
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 CS2203 姓名 王国豪 学号 U202215643

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

课程目标	1 (20%)	2 (15%)	3 (10%)	4 (5%)	5 (15%)	6 (20%)	7 (15%)	总分
得分								

教师评语：

教师签名：

给分日期：

目 录

实验一 SOCKET 编程实验	1
1.1 环境	1
1.2 系统功能需求	1
1.3 系统设计	2
1.4 系统实现	5
1.5 系统测试及结果说明	6
1.6 其它需要说明的问题	8
1.7 参考文献	8
实验二 数据可靠传输协议设计实验	9
2.1 环境	9
2.2 实验要求	9
2.3 协议的设计、验证及结果分析	9
2.4 其它需要说明的问题	19
2.5 参考文献	19
实验三 基于 CPT 的组网实验	20
3.1 环境	20
3.2 实验要求	20
3.3 基本部分实验步骤说明及结果分析	20
3.4 综合部分实验设计、实验步骤及结果分析	26
3.5 其它需要说明的问题	28
3.6 参考文献	28
心得体会与建议	29
4.1 心得体会	29
4.2 建议	29

实验一 Socket 编程实验

1.1 环境

1.1.1 开发平台

处理器：11th Gen Intel(R) Core(TM) i5-11260H @ 2.60GHz 2.61 GHz

操作系统：Windows 11

开发平台：VisualStudio 2022

开发语言：C/C++

1.1.2 运行平台

处理器：11th Gen Intel(R) Core(TM) i5-11260H @ 2.60GHz 2.61 GHz

操作系统：Windows 11

运行软件：VisualStudio 2022

1.2 系统功能需求

基本要求：

1. 可配置 Web 服务器的监听地址、监听端口和主目录（不得写在代码里面，不能每配置一次都要重编译代码）；
2. 能够单线程处理一个请求。当一个客户（浏览器，输入 URL：
http://202.103.2.3/index.html）连接时创建一个连接套接字；
3. 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；
4. 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；
5. 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。

高级要求：

1. 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；
2. 在服务器端的屏幕上输出请求的来源（IP 地址、端口号和 HTTP 请求命令行）；
3. 在服务器端的屏幕上能够输出对每一个请求处理的结果；
4. 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，并具备一定的异常情况处理能力。

1.3 系统设计

系统框架设计：

本系统基于 TCP/IP 协议 实现了一个简易的客户端-服务器通信框架，使用 Windows 套接字（Winsock2）库进行网络通信。系统的主要功能是通过 HTTP 协议 使客户端与 Web 服务器之间实现数据交互。考虑到开发过程中的调试和错误排查，本系统采用阻塞式 I/O 作为传输方式，既方便开发过程中的调试，又能保证数据的可靠传输，如图 1.1。

我们要实现的其实是一个 web 服务器，和客户机一样，都需要创建一个数据报的套接字，，服务器调用 bind（）函数给套接字分配一个公认的端口。一旦服务器将公认的端口分配给了套接字，客户机和服务器都能使用 sendto（）和 revfron（）来传递数据报。通信完毕调用 closesocket（）来关闭套接字。

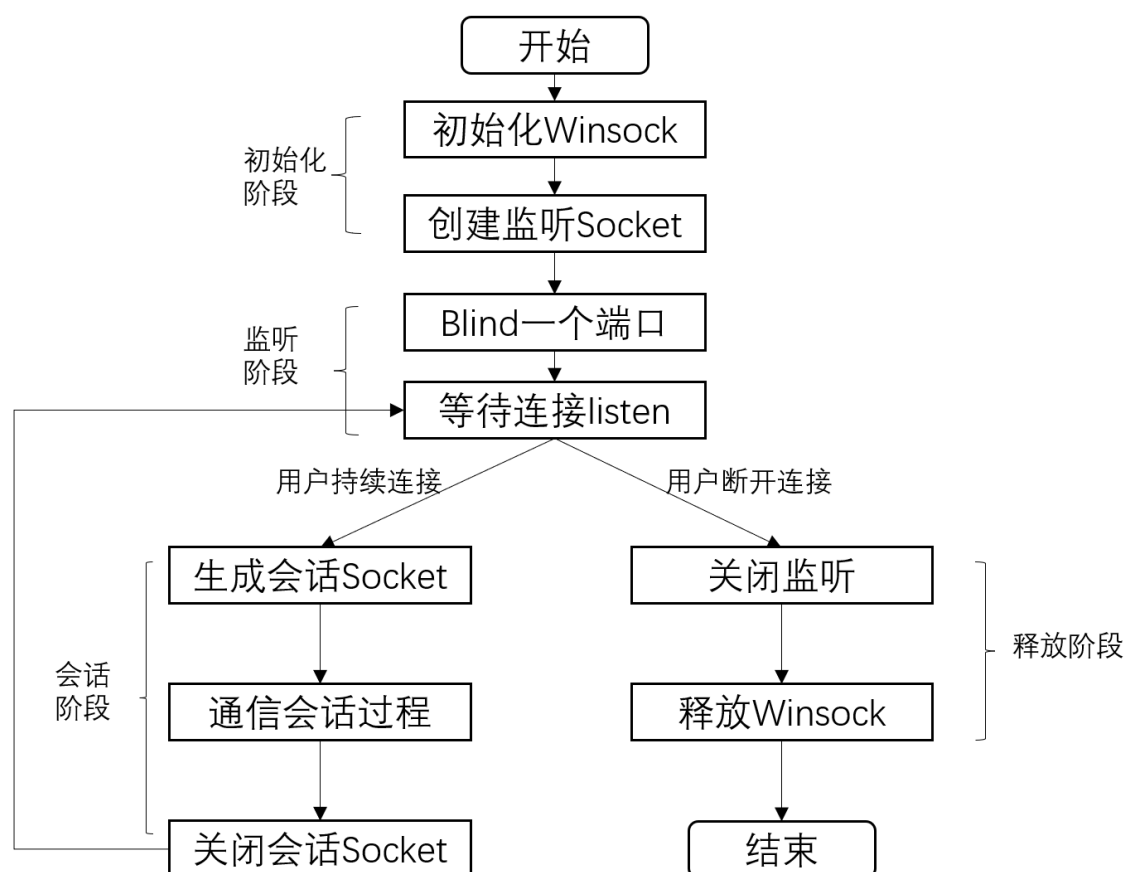


图 1.1 web 服务器系统框图

功能模块划分：

系统分为初始化模块、监听模块、会话模块和释放模块，各模块进一步分为多个具体过程以完成对应任务。

初始化模块：

下分 Winsock 初始化和监听 Socket 创建两个过程。

(1) Winsock 初始化：

使用 WSStartup 初始化 Winsock，并检查 Winsock 的版本号。若版本号不匹配或初始化失败，调用 WSAGetLastError 获取错误码并退出系统。输出日志信息确认 Winsock 初始化成功。

(2) 监听 Socket 创建：

使用 socket 函数创建一个监听 Socket。配置为支持 IPv4 (AF_INET)，使用流类型通信 (SOCK_STREAM) 和 TCP 协议 (IPPROTO_TCP)。若创建失败，调用 WSAGetLastError 显错误并退出。日志记录监听 Socket 的创建状态。

监听模块：

下分配置监听设置、监听绑定和等待连接状态设置三个过程。

(1) 配置监听设置

用户输入监听地址 (IP)、端口号以及提供服务的主目录路径。将输入数据写入 sockaddr_in 结构体，便于绑定操作。检查输入的地址与端口号是否合法，并在日志中记录。

(2) 监听绑定

使用 bind 函数将监听地址和端口绑定到监听 Socket。若绑定失败，调用 WSAGetLastError 显示错误信息并退出。成功绑定后，在日志中记录绑定的地址和端口号。

(3) 设置等待连接状态

使用 listen 函数设置监听 Socket 为等待连接状态。配置最大排队连接数 (例如 20)。若设置失败，调用 WSAGetLastError 显示错误信息并退出。记录监听 Socket 成功进入等待状态。

会话模块：

下分建立连接、请求数据处理、响应数据处理和关闭会话 Socket 四个过程

(1) 建立连接

使用 accept 函数接受客户端连接请求，生成会话 Socket。检查返回的会话 Socket 是否有效。记录客户端的 IP 地址和端口号。若建立连接失败，调用 WSAGetLastError 显示错误信息并继续监听其他连接。

(2) 请求数据处理

使用 recv 函数接收客户端的请求报文。对请求报文进行解析，提取请求的方法、路径和参数。

（3） 响应数据处理

检查解析出的文件路径，判断文件是否存在。文件不存在：返回 404 Not Found 状态码和对应 HTML 页面。权限不足：返回 403 Forbidden 状态码。文件存在：将文件以二进制形式打开，存入缓冲区。根据文件类型设置 Content-Type（如 text/html, image/png）。构建完整的 HTTP 响应报文，通过 send 函数发送给客户端。记录发送成功的字节数和响应状态。

（4） 关闭会话 Socket

在响应发送完成后，调用 closesocket 关闭会话 Socket。在日志中记录会话结束状态。释放模块：

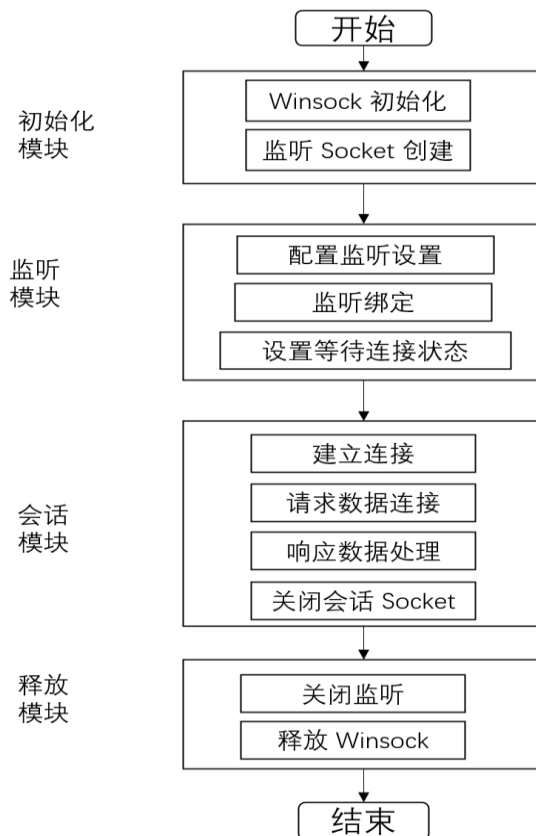
下分关闭监听和释放 Winsock 两个过程。

（1） 关闭监听

当客户端断开连接或程序终止时，调用 closesocket 关闭监听 Socket。确保监听 Socket 被正确释放，避免资源泄露。

（2） 释放 Winsock

调用 WSACleanup 释放 Winsock 库分配的资源。检查是否有未释放的 Socket，确保资源清理完整。日志记录释放状态。



图表 1.2 功能模块设计图

1.4 系统实现

下面将针对实验中实现系统的重难点技术进行分析。

1. 解析请求报文

如何在请求报文中将请求行中的 url 解析出来呢，我们注意到 url 前面和后面都有空格，我们可以先将这两个空格的位置找出，再利用 substr 函数即可解析出 url，与我们输入的主目录结合，构成请求报文的请求文件路径，请求报文见图 1.3。

```
GET /introduce.html HTTP/1.1 请求行
Host: 127.0.0.1:8080
Connection: keep-alive
Cache-Control: max-age=0
sec-ch-ua: "Microsoft Edge";v="131", "Chromium";v="131", "Not_A Brand";v="24"
sec-ch-ua-mobile: ?0
sec-ch-ua-platform: "Windows"
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/136.0.0.0 Edg/131.0.0.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,image/svg+xml,*/*;q=0.8
Accept-Encoding: gzip, deflate, br, zstd
Accept-Language: zh-CN,zh;q=0.9,en;q=0.8,en-GB;q=0.7,en-US;q=0.6
Cookie: _ga=GA1.1.2144652240.1706687523; _ga_69MPZE94D5=GS1.1.1706687523.1.1.1706689261.0.0.0
```

图 1.3 请求报文截图

2. 构造响应报文

基本的构造响应报文的过程分为判断文件是否能够打开，判断文件的扩展名，构建对应的响应报文内容。

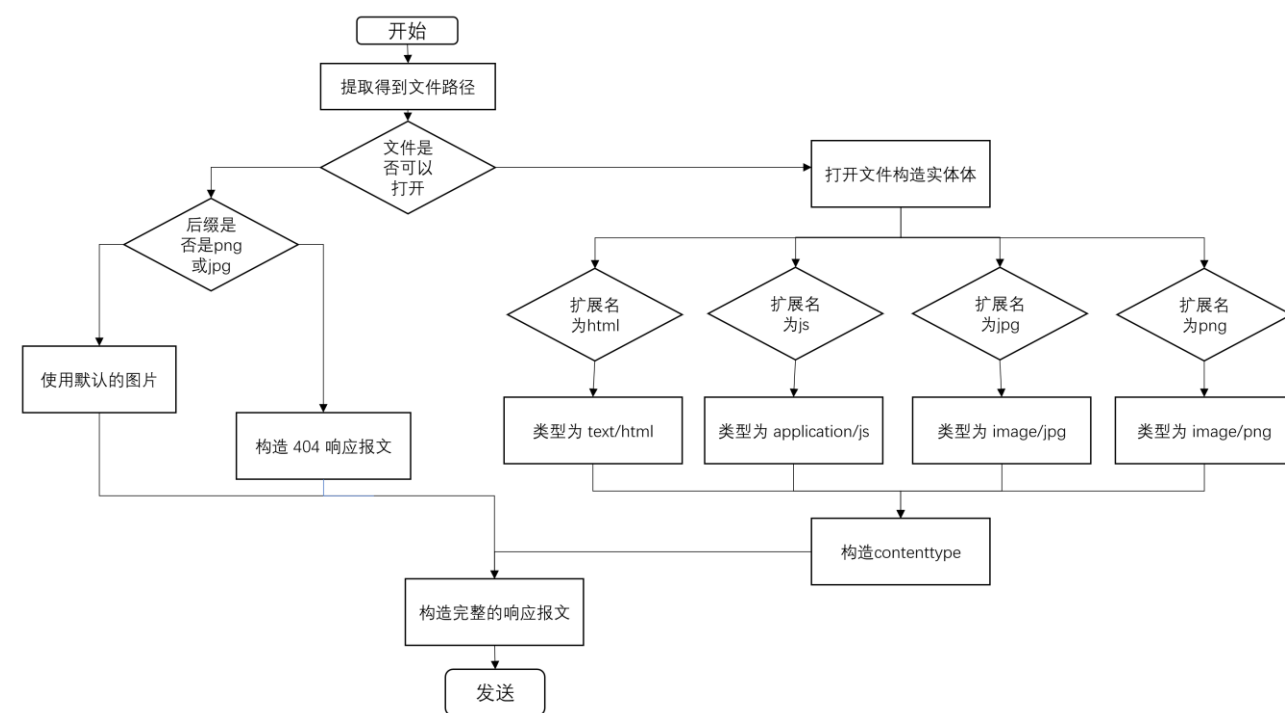


图 1.4 构造响应报文的流程图

首先我们需要根据文件是否存在以及文件类型，代码会生成不同的状态行，如果请求的文件不存在，首先检查是否为图片文件。如果不是图片请求，返回 HTTP/1.1 404 Not Found\r\n，表示文件未找到。如果请求的文件是图片且文件不存在，则尝试返回一个默认的图片 03.png。如果默认图片也无法找到，则返回 HTTP/1.1 404 Not Found\r\n。

根据文件名的扩展名，代码会选择相应的 Content-Type 类型，并构造出 HTTP 响应的首部行。例如，如果文件是 .html，则设置 Content-Type: text/html; charset=UTF-8。如果是图片文件 .png 或 .jpg，则设置对应的图片类型 (image/png、image/jpg)。

如果文件存在，代码会打开文件并读取文件内容。使用 std::ostringstream 将文件内容以二进制的形式加载到内存中，并转换为字符串 content，来构建实体体。

通过 send 函数将构建好的 HTTP 响应报文（包括状态行、首部行、实体体）发送给客户端。如果文件发送成功，会输出日志显示发送的字节数和文件名。

1.5 系统测试及结果说明

硬件测试环境：

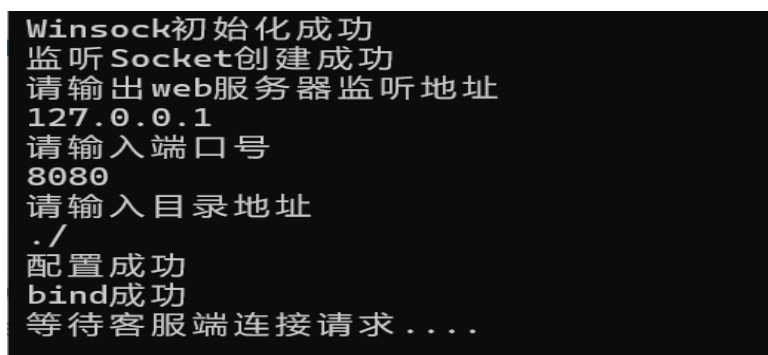
处理器： 11th Gen Intel(R) Core(TM) i5-11260H @ 2.60GHz 2.61 GHz

机带主存： 16.0 GB

测试结果与分析：

基础要求：

1. 可配置监听地址、监听端口和主目录，将监听信息与监听 Socket 绑定，且能够在监听端口上进行监听。细节见图 1.5。



```
Winsock初始化成功
监听Socket创建成功
请输出web服务器监听地址
127.0.0.1
请输入端口号
8080
请输入目录地址
./
配置成功
bind成功
等待客户端连接请求....
```

图 1.5 配置监听地址，端口以及主目录截图

2. 收到客户端请求时能创建连接套接字，能够响应客户端的请求，并定位相应的 html 文件，能够构造并发送可被客户端解析的响应报文。



图 1.6 响应客户端请求测试

我们还可以通过浏览器的网络工具查看响应报文的基本信息，如图 1.7 所示，状态码为



图 1.7 查看响应报文截图

高级要求：

1. 在服务器端的屏幕上输出每个请求的来源（IP 地址、端口号和 HTTP 请求命令行）。

在图 1.8 中可以看到 client 的 ip 是 127.0.0.1，端口是 52120，用 GET 方法，请求的 url 是 test.html，请求报文的大小为 816 字节。

```
IP Address: 127.0.0.1
Port: 52120
Received 816 bytes from client:
GET /test.html HTTP/1.1
Host: 127.0.0.1:8080
Connection: keep-alive
sec-ch-ua: "Microsoft Edge";v="131", "Chromium";v="131", "Not_A Brand";v="24"
sec-ch-ua-mobile: ?0
sec-ch-ua-platform: "Windows"
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like C
.36 Edg/131.0.0.0
```

图 1.8 请求的来源截图

2. 在服务器端的屏幕上能够输出对每一个请求处理的结果

```
http请求命令是 GET /test.html HTTP/1.1
Send 694 bytes to client
已发送filename:/test.html
发送完毕
```

图 1.9 处理结果截图

3. 对于无法成功定位文件的请求，能够根据错误原因，作相应的错误提示

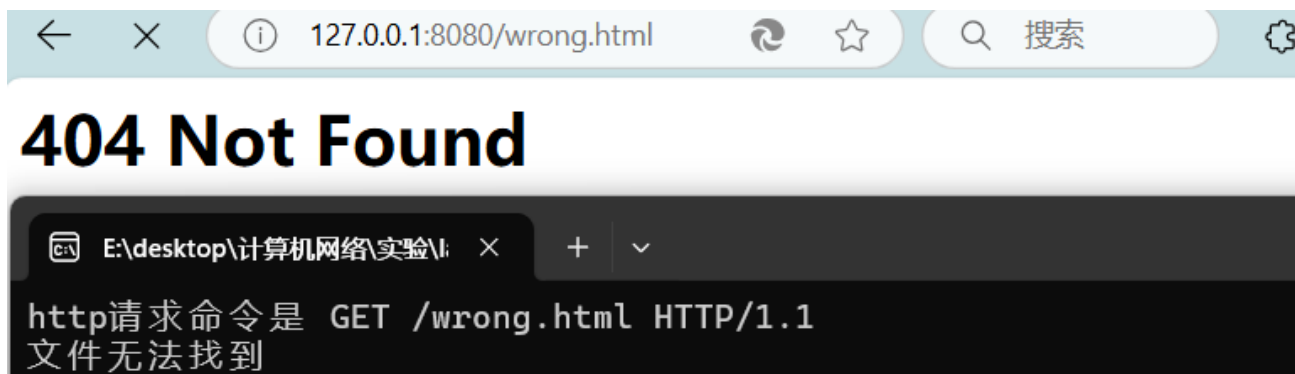


图 1.10 无法成功定位文件的处理截图

4. 支持一定的异常处理能力。

由于我们很多时候请求的文件里面涉及到很多图片，如果我们的图片路径存在问题的话，将找不到图片，故我设置了一个默认图片，如果找不到图片将用默认的图片，细节见图 1.11。

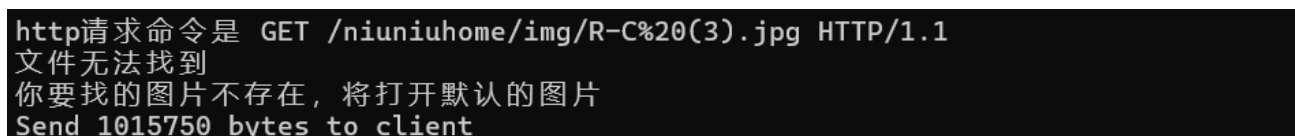


图 1.11 异常处理截图

1.6 其它需要说明的问题

在进行主目录的配置时，需以’/’作为分割线。原因在于客户端输入的 URL 一般以’/’为界线，为了方便进行路径的合并，在输入主目录时，同样也以’/’进行路径填写。

1.7 参考文献

- [1] (美)James F. Kurose, (美)Keith W. Ross 著, 陈鸣译. 计算机网络：自顶向下方法(原书 第 7 版). 北京：机械工业出版社,2018.5
- [2] 华中科技大学计算机科学与技术学院. Socket 编程实验指导手册

实验二 数据可靠传输协议设计实验

2.1 环境

- (1) 操作系统: Windows 11 家庭中文版
- (2) 处理器: 11th Gen Intel(R) Core(TM) i5-11260H @ 2.60GHz 2.61 GHz
- (3) 虚拟内存: 16.0 GB
- (4) 开发环境: Visual Studio 2022
- (5) 编程语言: C++
- (6) 依赖库: netsimlib.lib

2.2 实验要求

可靠运输层协议实验只考虑单向传输, 即: 只有发送方发生数据报文, 接收方仅仅接收报文并给出确认报文。要求实现具体协议时, 指定编码报文序号的二进制位数(例如 3 位二进制编码报文序号)以及窗口大小(例如大小为 4), 报文段序号必须按照指定的二进制位数进行编码。代码实现不需要基于 Socket API, 不需要利用多线程, 不需要任何 UI 界面。

本实验包括三个级别的内容, 具体包括:

- (1) 实现基于 GBN 的可靠传输协议。
- (2) 实现基于 SR 的可靠传输协议。
- (3) 在实现 GBN 协议的基础上, 根据 TCP 的可靠数据传输机制实现一个简化版的 TCP 协议, 要求:

报文段格式、接收方缓冲区大小和 GBN 协议一样保持不变; 报文段序号按照报文段为单位进行编号; 单一的超时计时器, 不需要估算 RTT 动态调整定时器 Timeout 参数; 支持快速重传和超时重传, 重传时只重传最早发送且没被确认的报文段; 确认号为收到的最后一个报文段序号; 不考虑流量控制、拥塞控制。

2.3 协议的设计、验证及结果分析

实验中我们报文的序号采取三位编码, 表示范围为 $[0, 7]$, 发送方的窗口都设置为 4, 接收方窗口根据不同的协议具体考虑。

2.3.1 GBN 协议的设计、验证及结果分析

GBN 协议的设计：

由于采取的是累计确认的方式，接收方只能按顺序接收分组，无法接受和缓存失序的分组。故接收方的窗口设置为 1，对于发送方和接收方面对各种情况的动作见图 2.1，将 GBN 协议的发送方和接收方设计并实现为两个对象 GBN RdtSender 和 GBN RdtReceiver。

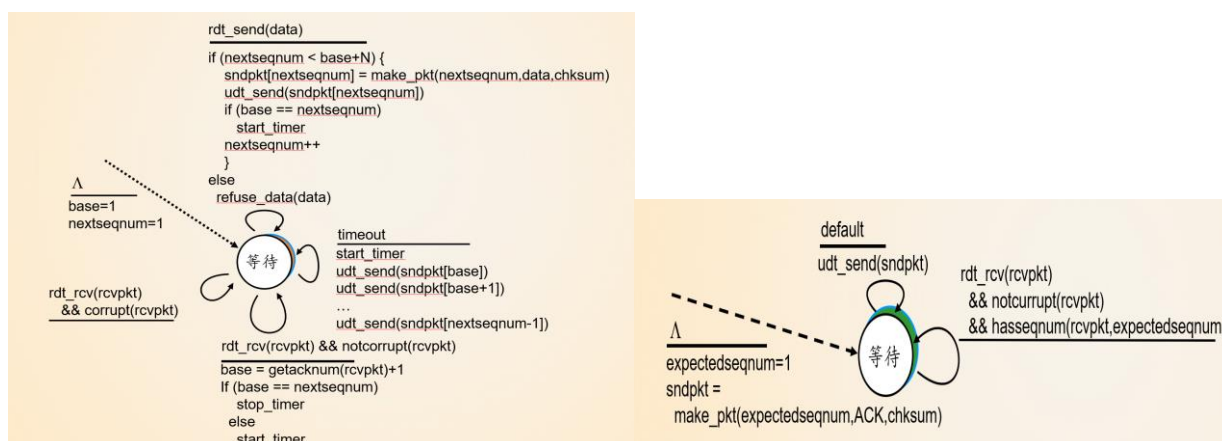


图 2.1 发送方和接收方的 FSM 示意图

设计过程中有一些细节问题需要处理，见表 2.1。

发送方的计时器维护什么	接收方收到失序的分组	超时时候发送方重发哪些分组
发送窗口中最早未被确认的分组	丢弃失序的分组，继续等待期望的分组	重发所有从超时分组开始的分组（包括超时的分组）

表 2.1 GBN 设计中细节处理表

经过上面的分析，给出类 GBN RdtSender 和 GBN RdtReceiver 的代码实现。

```

class GBNSender :public RdtSender
{
private:
    int expectSequenceNumberSend;    // 下一个发送序号
    bool waitingState;               // 是否处于等待 Ack 的状态
    int base;                        // 当前窗口基序号
    int winlen;                      // 窗口大小
    int seqlen;                     // 序号宽度
    std::deque<Packet> window;       // 窗口队列
    Packet packetWaitingAck;         // 已发送并等待 Ack 的数据包

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:
    GBNSender();
    virtual ~GBNSender();
};

```

```

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

```

关键函数的分析:

1. `bool GBNSender::send(const Message& message);`

该函数负责接收应用层的数据 `message`, 将其封装为传输层的数据包 `Packet`。在 GBN 协议中, 由于采用累计确认机制, 仅需一个计时器来管理所有未确认的分组。当最早的未确认分组序号与下一个待发送分组序号相同时, 表明接收方已接收全部数据包, 此时重新发送数据包时需重启计时器。随后, 调用 `sendToNetworkLayer` 函数发送数据包。发送完成后, 滑动窗口会循环右移, 并打印出当前的窗口状态。该函数接收应用层数据 `message` 作为输入, 并返回一个 `bool` 类型的变量, 指示数据发送是否成功。

2. `void GBNSender::receive(const Packet& ackPkt);`

该函数负责处理来自接收方的 ACK 报文 `ackPkt`。首先, 它会通过计算校验和来检查接收到的 ACK 是否损坏; 如果 ACK 损坏, 则不进行任何处理。随后, 函数会更新滑动窗口的状

态。由于 GBN 协议采用累计确认机制，一旦 ACK 被确认有效，它会将滑动窗口的基序号 base 直接设置为 $(ack+1) \% seqsize$ 。如果此时 base 序号与下一个待确认的分组序号相同，这意味着接收方已经成功接收了所有数据包，因此函数会停止计时器。如果条件不满足，则会重启计时器。该函数接收 ACK 报文 ackPkt 作为输入参数，并且不返回任何值。

3. `void GBNSender::timeoutHandler(int seqNum);`

判断下一个待确认分组序号与 base 序号是否相同，若相同则不做处理，否则重启 计时器，并将从 base 序号到下一个待确认分组序号前的所有数据包重传。

4. `void GBNReceiver::receive(const Packet& packet);`

当 GBN 接收方接收到来自发送方的数据包时，会调用相应的处理函数。该函数首先会检查数据包中的校验和字段。如果校验和正确，并且数据包的序号与接收方当前期待的序号相匹配，那么函数会将数据包中的数据解封装出来，并递交给应用层进行处理。同时，接收方会向网络层发送一个确认报文（ACK），以通知发送方该数据包已被成功接收，并据此更新接收方的窗口状态。如果校验和错误，或者数据包的序号与接收方当前期待的序号不匹配，那么接收方将丢弃该数据包，并向网络层重新发送上一次成功接收数据包所对应的 ACK，以保持通信的可靠性。

GBN 协议的验证与结果分析：

通过测试脚本将规定文本内容在模拟网络环境中进行十次随机收发测试并对比发 送数据与接收数据的差异，在十次随机测试中收发数据均保持一致，找不到差异，见图 2.2。

```
C:\Windows\system32\cmd.exe
Test "E:\desktop\计算机网络\实验\second\rdt\Debug\GBN.exe" 1:
正在比较文件 E:\DESKTOP\计算机网络\实验\SECOND\CHECK\input.txt 和 E:\DESKTOP\计算机网络\实验\SECOND\CHECK\OUTPUT_GBN.TXT
FC: 找不到差异

Test "E:\desktop\计算机网络\实验\second\rdt\Debug\GBN.exe" 2:
正在比较文件 E:\DESKTOP\计算机网络\实验\SECOND\CHECK\input.txt 和 E:\DESKTOP\计算机网络\实验\SECOND\CHECK\OUTPUT_GBN.TXT
FC: 找不到差异

Test "E:\desktop\计算机网络\实验\second\rdt\Debug\GBN.exe" 3:
正在比较文件 E:\DESKTOP\计算机网络\实验\SECOND\CHECK\input.txt 和 E:\DESKTOP\计算机网络\实验\SECOND\CHECK\OUTPUT_GBN.TXT
FC: 找不到差异

Test "E:\desktop\计算机网络\实验\second\rdt\Debug\GBN.exe" 4:
正在比较文件 E:\DESKTOP\计算机网络\实验\SECOND\CHECK\input.txt 和 E:\DESKTOP\计算机网络\实验\SECOND\CHECK\OUTPUT_GBN.TXT
FC: 找不到差异

Test "E:\desktop\计算机网络\实验\second\rdt\Debug\GBN.exe" 5:
正在比较文件 E:\DESKTOP\计算机网络\实验\SECOND\CHECK\input.txt 和 E:\DESKTOP\计算机网络\实验\SECOND\CHECK\OUTPUT_GBN.TXT
FC: 找不到差异

Test "E:\desktop\计算机网络\实验\second\rdt\Debug\GBN.exe" 6:
正在比较文件 E:\DESKTOP\计算机网络\实验\SECOND\CHECK\input.txt 和 E:\DESKTOP\计算机网络\实验\SECOND\CHECK\OUTPUT_GBN.TXT
FC: 找不到差异

Test "E:\desktop\计算机网络\实验\second\rdt\Debug\GBN.exe" 7:
正在比较文件 E:\DESKTOP\计算机网络\实验\SECOND\CHECK\input.txt 和 E:\DESKTOP\计算机网络\实验\SECOND\CHECK\OUTPUT_GBN.TXT
FC: 找不到差异

Test "E:\desktop\计算机网络\实验\second\rdt\Debug\GBN.exe" 8:
正在比较文件 E:\DESKTOP\计算机网络\实验\SECOND\CHECK\input.txt 和 E:\DESKTOP\计算机网络\实验\SECOND\CHECK\OUTPUT_GBN.TXT
FC: 找不到差异

Test "E:\desktop\计算机网络\实验\second\rdt\Debug\GBN.exe" 9:
正在比较文件 E:\DESKTOP\计算机网络\实验\SECOND\CHECK\input.txt 和 E:\DESKTOP\计算机网络\实验\SECOND\CHECK\OUTPUT_GBN.TXT
FC: 找不到差异

Test "E:\desktop\计算机网络\实验\second\rdt\Debug\GBN.exe" 10:
正在比较文件 E:\DESKTOP\计算机网络\实验\SECOND\CHECK\input.txt 和 E:\DESKTOP\计算机网络\实验\SECOND\CHECK\OUTPUT_GBN.TXT
FC: 找不到差异
```

图 2.2 GBN 协议测试截图

我们的中间过程都会重定向到 result.txt 文件中, 我将截取一些片段进行分析, 见图 2.3 。图中展示了发送方窗口因为收到 ack 后的移动, 以及超时的时候发送方发送了当前窗口中还未收到确认的所有分组, 还展示了接收方收到失序分组的时候会丢弃报文, 重复上次的 ack。

```
接收方发送确认报文: seqnum = -1, acknum = 4, checksum = 12847, .....
接收方收到报文后的窗口:[ 5N ] 下一个期待的报文编号
超时了, 开始处理发送方定时器时间到, 重发窗口报文: seqnum = 2, acknum = -1, checksum = 24414,
CCCCCCCCCCCCCCCCCCCC
发送方定时器时间到, 重发窗口报文: seqnum = 3, acknum = -1, checksum = 21843,
DDDDDDDDDDDDDDDDDDDDDD
发送方定时器时间到, 重发窗口报文: seqnum = 4, acknum = -1, checksum = 19272, EEEEEEEEEEEEEEEEEEE
发送方收到ack:3 收到ack=3, 将3以及之前的报文设置为收到
发送方窗口:[ 2Y 3Y 4N 5* ]
发送方正确收到确认: seqnum = -1, acknum = 3, checksum = 12848, .....
发送方滑动后窗口:[ 4N 5* 6 7 ]
接收方没有正确收到发送方的报文,报文序号不对: seqnum = 2, acknum = -1, checksum = 24414,
CCCCCCCCCCCCCCCCCCCC
接收方重新发送上次的确认报文: seqnum = -1, acknum = 4, checksum = 12847, .....
失序收到报文, 丢弃, 重发上次的ack
```

图 2.3 GBN 协议运行结果分析

2.3.2 SR 协议的设计、验证及结果分析

SR 协议的设计:

SR 协议和 GBN 协议类似, 但是 SR 不再采取累计确认的方式, 提高效率的同时也意味着需要缓存收到的失序的报文, 我们的接收方窗口也设置为 4, 其他的细节问题见表 2.3。

发送方的计时器维护什么	接收方收到失序的分组	超时时候发送方重发哪些分组
每个未确认的分组都有一个独立的计时器	缓存失序的分组, 等待缺失分组到达后进行处理	只重发超时的那个分组

表 2.3 SR 协议设计中的细节表

给出类 SR RdtSender 和 SR RdtReceiver 的代码实现。

```
struct waitPck {
    bool flag;    //当未接收到ACK时, 为false
    Packet winPck;
};
class SRSender :public RdtSender
{
private:
    int expectSequenceNumberSend; // 下一个发送序号
```



```

    bool waitingState;           // 是否处于等待Ack的状态
    int base;                     //当前窗口基序号
    int winlen;                   //窗口大小
    int seqlen;                   //序号宽度
    std::deque<waitPck> window;   //窗口队列
    Packet packetWaitingAck;      //已发送并等待Ack的数据包
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();
};

```

```

struct rcvPck {
    bool flag;           //指示该位置是否被占用, ture表示占用
    Packet winPck;       //存放数据包
};
class SRReceiver :public RdtReceiver
{
private:
    int expectSequenceNumberRcvd; // 期待收到的下一个报文序号
    int base;                     //当前窗口基序号
    int winlen;                   //窗口大小
    int seqlen;                   //序号宽度
    deque<rcvPck> window;         //窗口数组
    Packet lastAckPkt;            //上次发送的确认报文
public:
    SRReceiver();

```



```
virtual ~SRReceiver();  
public:  
    void receive(const Packet& packet); //接收报文，将被NetworkService调用  
};
```

关键函数实现：

1. `bool SRSender::send(const Message& message);`

当接收应用层传来的数据 `message` 时，会首先检查缓冲区是否已满。若缓冲区已满载，则发送操作失败，函数返回 `false`。反之，会将 `message`、当前的滑动窗口序号以及校验和等信息封装成一个传输层数据包 `Packet`，并递交给网络层进行发送。鉴于 SR 协议（选择重传协议）的特性，每个数据包都需要配置一个独立的计时器。数据包发送完毕后，系统会更新滑动窗口的状态，并打印出当前的窗口信息，最后函数返回 `true`，表示数据已成功发送。

2. `void SRSender::receive(const Packet& ackPkt);`

当接收方收到来自接收端的 ACK 报文 `ackPkt` 时，函数会首先通过计算校验和来验证 ACK 的完整性。如果校验和结果显示 ACK 已损坏，则将不对其进行任何处理。由于 SR 协议（选择重传协议）采用的是选择重传机制，一旦 ACK 被确认为有效，会将 `base` 序号更新为最后一个尚未确认的数据包的序号，并据此更新滑动窗口的状态。更新完成后，会打印出当前的滑动窗口信息。

3. `void SRSender::timeoutHandler(int seqNum);`

根据 SR 协议（选择重传协议）的运作机制，在发生超时事件时，系统仅需重传当前已超时但尚未确认的数据包，并重新启动该数据包的计时器。

4. `void SRSender::receive(const Packet& packet);`

当接收方收到来自发送方的数据报 `packet` 时，会首先计算其校验和以验证数据的正确性。如果校验和不正确，接收方会要求发送方重传上一次的数据报。此外，接收方还会检查数据报的序号是否与当前期望接收的序号相匹配（即序号是否在接收方的滑动窗口范围内）。如果序号不匹配，同样会要求发送方重传上一次的数据报。只有当校验和正确且序号匹配时，接收方才会从接收缓冲区中取出所有已按顺序到达的数据包，并将这些数据递交给应用层处理。随后，接收方会通过网络层向发送方发送一个确认报文，以确认数据的成功接收

SR 协议的验证与结果分析：

同样地，我们运行脚本将程序跑十遍，分析结果是否正确，见图 2.4，结果正确没问题。

```
C:\Windows\system32\cmd.e  X  +  v

Test "E:\desktop\计算机网络\实验\second\rdt\Debug\SR.exe" 4:
正在比较文件 E:\DESKTOP\计算机网络\实验\SECOND\CHECK\input.txt 和 E:\DESKTOP\计算机网络\实验\SECOND\CHECK\OUTPUT_SR.TXT
FC: 找不到差异

Test "E:\desktop\计算机网络\实验\second\rdt\Debug\SR.exe" 5:
正在比较文件 E:\DESKTOP\计算机网络\实验\SECOND\CHECK\input.txt 和 E:\DESKTOP\计算机网络\实验\SECOND\CHECK\OUTPUT_SR.TXT
FC: 找不到差异

Test "E:\desktop\计算机网络\实验\second\rdt\Debug\SR.exe" 6:
正在比较文件 E:\DESKTOP\计算机网络\实验\SECOND\CHECK\input.txt 和 E:\DESKTOP\计算机网络\实验\SECOND\CHECK\OUTPUT_SR.TXT
FC: 找不到差异

Test "E:\desktop\计算机网络\实验\second\rdt\Debug\SR.exe" 7:
正在比较文件 E:\DESKTOP\计算机网络\实验\SECOND\CHECK\input.txt 和 E:\DESKTOP\计算机网络\实验\SECOND\CHECK\OUTPUT_SR.TXT
FC: 找不到差异

Test "E:\desktop\计算机网络\实验\second\rdt\Debug\SR.exe" 8:
正在比较文件 E:\DESKTOP\计算机网络\实验\SECOND\CHECK\input.txt 和 E:\DESKTOP\计算机网络\实验\SECOND\CHECK\OUTPUT_SR.TXT
FC: 找不到差异

Test "E:\desktop\计算机网络\实验\second\rdt\Debug\SR.exe" 9:
正在比较文件 E:\DESKTOP\计算机网络\实验\SECOND\CHECK\input.txt 和 E:\DESKTOP\计算机网络\实验\SECOND\CHECK\OUTPUT_SR.TXT
FC: 找不到差异

Test "E:\desktop\计算机网络\实验\second\rdt\Debug\SR.exe" 10:
正在比较文件 E:\DESKTOP\计算机网络\实验\SECOND\CHECK\input.txt 和 E:\DESKTOP\计算机网络\实验\SECOND\CHECK\OUTPUT_SR.TXT
FC: 找不到差异

请按任意键继续. . .
```

图 2.4 SR 协议验证截图

具体的细节我们来查看 result.txt,这次我们将着重分析, SR 协议中接收方遇到失序的报文该如何处理。见图 2.5, 由于个别报文的丢失, 后续的报文到达接收方后, 并不会丢弃, 而是缓存起来, 等到超时的时候发送方会再次发送超序号的报文。

发送方发送报文后的窗口: [5N 6N 7N 0N]

四个待确认的报文

接收方正确收到发送方的报文: seqnum = 6, acknum = -1, checksum = 14130, GGGGGGGGGGGGGGGGGGGGGG

接收方收到报文后的窗口: [5N 6Y 7N 0N]

接收方滑动后窗口: [5N 6Y 7N 0N]

缓存失序报文6, 返回ack=6

接收方发送确认报文: seqnum = -1, acknum = 6, checksum = 12845,

接收方正确收到发送方的报文: seqnum = 7, acknum = -1, checksum = 11559, HHHHHHHHHHHHHHHHHHHHHH

接收方收到报文后的窗口: [5N 6Y 7Y 0N]

缓存失序报文7, 返回ack=7

接收方滑动后窗口: [5N 6Y 7Y 0N]

接收方发送确认报文: seqnum = -1, acknum = 7, checksum = 12844,

接收方正确收到发送方的报文: seqnum = 0, acknum = -1, checksum = 8996, IIIIIIIIIIIIIIIIIIIIIII

接收方收到报文后的窗口: [5N 6Y 7Y 0Y]

接收方滑动后窗口: [5N 6Y 7Y 0Y]

接收方发送确认报文: seqnum = -1, acknum = 0, checksum = 12851,

发送方收到ack:6

缓存失序报文0, 返回ack=0

发送方窗口: [5N 6Y 7N 0N]

发送方正确收到确认: seqnum = -1, acknum = 6, checksum = 12845,

由于报文5丢失导致的超时, 重发报文5

发送方滑动后窗口: [5N 6Y 7N 0N]

发送方定时器时间到, 重发上次发送的报文: seqnum = 5, acknum = -1, checksum = 16701, FFFFFFFFFFFFFFFFFFFFFF

图 2.5 SR 协议运行结果分析

2.3.3 简单 TCP/IP 协议的设计、验证及结果分析

简单 TCP/IP 协议的设计:

实验中的要实现的是一个简化版的 TCP/IP 协议, 是基于 GBN 协议改进的, 所以不具备缓存失序的报文的能力, 但是有一个很大的区别就是引入了快速重传机制, 我们只要稍微修改一

个 GBN 的发送方的代码就基本能够实现。

给出类 TCP RdtSender 和 TCP RdtReceiver 的代码实现。

```
class TCPSender :public RdtSender
{
private:
    int expectSequenceNumberSend; // 下一个发送序号
    bool waitingState;           // 是否处于等待Ack的状态
    int Rdnum;                   //记录冗余ACK数量
    int base;                    //当前窗口基序号
    int winlen;                  //窗口大小
    int seqlen;                  //序号宽度
    deque<Packet> window;        //窗口队列
    Packet packetWaitingAck;     //已发送并等待Ack的数据包
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:
    TCPSender();
    virtual ~TCPSender();
};
```

```
class TCPReceiver :public RdtReceiver
{
private:
    int expectSequenceNumberRcvd; // 期待收到的下一个报文序号
    int seqlen;                   //序号宽度
    Packet lastAckPkt;            //上次发送的确认报文
public:
    TCPReceiver();
};
```

```
virtual ~TCPReceiver();  
public:  
    void receive(const Packet& packet); //接收报文，将被NetworkService调用  
};
```

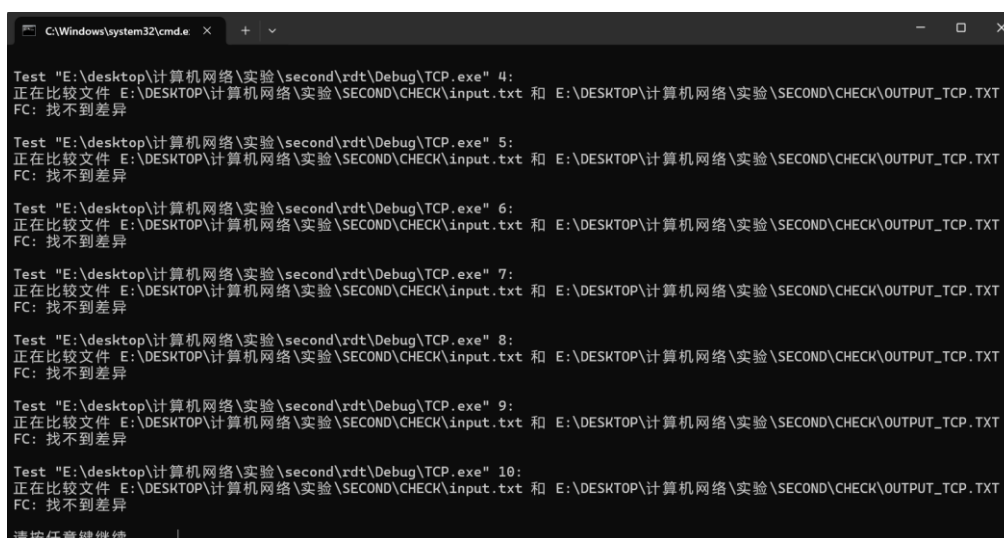
关键函数分析：

1. `void TCPSender::receive(const Packet& ackPkt)`

这个函数中基本上还是采取 SR 协议的代码，但是为了增加快速重传，我们要对代码进行稍微的改进，如果当前收到确认报文的序号等于上次收到的报文的序号，我们的计数变量 `Rdnum++`，如果 `Rdnum` 到了 3 说明收到了三个冗余的 `ack`，就开始快速启动重传。

TCP 协议的验证与结果分析：

我们还是运行脚本测试结果，见图 2.6，结果正确。



```
C:\Windows\system32\cmd.exe  
Test "E:\desktop\计算机网络\实验\second\rdt\Debug\TCP.exe" 4:  
正在比较文件 E:\DESKTOP\计算机网络\实验\SECOND\CHECK\input.txt 和 E:\DESKTOP\计算机网络\实验\SECOND\CHECK\OUTPUT_TCP.TXT  
FC: 找不到差异  
Test "E:\desktop\计算机网络\实验\second\rdt\Debug\TCP.exe" 5:  
正在比较文件 E:\DESKTOP\计算机网络\实验\SECOND\CHECK\input.txt 和 E:\DESKTOP\计算机网络\实验\SECOND\CHECK\OUTPUT_TCP.TXT  
FC: 找不到差异  
Test "E:\desktop\计算机网络\实验\second\rdt\Debug\TCP.exe" 6:  
正在比较文件 E:\DESKTOP\计算机网络\实验\SECOND\CHECK\input.txt 和 E:\DESKTOP\计算机网络\实验\SECOND\CHECK\OUTPUT_TCP.TXT  
FC: 找不到差异  
Test "E:\desktop\计算机网络\实验\second\rdt\Debug\TCP.exe" 7:  
正在比较文件 E:\DESKTOP\计算机网络\实验\SECOND\CHECK\input.txt 和 E:\DESKTOP\计算机网络\实验\SECOND\CHECK\OUTPUT_TCP.TXT  
FC: 找不到差异  
Test "E:\desktop\计算机网络\实验\second\rdt\Debug\TCP.exe" 8:  
正在比较文件 E:\DESKTOP\计算机网络\实验\SECOND\CHECK\input.txt 和 E:\DESKTOP\计算机网络\实验\SECOND\CHECK\OUTPUT_TCP.TXT  
FC: 找不到差异  
Test "E:\desktop\计算机网络\实验\second\rdt\Debug\TCP.exe" 9:  
正在比较文件 E:\DESKTOP\计算机网络\实验\SECOND\CHECK\input.txt 和 E:\DESKTOP\计算机网络\实验\SECOND\CHECK\OUTPUT_TCP.TXT  
FC: 找不到差异  
Test "E:\desktop\计算机网络\实验\second\rdt\Debug\TCP.exe" 10:  
正在比较文件 E:\DESKTOP\计算机网络\实验\SECOND\CHECK\input.txt 和 E:\DESKTOP\计算机网络\实验\SECOND\CHECK\OUTPUT_TCP.TXT  
FC: 找不到差异  
请按任意键继续...
```

图 2.6 TCP 协议验证结果截图

对于 `result.txt` 中记录的信息，我们着重分析快速重传的过程。

实验三 基于 CPT 的组网实验

3.1 环境

处理器: 11th Gen Intel(R) Core(TM) i5-11260H @ 2.60GHz 2.61 GHz

操作系统: Windows 11 家庭中文版

实验软件: Cisco Packet Tracer 6.0.0.0045

3.2 实验要求

熟悉使用 Cisco Packet Tracer 软件, 并使用该仿真软件完成本次实验。

本次实验共分为以下三个部分:

1. IP 地址规划与 Vlan 分配实验:

通过理解网络规划与配置的基本原理, 以及掌握网络连通性的分析与测试方法, 运用模拟软件创建一个包含多台个人电脑和路由器的网络拓扑。在此基础上, 按照特定的 IP 地址规划, 为路由器配置适当的端口地址, 并对各个个人电脑之间的连通性进行深入分析。

2. 路由配置实验:

此实验涉及三个主要方面, 要求使用模拟软件进行路由器配置和访问控制的实际操作。首先, 在基础内容的第一部分中, 需构建一个特定的网络拓扑, 配置路由器上的 RIP 协议, 确保各个个人电脑之间能够相互访问。其次, 在基础内容的第二部分中, 需要在路由器上配置 OSPF 协议。最后, 在基础内容 1 或 2 的基础上, 学生需对路由器进行进一步的访问控制设置。

3. 综合实验:

此实验的核心目标是为一个学校设计和搭建网络, 满足学院、图书馆、学生宿舍等不同地点的网络需求。通过充分利用提供的 IP 地址块, 设计并配置网络以满足复杂的连接和隔离需求, 解决实际生活中的网络问题。

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

I. 子实验 1

按照实验规定, 运用 Cisco Packet Tracer 建立一个与提供的拓扑图等效的网络结构。该

网络包含 8 台个人计算机、4 台交换机和 1 台路由器。

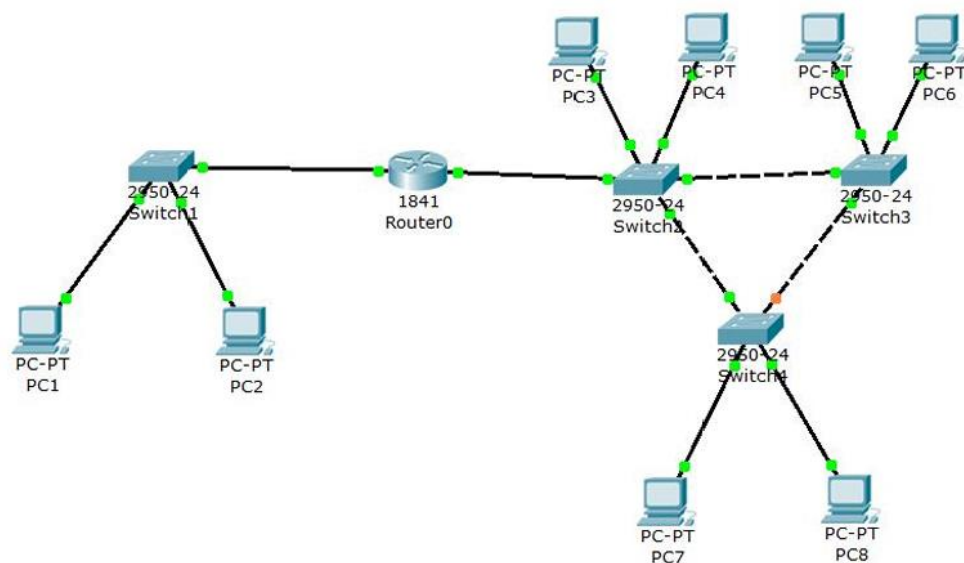


图 3.1

在实验中，我们采用直通线连接异类设备，而对于同类设备，则使用交叉线进行连接。连接计算机与交换机的过程中，我们使用了直通线以确保它们之间正常通信。同时，连接交换机与交换机的过程中，我们选择了交叉线，以保障它们之间的有效连接。最后，我们采用直通线将交换机与路由器相连。

一旦完成连接，我们为各个计算机和路由器分配了 IP 地址。在路由器的左侧，Gateway 配置为 192.168.0.1，而路由器右侧 PC 的 Gateway 配置为 192.168.1.1。PC1 和 PC2 分别获得了 IP 地址 192.168.0.2 和 192.168.0.3，而 PC3 到 PC8 则分别获得了 IP 地址从 192.168.1.2 到 192.168.1.7 的范围。此外，网络配置信息可通过将鼠标悬停在路由器或计算机上进行查看。

之后可以通过 Cisco Packet Tracer 提供的工具令各个 PC 之间互相通信，实验结果见图 3.2。

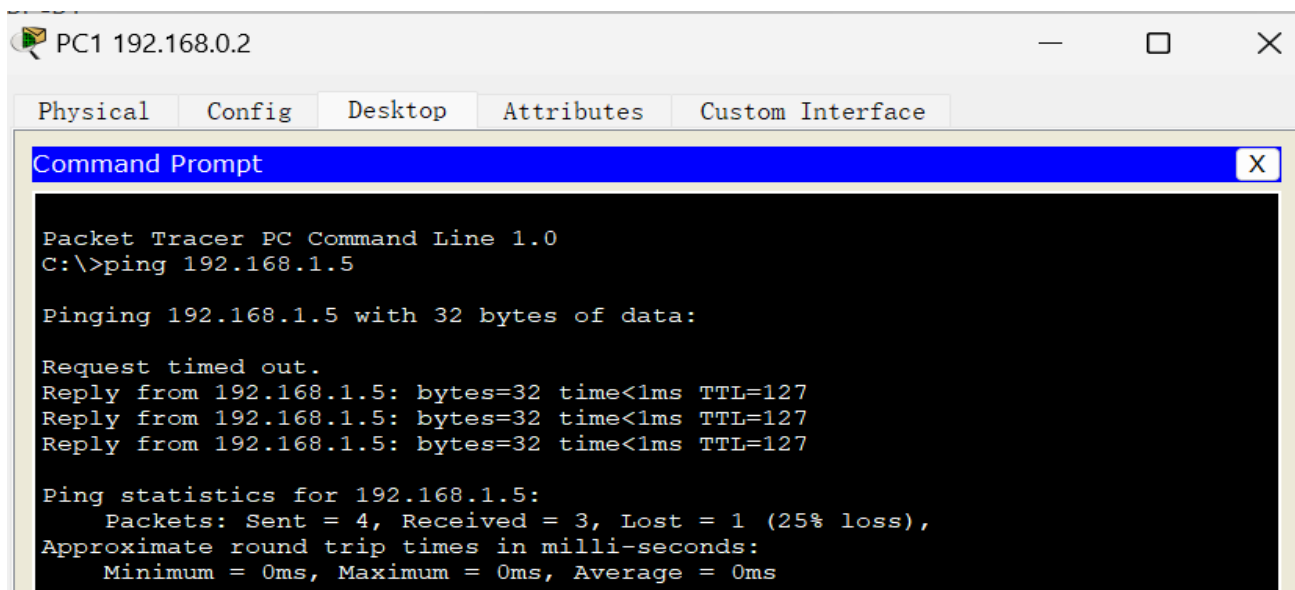


图 3.2

II. 子实验 2

在子实验 1 的基础上，需要对 PC4、PC6、PC8 进行重新配置，将它们从 192.168.1.0 子网中分离出来，并将它们的 IP 重新分配到 192.168.2.0 子网。这将导致 PC4、PC6、PC8 与路由器右端接口不再位于同一子网中。由此，192.168.2.0 子网中的设备将无法与其他子网中的设备进行通信，我们 通过配置好路由器的子端口信息就可以实现不同 vlan 之间的通讯了，见图 3.3。

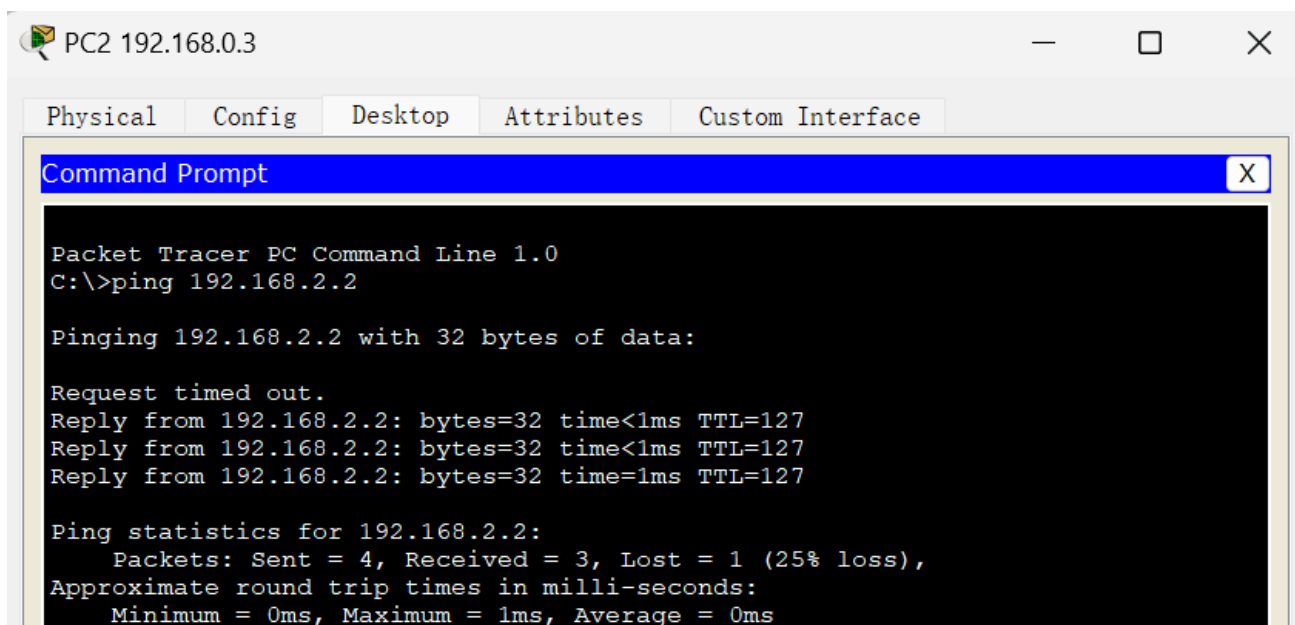


图 3.3

III. 结果分析

网络设备的端口具有配置 IP 地址的能力，而连接到这些端口的两个网段的个人电脑设置

了路由器端口的 IP 地址作为网关。这使得路由器能够转发消息，从而实现了这两个网段之间的通信。

此外，通过在交换机上划分虚拟局域网（VLAN），各个主机被分配到不同的 VLAN 中。每个 VLAN 都被视为一个独立的逻辑实体，不受物理空间和子网的限制。因此，位于同一 VLAN 的 PC1 和 PC2 能够相互通信。如果在路由器上没有配置 VLAN，VLAN 内的主机可以互相通信，但无法与其他 VLAN 中的主机通信，形成一种隔离状态。只有在路由器上进行了 VLAN 的配置，VLAN 内的消息才能被路由器正确识别和转发。否则，路由器无法确定消息的目的地，因为本地没有相关的 VLAN 可供选择。

3.3.2 路由配置实验的步骤及结果分析

I. 子实验 1

按照实验规定，运用 Cisco Packet Tracer 建立一个与提供的拓扑图等效的网络结构。该网络包含 8 台个人计算机、3 台交换机和 4 台路由器。最终布局如图 3.4。

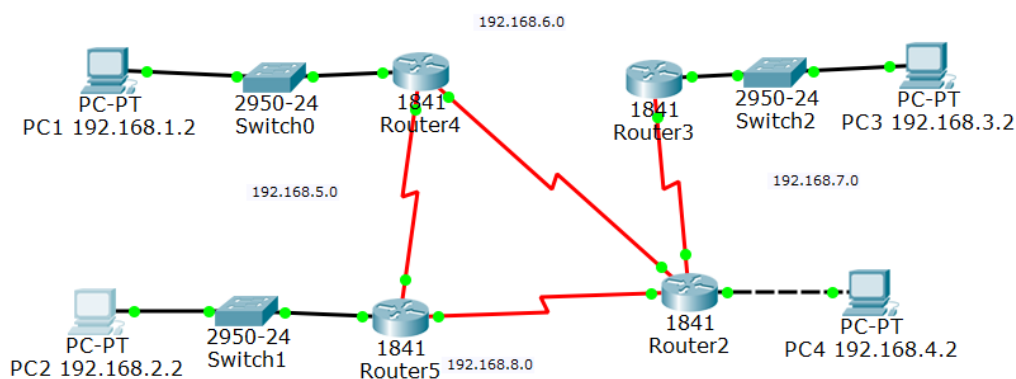


图 3.4

本次实验涉及四个路由器，每个负责管理一个特定子网，分别对应于 192.168.1.0、192.168.2.0、192.168.3.0 和 192.168.4.0 网段。为确保连接在路由器与 PC 机或交换机之间的端口 IP 与相应网段的 IP 地址匹配，必须进行配置。

RIP（Routing Information Protocol）是一种专为自治系统内设计的动态路由协议，用于实现路由器之间的路由信息交换，从而实现自动路由更新。在配置 RIP 协议的路由器时，只需指定每个端口的 IP 地址所在的网段。在这个实验中，要配置 RIP 协议，这是一种基于跳数评估路由优劣的距离矢量路由协议。为了为 routerA 配置 RIP 协议，需要进入“配置->路由配置->RIP”选项，然后在相应的界面中输入与这三个 IP 地址相对应的网段，并点击添加。配置完成后，路由器成功完成了 RIP 协议的设置，从而实现了动态路由更新的功能。



图 3.5

其他计算机的 IP 地址和网关设置，以及路由器的配置方法也是相似的，在完成配置后，可以随意选择两台计算机进行相互通信。在测试过程中，可能会出现第一次访问请求失败的情况，但在之后的第二次尝试中通信通常会成功。这可能是由于软件本身存在的问题，但也表明两台计算机可以正常访问。

II. 子实验 2

在第二个子实验中，与配置 RIP 协议的子实验相比，配置路由 OSPF 协议有所不同，配置 RIP 协议可以在可视化界面快速填写，OSPF 要在命令行配置，配置的具体命令可以在任务书上面查看，配置完的截图如下图

```
Router(config)#router ospf 1
Router(config-router)#network 192.168.1.0 0.0.0.255 area 0
Router(config-router)#network 192.168.5.0 0.0.0.255 area 0
Router(config-router)#network 192.168.6.0 0.0.0.255 area 0
Router(config-router)#end
```

图 3.6

最后进行测试，截图见图 3.7。

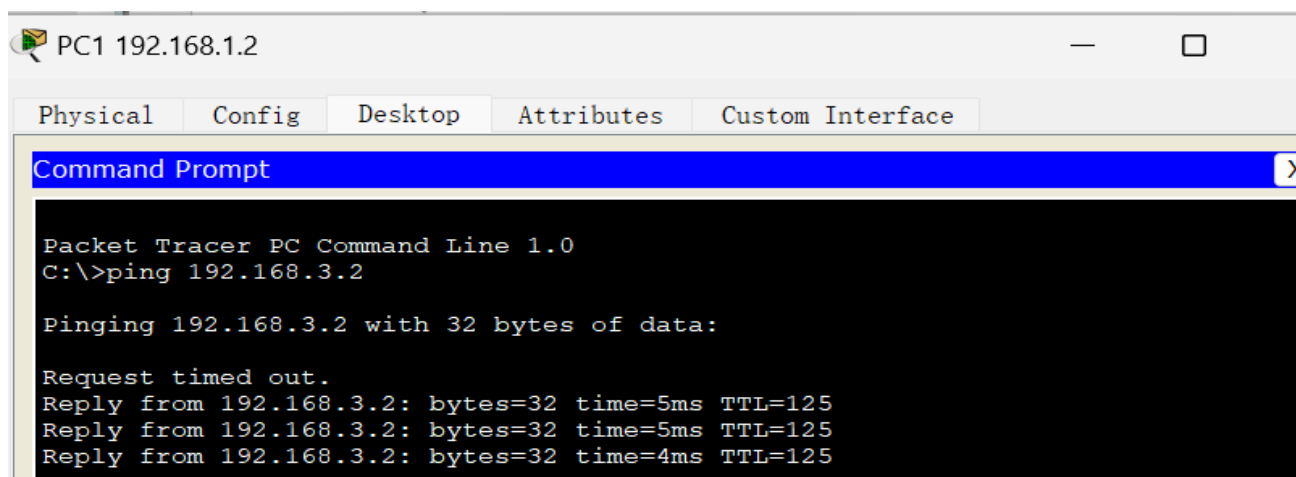


图 3.7

III. 子实验 3

在第三个子实验的首个任务中，要对路由器进行访问控制列表的配置。此处使用命令行对路由器 B 进行设置，以使得计算机 A 无法与其他网络子网通信，同时不受其他网络子网的访问。

在路由器 B 与交换机 1 相连接的端口上执行配置，使用 `access-list` 命令创建一个访问控制列表，选择 ACL 编号在 0 到 99 之间的数字。使用 `deny` 关键词来拦截特定网络子网的通信，而使用 `permit` 关键词则表示允许特定网络子网的通信。在这里，我们采用 `deny` 规则来屏蔽计算机 A 的通信。最后，通过 `access-group` 命令将 ACL 与路由器关联。我们只用将这个规则配置为特定端口的 `in` 或者 `out`，其实效果是一样的，最后配置完的截图见 3.7。

```

Router#show access-list
Standard IP access list 10
 10 deny 192.168.1.0 0.0.0.255
 20 permit any

```

图 3.8

下图是网络拓扑图检测，其中 PC1 无法成功发起对其他 PC 的请求，同时其他 PC 也无法访问 PC1，效果见图 3.8。

激活	最后状态	来源设备	目的设备
	失败	PC1	PC2
	成功	PC2	PC4
	成功	PC2	PC3

图 3.9

在实验的第三个子任务的第二个任务中，需要对路由器进行访问控制列表的设置，通过命令行对路由器 A 进行调整。此时，要实现 PC1 无法与 PC2 进行通信，但可以与其他计算机进行通讯。这时候我们就要用到扩展 `acl` 控制的命令，序号大于等于 100。我们需要指定源 `ip` 和目的 `ip`，这样才能达到效果，如下图 3.9 所示。

```
Router>enable
Router#show access-list
Extended IP access list 100
 10 deny ip host 192.168.1.2 host 192.168.2.2
 20 permit ip any any
```

图 3.10

以下为对网络拓扑的检测，其中 PC1 对 PC2 的请求失败，PC2 对 PC1 的请求也失败，而其他 PC 与 PC1、PC2 的访问正常。

激活	最后状态	来源设备	目的设备
	失败	PC1	PC2
	失败	PC2	PC1
	成功	PC1	PC4
	成功	PC2	PC3
	成功	PC3	PC4

图 3.11

IV. 结果分析

两个路由选择协议，在配置的时候有相似之处，通过学习任务书的配置方法，也对每一个路由器成功进行了初始化操作。

此外，访问控制列表（ACL）被用于管理路由器和交换机端口的数据包进出。通过在路由器 A 的端口上使用 deny 规则，我们可以阻止来自 PC1 所在网段的消息，有效过滤与 PC1 相关的所有通信，使得 PC1 无法与外部进行通信。如果需要同时允许某些消息通过并阻止其他消息，我们可以结合使用 deny 和 permit 指令，实现更精细的访问控制。灵活使用扩展的 acl 可以简化我们的配置。

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

这个实验本质上是对之前每个小实验的结合和运用，我们要学会分配 ip，合理分配 vlan，以及配置好控制访问 acl。

首先我们进行 ip 的分配，先要分配宿舍的 IP 地址，每个宿舍需要支持 200 台主机，其中 8 位用于子网编码，剩余的 50 多台主机则可等待后续使用。学校获得的 IP 地址块是 211.69.4/22，可以利用低 10 位进行划分。因此，通过限定第 8、9 位，低 8 位用于编码，可以分别使用 211.69.4.0/24、211.69.5.0/24、211.69.6.0/24 作为三个宿舍的 IP 地址。对于图书馆，需要支持 100 台主机，至少需要 7 位，因此限定第 7 位，低 7 位用于编址。图书馆可以使用 211.69.7.0/25 网段。最后，还有三个学院需要分配 IP 地址，每个学院有 20 台主机。考虑前面剩余的网段还有 7 位可用，因此使用低 5 位进行每个学院的 IP 编址，第 5、6 位用于学院

的编码。在确定了 IP 地址划分方案后，得到了初步的设计图，并在 Cisco Packet Tracer 中绘制了网络拓扑图。

我们每一个宿舍和每一个学院可以设置为一个 vlan，方便我们进行管理。

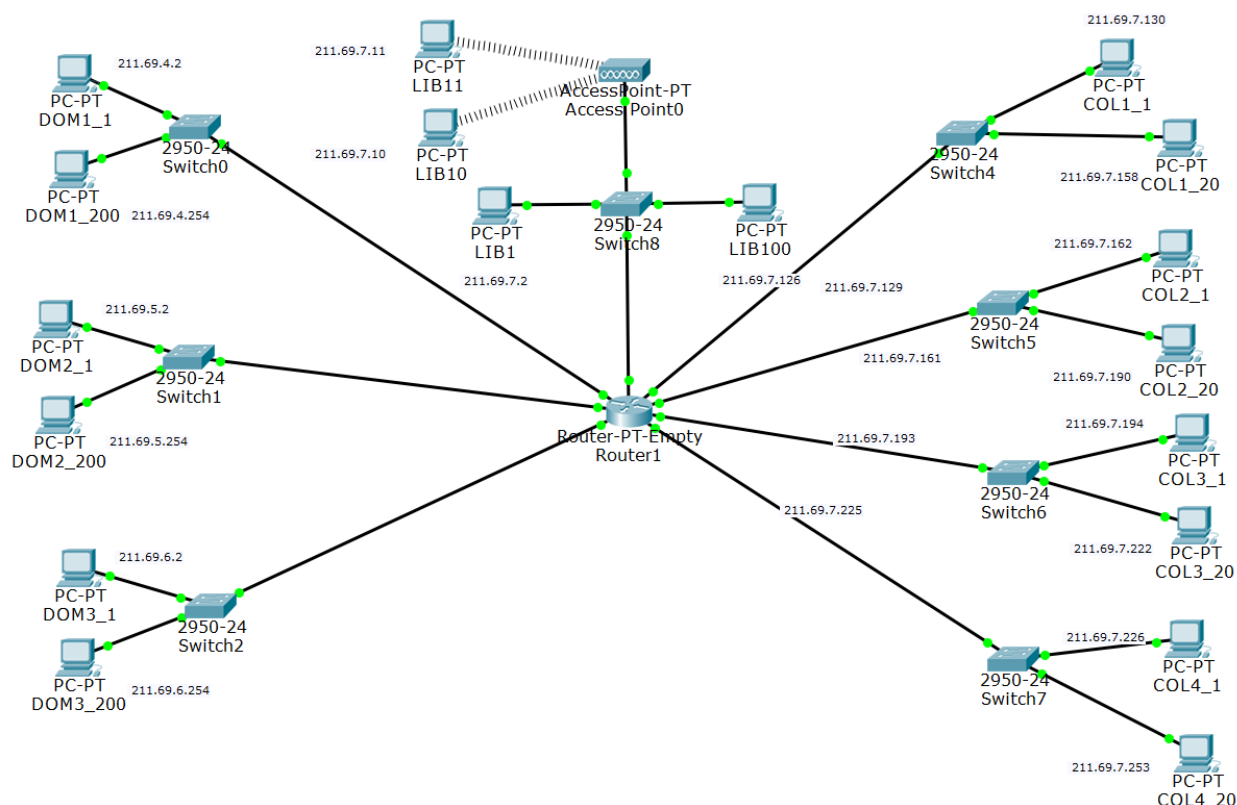


图 3.12

3.4.2 实验步骤

按照我们规划的 ip 分配进行设置，每一组包含两台计算机，代表这部分的网络的主机，依照总体网络拓扑结构进行连接，确保同类设备之间采用互交线，异类设备之间使用直通线原则。配置以后我们就可以设置 vlan，见图 3.11，接下来设置宿舍和学院网关路由器的 ACL，使宿舍与学院不能互相访问。最后为了实现图书馆的有线和无线上网，我们还要设置图书馆的 DHCP 服务器和无线接入点（采用 WPA2-PSA 认证），用搭载无线网卡的 Laptop 连接无线接入点。

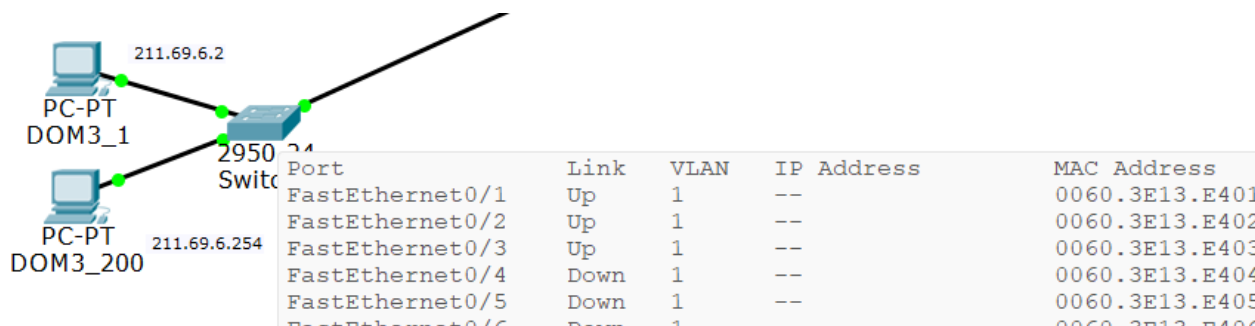


图 3.13 部分 vlan 设置截图

```
Router>enable
Router#show access-list
Extended IP access list 102
 10 deny ip 211.69.4.0 0.0.0.255 211.69.7.128 0.0.0.127
 20 permit ip any any
Extended IP access list 103
 10 deny ip 211.69.5.0 0.0.0.255 211.69.7.128 0.0.0.127
 20 permit ip any any
Extended IP access list 104
 10 deny ip 211.69.6.0 0.0.0.255 211.69.7.128 0.0.0.127
 20 permit ip any any
```

图 3.14 路由器的 acl 设置截图

3.4.3 结果分析

3.5 其它需要说明的问题

暂无问题。

3.6 参考文献

- [1] (美)James F. Kurose, (美)Keith W. Ross 著, 陈鸣译. 计算机网络: 自顶向下方法(原书 第 7 版). 北京: 机械工业出版社, 2018. 5
- [2] 华中科技大学计算机科学与技术学院. 基于 CPT 的组网实验指导手册

心得体会与建议

4.1 心得体会

这三次的实验自己动手做下来，对我的理论知识既有检验也有牢固的作用，纸上得来终究浅，期间我也查阅了很多资料，了解 Winsock 库的基本操作，探究三种流水线传输协议的区别，查询如何使用 CPT 实验中用到的软件等，这也极大锻炼了我的动手能力，每次做完一个实验都是一件令人开心的事情，感谢老师在我遇到困难的时候对我的指导！

4.2 建议

1. 实验二中 TCP 是一个简单的版本，是在 GBN 的基础上修改，我希望以后可以增加难度，例如可以给接收方增加一个缓存的空间（因为实际生活中 TCP 一般是缓存失序的报文），这样可以模拟实际的生活的情况。
2. 实验三中的指导书中可以增加指导如何无线上网的内容。
3. 建议可以提供报告模版的 Latex 版本。