
华中科技大学计算机学院

《计算机通信与网络》实验报告

班级_____ 姓名_____ 学号_____

项目	Socket 编程 (40%)	数据可靠传输协议设计 (20%)	CPT 组网 (20%)	平时成绩 (20%)	总分
得分					

教学目标达成情况一览表

实验 \ 目标	目标 1	目标 2	目标 3	目标 4	目标 5	目标 6	合计
Socket 编程	/10	/7	/6	/5	/4	/8	/40
数据可靠传输 协议设计	/5	/5	/3	/3	/4		/20
CPT 组网	/5	/6		/3	/4	/2	/20
小计	/20	/18	/9	/11	/12	/10	/80

教师评语：

教师签名：

给分日期：

目 录

实验一 SOCKET 编程实验	3
1.1 环境	3
1.2 系统功能需求	3
1.3 系统设计	4
1.4 系统实现	5
1.5 系统测试及结果说明	8
1.6 其它需要说明的问题	12
实验二 数据可靠传输协议设计实验	13
2.1 环境	13
2.2 实验要求	13
2.3 协议的设计、验证及结果分析	13
2.4 其它需要说明的问题	32
实验三 基于 CPT 的组网实验	33
3.1 环境	33
3.2 实验要求	33
3.3 基本部分实验步骤说明及结果分析	35
3.4 综合部分实验设计、实验步骤及结果分析	44
3.5 其它需要说明的问题	47
心得体会与建议	48
4.1 心得体会	48
4.2 建议	49
参考文献	50

实验一 Socket 编程实验

1.1 环境

1.1.1 开发平台

Visual Studio 2017

1.1.2 运行平台

操作系统: Windows 10;

CPU: Inter-Core-i7-7500U;

RAM: 16G

1.2 系统功能需求

编写一个支持多线程处理的 Web 服务器软件, 要求如下:

第一级:

- ✧ 可配置 Web 服务器的监听地址、监听端口和虚拟路径。
- ✧ 能够单线程处理一个请求。当一个客户 (浏览器, 输入 URL : `http://127.0.0.1/index.html`) 连接时创建一个连接套接字;
- ✧ 从连接套接字接收 http 请求报文, 并根据请求报文的确定用户请求的网页文件;
- ✧ 从服务器的文件系统获得请求的文件。创建一个由请求的文件组成的 http 响应报文。(报文包含状态行+实体)。
- ✧ 经 TCP 连接向请求的浏览器发送响应, 浏览器可以正确显示网页的内容;
- ✧ 服务可以启动和关闭。

第二级:

- ✧ 支持多线程, 能够针对每一个新的请求创建新的线程, 每个客户请求启动一个线程为该客户服务;
- ✧ 在服务器端的屏幕上输出每一个请求的来源 (IP 地址、端口号和 HTTP 请求命令行)
- ✧ 支持一定的异常情况处理能力。

第三级:

- ✧ 能够传输包含多媒体（如图片）的网页给客户端，并能在客户端正确显示；
- ✧ 对于无法成功定位文件的请求，根据错误原因，作相应错误提示。
- ✧ 在服务器端的屏幕上能够输出对每一个请求处理的结果。
- ✧ 具备完成所需功能的基本图形用户界面（GUI），并具友好性

1.3 系统设计

1.3.1 系统架构

本实验要求实现 web 服务器，通讯双方为 Web 服务器端(服务器端)和浏览器(客户端)。服务器需要处理来自浏览器的 http 请求，分析报文后给出应答报文，并发送相应的文件或错误代码。采用了异步实现的方式监听选定的地址和端口。

1.3.2 功能模块划分

系统分为四个主要模块，包括初始化系统 Socket 服务，解析 HTTP 请求报文，发送文件，发送错误响应报文，如图 1.1 所示。

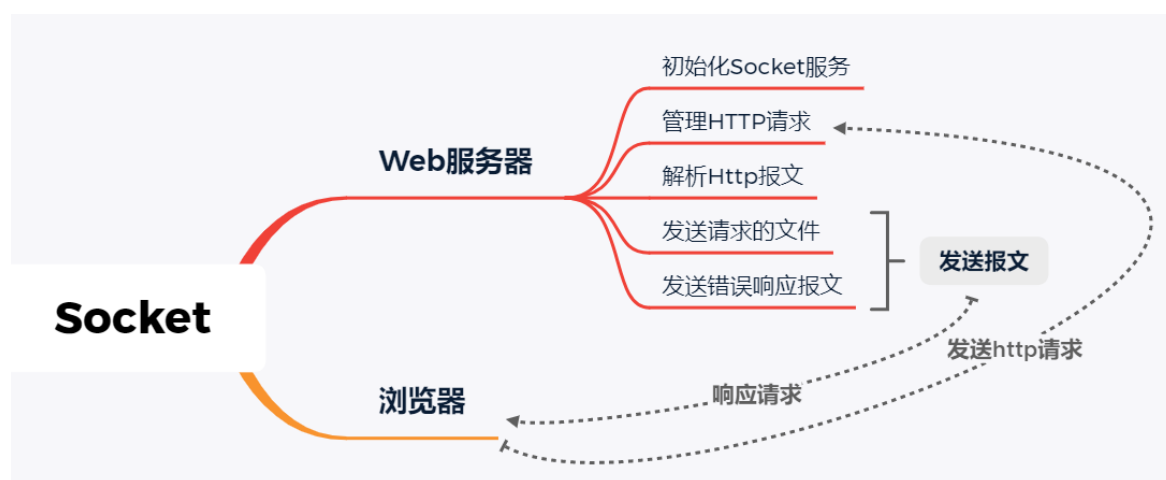


图 1.1 功能模块划分

(1) 初始化 Socket 模块

本模块实现系统 socket 服务的初始化，包括初始化 WINSOCK，配置服务器的监听地址，监听端口和虚拟路径，若启动失败则打印错误报文并关闭 socket 服务。

(2) HTTP 请求管理模块

本模块利用 fd 和 select 操作实现对请求的异步管理，对于长时间未发出请求的 TCP 链接主动关闭。

(3) HTTP 报文解析模块

本模块从 http 请求报文中提取出对应的请求文件，并分析系统中文件的存在情况，并根据文件情况选择调用文件发送或错误响应报文发送。

(4) 请求文件发送模块

本模块依据请求文件的类型构建返回报文，并在对应的 tcp 套接字中发送文件的内容，支持 jpg, html, txt, MP3, MP4 等多种件格式。

(5) 错误报文响应模块

本模块在无法定位请求的文件时构建错误报文并返回给对应套接字。

1.4 系统实现

(1) 初始化系统 socket 服务

函数原型：int server_initialize(WORD w, WSADATA &wsaData, SOCKET &s, SOCKADDR_IN &srvAddr);

初始化如流程图所示：

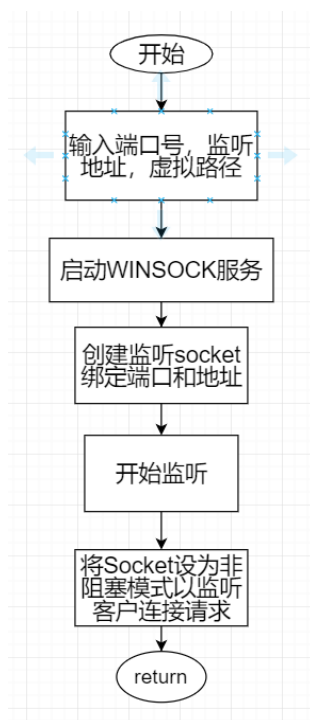


图 1.2 初始化 socket

(2) 管理 HTTP 请求

数据结构及变量定义：

```
typedef list<SOCKET> ListCONN;
```

```
typedef list<SOCKET> ListConErr;
```

```
ListCONN conList;      //保存所有有效的会话 SOCKET
```

```
ListConErr conErrList; //保存所有失效的会话 SOCKET
```

```
map<SOCKET, clock_t> conn_isvalid; //记录 socket 的有效情况
```

```
FD_SET rfd;
```

首先定义了两个 list(`conList` 和 `conErrList`)用来存储创建的会话 `socket` 和已经失效的 `socket` (超过一定时间未发送文件请求则认为其已经失效)。主程序为一个大的循环体,在循环的初始进行键值判断,若按下 `p` 键,则退出循环,作为关闭服务器的标志。循环开始会关闭 `conErrList` 中已失效的 `socket` 并对 `conList` 进行更新,以保证服务器始终有足够的资源区满足那些正在发送请求的 `socket`,而忽略掉长时间未使用的 `socket`。

接着会将 `conList` 中的 `socket` 设置为非阻塞模式并加入到 `rfds` 中(`rfds` 为 `struct fd_set`,可以理解为一个集合,这个集合中存放的是文件描述符(file descriptor),即文件句柄,这可以说是我们所说的普通意义的文件,一个 `socket` 就是一个文件,`socket` 句柄就是一个文件描述符。`fd_set` 集合可以通过一些宏由人为来操作,比如清空集合 `FD_ZERO(fd_set *)`,将一个给定的文件描述符加入集合之中 `FD_SET(int,fd_set *)`,将一个给定的文件描述符从集合中删除 `FD_CLR(int,fd_set*)`,检查集合中指定的文件描述符是否可以读写 `FD_ISSET(int,fd_set*)`,并将监听 `socket` 添加到 `rfds` 中(`FD_SET(srvSock, &rfds)`),从而以异步的方式实现对于浏览器请求的监听。

接下来调用 `select` 函数,该函数是阻塞的,为该函数设置一个超时时间保证程序不会一直阻塞在这里。调用 `FD_ISSET` 对于 `srvsock` 和 `conList` 中的每一项进行可读性检测,`srvsock` 可读说明有新的请求需要处理,将其加入 `conList` 中;`conList` 中某些项可读则说明当前已建立的 `TCP` 连接中有新的文件请求需要处理,解析其报文并返回响应文件或错误报文即可。

之后需要根据 `socket` 的未响应时间对其进行失效性超时判断,将超时失效的 `socket` 加入 `conErrList` 中,并进入下一次主循环。

整个流程如图 1.3 所示:

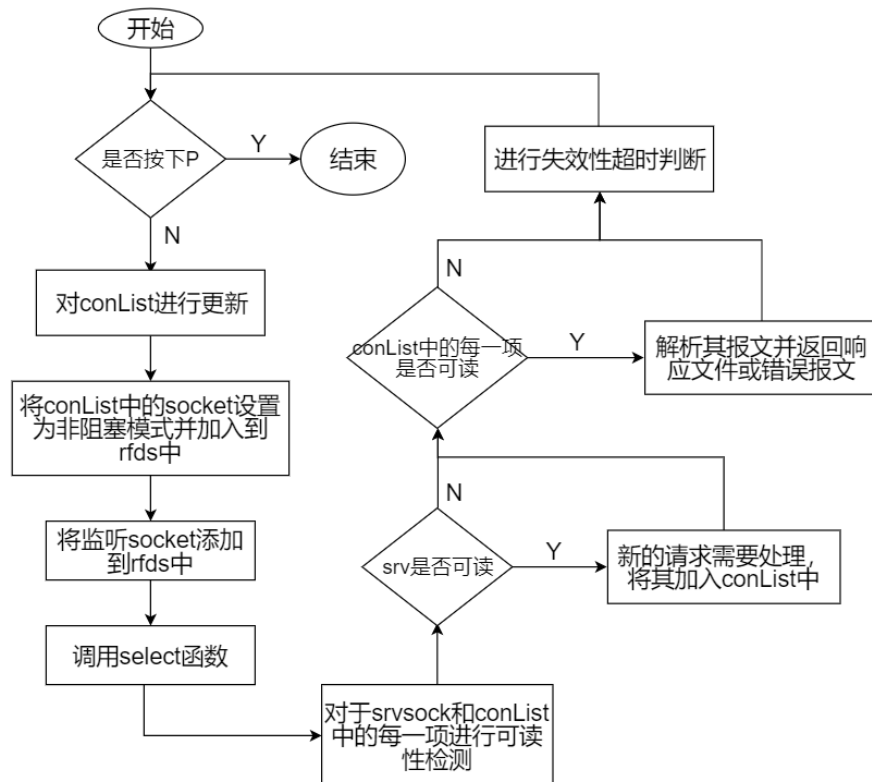


图 1.3 管理 HTTP 请求

(3) HTTP 报文解析

对于请求报文，首先提取出第一行即请求报文头，通过字符串操作解析出请求的文件名称。文件名称加上虚拟路径即可构成服务器索引文件的绝对路径。

处理 GET 请求，解析出请求的文件名称，具体过程如图 1.4 所示：

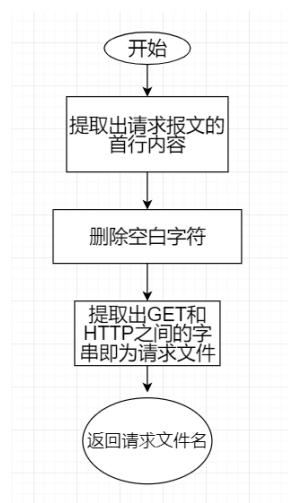


图 1.4 HTTP 报文解析

(4) 请求文件发送

首先根据文件名加上虚拟路径构成的绝对路径尝试打开对于文件，若打开失败，进入错误报文响应流程；若打开成功，则构造文件响应报文。响应报文应该包括的内容有 http 类型及响应码：“HTTP/1.1 200 OK\r\n”，类型“Content-Type: image/jpeg\r\n”，长连接确认：“Connection: keep-alive\r\n\r\n”，content 长度：“Content-Length: %d\r\n”。之后通过 send 函数将内容发到对应的套接字即可，当发送内容过大的时候需要进行分块处理，多次发送。

(5) 错误报文响应

构造和(4)相同的响应报文，但要将返回码设为 404，即“HTTP/1.1 404 Not Found OK\r\n”。

(6) 流程控制

Main 函数包括一个循环，当键入 S 时启动服务器，键入 P 时停止服务器（停止逻辑在(2)中实现，停止后返回 Main 函数），如流程图 1.5 所示：

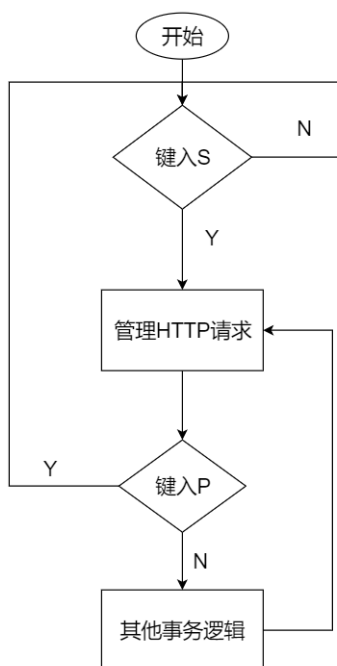


图 1.5 程序流程控制

1.5 系统测试及结果说明

(1) 配置 Web 服务器的监听地址、监听端口和虚拟路径；启动和关闭服务器(S, P)


```

s:启动服务器;  p:停止服务器

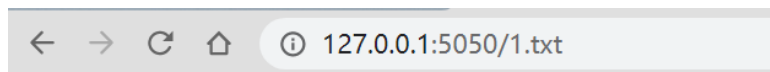
input listen PORT: 5050
input listen IP_ADDR: 127.0.0.1
input listen VIRTUAL_ADDR: C:\Users\朱志成\Desktop\server_v\

Server's winsock initialized!
Server TCP socket creat OK!
Server socket bind OK!
Server start listen port 5050

connSock: 496
IP: 127.0.0.1 PORT: 56612
http request: GET /1.jpg HTTP/1.1
find file : C:\Users\朱志成\Desktop\server_v\1.jpg
send file success!

```

图 1.6 服务器启动配置



计算机网络是指将地理位置不同的具有独立功能的多台计算机及其外部设

图 1.7 文件访问测试

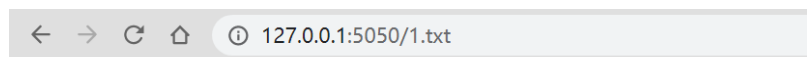
```

正在关闭浏览器socket...
关闭server socket...
server 已关闭!

```

图 1.8 关闭测试

(2) 请求各种格式的文件



计算机网络是指将地理位置不同的具有独立功能的多台计算机及其外部设备，通过通信线路连接

图 1.9 txt 文件请求

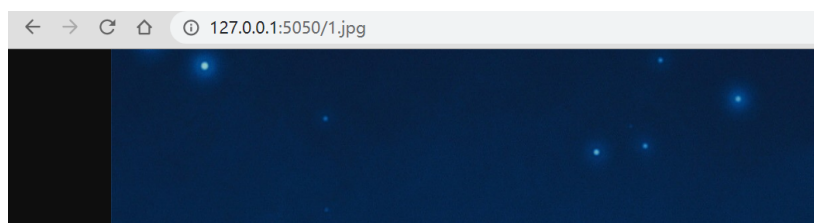


图 1.10 jpg 文件请求

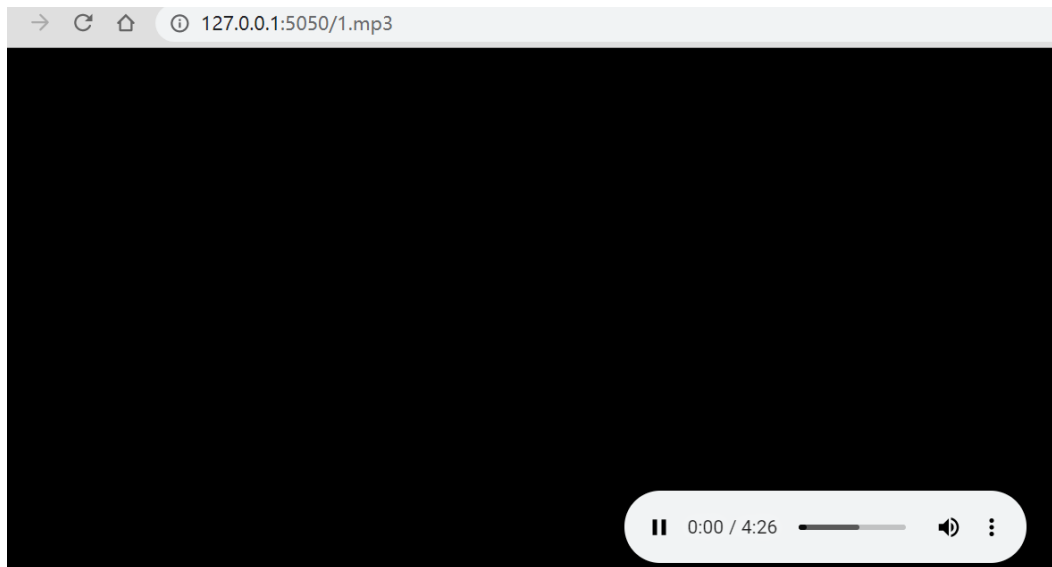


图 1.11 MP3 文件请求

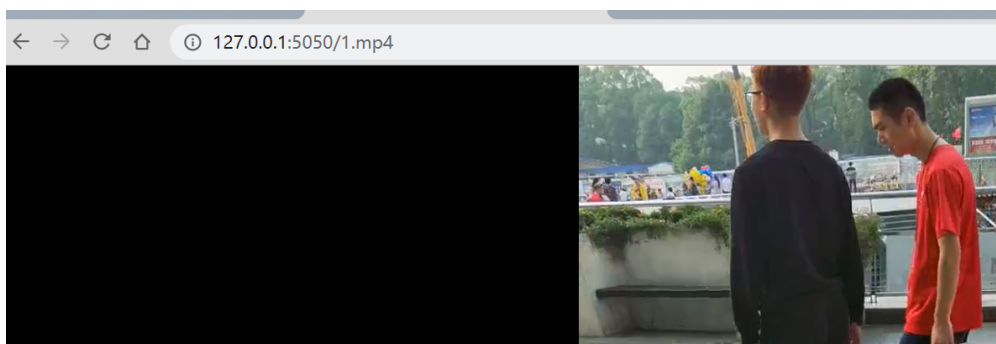


图 1.12 MP4 文件请求

← → ↻ 🏠 ⓘ 127.0.0.1:5050/index.html

ORAL settings 在线测试 错题分析 我的账户 ▾

题目类型: 加法 减法 乘法 除法

运算项个数: 求解目标:

题目数目: ⚙️ 详细设置

年 级: 一年级 二年级 三年级

立即生成!

图 1.12 包含多引用文件的 html 文件请求

(3) 输出每一个请求的来源和处理结果

```
IP: 127.0.0.1 PORT: 45349
http request: GET /js/bmap.min.js HTTP/1.1
find file : C:\Users\朱志成\Desktop\server_v\js/bmap.min.js
send file success!

connSock: 516
IP: 127.0.0.1 PORT: 45605
http request: GET /js/bootstrap.min.js HTTP/1.1
find file : C:\Users\朱志成\Desktop\server_v\js/bootstrap.min.js
send file success!

connSock: 284
IP: 127.0.0.1 PORT: 44837
http request: GET /js/dataTool.min.js HTTP/1.1
find file : C:\Users\朱志成\Desktop\server_v\js/dataTool.min.js
send file success!
```

图 1.13 输出请求的来源和处理结果

(4) 异常处理（请求不存在或无法定位的文件）

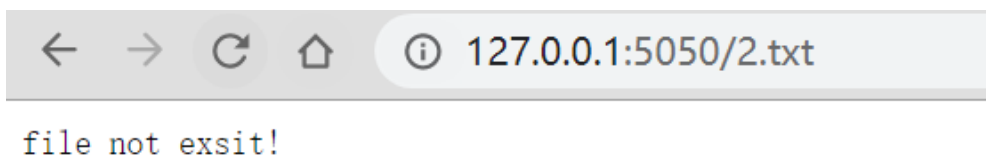


图 1.14 文件不存在时的请求结果

```
connSock: 540
IP: 127.0.0.1 PORT: 31014
http request: GET /2.txt HTTP/1.1
find file : C:\Users\朱志成\Desktop\server_v\2.txt
请求的文件不存在!
```

图 1.15 文件不存在时的处理结果

(5) 多线程和压力测试

实验二 数据可靠传输协议设计实验

2.1 环境

2.1.1 开发平台

Visual Studio 2017

2.1.2 运行平台

操作系统: Windows 10;

CPU: Inter-Core-i7-7500U;

RAM: 16G

2.2 实验要求

本实验包括三个级别的内容，具体包括：

实现基于 GBN 的可靠传输协议。

实现基于 SR 的可靠传输协议。

在实现 GBN 协议的基础上，根据 TCP 的可靠数据传输机制（包括超时后只重传最早发送且没被确认的报文、快速重传）实现一个简化版的 TCP 协议。报文段格式、报文段序号编码方式和 GBN 协议一样保持不变，不考虑流量控制、拥塞控制，不需要估算 RTT 动态调整定时器 Timeout 参数。

2.3 协议的设计、验证及结果分析

2.3.1 GBN 协议的设计、验证及结果分析

(1) 总体设计

GBN 即 Go-Back-N 协议，允许发送方一次发送多个分组而不需等待每一次确认，但等待确认 ACK 的总分组数目不能超过滑动窗口最大值 N。接收方只会顺序接受正确分组，不缓存分组，错误分组直接丢弃并返回上一次的确认 ACK。发送方和接收方流程如下所示

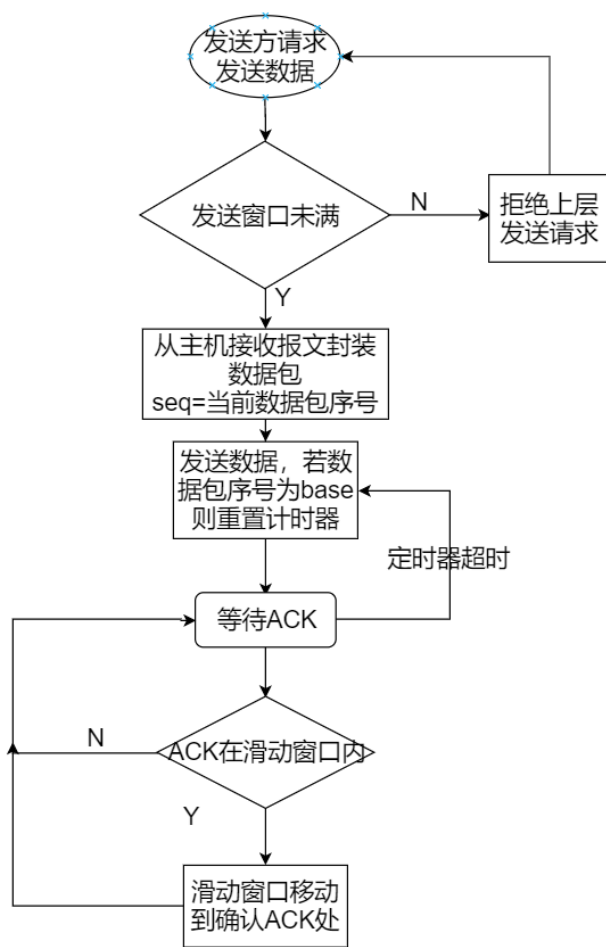


图 2.1 发送方流程图

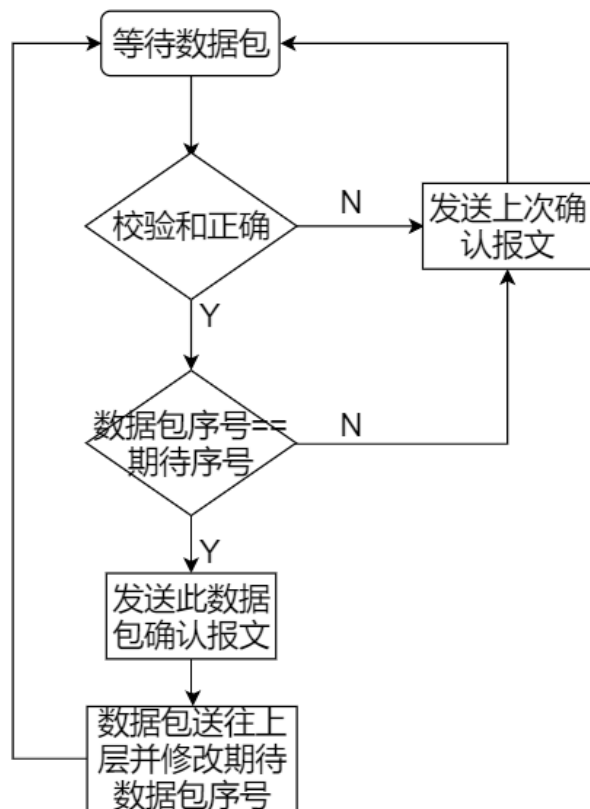


图 2.2 接收方流程图

(2) 发送方实现

接口及函数定义：

#define WINDOW_SIZE 4//窗口大小

#define ORDER_SIZE 8//序号大小

```
class GBNSender :public RdtSender
```

```
{
```

```
private:
```

```
int base;           //发送窗口头
```

```
int nextseqnum;     //下一个发送的序号
```

```
list<Packet> sndpkt; //已发送的数据包
```

```
bool waitingState;  //是否处于滑动窗口已满等待 Ack 的状态
```

```
public:
```

```
bool getWaitingState();
```

```
bool send(const Message &message); //发送应用层下来的 Message，由
NetworkServiceSimulator 调用,如果发送方成功地将 Message 发送到网络层，返回 true;如果
因为发送方处于等待正确确认状态而拒绝发送 Message，则返回 false
```

```
void receive(const Packet &ackPkt);           //接受确认 Ack, 将被 NetworkServiceSimulator
调用
void timeoutHandler(int seqNum);              //Timeout handler, 将被
NetworkServiceSimulator 调用
GBNSender();
virtual ~GBNSender();
};
```

base 为当前最早的未确认分组的序号, nextseqnum 为当下一个待发分组的序号, 可以将序号范围分割成 4 段, [0, base-1] 段内的序号对应于已经发送并确认的分组, [base,nextseqnum-1] 段对应已经发送但未被确认的分组, [nextseqnum, base+N-1] 段内的序号对应允许发送的分组, 大于或等于 base+N 的序号在滑动窗口外, 不允许使用, 直到收到确认 ACK 使 base 改变才能使用。

函数实现:

① 从应用层收到新数据分组

函数名: bool StopWaitRdtSender::send(Message &message)

入口参数: 数据 Message

返回值: 无

算法描述:

- If(发送方发送窗口已满)
- 向上层发送拒绝信息后返回;
- 将 nextseqnum 作为新数据分组序号;
- 加入校验和;
- 发送新数据分组;
- If (当前所发数据分组的序号==base)
- 启动计时器;
- 更新 nextseqnum

② 计时器超时

函数名: void StopWaitRdtSender::timeoutHandler(int seqNum)

入口参数: 数据分组序号

返回值: 无

算法描述:

- 关闭计时器;

重新启动计时器；
顺次发送滑动窗口中所有数据分组。

③ 收到确认分组

函数名：void StopWaitRdtSender::receive(Packet &ackPkt)

输入参数：确认报文

返回值：无

算法实现：

```
If(校验和正确且 ACK 在滑动窗口内)
{
    滑动 base 到 ACK 处;
    将已确认分组从 sndpkt 中移出;
    If (nextseqnum == base)
        停止计时器;
    Else
        重启计时器;
}
Else
    丢弃错误分组;
```

(3) 接收方实现

接口及函数定义：

```
#define ORDER_SIZE 8 //窗口序号
```

```
class GBNReceiver :public RdtReceiver
```

```
{
```

```
private:
```

```
    int expectSequenceNumberRcvd; // 期待收到的下一个报文序号
```

```
    Packet lastAckPkt;           //上次发送的确认报文
```

```
public:
```

```
    GBNReceiver();
```

```
    virtual ~GBNReceiver();
```

```
    void receive(const Packet &packet); //接收报文，将被 NetworkService 调用
```

```
};
```

接收方实现接受数据包校验后发送确认报文的功能，具体实现为：

函数名: void StopWaitRdtReceiver::receive(Packet &packet)

输入参数: 数据包

返回值: 无

算法描述:

```
If(校验和正确, 接受报文的序号与接收方期待序号一致){  
    取出 Message, 向上递交给应用层;  
    确认序号等于收到的报文序号;  
    添加校验和;  
    发送确认报文;  
    期待序号加一;  
}  
Else{  
    丢失该分组;  
    重发上次确认报文;  
}
```

(4) 验证结果

用 check_win 进行测试, 运行 10 次程序并比较两文件中数据, 结果如图 2.3 所示

```
Test "GBN.exe" 4:  
正在比较文件 input.txt 和 OUTPUT.TXT  
FC: 找不到差异  
  
Test "GBN.exe" 5:  
正在比较文件 input.txt 和 OUTPUT.TXT  
FC: 找不到差异  
  
Test "GBN.exe" 6:  
正在比较文件 input.txt 和 OUTPUT.TXT  
FC: 找不到差异  
  
Test "GBN.exe" 7:  
正在比较文件 input.txt 和 OUTPUT.TXT  
FC: 找不到差异  
  
Test "GBN.exe" 8:  
正在比较文件 input.txt 和 OUTPUT.TXT  
FC: 找不到差异  
  
Test "GBN.exe" 9:  
正在比较文件 input.txt 和 OUTPUT.TXT  
FC: 找不到差异  
  
Test "GBN.exe" 10:  
正在比较文件 input.txt 和 OUTPUT.TXT  
FC: 找不到差异  
  
请按任意键继续. . . |
```

图 2.3 GBN 测试

多次运行信息传输无误, 实现了 GBN 可靠信息传输。

(5) 结果分析

分析体现 GBN 协议特点（超时重传）的地方如下：

```

发送方正确收到ack包: seqnum = -1, acknum = 5, checksum = 12846, .....
滑动窗口为: 6 7 0
接收方没有正确收到发送方的报文,数据校验错误: seqnum = 6, acknum = -1, checksum = 14130, HGGGGGGGGGGGGGGGGGGGG
接收方重新发送上次的确认报文: seqnum = -1, acknum = 5, checksum = 12846, .....
接收方没有正确收到发送方的报文,报文序号不对: seqnum = 7, acknum = -1, checksum = 11559, HHHHHHHHHHHHHHHHHHHHH
接收方重新发送上次的确认报文: seqnum = -1, acknum = 5, checksum = 12846, .....
接收方没有正确收到发送方的报文,报文序号不对: seqnum = 0, acknum = -1, checksum = 8996, IIIIIIIIIIIIIIIIIIIII
接收方重新发送上次的确认报文: seqnum = -1, acknum = 5, checksum = 12846, .....
发送定时器超时
重发上次发送的报文: seqnum = 6, acknum = -1, checksum = 14130, GGGGGGGGGGGGGGGGGGGGG
重发上次发送的报文: seqnum = 7, acknum = -1, checksum = 11559, HHHHHHHHHHHHHHHHHHHHH
重发上次发送的报文: seqnum = 0, acknum = -1, checksum = 8996, IIIIIIIIIIIIIIIIIIIII

```

图 2.4 GBN 控制台信息输出

收到 ACK=5 后，滑动窗口还剩下 6，7，0，此时由于发送定时器超时，会重发滑动窗口中的所有报文 6，7，0。

分析体现 GBN 累计确认的地方如下：

```

发送方正确收到ack包: seqnum = -1, acknum = 6, checksum = 12845, .....
滑动窗口为: 7 0
接收方正确收到发送方的报文: seqnum = 0, acknum = -1, checksum = 8996, IIIIIIIIIIIIIIIIIIIII
*****模拟网络环境*****: 向上递交给应用层数据: IIIIIIIIIIIIIIIIIIIII
接收方发送确认报文: seqnum = -1, acknum = 0, checksum = 12851, .....
发送方正确收到ack包: seqnum = -1, acknum = 0, checksum = 12851, .....
滑动窗口为:
发送方发送报文: seqnum = 1, acknum = -1, checksum = 6425, JJJJJJJJJJJJJJJJJJJJJ
发送方发送报文: seqnum = 2, acknum = -1, checksum = 3854, KKKKKKKKKKKKKKKKKKKKKK
发送方发送报文: seqnum = 3, acknum = -1, checksum = 1283, LLLLLLLLLLLLLLLLLLLLLLLL
发送方发送报文: seqnum = 4, acknum = -1, checksum = 64247, MMMMMMMMMMMMMMMMMMMMMMMMMM

```

图 2.5 GBN 控制台信息输出

初始滑动窗口为 7，0，收到 ACK=0 后直接滑过了数据包 7。

2.3.2 SR 协议的设计、验证及结果分析

(1) 总体设计

SR 即 Selective-Repeat 选择重传，与 GBN 协议的区别是，接收方缓存失序分组，发送方只重传错误和丢失分组。发送方和接收方的实现流程如下图：

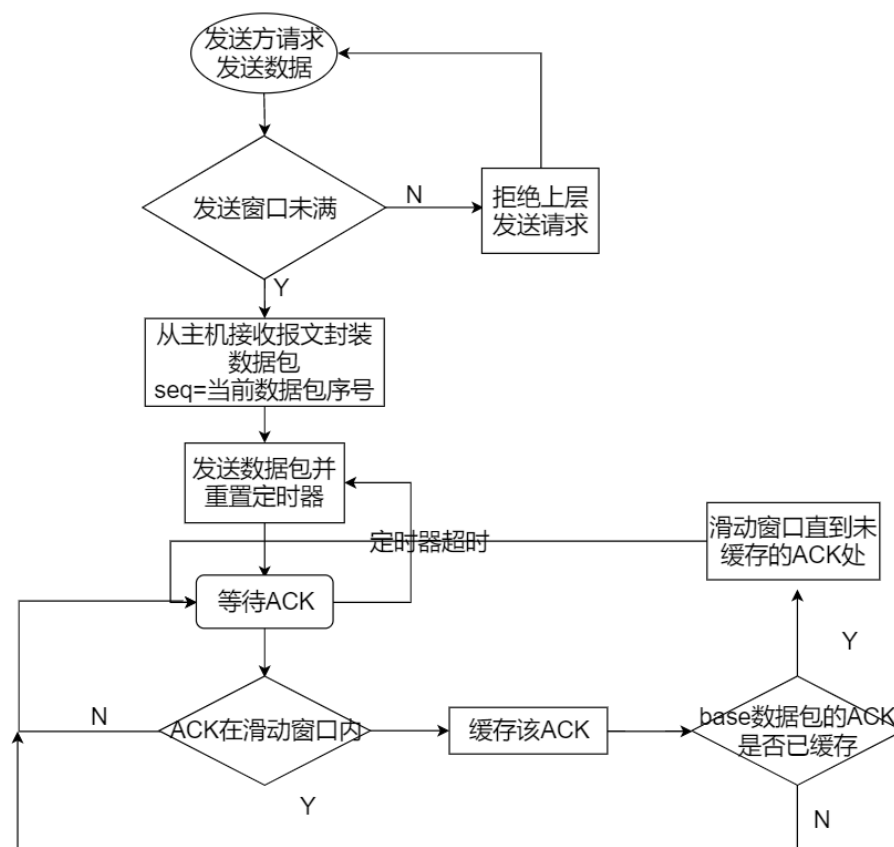


图 2.6 发送方处理流程

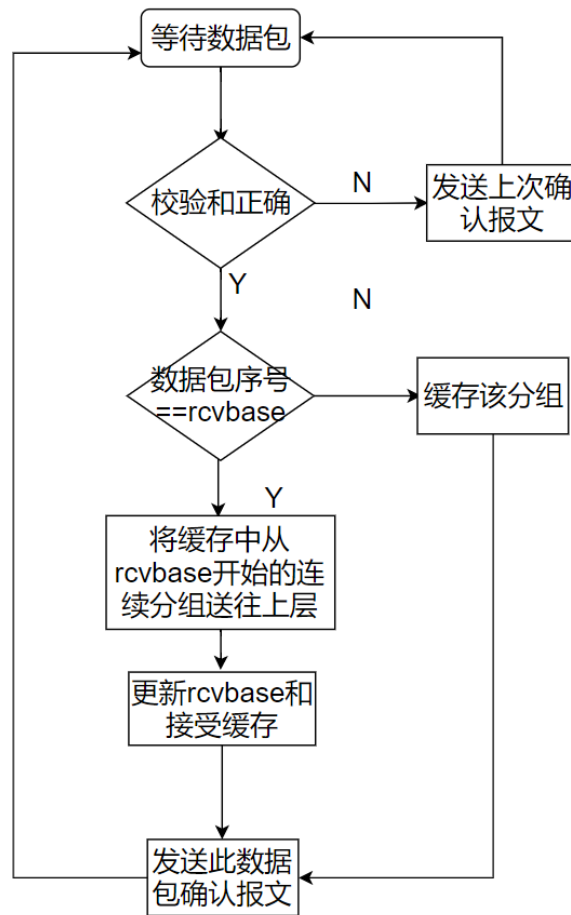


图 2.7 接收方处理流程

(2) 发送方实现

接口及函数定义：

```
#define WINDOW_SIZE 4//窗口大小
```

```
#define ORDER_SIZE 8//序号长度
```

```
class SRSEnder :public RdtSEnder
```

```
{
```

```
private:
```

```
    int base;                //发送窗口头
```

```
    int nextseqnum;          //下一个发送的序号
```

```
    list<Packet> sndpkt;      //已发送的数据包
```

```
    list<int> sndack; //等待收到的 ack
```

```
    bool waitingState;       // 是否处于滑动窗口已满等待 Ack 的状态
```

```
public:
```

```
    bool getWaitingState();
```

```
    bool send(const Message &message); // 发送应用层下来的
```

Message, 由 NetworkServiceSimulator 调用,如果发送方成功地将 Message 发送到网络层, 返回 true;如果因为发送方处于等待正确确认状态而拒绝发送 Message, 则返回 false

```
void receive(const Packet &ackPkt);           // 接受确认 Ack, 将被  
NetworkServiceSimulator 调用  
void timeoutHandler(int seqNum);              //Timeout handler, 将被  
NetworkServiceSimulator 调用
```

public:

```
    SRSender();  
    virtual ~SRSender();  
};
```

函数实现:

① 从应用层收到新数据分组

函数名: bool StopWaitRdtSender::send(Message &message)

入口参数: 数据 Message

返回值: 无

算法描述:

- If(发送方发送窗口已满)
- 向上层发送拒绝信息后返回;
- 将 nextseqnum 作为新数据分组序号;
- 加入校验和;
- 发送新数据分组;
- 启动对应报文序号的计时器;
- 更新 nextseqnum

② 计时器超时

函数名: void StopWaitRdtSender::timeoutHandler(int seqNum)

入口参数: 数据分组序号

返回值: 无

算法描述:

- 关闭计时器;
- 重新启动计时器;
- 重传超时的数据分组。

③ 收到确认分组

函数名: void StopWaitRdtSender::receive(Packet &ackPkt)

输入参数: 确认报文

返回值: 无

算法实现:

```
If(校验和正确且 ACK 在滑动窗口内)
{
    缓存确认 ACK 序号;
    根据缓存 ACK 列表更新 base 的值
    将已确认分组从 sndpkt 中移出并停止对应的计时器;
}
Else
    丢弃错误分组;
```

(3) 接收方实现

接口及函数定义:

```
#define WINDOW_SIZE 4 //接受窗口大小
#define ORDER_SIZE 8 //序号大小
class SRReceiver :public RdtReceiver
{
private:
    int rcvbase; //接受窗口头
    Packet sndpkt; //响应报文
    int pkt_count; //缓存报文数目
    list<Packet> rcvpkt; //缓存接收分组
public:
    SRReceiver();
    virtual ~SRReceiver();
    void receive(const Packet &packet); //接收报文，将被 NetworkService 调用
};
```

接收方实现接受数据缓存后发送确认报文的功能，具体实现为:

函数名: void StopWaitRdtReceiver::receive(Packet &packet)

输入参数: 数据包

返回值: 无

算法描述:

```
If(校验和正确){  
    If(收到的序号不在接收窗口范围内)  
        忽略该分组;  
    Else{  
        确认序号等于收到的报文序号;  
        通过网络层发送确认报文到对方;  
        缓存报文;  
        从 rcvbase 去除连续的报文 Message, 向上递交给应用层;  
        从缓存中清除已提交的分组  
        更新 rcvbase  
    }  
}  
Else{  
    丢失该分组;  
    重发上次确认报文;  
}
```

(4) 验证结果

用 check_win 进行测试, 运行 10 次程序并比较两文件中数据, 结果如图 2.8 所示

```
Test "SR.exe" 4:  
正在比较文件 input.txt 和 OUTPUT.TXT  
FC: 找不到差异  
  
Test "SR.exe" 5:  
正在比较文件 input.txt 和 OUTPUT.TXT  
FC: 找不到差异  
  
Test "SR.exe" 6:  
正在比较文件 input.txt 和 OUTPUT.TXT  
FC: 找不到差异  
  
Test "SR.exe" 7:  
正在比较文件 input.txt 和 OUTPUT.TXT  
FC: 找不到差异  
  
Test "SR.exe" 8:  
正在比较文件 input.txt 和 OUTPUT.TXT  
FC: 找不到差异  
  
Test "SR.exe" 9:  
正在比较文件 input.txt 和 OUTPUT.TXT  
FC: 找不到差异  
  
Test "SR.exe" 10:  
正在比较文件 input.txt 和 OUTPUT.TXT  
FC: 找不到差异  
  
请按任意键继续. . .
```

图 2.8 SR 测试结果

多次运行信息传输无误, 实现了 SR 可靠信息传输。

(5) 结果分析

分析体现 SR 协议特点（选择重传）的地方如下：

发送方正确收到ack包: seqnum = -1, acknum = 7, checksum = 65529,

滑动窗口为: 5 6 7 0

窗口内等待的ack有: 5 0

发送定时器超时

重发上次发送的报文: seqnum = 5, acknum = -1, checksum = 16701, FFFFFFFFFFFFFFFFFFFFFFFF

发送定时器超时

重发上次发送的报文: seqnum = 0, acknum = -1, checksum = 8996, IFFFFFFFFFFFFFFFFF

图 2.9 SR 控制台信息输出

分析可知，每次定时器超时所导致的重传都只会重传定时器所对应的一个数据包。

2.3.3 简单 TCP/IP 协议的设计、验证及结果分析

(1) 总体设计

TCP 为了保证不发生丢包，就给每个包一个序号，同时序号也保证了传送到接收端实体的包的按序接收。然后接收端实体对已成功收到的包发回一个相应的确认（ACK）；如果发送端实体在合理的往返时延（RTT）内未收到确认，那么对应的数据包就被假设为已丢失将会被进行重传。TCP 用一个校验和函数来检验数据是否有错误；在发送和接收时都要计算校验和。本实验在实现 GBN 协议的基础上，根据 TCP 的可靠数据传输机制（包括超时后只重传最早发送且没被确认的报文、快速重传）只实现一个简化版的 TCP 协议。发送方和接收方的实现流程如下图：

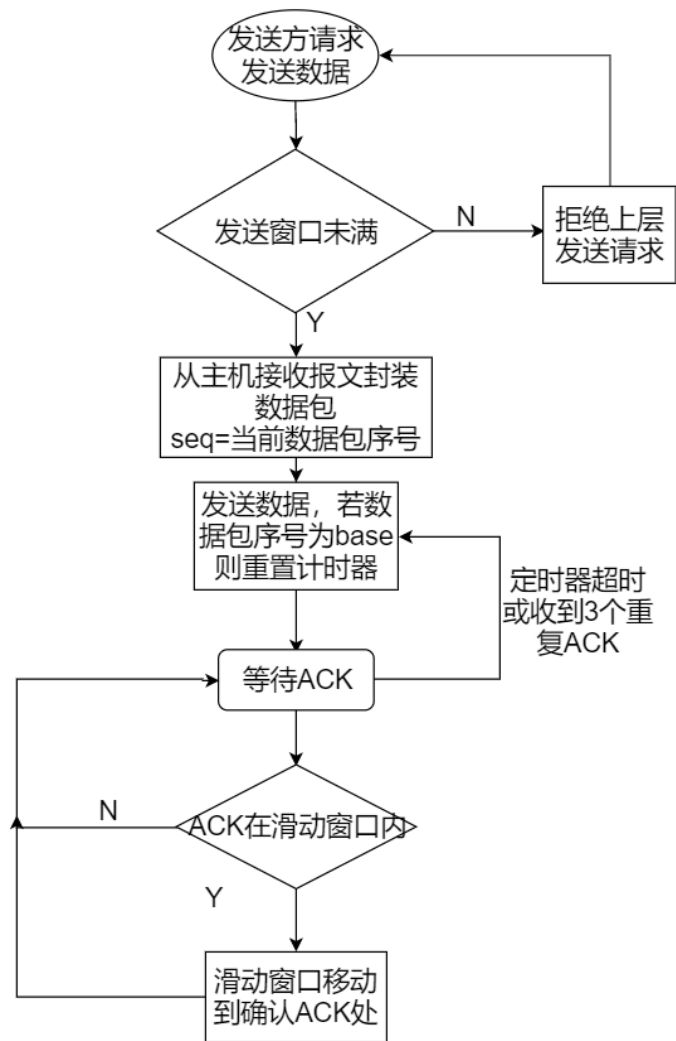


图 2.10 发送方处理流程

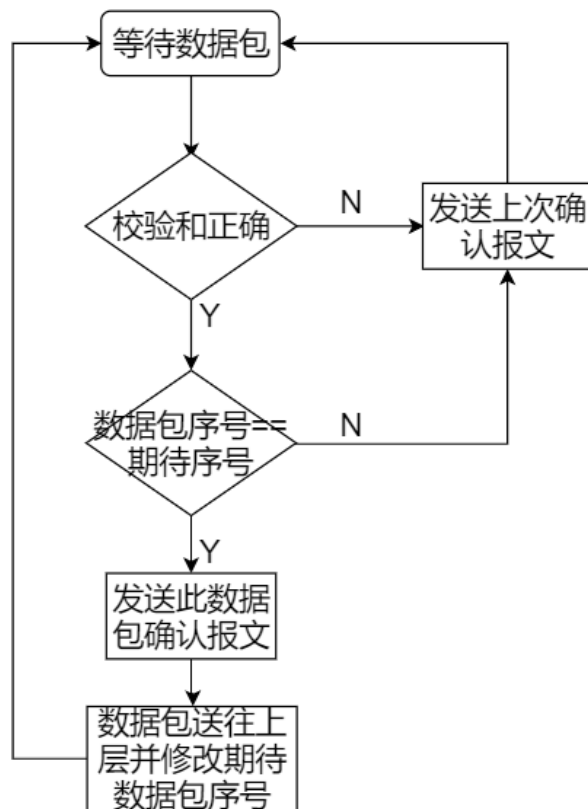


图 2.11 接收方处理流程

(2) 发送方实现

接口及函数定义：

```
#define WINDOW_SIZE 4//窗口大小
```

```
#define ORDER_SIZE 8//序号长度
```

```
class SRSEnder :public RdtSEnder
```

```
{
```

```
private:
```

```
    int base;                //发送窗口头
```

```
    int nextseqnum;          //下一个发送的序号
```

```
    list<Packet> sndpkt;     //已发送的数据包
```

```
    bool waitingState;       // 是否处于滑动窗口已满等待 Ack 的状态
```

```
    int repeat_count;        //记录 ack 的重复次数，用于快速重传
```

```
public:
```

```
    bool getWaitingState();
```

```
    bool send(const Message &message);           // 发 送 应 用 层 下 来 的
```

Message，由 NetworkServiceSimulator 调用,如果发送方成功地将 Message 发送到网络层，返回 true;如果因为发送方处于等待正确确认状态而拒绝发送 Message，则返回 false

```

        void receive(const Packet &ackPkt);                // 接 受 确 认 Ack ， 将 被
NetworkServiceSimulator 调用
        void timeoutHandler(int seqNum);                  //Timeout    handler    ，    将    被
NetworkServiceSimulator 调用

public:
    SRSender();
    virtual ~SRSender();
};

```

函数实现：

① 从应用层收到新数据分组

函数名：bool StopWaitRdtSender::send(Message &message)

入口参数：数据 Message

返回值：无

算法描述：

 If(发送方发送窗口已满)

 向上层发送拒绝信息后返回；

 将 nextseqnum 作为新数据分组序号；

 加入校验和；

 发送新数据分组；

 If（当前所发数据分组的序号==base）

 启动计时器；

 更新 nextseqnum

② 计时器超时

函数名：void StopWaitRdtSender::timeoutHandler(int seqNum)

入口参数：数据分组序号

返回值：无

算法描述：

 关闭计时器；

 重新启动计时器；

 重传 base 数据分组。

③ 收到确认分组

函数名: void StopWaitRdtSender::receive(Packet &ackPkt)

输入参数: 确认报文

返回值: 无

算法实现:

```
    If(校验和正确且)
    {
        If (ACK 在滑动窗口内且不为 base)
        {
            滑动窗口到 ACK 处;
            更新 base;
            将已确认分组从 sndpkt 中移出;
            If (nextseqnum == base)
                停止计时器;
            Else
                重启计时器;
            Repeat_count 清 0;
        }
        Else If (ACK==base)
        {
            Repeat_count 自增;
            If (Repeat==3)
            {
                重发 base 数据包;
                重启定时器;
                Repeat_count 清 0;
            }
        }
    }
    Else
        丢弃错误分组;
```

(3) 接收方实现

接口及函数定义:

```
#define WINDOW_SIZE 4 //接受窗口大小
```

```
#define ORDER_SIZE 8 //序号大小
```

```
class SRReceiver :public RdtReceiver
```

```

{
private:
    int expectSequenceNumberRcvd; // 期待收到的下一个报文序号
    Packet lastAckPkt;           //上次发送的确认报文
public:
    SRReceiver();
    virtual ~SRReceiver();
    void receive(const Packet &packet); //接收报文，将被 NetworkService 调用
};

```

接收方实现接受数据包校验后发送确认报文的功能，具体实现为：

函数名：void StopWaitRdtReceiver::receive(Packet &packet)

输入参数：数据包

返回值：无

算法描述：

```

    If(校验和正确，接受报文的序号与接收方期待序号一致){
        取出 Message，向上递交给应用层；
        确认序号等于收到的报文序号加一；
        添加校验和；
        发送确认报文；
        期待序号加一；
    }
    Else{
        丢失该分组；
        重发上次确认报文；
    }
}

```

(4) 验证结果

用 check_win 进行测试，运行 10 次程序并比较两文件中数据，结果如图 2.12 所示

```

Test "TCP.exe" 4:
正在比较文件 input.txt 和 OUTPUT.TXT
FC: 找不到差异

Test "TCP.exe" 5:
正在比较文件 input.txt 和 OUTPUT.TXT
FC: 找不到差异

Test "TCP.exe" 6:
正在比较文件 input.txt 和 OUTPUT.TXT
FC: 找不到差异

Test "TCP.exe" 7:
正在比较文件 input.txt 和 OUTPUT.TXT
FC: 找不到差异

Test "TCP.exe" 8:
正在比较文件 input.txt 和 OUTPUT.TXT
FC: 找不到差异

Test "TCP.exe" 9:
正在比较文件 input.txt 和 OUTPUT.TXT
FC: 找不到差异

Test "TCP.exe" 10:
正在比较文件 input.txt 和 OUTPUT.TXT
FC: 找不到差异

请按任意键继续. . .

```

图 2.12 TCP 测试结果

多次运行信息传输无误，实现了 TCP 可靠信息传输。

(5) 结果分析

分析体现 TCP 快速重传的地方如下：

```

发送方正正确收到ack包: seqnum = -1, acknum = 5, checksum = 12846, .....
接收方没有正确收到发送方的报文,报文序号不对: seqnum = 7, acknum = -1, checksum = 11559, HHHHHHHHHHHHHHHHHH
接收方重新发送上次的确认报文: seqnum = -1, acknum = 5, checksum = 12846, .....
发送定时器超时
重发报文: seqnum = 5, acknum = -1, checksum = 16701, FFFFFFFFFFFFFFFFFFFF

发送方正正确收到ack包: seqnum = -1, acknum = 5, checksum = 12846, .....
接收方没有正确收到发送方的报文,报文序号不对: seqnum = 0, acknum = -1, checksum = 8996, IHHHHHHHHHHHHHHHHH
接收方重新发送上次的确认报文: seqnum = -1, acknum = 5, checksum = 12846, .....

发送方正正确收到ack包: seqnum = -1, acknum = 5, checksum = 12846, .....

快速重传报文: seqnum = 5, acknum = -1, checksum = 16701, FFFFFFFFFFFFFFFFFFFF

接收方正正确收到发送方的报文: seqnum = 5, acknum = -1, checksum = 16701, FFFFFFFFFFFFFFFFFFFF

```

图 2.13 TCP 控制台信息输出

分析可知，发送方三次收到 ACK=5 后重传对应的 5 号数据包。

分析体现 TCP 累计确认的地方如下：

```
发送方正确收到ack包: seqnum = -1, acknum = 2, checksum = 12849, .....
滑动窗口为: 2 3 4
接收方正确收到发送方的报文: seqnum = 2, acknum = -1, checksum = 38549, WWWWWWWWWWWWWWWWWWWWWWWWW
*****模拟网络环境*****: 向上递交给应用层数据: WWWWWWWWWWWWWWWWWWWWWWWWW
接收方发送确认报文: seqnum = -1, acknum = 3, checksum = 12848, .....
接收方正确收到发送方的报文: seqnum = 3, acknum = -1, checksum = 35978, XXXXXXXXXXXXXXXXXXXXXXXX
*****模拟网络环境*****: 向上递交给应用层数据: XXXXXXXXXXXXXXXXXXXXXXXX
接收方发送确认报文: seqnum = -1, acknum = 4, checksum = 12847, .....

发送方正确收到ack包: seqnum = -1, acknum = 4, checksum = 12847, .....
滑动窗口为: 4
```

图 2.14 TCP 控制台信息输出

分析可知，发送方初始滑动窗口为 2，3，4，收到 ACK=4 后滑动窗口直接滑过了 3 号数据包变为 4。

2.4 其它需要说明的问题

无

实验三 基于 CPT 的组网实验

3.1 环境

开发环境: Cisco Packet Tracer

操作系统: Window 10

CPU: Inter-Core-i7-7500U;

RAM: 16G

3.2 实验要求

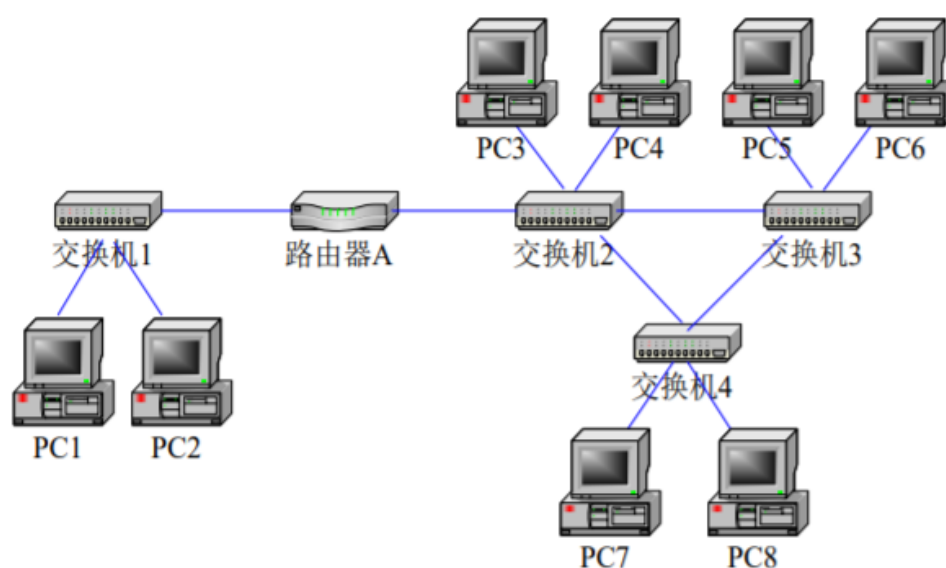


图 3.1 网络拓扑图

(1) 第一项实验——IP 地址规划与 Vlan 分配实验:

使用仿真软件描述网络拓扑图 3.1。

基本内容 1

将 PC1、PC2 设置在同一个网段, 子网地址是: 192.168.0.0/24;

将 PC3~PC8 设置在同一个网段, 子网地址是: 192.168.1.0/24;

配置路由器, 使得两个子网的各 PC 机之间可以自由通信。

基本内容 2

将 PC1、PC2 设置在同一个网段，子网地址是：192.168.0.0/24；
将 PC3、PC5、PC7 设置在同一个网段，子网地址是：192.168.1.0/24；
将 PC4、PC6、PC8 设置在同一个网段，子网地址是：192.168.2.0/24；
配置交换机 1、2、3、4，使得 PC1、PC2 属于 Vlan2，PC3、PC5、PC7 属于 Vlan3，
PC4、PC6、PC8 属于 Vlan4；
测试各 PC 之间的连通性，并结合所学理论知识进行分析；
配置路由器，使得拓扑图上的各 PC 机之间可以自由通信，结合所学理论对你的路由器配置过程进行详细说明。

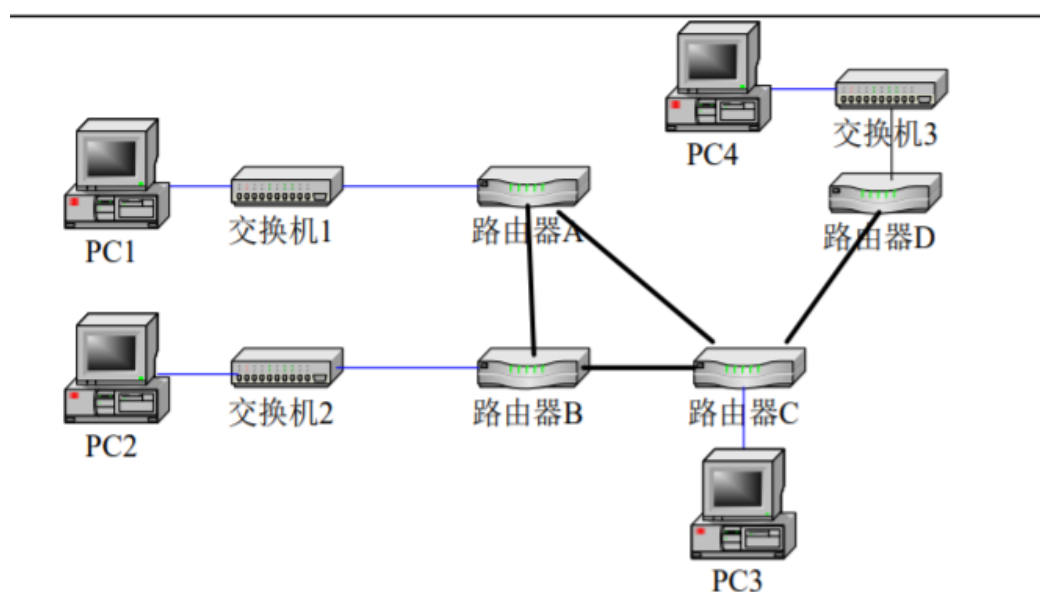


图 3.2 网络拓扑图

(2) 第二项实验——路由配置实验

使用仿真软件描述网络拓扑图 3.2

基本内容 1

将 PC1 设置在 192.168.1.0/24 网段；

将 PC2 设置在 192.168.2.0/24 网段；

将 PC3 设置在 192.168.3.0/24 网段；

将 PC4 设置在 192.168.4.0/24 网段

设置路由器端口的 IP 地址

在路由器上配置 RIP 协议，使各 PC 机能互相访问

基本内容 2

将 PC1 设置在 192.168.1.0/24 网段；

将 PC2 设置在 192.168.2.0/24 网段；

将 PC3 设置在 192.168.3.0/24 网段；

将 PC4 设置在 192.168.4.0/24 网段

设置路由器端口的 IP 地址

在路由器上配置 OSPF 协议，使各 PC 机能互相访问

基本内容 3

在基本内容 1 或者 2 的基础上，对路由器 1 进行访问控制配置，使得 PC1 无法访问其它 PC，也不能被其它 PC 机访问。

在基本内容 1 或者 2 的基础上，对路由器 1 进行访问控制配置，使得 PC1 不能访问 PC2，但能访问其它 PC 机

(3) 综合实验

实验背景：

某学校申请了一个前缀为 211.69.4.0/22 的地址块，准备将整个学校连入网络。该学校有 4 个学院，1 个图书馆，3 个学生宿舍。每个学院有 20 台主机，图书馆有 100 台主机，每个学生宿舍拥有 200 台主机。

组网需求：

图书馆能够无线上网

学院之间可以相互访问

学生宿舍之间可以相互访问

学院和学生宿舍之间不能相互访问

学院和学生宿舍皆可访问图书馆。

实验任务要求：

完成网络拓扑结构的设计并在仿真软件上进行绘制(要求具有足够但最少的设备，不需要考虑设备冗余备份的问题)

根据理论课的内容，对全网的 IP 地址进行合理的分配

在绘制的网络拓扑结构图上对各类设备进行配置，并测试是否满足组网需求，如有无法满足之处，请结合理论给出解释和说明

3.3 基本部分实验步骤说明及结果分析

3.3.1 IP 地址规划与 Vlan 分配实验的步骤及结果分析

(1) 基本内容一：

- ① 连接拓扑图如图 3.3，相同设备用交叉线连接，不同设备用直连线连接

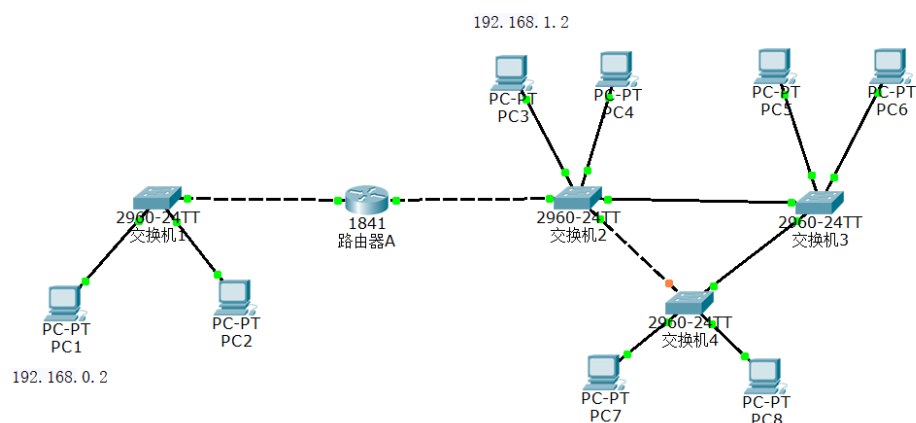


图 3.3 拓扑图

② 配置路由器和终端如下表

表 3.1 路由器和终端配置

设备	接口	IP 地址	子网掩码
ROUTER1	Fa0/0	192.168.0.1/24	255.255.255.0
	Fa0/1	192.168.1.1/24	
PC1		192.168.0.2/24	255.255.255.0
PC2		192.168.0.3/24	255.255.255.0
PC3		192.168.1.2/24	255.255.255.0
PC4		192.168.1.3/24	255.255.255.0
PC5		192.168.1.4/24	255.255.255.0
PC6		192.168.1.5/24	255.255.255.0
PC7		192.168.1.6/24	255.255.255.0
PC8		192.168.1.7/24	255.255.255.0

③ 测试连通性

各 PC 间皆可自由通信

PC1→PC2

	成功	PC1	PC2	ICMP		0.000	N	0
--	----	-----	-----	------	--	-------	---	---

PC1→PC3

	失败	PC1	PC3	ICMP		0.000	N	1	(3)
	成功	PC1	PC3	ICMP		0.000	N	2	(3)

PC3→PC4

	成功	PC3	PC4	ICMP		0.000	N	3	(3)
--	----	-----	-----	------	--	-------	---	---	-----

PC3→PC1

由分析可知，同一子网 PC 可以通信，不同子网间 PC 也可以通信，因为两个不同子网使用的是同一路由器的两个不同接口。

(2) 基本内容二：

- ① 连接拓扑图，外观同图 3.3
- ② 划分路由器和交换机、配置终端的网络地址
- ③ 划分 Vlan，将 PC 的接口划分到不同的 VLAN 上
- ④ 配置路由器与交换机、交换机与交换机间的通路为 trunk 模式
- ⑤ 配置路由器 VLAN：由于路由器仅通过接口 Fa0/1 与交换机 2 相连，而该接口实际上连接了两个 VLAN，因此需要针对两个 VLAN 创建两个子接口：Fa0/1.1 和 Fa0/1.2，
- ⑥ 配置路由器和终端如下表

表 3.2 网络配置

设备	接口	网络地址	掩码	VLAN
ROUTER1	Fa0/0.1	192.168.0.1/24	255.255.255.0	
	Fa0/1.1	192.168.1.1/24	255.255.255.0	
	Fa0/1.2	192.168.2.1/24	255.255.255.0	
SWITCH1	Fa0/2			VLAN2
	Fa0/3			VLAN2
SWITCH2	Fa0/1			VLAN3
	Fa0/3			VLAN4
SWITCH3	Fa0/2			VLAN3
	Fa0/3			VLAN4
SWITCH4	Fa0/1			VLAN3
	Fa0/2			VLAN4
PC1		192.168.0.2/24	255.255.255.0	
PC2		192.168.0.3/24	255.255.255.0	
PC3		192.168.1.2/24	255.255.255.0	
PC4		192.168.2.2/24	255.255.255.0	
PC5		192.168.1.3/24	255.255.255.0	
PC6		192.168.2.3/24	255.255.255.0	
PC7		192.168.1.4/24	255.255.255.0	
PC8		192.168.2.4/24	255.255.255.0	

⑦ 连通性测试

PC 间皆可自由通信

PC1→PC2

PC1→PC3

失败	PC1	PC3	ICMP	0.000	N	1	(编辑)
成功	PC1	PC3	ICMP	0.000	N	2	(编辑)

PC3→PC1

成功	PC3	PC1	ICMP	0.000	N	0	(编辑)
----	-----	-----	------	-------	---	---	------

PC3→PC4

失败	PC3	PC4	ICMP	0.000	N	0	(编辑)
成功	PC3	PC4	ICMP	0.000	N	1	(编辑)

PC3→PC7

成功	PC3	PC7	ICMP	0.000	N	0	(编辑)
----	-----	-----	------	-------	---	---	------

分析可知，PC 间皆可自由通信。

3.3.2 路由配置实验的步骤及结果分析

(1) 基本内容 1

① 如图 3.4 所示，合理选用线路、分配接口，连接拓扑图

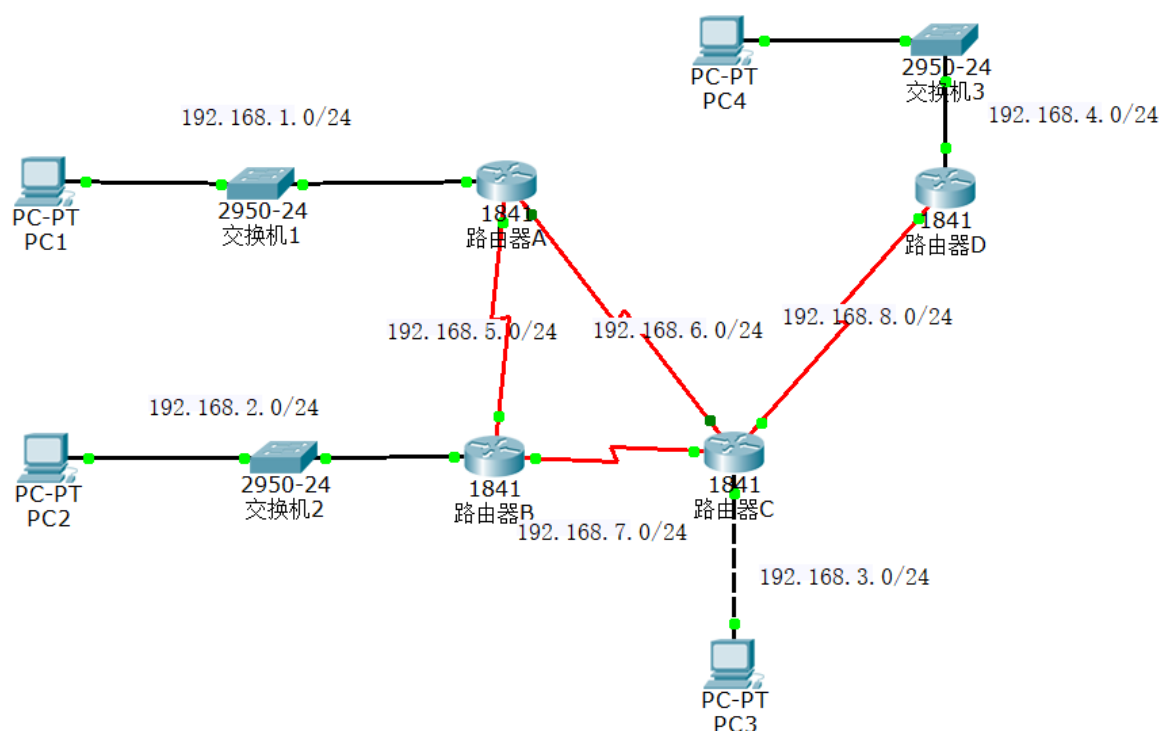


图 3.4 拓扑图

② 规划终端和路由器 IP 地址

表 3.3 网络配置表

设备	接口	网络地址	掩码
ROUTERA	Fa0/0	192. 168. 1. 1/24	255. 255. 255. 0
	Se0/0/0	192. 168. 5. 1/24	255. 255. 255. 0
	Se0/0/1	192. 168. 6. 1/24	255. 255. 255. 0
ROUTERB	Fa0/0	192. 168. 2. 1/24	255. 255. 255. 0
	Se0/0/0	192. 168. 5. 2/24	255. 255. 255. 0
	Se0/0/1	192. 168. 7. 1/24	255. 255. 255. 0
ROUTERC	Fa0/0	192. 168. 3. 1/24	255. 255. 255. 0
	Se0/0/0	192. 168. 6. 2/24	255. 255. 255. 0
	Se0/1/0	192. 168. 7. 2/24	255. 255. 255. 0
	Se0/1/1	192. 168. 8. 2/24	255. 255. 255. 0
ROUTERD	Fa0/0	192. 168. 4. 1/24	255. 255. 255. 0
	Se0/0/1	192. 168. 8. 2/24	255. 255. 255. 0
PC1		192. 168. 1. 2/24	255. 255. 255. 0
PC2		192. 168. 2. 2/24	255. 255. 255. 0
PC3		192. 168. 3. 2/24	255. 255. 255. 0
PC4		192. 168. 4. 2/24	255. 255. 255. 0

③ 路由器配置 RIP 协议

表 3.3 路由器配置 RIP

路由器	RIP routing
ROUTERA	192.168.1.0 192.168.5.0 192.168.6.0
ROUTERB	192.168.2.0 192.168.5.0 192.168.7.0
ROUTERC	192.168.3.0 192.168.6.0 192.168.7.0 192.168.8.0
ROUTERD	192.168.4.0 192.168.8.0

④ 连通性测试

各 PC 间皆可通信

PC1->PC2

	失败	PC1	PC2	ICMP		0.000	N	0	(编辑)
	成功	PC1	PC2	ICMP		0.000	N	1	(编辑)

PC1→PC3

	失败	PC1	PC3	ICMP		0.000	N	2	(编辑)
	成功	PC1	PC3	ICMP		0.000	N	3	(编辑)

PC1→PC4

	失败	PC1	PC4	ICMP		0.000	N	0	(编辑)
	成功	PC1	PC4	ICMP		0.000	N	1	(编辑)

PC2→PC3

	成功	PC2	PC3	ICMP		0.000	N	0	(编辑)
--	----	-----	-----	------	--	-------	---	---	------

PC2→PC4

	成功	PC2	PC4	ICMP		0.000	N	2	(编辑)
--	----	-----	-----	------	--	-------	---	---	------

PC3→PC4

	成功	PC3	PC4	ICMP		0.000	N	3	(编辑)
--	----	-----	-----	------	--	-------	---	---	------

由分析知，配置了 RIP 后，不同的子网的主机之间可以相互 ping 通。

(2) 基本内容 2

- ① 配置拓扑图、路由器和终端 IP 与基本内容 2 相同
- ② 配置 OSPF 协议

配置完成的 OSPF 如下表所示

表 3.4 OSPF 协议配置





路由器	OSPF neighbor						
ROUTERA	Neighbor ID	Pri	State	Dead Time	Address	Interface	
	192.168.7.1	0	FULL/ -	00:00:39	192.168.5.2	Serial0/0/0	
	192.168.8.2	0	FULL/ -	00:00:33	192.168.6.2	Serial0/0/1	
ROUTERB	Neighbor ID	Pri	State	Dead Time	Address	Interface	
	192.168.6.1	0	FULL/ -	00:00:36	192.168.5.1	Serial0/0/0	
	192.168.8.2	0	FULL/ -	00:00:35	192.168.7.2	Serial0/0/1	

ROUTERC	Neighbor ID	Pri	State	Dead Time	Address	Interface
	192.168.8.1	0	FULL/ -	00:00:31	192.168.8.1	Serial0/1/1
	192.168.6.1	0	FULL/ -	00:00:33	192.168.6.1	Serial0/0/0
	192.168.7.1	0	FULL/ -	00:00:33	192.168.7.1	Serial0/1/0
ROUTERD	Neighbor ID	Pri	State	Dead Time	Address	Interface
	192.168.8.2	0	FULL/ -	00:00:37	192.168.8.2	Serial0/0/0





③ 连通性测试

各 PC 间皆可通信





PC1→PC2

	失败	PC1	PC2	ICMP		0.000	N	0	(编辑)
	成功	PC1	PC2	ICMP		0.000	N	1	(编辑)



PC1→PC3

	失败	PC1	PC3	ICMP		0.000	N	0	(编辑)
	成功	PC1	PC3	ICMP		0.000	N	1	(编辑)


PC1→PC4

	失败	PC1	PC4	ICMP		0.000	N	0	(编辑)
	成功	PC1	PC4	ICMP		0.000	N	1	(编辑)



PC2→PC3

	成功	PC2	PC3	ICMP		0.000	N	0	
---	----	-----	-----	------	--	-------	---	---	--

PC2→PC4

	成功	PC2	PC4	ICMP		0.000	N	0	
---	----	-----	-----	------	--	-------	---	---	--

PC3→PC4

	成功	PC3	PC4	ICMP		0.000	N	0	
---	----	-----	-----	------	--	-------	---	---	--

由分析知，配置了 OSPF 后，不同的子网的主机之间可以相互 ping 通。

(3) 基本内容 3.1

- ① 配置拓扑图、路由器和终端 IP 与基本内容 2 相同
- ② 对路由器 A 进行 ACL 配置如图所示，使得 PC1 无法访问其它 PC，也不能被其它 PC 机访问。

```

Router#show ip access
Extended IP access list acl_1
 10 deny ip any 192.168.1.0 0.0.0.255
 20 deny ip 192.168.1.0 0.0.0.255 any
 30 permit ip any any





```

图 3.5 路由器 A 的 ACL 配置





③ 连通性测试

PC1 无法访问其它 PC，也不能被其它 PC 机访问；PC2，PC3，PC4 可以互相访问





PC1→PC2

	失败	PC1	PC2	ICMP		0.000	N	0	(编辑)
	失败	PC1	PC2	ICMP		0.000	N	1	(编辑)





PC1→PC3

	失败	PC1	PC3	ICMP		0.000	N	0	(编辑)
	失败	PC1	PC3	ICMP		0.000	N	1	(编辑)





PC1→PC4

	失败	PC1	PC4	ICMP		0.000	N	0	(编辑)
	失败	PC1	PC4	ICMP		0.000	N	1	(编辑)





PC2→PC1

	失败	PC2	PC1	ICMP		0.000	N	0	(编辑)
	失败	PC2	PC1	ICMP		0.000	N	1	(编辑)



PC3→PC1

	失败	PC3	PC1	ICMP		0.000	N	0	(编辑)
	失败	PC3	PC1	ICMP		0.000	N	1	(编辑)



PC4→PC1

	失败	PC4	PC1	ICMP		0.000	N	0	(编辑)
	失败	PC4	PC1	ICMP		0.000	N	1	(编辑)



PC2→PC3

	成功	PC2	PC3	ICMP		0.000	N	0	
---	----	-----	-----	------	--	-------	---	---	--

PC2->PC4

	成功	PC2	PC4	ICMP		0.000	N	0
---	----	-----	-----	------	---	-------	---	---

PC3->PC4

	成功	PC3	PC4	ICMP		0.000	N	0
---	----	-----	-----	------	---	-------	---	---

由分析知，配置了 ACL 后，PC1 无法访问其它 PC，也不能被其它 PC 机访问；PC2，PC3，PC4 可以互相访问。

(4) 基本内容 3.2





- ① 配置拓扑图、路由器和终端 IP 与基本内容 2 相同
- ② 对路由器 A 进行 ACL 配置如图所示，使得 PC1 不能访问 PC2，但能访问其它 PC 机。

```
Router#show ip access
Extended IP access list acl_1
  10 deny ip host 192.168.1.2 host 192.168.2.2
  20 permit ip any any
Extended IP access list acl_2
  10 deny ip host 192.168.2.2 host 192.168.1.2
  20 permit ip any any
```





图 3.6 路由器 A 的 ACL 配置

③ 连通性测试





PC1->PC2

	失败	PC1	PC2	ICMP		0.000	N	0	(编辑)
	失败	PC1	PC2	ICMP		0.000	N	1	(编辑)

PC1->PC3

	失败	PC1	PC3	ICMP		0.000	N	2	(编辑)
	成功	PC1	PC3	ICMP		0.000	N	3	(编辑)

PC1->PC4

	失败	PC1	PC4	ICMP		0.000	N	0	(编辑)
	成功	PC1	PC4	ICMP		0.000	N	1	(编辑)

由分析知，配置了 ACL 后 PC1 不能访问 PC2，但能访问其它 PC 机。

3.4 综合部分实验设计、实验步骤及结果分析

3.4.1 实验设计

(1) IP 地址划分

图书馆有 100 台主机，网络前缀长度为 25，掩码为 255.255.255.128

每个学生宿舍有 200 台主机，网络前缀长度为 24，掩码为 255.255.255.0

每个学院有 20 台主机，网络前缀长度为 27，掩码为 255.255.255.224

表 3.5 IP 地址分配

地点	IP 网络前缀	掩码
图书馆	211.69.4.129/25	255.255.255.128
宿舍 1	211.69.5.1/24	255.255.255.0
宿舍 2	211.69.6.1/24	255.255.255.0
宿舍 3	211.69.7.1/24	255.255.255.0
学院 A	211.69.4.1/26	255.255.255.224
学院 B	211.69.4.33/26	255.255.255.224
学院 C	211.69.4.65/26	255.255.255.224
学院 D	211.69.4.97/26	255.255.255.224

(2) 路由器 IP 分配、OSPF 配置及 ACL 配置

表 3.6 路由器设置

路由器	接口	IP 地址	OSPF neighbor	ACL 配置
Router0	Fa0/0 Se0/0/0	211.69.5.1 192.168.2.1	Neighbor ID 211.69.4.129	Standard IP access list 10 deny 211.69.4.0 0.0.0.127 permit any
Router1	Fa0/0 Se0/0/0	211.69.6.1 192.168.3.1	Neighbor ID 211.69.4.129	Standard IP access list 10 deny 211.69.4.0 0.0.0.127 permit any
Router2	Fa0/0 Se0/0/0	211.69.7.1 192.168.4.1	Neighbor ID 211.69.4.129	Standard IP access list 10 deny 211.69.4.0 0.0.0.127 permit any
Router3	Fa0/0 Se0/0/0 Se0/0/1 Se0/1/0 Se0/1/1	211.69.4.128 192.168.2.1 192.168.3.1 192.168.4.1 192.168.1.1	Neighbor ID 211.69.4.1 211.69.5.1 211.69.6.1 211.69.7.1	无
Router4	Fa0/0.1 Fa0/0.2 Fa0/0.3 Fa0/0.4 Se0/0/0	211.69.4.1 211.69.4.33 211.69.4.65 211.69.4.97 192.168.1.1	Neighbor ID 211.69.4.129	Standard IP access list 10 deny 211.69.5.0 0.0.0.255 deny 211.69.6.0 0.0.0.255 deny 211.69.7.0 0.0.0.255 permit any

3.4.2 实验步骤

(1) 连接拓扑图，合理分配接口，连接线路如下图

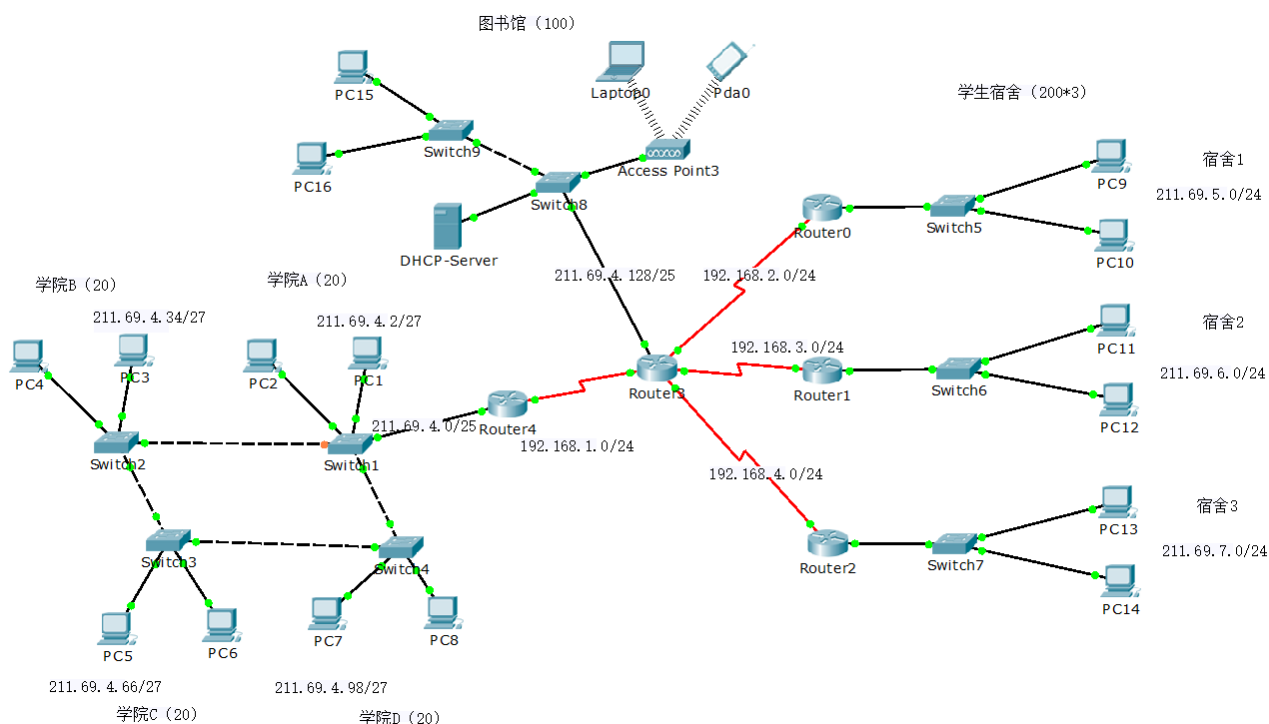


图 3.7 拓扑图

- (2) 根据表 3.5，为 PC 和路由器分配 IP 地址及掩码
- (3) 交换机接口划分 VLAN
- (4) 配置交换机间及与路由器间 trunk 链路
- (5) 根据表 3.5 和 3.6，为路由器划分 VLAN 并指定接口
- (6) 根据表 3.6，为路由器配置 OSPF 协议
- (7) 根据表 3.6，对路由器进行 ACL 配置。
- (8) 配置图书馆 DHCP 服务器如下

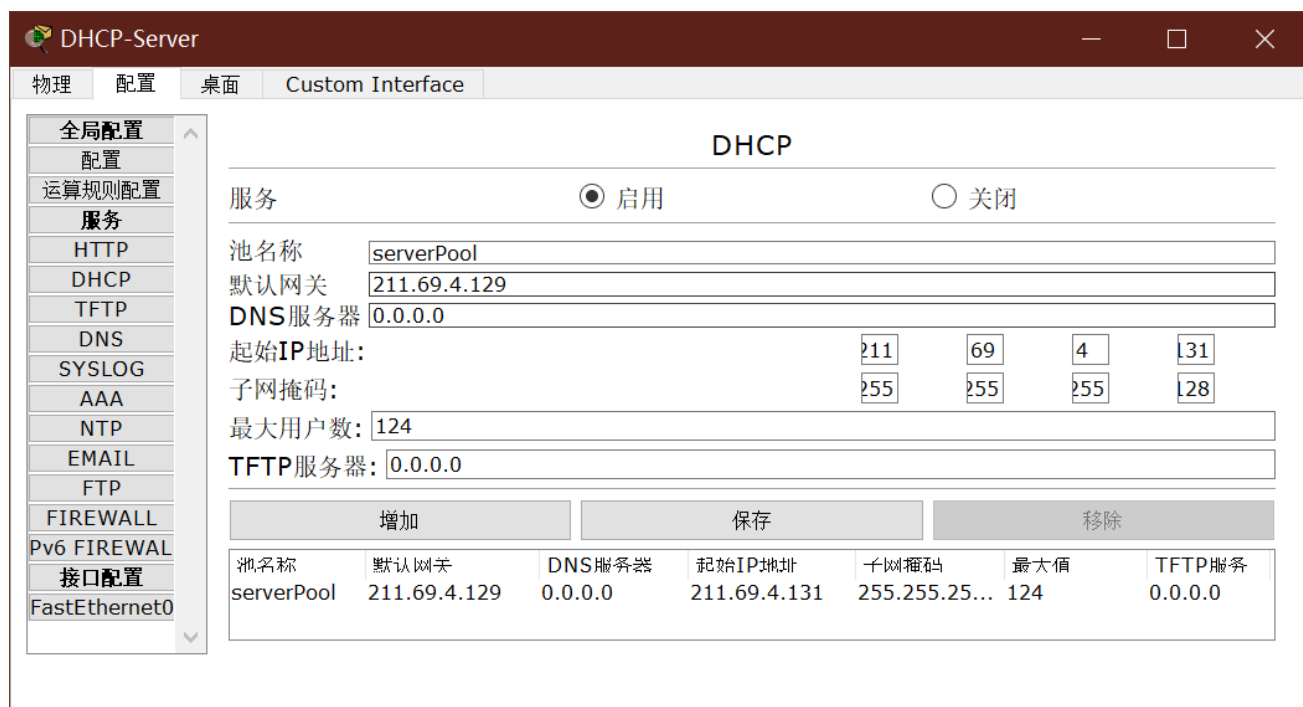


图 3.7 图书馆 DHCP 服务器

3.4.3 结果分析

学院 A ping 学院 D（学院之间可以相互访问）

	失败	PC1	PC7	ICMP		0.000	N	0	(编辑)
	成功	PC1	PC7	ICMP		0.000	N	1	(编辑)

宿舍 1 ping 宿舍 2（学生宿舍之间可以相互访问）

	失败	PC9	PC11	ICMP		0.000	N	2	(编辑)
	成功	PC9	PC11	ICMP		0.000	N	3	(编辑)

学院 A ping 宿舍 2（学院不能访问学生宿舍）

	失败	PC1	PC11	ICMP		0.000	N	0	(编辑)
	失败	PC1	PC11	ICMP		0.000	N	1	(编辑)







宿舍 3 ping 学院 B（学生宿舍不能访问学院）

	失败	PC13	PC3	ICMP		0.000	N	0	(编辑)
	进行中	PC13	PC3	ICMP		0.000	N	1	(编辑)

学院 C ping 图书馆（学院可以访问图书馆）

	失败	PC5	PC15	ICMP		0.000	N	2	(编辑)
	成功	PC5	PC15	ICMP		0.000	N	3	(编辑)

宿舍 2 ping 图书馆（学生宿舍可以访问图书馆）

	成功	PC11	PC15	ICMP		0.000	N	0	(编辑)
图书馆 ping 无线（图书馆可以无线上网）									
	成功	Laptop0	PC15	ICMP		0.000	N	0	(编辑)
	成功	PC15	Laptop0	ICMP		0.000	N	1	(编辑)

经以上分析，图书馆能够无线上网；学院之间可以相互访问；学生宿舍之间可以相互访问；学院和学生宿舍之间不能相互访问；学院和学生宿舍皆可访问图书馆，满足实验要求。

3.5 其它需要说明的问题

无

心得体会与建议

4.1 心得体会

通过三次实验，对于计算机网络有了整体上的把握，对于各种信息的传输方式有了较为细致的了解，对于实际中的配置操作也掌握较好。实验中碰到了很多困难，在这个过程中学习了大量新的知识，自己的编程能力和解决问题的能力有了较大的提升，最终较为理想地完成了实验内容。

第一次实验为 socket 实验，要求实现 web 服务器处理浏览器请求，最初对于整个的处理流程不是很了解，就先实现了一个服务器和客户端之间的交互聊天室，到以异步的形式处理请求。之后了解报文格式后再解析 HTTP 请求报文，了解长连接的建立过程和保持，失效连接的处理，各种格式文件的发送，状态码的解析，通过一个一个模块的实现最终构成了完整的异步 Web 请求服务器。但是之后把整个项目向 QT 上迁移的时候碰到了预期之外的问题，界面体验不友好，造成了程序的不稳定，最终由于时间问题放弃了 GUI，有些许遗憾，设计程序的整体流程还需要改进。第一个实验很好的体现出了自顶向下的思想，与浏览器请求的交互处理更是需要严谨实现，一些小 bug 往往会导致整个程序的崩溃，对我们编写程序有较高的要求。

第二次实验设计数据传输的过程，分别以 GBN, SR, TCP 对数据实现可靠传输，我对于不同协议的传输过程有了细致的了解，也更能理解在传输过程中对于数据可靠性的要求。

第三次实验 TCP 组网，感觉比前两次实验更加具有实际应用意义，对于 IP 协议、网络层协议和数据链路层协议的工作原理及机制有了一定的了解，对于一些常用的组网技术如 VLAN 划分、ACL 过滤等掌握了配置方式，学会了路由协议的配置方法、路由器及二/三层交换机的配置方法。在最后的综合实验中，对于整个网络 IP 地址进行划分并一步步实现不同的组网要求，实践了上述方法，并在整体上对于组网有了一定的认识。

总的来说，三次网络实验是有一定难度的，主要体现在：

1. 新工具的使用

第一次实验要求写 GUI 界面，一般会选择 QT 作为开发界面的工具，但我对 QT 的了解有限，通过短时间的学习也仅停留在写一些小的界面程序的层次，拿来做 Web 服务器多线程和异步开发有一定的难度。

2. 陌生的开发情境

本质上来说这是第一次根据完整的开发文档写程序，对于很多没有样例示范的地方理解上可能存在一定的偏差，这些不太明白的地方往往会成为程序中隐藏的 BUG，调试起来也较为困难；短时间内接触大量新的东西并投入实际应用有些接受不了。比如实

验一中各种函数文档，实验二的模拟环境，实验三的诸多命令，在刚开始接触的时候带给了我一些焦虑，经过一段时间的适应才静下来慢慢的去实践。

3. 理论知识和实际的偏离

对于书本上的内容有了较好的掌握，但是实际写程序的时候还是会有一些难以下手的感觉，很多地方仅仅停留在知识层面，没有形成完整的体系，因此刚时候有一种对着书本知识点写代码翻译的感觉，无法正常运行的程序也难以定位到错误，几次实验下来才对于整个数据传输的过程有了较好的理解。另一方面，实际实现的时候考虑到实际需要，实现方式可能于书本上的理论知识有出入，这种时候有需要根据实际进行调整。

4.2 建议

1. 希望实验的开始能有较为细致的指导，帮助同学们少走弯路，节约时间，快速入门。
2. 希望各科实验的时间安排能够考虑到时间冲突，这个学期由于几个大课程设计安排在了一起，时间太紧张。建议实验可以早一些开设或者跟着课程的进度走。
3. 计网实验不一定要局限于 c++ 开发，其他一些语言或许也具有很好的实践性。

参考文献

[1] □James F. Kurose Keith W. Ross. 计算机网络-自顶向下方法. (第七版). 机械工业出版社, 2018-9

[2] □Kevin R. Fall W. Richard Stevens. TCP/IP 详解 卷 1: 协议. (第二版). 机械工业出版社, 2016-6-23