (recvSize < 0) {  waitCount++;  if (waitCount > 20) {  timeoutHandler();  waitCount = 0;  }  }  else {  if (buffer[1] == '0') {  flg = 0;  break;  }  ackHandler(buffer[0]);  waitCount = 0;  }  Sleep(20);  break;  }  if (flg == 0) break;  }  if (flg == 0) {  printf("File transfer complete\n");  ZeroMemory(buffer, sizeof(buffer));  continue;  }  }  }  sendto(socketClient, buffer, 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, "Good bye!")) {  break;  }  }  closesocket(socketClient);  WSACleanup();  return 0;  }  GBN协议服务器端代码：  lab2\_gbn\_server.cpp  #include <stdlib.h>  #include <time.h>  #include <WinSock2.h>  #include <windows.h>  #include <fstream>  #include <sstream>  #include <cstdio>  #pragma comment(lib,"ws2\_32.lib")  #pragma warning(disable:4996)  #define SERVER\_PORT 12340  #define SERVER\_IP "0.0.0.0"  using namespace std;  const int BUFFER\_LENGTH = 1027;  const int SEND\_WIND\_SIZE = 10;  const int SEQ\_SIZE = 20;  BOOL ack[SEQ\_SIZE];  int curSeq;  int curAck;  int totalSeq;  int totalPacket;  int waitSeq;  void getCurTime(char\* ptime) {  char buffer[128];  memset(buffer, 0, sizeof(buffer));  SYSTEMTIME sys;  GetLocalTime(&sys);  sprintf\_s(buffer, "%4d/%02d/%02d %02d:%02d:%02d",  sys.wYear,  sys.wMonth,  sys.wDay,  sys.wHour,  sys.wMinute,  sys.wSecond);  strcpy\_s(ptime, sizeof(buffer), buffer);  }  bool seqIsAvailable() {  int step;  step = curSeq - curAck;  step = step >= 0 ? step : step + SEQ\_SIZE;  if (step >= SEND\_WIND\_SIZE) {  return false;  }  if (ack[curSeq]) {  return true;  }  return false;  }  void timeoutHandler() {  printf("Timer out error.\n");  int index;  for (int i = 0; i < (curSeq - curAck + SEQ\_SIZE) % SEQ\_SIZE; ++i) {  index = (i + curAck) % SEQ\_SIZE;  ack[index] = TRUE;  }  totalSeq = totalSeq - ((curSeq - curAck + SEQ\_SIZE) % SEQ\_SIZE);  curSeq = curAck;  }  void ackHandler(char c) {  unsigned char index = (unsigned char)c - 1;  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 {  for (int i = curAck; i < SEQ\_SIZE; ++i) {  ack[i] = TRUE;  }  for (int i = 0; i <= index; ++i) {  ack[i] = TRUE;  }  curAck = index + 1;  }  }  BOOL lossInLossRatio(float lossRatio) {  int lossBound = (int)(lossRatio \* 100);  int r = rand() % 100;  if (r < lossBound) {  return TRUE;  }  return FALSE;  }  int main()  {  WORD wVersionRequested;  WSADATA wsaData;  int err;  wVersionRequested = MAKEWORD(2, 2);  err = WSAStartup(wVersionRequested, &wsaData);  if (err != 0) {  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);  int iMode;  SOCKADDR\_IN addrServer;  addrServer.sin\_addr.S\_un.S\_addr = htonl(INADDR\_ANY);  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;  int length = sizeof(SOCKADDR);  char buffer[BUFFER\_LENGTH];  ZeroMemory(buffer, sizeof(buffer));  int recvSize;  int loct = 0;  int waitCount = 0;  float packetLossRatio = 0.2;  float ackLossRatio = 0.2;  srand((unsigned)time(NULL));  while (true) {  recvSize = recvfrom(sockServer, buffer, BUFFER\_LENGTH, 0, ((SOCKADDR\*)&addrClient), &length);  printf("recv from client: %s\n", buffer);  if (strcmp(buffer, "-time") == 0) {  getCurTime(buffer);  }  else if (strcmp(buffer, "-quit") == 0) {  strcpy\_s(buffer, strlen("Good bye!") + 1, "Good bye!");  }  else {  char filename[100];  char operation[10];  char cmd[10];  int ret;  unsigned char u\_code;  unsigned short seq;  unsigned short recvSeq;  unsigned short waitSeq;  unsigned short recvPacket;  int sendack = 0;  int stage = 0;  ret = sscanf(buffer, "%s %f %f %s %s", &cmd, &packetLossRatio, &ackLossRatio, &operation, &filename);  if (!strcmp(cmd, "gbn")) {  if (!strcmp(operation, "download")) {  iMode = 1;  int flg = 1;  ioctlsocket(sockServer, FIONBIO, (u\_long FAR\*) & iMode);  std::ifstream fin;  fin.open(filename, ios\_base::in);  if (!fin.is\_open()) {  printf("Unable to open file");  iMode = 0;  ioctlsocket(sockServer, FIONBIO, (u\_long FAR\*) & iMode);  continue;  }  char buff[1024] = { 0 };  char data[1024 \* 113];  loct = 0;  while (fin.getline(buff, sizeof(buff))) {  if (buff[0] == '0') break;  memcpy(data + 1024 \* loct, buff, 1024);  ++loct;  }  fin.close();  totalPacket = loct;  ZeroMemory(buffer, sizeof(buffer));  int recvSize;  waitCount = 0;  printf("Begain to test GBN protocol,please don't abort the process\n");  int stage = 0;  bool runFlag = true;  while (runFlag) {  switch (stage) {  case 0:  buffer[0] = 205;  sendto(sockServer, buffer, strlen(buffer) + 1, 0, (SOCKADDR\*)&addrClient, sizeof(SOCKADDR));  Sleep(100);  stage = 1;  break;  case 1:  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", totalPacket \* 1024, totalPacket);  curSeq = 0;  curAck = 0;  totalSeq = 0;  waitCount = 0;  stage = 2;  for (int i = 0; i < SEQ\_SIZE; ++i) {  ack[i] = TRUE;  }  }  }  break;  case 2:  if (seqIsAvailable() && totalSeq < loct) {  buffer[0] = curSeq + 1;  if (totalSeq == loct - 1) buffer[1] = '0';  else buffer[1] = '1';  ack[curSeq] = FALSE;  memcpy(&buffer[2], data + 1024 \* totalSeq, 1024);  printf("send a packet with a seq of %d\n", curSeq);  sendto(sockServer, buffer, BUFFER\_LENGTH, 0, (SOCKADDR\*)&addrClient, sizeof(SOCKADDR));  ++curSeq;  curSeq %= SEQ\_SIZE;  ++totalSeq;  Sleep(500);  }  recvSize = recvfrom(sockServer, buffer, BUFFER\_LENGTH, 0, ((SOCKADDR\*)&addrClient), &length);  if (recvSize < 0) {  waitCount++;  if (waitCount > 20) {  timeoutHandler();  waitCount = 0;  }  }  else {  if (buffer[1] == '0')  {  flg = 0;  break;  }  ackHandler(buffer[0]);  waitCount = 0;  }  Sleep(500);  break;  }  if (flg == 0) break;  }  if (flg == 0) {  printf("File transfer complete\n");  iMode = 0;  ioctlsocket(sockServer, FIONBIO, (u\_long FAR\*) & iMode);  ZeroMemory(buffer, sizeof(buffer));  continue;  }  }  else if (!strcmp(operation, "upload")) {  char data[1024 \* 113];  loct = 0;  int flg = 1;  BOOL b;  while (true) {  recvfrom(sockServer, buffer, BUFFER\_LENGTH, 0, (SOCKADDR\*)&addrClient, &length);  switch (stage) {  case 0:  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(sockServer, buffer, 2, 0, (SOCKADDR\*)&addrClient, sizeof(SOCKADDR));  stage = 1;  recvSeq = 0;  waitSeq = 1;  loct = 0;  }  break;  case 1:  seq = (unsigned short)buffer[0];  b = lossInLossRatio(packetLossRatio);  if (b) {  printf("The packet with a seq of %d loss\n", seq - 1);  continue;  }  printf("recv a packet with a seq of %d\n", seq - 1);  if (!(waitSeq - seq)) {  if (buffer[1] == '0') flg = 0;  memcpy(data + 1024 \* loct, buffer + 2, 1024);  ++loct;  ++waitSeq;  if (waitSeq == 21) {  waitSeq = 1;  }  buffer[0] = seq;  recvSeq = seq;  recvPacket = (unsigned short)buffer[1];  buffer[2] = '\0';  }  else {  if (!recvSeq) {  continue;  }  buffer[0] = recvSeq;  buffer[1] = recvPacket;  buffer[2] = '\0';  }  b = lossInLossRatio(ackLossRatio);  if (b) {  printf("The ack of %d loss\n", (unsigned char)buffer[0] - 1);  continue;  }  sendto(sockServer, buffer, 3, 0, (SOCKADDR\*)&addrClient, sizeof(SOCKADDR));  printf("send a ack of %d\n", (unsigned char)buffer[0] - 1);  break;  }  if (flg == 0) {  printf("File transfer complete\n");  break;  }  Sleep(500);  }  ofstream ofs;  ofs.open(filename, ios::out);  char buff[1300];  printf("%d", loct);  for (int i = 0; i < loct; ++i) {  memcpy(buff, data + 1024 \* i, 1024);  ofs << buff << endl;  }  ofs.close();  if (flg == 0) {  ZeroMemory(buffer, sizeof(buffer));  continue;  }  }  }  }  sendto(sockServer, buffer, strlen(buffer) + 1, 0, (SOCKADDR\*)&addrClient, sizeof(SOCKADDR));  Sleep(500);  }  closesocket(sockServer);  WSACleanup();  return 0;  }  SR协议客户端代码：  lab2\_client.cpp  #include <stdlib.h>  #include <time.h>  #include <WinSock2.h>  #include <WS2tcpip.h>  #include <fstream>  #pragma comment(lib,"ws2\_32.lib")  #define SERVER\_PORT 8080 // 接收数据的端口号  #define SERVER\_IP "127.0.0.1" // 服务器的 IP 地址  #define BUFFER\_SIZE 1024 // 缓冲区大小  #define SEQ\_SIZE 16 // 序列号个数  #define SWIN\_SIZE 8 // 发送窗口大小  #define RWIN\_SIZE 8 // 接收窗口大小  #define LOSS\_RATE 0.1 // 丢包率  using namespace std;  char cmdBuffer[50];  char buffer[BUFFER\_SIZE];  char cmd[10];  char fileName[40];  char filePath[50];  char file[1024 \* 1024];  int len = sizeof(SOCKADDR);  int recvSize;  int Deliver(char\* file, int ack);  int Send(ifstream& infile, int seq, SOCKET socket, SOCKADDR\* addr);  int MoveSendWindow(int seq);  int Read(ifstream& infile, char\* buffer);  struct Cache {  bool used;  char buffer[BUFFER\_SIZE];  Cache() {  used = false;  ZeroMemory(buffer, sizeof(buffer));  }  }recvWindow[SEQ\_SIZE];  struct DataFrame {  clock\_t start;  char buffer[BUFFER\_SIZE];  DataFrame() {  start = 0;  ZeroMemory(buffer, sizeof(buffer));  }  }sendWindow[SEQ\_SIZE];  int main(int argc, char\* argv[]) {  // 加载套接字库  WORD wVersionRequested;  WSADATA wsaData;  // 版本 2.2  wVersionRequested = MAKEWORD(2, 2);  int err = WSAStartup(wVersionRequested, &wsaData);  if (err != 0) {  printf("Winsock.dll 加载失败，错误码: %d\n", err);  return -1;  }  if (LOBYTE(wsaData.wVersion) != LOBYTE(wVersionRequested) || HIBYTE(wsaData.wVersion) != HIBYTE(wVersionRequested)) {  printf("找不到 %d.%d 版本的 Winsock.dll\n", LOBYTE(wVersionRequested), HIBYTE(wVersionRequested));  WSACleanup();  return -1;  }  else {  printf("Winsock %d.%d 加载成功\n", LOBYTE(wVersionRequested), HIBYTE(wVersionRequested));  printf("客户端启动成功\n");  printf("上传文件： upload 文件名\n");  printf("下载文件： download 文件名\n\n");  }  // 创建客户端套接字  SOCKET socketClient = socket(AF\_INET, SOCK\_DGRAM, 0);  // 设置为非阻塞模式  int iMode = 1;  ioctlsocket(socketClient, FIONBIO, (u\_long FAR\*) & iMode);  SOCKADDR\_IN addrServer;  inet\_pton(AF\_INET, SERVER\_IP, &addrServer.sin\_addr);  addrServer.sin\_family = AF\_INET;  addrServer.sin\_port = htons(SERVER\_PORT);  srand((unsigned)time(NULL));  int status = 0;  clock\_t start;  clock\_t now;  int seq;  int ack;  while (true) {  gets\_s(cmdBuffer, 50);  sscanf\_s(cmdBuffer, "%s%s", cmd, sizeof(cmd) - 1, fileName, sizeof(fileName) - 1);  if (!strcmp(cmd, "upload")) {  printf("上传文件: %s\n", fileName);  strcpy\_s(filePath, "./");  strcat\_s(filePath, fileName);  ifstream infile(filePath);  start = clock();  seq = 0;  status = 0;  sendWindow[0].buffer[0] = 10;  strcpy\_s(sendWindow[0].buffer + 1, strlen(cmdBuffer) + 1, cmdBuffer);  sendWindow[0].start = start - 1000L;  while (true) {  recvSize = recvfrom(socketClient, buffer, BUFFER\_SIZE, 0, (SOCKADDR\*)&addrServer, &len);  switch (status)  {  case 0:  if (recvSize > 0 && buffer[0] == 100) {  if (!strcmp(buffer + 1, "OK")) {  start = clock();  status = 1;  sendWindow[0].start = 0L;  continue;  }  else if (!strcmp(buffer + 1, "NO")) {  status = -1;  break;  }  }  now = clock();  if (now - sendWindow[0].start >= 1000L) {  sendWindow[0].start = now;  sendto(socketClient, sendWindow[0].buffer, strlen(sendWindow[0].buffer) + 1, 0, (SOCKADDR\*)&addrServer, sizeof(SOCKADDR));  }  break;  case 1:  if (recvSize > 0 && buffer[0] == 101) {  start = clock();  ack = buffer[1];  ack--;  sendWindow[ack].start = -1L;  if (ack == seq) {  seq = MoveSendWindow(seq);  }  printf("接收 ack = %d, 当前起始 seq = %d\n", ack + 1, seq + 1);  }  if (!Send(infile, seq, socketClient, (SOCKADDR\*)&addrServer)) {  printf("上传完毕...\n");  status = 2;  start = clock();  sendWindow[0].buffer[0] = 10;  strcpy\_s(sendWindow[0].buffer + 1, 7, "Finish");  sendWindow[0].start = start - 1000L;  continue;  }  break;  case 2:  if (recvSize > 0 && buffer[0] == 100) {  if (!strcmp(buffer + 1, "OK")) {  buffer[0] = 10;  strcpy\_s(buffer + 1, 3, "OK");  sendto(socketClient, buffer, strlen(buffer) + 1, 0, (SOCKADDR\*)&addrServer, sizeof(SOCKADDR));  status = 3;  break;  }  }  now = clock();  if (now - sendWindow[0].start >= 1000L) {  sendWindow[0].start = now;  sendto(socketClient, sendWindow[0].buffer, strlen(sendWindow[0].buffer) + 1, 0, (SOCKADDR\*)&addrServer, sizeof(SOCKADDR));  }  default:  break;  }  if (status == -1) {  printf("服务器拒绝请求\n");  infile.close();  break;  }  if (status == 3) {  printf("上传成功，结束通信\n");  infile.close();  break;  }  if (clock() - start >= 5000L) {  printf("通信超时，结束通信\n");  infile.close();  break;  }  if (recvSize <= 0) {  Sleep(200);  }  }  }  else if (!strcmp(cmd, "download")) {  printf("申请下载文件 %s\n", fileName);  strcpy\_s(filePath, "./");  strcat\_s(filePath, fileName);  ofstream outfile(filePath);  start = clock();  ack = 0;  status = 0;  sendWindow[0].buffer[0] = 10;  strcpy\_s(sendWindow[0].buffer + 1, strlen(cmdBuffer) + 1, cmdBuffer);  sendWindow[0].start = start - 1000L;  while (true) {  recvSize = recvfrom(socketClient, buffer, BUFFER\_SIZE, 0, (SOCKADDR\*)&addrServer, &len);  if ((float)rand() / RAND\_MAX < LOSS\_RATE) {  recvSize = 0;  buffer[0] = 0;  }  switch (status)  {  case 0:  if (recvSize > 0 && buffer[0] == 100) {  if (!strcmp(buffer + 1, "OK")) {  printf("申请通过, 准备下载...\n");  start = clock();  status = 1;  sendWindow[0].buffer[0] = 10;  strcpy\_s(sendWindow[0].buffer + 1, 3, "OK");  sendWindow[0].start = start - 1000L;  continue;  }  else if (!strcmp(buffer + 1, "NO")) {  status = -1;  break;  }  }  now = clock();  if (now - sendWindow[0].start >= 1000L) {  sendWindow[0].start = now;  sendto(socketClient, sendWindow[0].buffer, strlen(sendWindow[0].buffer) + 1, 0, (SOCKADDR\*)&addrServer, sizeof(SOCKADDR));  }  break;  case 1:  if (recvSize > 0 && (unsigned char)buffer[0] == 200) {  printf("开始下载...\n");  start = clock();  seq = buffer[1];  printf("接收数据帧 seq = %d, data = %s, 发送ack = %d\n", seq, buffer + 2, seq);  seq--;  recvWindow[seq].used = true;  strcpy\_s(recvWindow[seq].buffer, strlen(buffer + 2) + 1, buffer + 2);  if (ack == seq) {  ack = Deliver(file, ack);  }  status = 2;  buffer[0] = 11;  buffer[1] = seq + 1;  buffer[2] = 0;  sendto(socketClient, buffer, strlen(buffer) + 1, 0, (SOCKADDR\*)&addrServer, sizeof(SOCKADDR));  continue;  }  now = clock();  if (now - sendWindow[0].start >= 1000L) {  sendWindow[0].start = now;  sendto(socketClient, sendWindow[0].buffer, strlen(sendWindow[0].buffer) + 1, 0, (SOCKADDR\*)&addrServer, sizeof(SOCKADDR));  }  break;  case 2:  if (recvSize > 0) {  if ((unsigned char)buffer[0] == 200) {  seq = buffer[1];  int temp = seq - 1 - ack;  if (temp < 0) {  temp += SEQ\_SIZE;  }  start = clock();  seq--;  if (temp < RWIN\_SIZE) {  if (!recvWindow[seq].used) {  recvWindow[seq].used = true;  strcpy\_s(recvWindow[seq].buffer, strlen(buffer + 2) + 1, buffer + 2);  }  if (ack == seq) {  ack = Deliver(file, ack);  }  }  printf("接收数据帧 seq = %d, data = %s, 发送 ack = %d, 起始 ack = %d\n", seq + 1, buffer + 2, seq + 1, ack + 1);  buffer[0] = 11;  buffer[1] = seq + 1;  buffer[2] = 0;  sendto(socketClient, buffer, strlen(buffer) + 1, 0, (SOCKADDR\*)&addrServer, sizeof(SOCKADDR));  }  else if (buffer[0] == 100 && !strcmp(buffer + 1, "Finish")) {  status = 3;  outfile.write(file, strlen(file));  buffer[0] = 10;  strcpy\_s(buffer + 1, 3, "OK");  sendto(socketClient, buffer, strlen(buffer) + 1, 0, (SOCKADDR\*)&addrServer, sizeof(SOCKADDR));  continue;  }  }  break;  default:  break;  }  if (status == -1) {  printf("服务器拒绝请求\n");  outfile.close();  break;  }  if (status == 3) {  printf("下载成功, 结束通信\n");  outfile.close();  break;  }  if (clock() - start >= 5000L) {  printf("通信超时, 结束通信\n");  outfile.close();  break;  }  if (recvSize <= 0) {  Sleep(20);  }  }  }  else if (!strcmp(cmd, "quit")) {  break;  }  }  closesocket(socketClient);  printf("关闭套接字\n");  WSACleanup();  return 0;  }  int Read(ifstream& infile, char\* buffer) {  if (infile.eof()) {  return 0;  }  infile.read(buffer, 3);  int cnt = infile.gcount();  buffer[cnt] = 0;  return cnt;  }  int Deliver(char\* file, int ack) {  while (recvWindow[ack].used) {  recvWindow[ack].used = false;  strcat\_s(file, strlen(file) + strlen(recvWindow[ack].buffer) + 1, recvWindow[ack].buffer);  ack++;  ack %= SEQ\_SIZE;  }  return ack;  }  int Send(ifstream& infile, int seq, SOCKET socket, SOCKADDR\* addr) {  clock\_t now = clock();  for (int i = 0; i < SWIN\_SIZE; i++) {  int j = (seq + i) % SEQ\_SIZE;  if (sendWindow[j].start == -1L) {  continue;  }  if (sendWindow[j].start == 0L) {  if (Read(infile, sendWindow[j].buffer + 2)) {  sendWindow[j].start = now;  sendWindow[j].buffer[0] = 20;  sendWindow[j].buffer[1] = j + 1;  }  else if (i == 0) {  return 0;  }  else {  break;  }  }  else if (now - sendWindow[j].start >= 1000L) {  sendWindow[j].start = now;  }  else {  continue;  }  printf("发送数据帧 seq = %d, data = %s\n", j + 1, sendWindow[j].buffer + 2);  sendto(socket, sendWindow[j].buffer, strlen(sendWindow[j].buffer) + 1, 0, addr, sizeof(SOCKADDR));  }  return 1;  }  //移动发送的滑动窗口  int MoveSendWindow(int seq) {  while (sendWindow[seq].start == -1L) {  sendWindow[seq].start = 0L;  seq++;  seq %= SEQ\_SIZE;  }  return seq;  }  SR协议服务器端代码：  lab2\_server.cpp  #include <stdlib.h>  #include <time.h>  #include <WinSock2.h>  #include <WS2tcpip.h>  #include <fstream>  #pragma comment(lib,"ws2\_32.lib")  #define SERVER\_PORT 8080 // 端口号  #define SERVER\_IP "0.0.0.0" // IP 地址  #define SEQ\_SIZE 16 // 序列号个数  #define SWIN\_SIZE 8 // 发送窗口大小  #define RWIN\_SIZE 8 // 接收窗口大小  #define BUFFER\_SIZE 1024 // 缓冲区大小  #define LOSS\_RATE 0.1 //丢包率  using namespace std;  struct recv {  bool used;  char buffer[BUFFER\_SIZE];  recv() {  used = false;  ZeroMemory(buffer, sizeof(buffer));  }  }recvWindow[SEQ\_SIZE];  struct send {  clock\_t start;//由于使用的是SR，因此每一个窗口位置都需要设置一个计时器  char buffer[BUFFER\_SIZE];  send() {  start = 0;  ZeroMemory(buffer, sizeof(buffer));  }  }sendWindow[SEQ\_SIZE];  char cmdBuffer[50];  char buffer[BUFFER\_SIZE];  char cmd[10];  char fileName[40];  char filePath[50];  char file[1024 \* 1024];  int len = sizeof(SOCKADDR);  int recvSize;  int Deliver(char\* file, int ack);  int Send(ifstream& infile, int seq, SOCKET socket, SOCKADDR\* addr);  int MoveSendWindow(int seq);  int Read(ifstream& infile, char\* buffer);  //主函数  int main(int argc, char\* argv[]) {  // 加载套接字库  WORD wVersionRequested;  WSADATA wsaData;  // 版本 2.2  wVersionRequested = MAKEWORD(2, 2);  int err = WSAStartup(wVersionRequested, &wsaData);  if (err != 0) {  printf("Winsock.dll 加载失败，错误码: %d\n", err);  return -1;  }  if (LOBYTE(wsaData.wVersion) != LOBYTE(wVersionRequested) || HIBYTE(wsaData.wVersion) != HIBYTE(wVersionRequested)) {  printf("找不到 %d.%d 版本的 Winsock.dll\n", LOBYTE(wVersionRequested), HIBYTE(wVersionRequested));  WSACleanup();  return -1;  }  else {  printf("Winsock %d.%d 加载成功\n", LOBYTE(wVersionRequested), HIBYTE(wVersionRequested));  printf("服务端启动成功\n\n");  }  // 创建服务器套接字  SOCKET socketServer = socket(AF\_INET, SOCK\_DGRAM, IPPROTO\_UDP);  // 设置为非阻塞模式  int iMode = 1;  ioctlsocket(socketServer, FIONBIO, (u\_long FAR\*) & iMode);  SOCKADDR\_IN addrServer;  inet\_pton(AF\_INET, SERVER\_IP, &addrServer.sin\_addr);  addrServer.sin\_family = AF\_INET;  addrServer.sin\_port = htons(SERVER\_PORT);  // 绑定端口  if (err = bind(socketServer, (SOCKADDR\*)&addrServer, sizeof(SOCKADDR))) {  err = GetLastError();  printf("绑定端口 %d 失败，错误码: % d\n", SERVER\_PORT, err);  WSACleanup();  return -1;  }  else {  printf("绑定端口 %d 成功", SERVER\_PORT);  }  SOCKADDR\_IN addrClient;  int status = 0;  clock\_t start;  clock\_t now;  int seq;  int ack;  ofstream outfile;  ifstream infile;  cmdBuffer[0] = 'Y';  cmdBuffer[1] = '\0';  //进入接收状态，注意服务器主要处理的任务是接收客户机请求，共有上载和下载两种任务  while (true) {  recvSize = recvfrom(socketServer, buffer, BUFFER\_SIZE, 0, ((SOCKADDR\*)&addrClient), &len);  if ((float)rand() / RAND\_MAX < LOSS\_RATE) {  recvSize = 0;  buffer[0] = 0;  }  switch (status)  {  case 0://接收请求  if (recvSize > 0 && buffer[0] == 10) {  char addr[100];  ZeroMemory(addr, sizeof(addr));  inet\_ntop(AF\_INET, &addrClient.sin\_addr, addr, sizeof(addr));  sscanf\_s(buffer + 1, "%s%s", cmd, sizeof(cmd) - 1, fileName, sizeof(fileName) - 1);  if (strcmp(cmd, "upload") && strcmp(cmd, "download")) {  continue;  }  strcpy\_s(filePath, "./");  strcat\_s(filePath, fileName);  printf("收到来自客户端 %s 的请求: %s\n", addr, buffer);  if (!strcmp(cmdBuffer, "Y")) {  buffer[0] = 100;  strcpy\_s(buffer + 1, 3, "OK");  if (!strcmp(cmd, "upload")) {  file[0] = 0;  start = clock();  ack = 0;  status = 1;  outfile.open(filePath);  }  else if (!strcmp(cmd, "download")) {  start = clock();  seq = 0;  status = -1;  infile.open(filePath);  }  }  else {  buffer[0] = 100;  strcpy\_s(buffer + 1, 3, "NO");  }  sendto(socketServer, buffer, strlen(buffer) + 1, 0, (SOCKADDR\*)&addrClient, sizeof(SOCKADDR));  }  break;  case 1://客户机请求上传，也就是服务器端是接收方  if (recvSize > 0) {  if (buffer[0] == 10) {  if (!strcmp(buffer + 1, "Finish")) {  printf("传输完毕...\n");  start = clock();  sendWindow[0].start = start - 1000L;  sendWindow[0].buffer[0] = 100;  strcpy\_s(sendWindow[0].buffer + 1, 3, "OK");  outfile.write(file, strlen(file));  status = 2;  }  buffer[0] = 100;  strcpy\_s(buffer + 1, 3, "OK");  sendto(socketServer, buffer, strlen(buffer) + 1, 0, (SOCKADDR\*)&addrClient, sizeof(SOCKADDR));  }  else if (buffer[0] == 20) {  seq = buffer[1];  int temp = seq - 1 - ack;  if (temp < 0) {  temp += SEQ\_SIZE;  }  start = clock();  seq--;  if (temp < RWIN\_SIZE) {  if (!recvWindow[seq].used) {  recvWindow[seq].used = true;  strcpy\_s(recvWindow[seq].buffer, strlen(buffer + 2) + 1, buffer + 2);  }  if (ack == seq) {  ack = Deliver(file, ack);  }  }  printf("接收数据帧 seq = %d, data = %s, 发送 ack = %d, 起始 ack = %d\n", seq + 1, buffer + 2, seq + 1, ack + 1);  buffer[0] = 101;  buffer[1] = seq + 1;  buffer[2] = 0;  sendto(socketServer, buffer, strlen(buffer) + 1, 0, (SOCKADDR\*)&addrClient, sizeof(SOCKADDR));  }  }  break;  case 2://接收完成  if (recvSize > 0 && buffer[0] == 10 && !strcmp(buffer + 1, "OK")) {  printf("传输成功，结束通信\n");  status = 0;  outfile.close();  }  now = clock();  if (now - sendWindow[0].start >= 1000L) {  sendWindow[0].start = now;  sendto(socketServer, sendWindow[0].buffer, strlen(sendWindow[0].buffer) + 1, 0, (SOCKADDR\*)&addrClient, sizeof(SOCKADDR));  }  break;  case -1://客户机请求下载，也就是服务器端充当发送方  if (recvSize > 0) {  if (buffer[0] == 10) {  if (!strcmp(buffer + 1, "OK")) {  printf("开始传输...\n");  start = clock();  status = -2;  }  buffer[0] = 100;  strcpy\_s(buffer + 1, 3, "OK");  sendto(socketServer, buffer, strlen(buffer) + 1, 0, (SOCKADDR\*)&addrClient, sizeof(SOCKADDR));  }  }  break;  case -2://服务器端发送数据  if (recvSize > 0 && buffer[0] == 11) {  start = clock();  ack = buffer[1];  ack--;  sendWindow[ack].start = -1L;  if (ack == seq) {  seq = MoveSendWindow(seq);  }  printf("接收 ack = %d, 当前起始 seq = %d\n", ack + 1, seq + 1);  }  if (!Send(infile, seq, socketServer, (SOCKADDR\*)&addrClient)) {  printf("传输完毕...\n");  status = -3;  start = clock();  sendWindow[0].buffer[0] = 100;  strcpy\_s(sendWindow[0].buffer + 1, 7, "Finish");  sendWindow[0].start = start - 1000L;  }  break;  case -3://请求完成  if (recvSize > 0 && buffer[0] == 10) {  if (!strcmp(buffer + 1, "OK")) {  printf("传输成功，结束通信\n");  infile.close();  status = 0;  break;  }  }  now = clock();  if (now - sendWindow[0].start >= 1000L) {  sendWindow[0].start = now;  sendto(socketServer, sendWindow[0].buffer, strlen(sendWindow[0].buffer) + 1, 0, (SOCKADDR\*)&addrClient, sizeof(SOCKADDR));  }  default:  break;  }  if (status != 0 && clock() - start > 20000L) {  printf("通信超时, 结束通信\n");  status = 0;  outfile.close();  continue;  }  if (recvSize <= 0) {  Sleep(10);  }  }  //关闭套接字，卸载库  closesocket(socketServer);  WSACleanup();  return 0;  }  int Read(ifstream& infile, char\* buffer) {  //从文件中读取需要发送的数据  if (infile.eof()) {  return 0;  }  infile.read(buffer, 3);  int cnt = infile.gcount();  buffer[cnt] = 0;  return cnt;  }  int Deliver(char\* file, int ack) {  while (recvWindow[ack].used) {  recvWindow[ack].used = false;  strcat\_s(file, strlen(file) + strlen(recvWindow[ack].buffer) + 1, recvWindow[ack].buffer);  ack++;  ack %= SEQ\_SIZE;  }  return ack;  }  int Send(ifstream& infile, int seq, SOCKET socket, SOCKADDR\* addr) {  //发送数据  clock\_t now = clock();  for (int i = 0; i < SWIN\_SIZE; i++) {  int j = (seq + i) % SEQ\_SIZE;  if (sendWindow[j].start == -1L) {//传输超时，不需要  continue;  }  if (sendWindow[j].start == 0L) {//开始计时  if (Read(infile, sendWindow[j].buffer + 2)) {  sendWindow[j].start = now;  sendWindow[j].buffer[0] = 200;  sendWindow[j].buffer[1] = j + 1;  }  else if (i == 0) {  return 0;  }  else {  break;  }  }  else if (now - sendWindow[j].start >= 1000L) {//更新时间  sendWindow[j].start = now;  }  else {  continue;  }  printf("发送数据帧 seq = %d, data = %s\n", j + 1, sendWindow[j].buffer + 2);  sendto(socket, sendWindow[j].buffer, strlen(sendWindow[j].buffer) + 1, 0, addr, sizeof(SOCKADDR));  }  return 1;  }  int MoveSendWindow(int seq) {  //移动窗口  while (sendWindow[seq].start == -1L) {  sendWindow[seq].start = 0L;  seq++;  seq %= SEQ\_SIZE;  }  return seq;  } |
| 实验结果： |
| GBN协议运行结果：  1.测试客户端下载，设置丢包率为20%，以5个数据包为例：    可以看到服务器发送2和3发生了丢包，超时重传。  下面测试客户端上传，设置丢包率为20%，以5个数据包为例：      服务器端没有收到数据包3，于是发送ack=2，触发客户端重传。  SR协议运行结果：默认丢包率为0.1  客户端上传：客户端到服务端    本例子中seq=1没有及时收到，但是接收方已经缓存，当收到该报文时，ack由1变成8  客户端下载：服务端到客户端。    可以看到本例子中ack=3发生了丢失，存在一定的丢包情况。  传输应用的实现：  可以看到服务端产生了一个客户端传过来的test.txt，客户端也产生了从服务端传的test.txt:      测试成功 |
| 问题讨论： |
| 所设计的GBN协议的数据分组格式、确认分组格式、各个域作用：  数据分组格式：    Seq ： 为 1 个字节， 取值为 0~255， （故序列号最多为 256 个）；  Data ： 小于等于 1024 个字节， 为传输的数据；  0 ： 最后一个字节放入 EOF0， 表示结尾。  确认分组格式：    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。  GBN协议的典型交互过程：    在代码中SR协议中，为了更加清晰和方便，Data设置成了3个字节。  SR协议的典型交互过程：    文字描述如下：  初始化阶段： 在连接建立时，发送方和接收方都会初始化各自的窗口，并设置初始的序列号。双方协商确定的窗口大小将决定在任意时间段内可以未确认发送或接收的数据包数量。  数据包发送： 发送方根据序列号依次从窗口中的第一个数据包开始发送，每个已发送的数据包都会启动一个计时器。发送方同时记录已发送但尚未确认的数据包序列号。  数据包接收： 接收方收到数据包后，检查其序列号是否在窗口的范围内。若序列号有效，接收方会发送确认消息（ACK）给发送方，并更新窗口以期望下一个序列号。如果序列号超出窗口范围或有错误，接收方将丢弃该数据包，并可能发送否定确认（NACK）或静默等待下一个期望的数据包。  处理确认（ACK）： 当发送方接收到ACK时，它会从未确认的数据包列表中移除对应序列号的包。如果窗口中还有未发送的数据包，发送方将继续传输下一个数据包。  处理丢失或损坏的数据包： 若发送方在计时器到期前未收到某个数据包的ACK，发送方将假定该数据包已丢失或损坏，并重新传输，直到收到相应的ACK为止。  流量控制： 接收方通过ACK消息告知发送方其当前窗口大小和下一个期望的序列号，以此控制发送速率。当所有数据成功传输并得到确认后，双方通过控制消息来关闭连接，完成通信。数据丢包模拟验证方法：    将随机生成的随机数比上最大的随机数，小于定义的丢包率，那么就将缓冲区，接受窗口大小设为0。  程序两端流程图：  客户端流程图：      服务端代码流程图：    附上GBN协议的有限状态机描述： |
| 心得体会： |
| 在本次实验中，我深入学习了两种协议的工作原理，并亲身体验了它们在实际数据传输中的应用。这次实验不仅加深了我对计算机网络底层通信机制的理解，也让我对数据传输的可靠性与效率有了更加直观的认识。  首先，GBN（Go-Back-N）协议是一种基于累计确认的滑动窗口协议。在实验中，通过模拟发送方和接收方的交互，我切身体会到GBN协议在数据丢失时的处理方式。当出现数据包丢失时，GBN协议要求发送方重传从丢失包开始的所有后续数据包。尽管这种方式比较简单，但在某些情况下会导致大量不必要的重传，浪费了资源。此外，实验还让我意识到，窗口大小的设置对GBN协议的性能至关重要，它直接影响到数据传输的效率和系统的可靠性。  与GBN协议相比，SR（Selective Repeat）协议采用了选择性重传机制。在实验中，我观察到SR协议在丢包时只重传丢失的数据包，而不会像GBN那样重传所有后续包。此机制有效提高了传输效率，减少了重传的冗余开销。此外，SR协议通过维护发送和接收窗口，确保数据的顺序性和完整性。在实际操作中，我深刻感受到SR协议的灵活性和高效性，特别是在应对网络传输中的各种异常情况时，表现更加优越。  总的来说，这次实验使我对GBN和SR协议在不同场景下的优势和不足有了更清晰的认识，也让我对可靠高效的数据传输有了更深入的理解。 |