

GBN协议实验报告

1120221680

陈思达

1. Requirement Analysis

本项目要求使用 UDP 协议实现基于滑动窗口的 Go-Back-N (GBN) 协议，具备可靠文件传输能力。由于 UDP 是不可靠协议，因此需在应用层自行实现超时重传、确认应答、窗口控制等机制，以实现可靠性。

主要功能需求包括：

- 支持滑动窗口机制，窗口大小可配置；
- 支持可变序列号位数及序列号空间管理；
- 基于定时器实现的超时重传；
- 支持配置文件控制：UDP端口、错误率、丢包率、起始序列号、超时时间等参数；
- 发送方与接收方的结构分离，通过结构体和接口封装；
- 实现日志记录功能，记录数据发送与接收过程；
- 实现校验和功能，检测数据包损坏；
- 设计良好的系统初始化逻辑和清理机制。

项目重点：可靠传输的协议逻辑（序列号、超时、确认、窗口滑动）、配置参数解耦设计、调试和错误模拟。

2. Design

2.1 数据帧结构（PDU）

位于 `proto.h` 头文件中。包含 `seqNo`、`length`、`data`、`checksum`、`totalPackets` 字段和若干内置函数，介绍如下：

```
// 定义 PDU（协议数据单元）结构体
#pragma pack(1) // 结构体紧凑对齐
struct PDU
{
    int32_t totalPackets; // 包总数
    uint32_t seqNo;       // 序号
    uint16_t length;      // 数据部分的长度（不超过4096字节）
    char* data;           // 数据部分的内容
    uint16_t checksum;    // CRC 校验
```

2.2 核心机制

- 校验和：发送时使用伪校验算法生成 `checksum`；接收端验证是否匹配；

- 滑动窗口：发送端维护 **base** 与 **nextSeqNum** 指针，窗口大小为配置文件指定；
- 序列号空间：序列号取值范围根据配置的位数决定（如3位支持0~7）；
- **ACK** 和重传机制：
 - 接收方收到任意数据帧后，发送当前对应的 **ACK**；
 - 发送方维护超时定时器，若未收到 **ACK** 则超时重传窗口内全部未确认数据；若收到新的 **ACK**，则更新当前窗口位置，继续发送窗口内数据
- 配置文件：**config.cfg** 中设定各类参数，使用标准 C++ 文件读入模块初始化，修改配置无需反复编译文件
- 日志：使用标准输出 + 日志文件记录每次数据包的收发及事件；
- 模拟丢包和错误：在发送端根据丢包率和错误率设定，模拟数据损坏或丢失。

2.2 校验和控制

- 发送方通过CRC算法，在每个数据包装填数据后计算CRC校验码，并赋值到**checksum**字段
- 接收方通过校验码，判断每个接收包的数据正误

2.3 组织组件

代码分为发送端（`sender.exe`）和接收端（`receiver.exe`）

- 发送端：读取文件，按照 `dataSize` 将文件分割成多个 PDU，装填数据并向接收端发送
- 接收端：校验并识别数据包，向发送端返回ACK确认包

2.4 配置文件

代码通过配置文件（`config.cfg`）控制两端应用的行为，修改配置文件会直接影响应用表现，无需重新编译。

- `config.cfg` 中含有 `port`, `filePath`, `windowSize`, `timeout`, `lostRate`, `errorRate` 等参数
- 通过 `loadConfig` 读取后传入各功能模块

2.5 日志功能

代码实现了发送端和接收端的双端日志功能。每个数据包的发送和接收都会被写入对应日志。日志格式如下：

发送端日志：

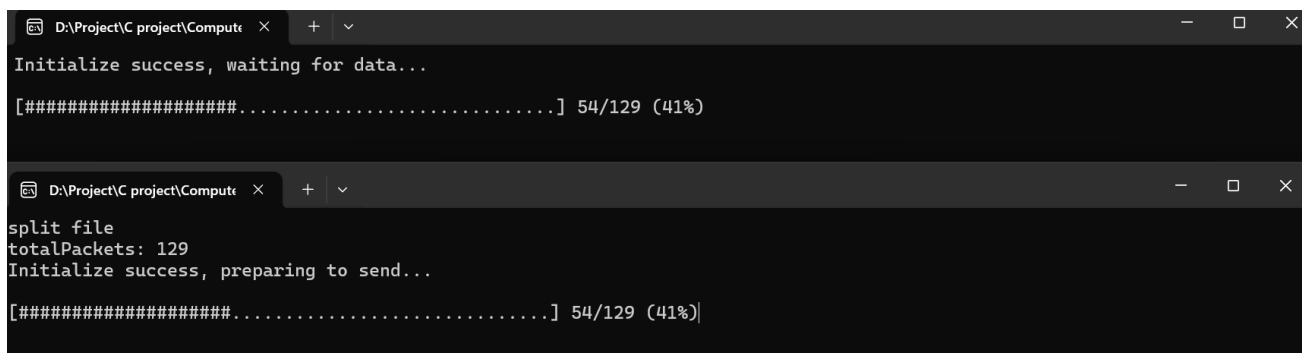
- `[2025-05-07 22:18:42] 1, pdu_to_send=13, status=NEW, ackedNo=4294967295`
 - `[2025-05-07 22:18:42]` :时间戳
 - `1`: 该数据包的发送次数
 - `pdu_to_send`: 该数据包的序号
 - `status`: 该数据包的发送状态, NEW/ TO/ RT 对应 初次发送/ 超时重传/ 丢包或错误重传

接收端日志：

- `[2025-05-07 23:00:58] 1, pdu_exp=5, pdu_recv=6, status=NoErr`
 - `[2025-05-07 23:00:58]`: 时间戳
 - `1`: 该数据包的接收次数
 - `pdu_exp`: 期望接收的包序号
 - `pdu_recv`: 实际接收的包序号
 - `status`: 该数据包的接收状态: OK/ NoErr/ DataErr 对应 正确数据包/ 序号错误的包/ 数据错误的包

2.6 终端展示

代码编写了简易的进度条输出, 隐藏了所有其余输出, 仅保留 `已确认包/ 总包数` 的进度条, 将所有具体情况输出到日志文件。



The image shows two terminal windows side-by-side. The top window has a tab labeled 'D:\Project\C project\Compute' and displays the text: 'Initialize success, waiting for data...' followed by a progress bar '[#####.....] 54/129 (41%)'. The bottom window also has a tab labeled 'D:\Project\C project\Compute' and displays: 'split file', 'totalPackets: 129', 'Initialize success, preparing to send...' followed by the same progress bar '[#####.....] 54/129 (41%)'.

3. Development and Implementation

3.1 开发环境

- 操作系统：Windows 11
- 编程语言：C++
- 工具链：MinGW / g++、Visual Studio Code
- 网络库：Winsock（Windows）
- 构建方式：头文件（proto.h）、主函数（sender.cpp、receiver.cpp）

3.2 项目结构



3.3 核心代码

进行CRC校验计算

位于 `proto.h`，计算当前数据帧的CRC码并赋值给 `checksum` 字段

```
// 进行CRC校验计算
void calculateChecksum() {
    vector<char> buffer(sizeof(totalPackets) + sizeof(seqNo) + sizeof(length) + length);

    memcpy(buffer.data(), &totalPackets, sizeof(totalPackets));
    memcpy(buffer.data() + sizeof(totalPackets), &seqNo, sizeof(seqNo));
    memcpy(buffer.data() + sizeof(totalPackets) + sizeof(seqNo), &length, sizeof(length));
    memcpy(buffer.data() + sizeof(totalPackets) + sizeof(seqNo) + sizeof(length), data, length);

    // printBufferHex(buffer.data(), buffer.size());
    this->checksum = crc16(buffer.data(), buffer.size());
}
```

待传输文件切分

```
// 读取文件并切分为多个 PDU
vector<PDU> splitFileToPackets(const string &filename,
int dataSize, int initSeq, int &totalPackets)
{
    cout << "split file" << endl;
    ifstream file(filename, ios::binary | ios::ate);
    if (!file.is_open())
    {
        cerr << "Failed to open file: " << filename <<
endl;
```

```

        exit(1);
    }

    streamsize fileSize = file.tellg(); // 获取文件大小
    file.seekg(0, ios::beg);           // 回到文件开头

    totalPackets = static_cast<int>
(ceil((double)fileSize / dataSize));
    vector<PDU> packets;
    packets.reserve(totalPackets);

    for (int i = initSeq; i < totalPackets + initSeq;
++i)
    {
        int index = i - initSeq; // 相对索引 (0开始)

        // 每个包的大小, 考虑最后一个包需要额外切分
        int thisSize = (index < totalPackets - 1)
? dataSize
: static_cast<int>(fileSize - (totalPackets -
1) * dataSize);

        PDU pdu;
        pdu.totalPackets = totalPackets; // 设置总包数
        pdu.seqNo = i;
        pdu.length = thisSize;
        pdu.allocateData(thisSize); // 分配数据空间
        file.read(pdu.data, thisSize); // 读取对应内容
        pdu.calculateChecksum(); // 计算校验和

        packets.push_back(pdu);
    }

```



```
file.close();  
return packets;  
}
```

发送端确认与重传机制

发送端初始化参数后，按序发送窗口内的所有数据帧

```
// 发送窗口内的数据包  
while (seq < totalPackets + initSeq)  
{  
    // 当下一个要发送的包还在窗口内时发送数据包  
    while (nextSeqNum < seq + swSize && nextSeqNum < totalPackets + initSeq)  
    {  
        // 从切分好的列表中获取当前要发送的包  
        PDU &pdu = packets[nextSeqNum - initSeq];  
  
        // 查map获取当前包的重传次数  
        int sendCount = retransmitCount[nextSeqNum];  
        retransmitCount[nextSeqNum] = ++sendCount; // 更新重传次数  
  
        // 定义当前包的发送状态：初次发送/超时/重传  
        string status = sendCount == 1 ? "NEW" : (timeoutFlag ? "TO " : "RT ");  
  
        // 有出错概率地发送数据包  
        sendWithError(sock, destAddr, pdu, sendCount, status, lostRate, errorRate, ackReceive  
  
        nextSeqNum++; // 更新下一个序列号  
    }  
}
```

在当前窗口内所有数据包发送完毕后，开始阻塞等待ACK。若超时未收到ACK，将当前窗口回退至最近收到的ACK对应序号的数据包处，重新发送数据包。

```

// 将窗口内数据包全部发送完毕，开始等待ACK
bool ackFlag = false; // 当前超时期内是否收到ACK的标志
auto startTime = chrono::high_resolution_clock::now(); // 记录开始等待时间

// 等待ACK确认
while (!ackFlag)
{
    // 检查超时
    auto currentTime = chrono::high_resolution_clock::now();
    auto elapsed = chrono::duration_cast<std::chrono::milliseconds>(currentTime - startTime);

    // 超时处理：更新窗口位置，并发送窗口内所有未确认包
    if (elapsed.count() >= timeout)
    {
        timeoutFlag = true; // 设置超时标志
        seq = max(ackReceived + 1, initSeq); // 更新窗口起始位置
        nextSeqNum = seq; // 重置下一个要发送的包序列号
        break;
    }
}

```

若接收到ACK，更新当前的窗口状态。

```

// 未超时，正常接收 ACK
int ret = recvfrom(sock, recvBuf, sizeof(recvBuf), 0, (sockaddr *)&destAddr, &receiverLen);
if (ret > 0)
{
    PDU ack = deserializePDU(recvBuf, ret);

    // 若收到ACK，则更新窗口
    if (ack.isValid())
    {
        ackFlag = true; // 收到有效的ACK
        ackReceived = max(int(ack.seqNo), ackReceived); // 更新已收到的最新ACK序列号
        seq = ackReceived + 1; // 更新窗口的起始位置
        printProgressBar(ackReceived - initSeq + 1, totalPackets); // 打印进度条
        break;
    }
    else
        continue; // 无效的ACK，继续等待
}
}

```

4. System Deployment, Startup, and Use

1. 修改 `config.cfg` 中参数（UDP端口、文件路径、窗口大小、错误率等），参考配置如下：

```
UDPPort=41680
DataSize=8192
ErrorRate=10
LostRate=10
SWSize=30
InitSeqNo=1
Timeout=150
SendLogPath=./log/sender_log.txt
RecvLogPath=./log/receiver_log.txt
InputPath=./data/input.png
OutputPath=./data/output.png
```

（注：上图是在传输3.62MB的png文件时的参考配置）

2. 编译项目：

```
cd $dir && chcp 65001 && g++ $fileName -o  
$fileNameWithoutExt -finput-charset=UTF-8 -  
fexec-charset=UTF-8 -lpsapi -lkernel32 -lws2_32  
&& $dir$fileNameWithoutExt
```

3. 启动接收端：

```
./receiver.exe
```

4. 启动发送端：

```
./sender.exe
```

5. 等待文件传输完成即可

5. System Test

单元测试

编写了 `testSender.cpp`，尝试向接收端按序发送了100个自定义的准确无误的数据包。经 `testSender` 测试，如下功能已成功实现：

- PDU 序列化与反序列化正确
- 校验和函数通过模拟错误验证

```

// 手动生成并组装100个PDU进行测试
for (int i = 0; i < totalPackets; ++i)
{
    PDU pdu;
    pdu.totalPackets = totalPackets; // 设置总包数
    pdu.seqNo = i;
    pdu.length = dataSize;           // 设置数据长度
    pdu.allocateData(dataSize);      // 分配数据空间
    memset(pdu.data, 'A' + i % 26, dataSize); // 设置数据区内容
    pdu.calculateChecksum();          // 计算校验和

    // 序列化 PDU
    int packetLen;
    char *serialized = serializePDU(pdu, packetLen);

    sendto(sock, serialized, packetLen, 0, (sockaddr *)&destAddr, sizeof(destAddr));
    delete[] serialized; // 释放序列化后的数据
}

```

集成测试

分别使用16KB的txt文件、3.62MB的png文件、9.47MB的pptx文件进行文件传输测试，过程中：

- 正常 ACK 收发，日志完整；
- 设置错误率=0.1、丢包率=0.1，模拟测试能最终完成文件传输。

测试截图

D:\Project\C project\Compute × + ▾

```
split file
totalPackets: 929
Initialize success, preparing to send...

[#####.....] 843/929 (90%)|
```

D:\Project\C project\Compute × + ▾

```
Initialize success, waiting for data...

[#####.....] 843/929 (90%)|
```

D:\Project\C project\Compute × + ▾

```
split file
totalPackets: 929
Initialize success, preparing to send...

[#####] 929/929 (100%)All packets acknowledged, exiting...

[INFO] Total transmission time: 38 s

Total packets sent: 2475
Total Retransmissions: 1546
Average Retransmissions: 1.66
Timeout Retransmissions: 104 / 2475 = 4.20%
Error or Lost Retransmissions: 1442 / 2475 = 58.26%
请按任意键继续. . .|
```

D:\Project\C project\Compute × + ▾

```
split file
totalPackets: 929
Initialize success, preparing to send...

[#####] 929/929 (100%)All packets acknowledged, exiting...

[INFO] Total transmission time: 87 s

Total packets sent: 4534
Total Retransmissions: 3605 / 4534 = 79.51%
Timeout Retransmissions: 242 / 4534 = 5.34%
Error or Lost Retransmissions: 3363 / 4534 = 74.17%
请按任意键继续. . .|
```



文件

编辑

查看

```
[2025-05-07 23:28:31] 4, pdu exp=3689, pdu recv=3689, status=OK
[2025-05-07 23:28:31] 4, pdu exp=3690, pdu recv=3690, status=OK
[2025-05-07 23:28:31] 3, pdu exp=3691, pdu recv=3691, status=OK
[2025-05-07 23:28:31] 4, pdu exp=3692, pdu recv=3692, status=OK
[2025-05-07 23:28:31] 4, pdu exp=3693, pdu recv=3693, status=OK
[2025-05-07 23:28:31] 2, pdu exp=3694, pdu recv=3694, status=OK
[2025-05-07 23:28:31] 4, pdu exp=3695, pdu recv=3695, status=OK
[2025-05-07 23:28:31] 4, pdu exp=3696, pdu recv=3696, status=OK
[2025-05-07 23:28:31] 3, pdu exp=3697, pdu recv=3697, status=OK
[2025-05-07 23:28:31] 2, pdu exp=3698, pdu recv=3698, status=OK
[2025-05-07 23:28:31] 2, pdu exp=3699, pdu recv=3699, status=OK
[2025-05-07 23:28:31] 2, pdu exp=3700, pdu recv=3700, status=OK
[2025-05-07 23:28:31] 2, pdu exp=3701, pdu recv=3701, status=OK
[2025-05-07 23:28:31] 1, pdu exp=3702, pdu recv=3702, status=OK
[2025-05-07 23:28:31] 1, pdu exp=3703, pdu recv=3703, status=OK
[2025-05-07 23:28:31] 1, pdu exp=3704, pdu recv=3704, status=OK
[2025-05-07 23:28:31] 1, pdu exp=3705, pdu recv=3705, status=OK
[2025-05-07 23:28:31] 1, pdu exp=3706, pdu recv=3706, status=OK
[2025-05-07 23:28:31] 1, pdu exp=3707, pdu recv=3707, status=OK
[2025-05-07 23:28:31] 1, pdu exp=3708, pdu recv=3708, status=OK
[2025-05-07 23:28:31] 1, pdu exp=3709, pdu recv=3709, status=OK
[2025-05-07 23:28:31] 1, pdu exp=3710, pdu recv=3711, status=NoErr
[2025-05-07 23:28:31] 1, pdu exp=3710, pdu recv=3712, status=NoErr
[2025-05-07 23:28:31] 1, pdu exp=3710, pdu recv=3713, status=NoErr
[2025-05-07 23:28:31] 1, pdu exp=3710, pdu recv=3714, status=NoErr
[2025-05-07 23:28:31] 1, pdu exp=3710, pdu recv=3715, status=NoErr
[2025-05-07 23:28:32] 1, pdu exp=3710, pdu recv=3710, status=OK
[2025-05-07 23:28:32] 2, pdu exp=3711, pdu recv=3711, status=OK
[2025-05-07 23:28:32] 2, pdu exp=3712, pdu recv=3712, status=OK
[2025-05-07 23:28:32] 2, pdu exp=3713, pdu recv=3713, status=OK
[2025-05-07 23:28:32] 2, pdu exp=3714, pdu recv=3714, status=OK
[2025-05-07 23:28:32] 2, pdu exp=3715, pdu recv=3715, status=OK
```

6. Performance and Analysis

为了优化代码性能，实验通过修改配置文件，分析了各种情形下传输同一文件所需的时间，并统计其需要的平均重传次数（包括超时重传和丢包/错包重传），分析结果如下：

错误率	丢包率	包大小	窗口大小	超时限 时	平均重传次数	耗时
0.05	0.05	4096 byte	15	300ms	1.66	38s
0.1	0.1	4096 byte	15	300ms	4.88	87s
0.1	0.1	4096byte	30	300ms	7.99	92s
0.05	0.05	8192 byte	15	300ms	2.85	30s
0.1	0.1	8192 byte	15	300ms	5.08	53s
0.1	0.1	8192byte	30	300ms	8.78	46s
0.1	0.1	8192byte	30	150ms	8.78	25s

（上述测试结果基于传输相同的3.62MB的png文件测试）

部分分析结果截图：

```
split file
totalPackets: 465
Initialize success, preparing to send...

[#####]

[INFO] Total transmission time: 53 s

Total packets sent: 2828
Total Retransmissions: 2363
Average Retransmissions: 5.08
Timeout Retransmissions: 159 / 2828 = 5.62%
Error or Lost Retransmissions: 2204 / 2828 = 77.93%
请按任意键继续 . . . |
```

```
split file
totalPackets: 465
Initialize success, preparing to send...

[#####]

[INFO] Total transmission time: 46 s

Total packets sent: 4547
Total Retransmissions: 4082
Average Retransmissions: 8.78
Timeout Retransmissions: 138 / 4547 = 3.03%
Error or Lost Retransmissions: 3944 / 4547 = 86.74%
请按任意键继续 . . . |
```

```
split file
totalPackets: 465
Initialize success, preparing to send...

[#####]

[INFO] Total transmission time: 25 s

Total packets sent: 4547
Total Retransmissions: 4082
Average Retransmissions: 8.78
Timeout Retransmissions: 138 / 4547 = 3.03%
Error or Lost Retransmissions: 3944 / 4547 = 86.74%
请按任意键继续 . . . |
```

现象分析

- 随着错误率和丢包率的增加，平均重传次数和传输时长显著增加。例如，当错误率和丢包率从0.05增加到0.1时，平均重传次数从1.66增加到4.88，传输时长从38s增加到了87s。
- 包大小的增加会导致平均重传次数的增加，但总的传输时间却下降了。例如，包大小从4096 byte增加到8192 byte时，平均重传次数从1.66增加到2.85，而传输时长从38s降低到了30s。
- 窗口大小的增加会导致平均重传次数的增加，但却不一定影响总的传输时间。包大小4096byte时，将窗口大小从15增大到30，导致平均重传次数从4.88增长到7.99，传输市场从87s增长到92s。然而，在包大小8192byte时，将窗口从15增大到30，同样导致平均重

传次数增大，从5.08增长到8.78，但传输时长从53s缩减到了46s

结论

- 错误率和丢包率是影响重传次数和传输时间的主要因素。降低错误率和丢包率可以显著减少重传次数和传输时间。
 - 包大小的增加会增加重传次数，尤其是在网络状况不佳的情况下。
 - 窗口大小的增加可以提高传输效率，减少总传输时间，但在高错误率和丢包率的情况下可能会导致更多的重传和更久的总传输时间。
 - 超时限时的减少可以显著减少传输时间，尤其是在网络状况较好的情况下。
-

7. Summary or Conclusions

本项目深入理解了 GBN 协议的核心机制，包括窗口管理、超时重传、确认应答及数据帧结构设计等内容。在实践中，我们通过模拟真实网络环境下的丢包、误码情况，验证了该协议的可靠性与健壮性。

项目中遇到的关键问题如序列号回绕、超时精度、回退控制等均已解决，最终系统运行稳定，结构清晰，模块职责明确。通过本实验，对面向连接的协议设计、网络编程、协议仿真等能力有显著提升。

8. References

1. Tanenbaum, A. S., & Wetherall, D. J. (2010). *Computer Networks (5th ed.)*. Pearson.
 2. 李新华. 计算机网络教程. 清华大学出版社.
-

9. Comments

本课程实践性强，涵盖网络协议从原理到实现的全过程，建议在项目阶段提供更多错误模拟与调试辅助工具示例。此外，可加入 TCP 与 GBN 的对比分析环节，加强理论联系实际的能力。