

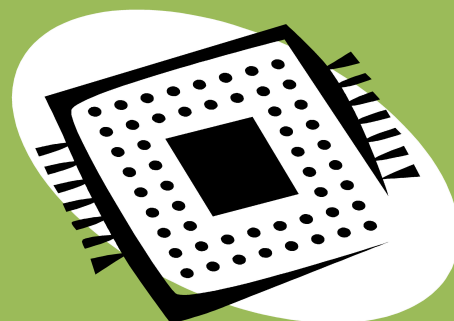
华中科技大学

2021

计算机网络

· 实验报告 ·

专 业:	计算机科学与技术
班 级:	CSXJ1902
学 号:	U201913821
姓 名:	朱旭鹏
电 话:	18307576517
邮 件:	1064687807@qq.com
完成日期:	2021-11-12



计算机科学与技术学院

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 校交 1902

姓名 朱旭鹏

学号 U201913821

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	3
1.4 系统实现	4
1.5 系统测试及结果说明	5
1.6 其他问题	7
我的思考	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	17
实验三 基于 CPT 的组网实验	18
3.1 环境	18
3.2 实验要求	18
3.3 各部分的实验实现细节	18

实验一 Socket 编程实验

1.1 开发环境

1.1.1 开发平台

- (1) 操作系统: MacOS Catalina 10.15.7;
- (2) 处理器: 2.3 GHz 八核 Intel Core i9;
- (3) 物理内存: 16 GB 2667 MHz DDR4;
- (4) IDE: IntelliJ IDEA 2020.1.4 (Community Edition)
Build #IC-201.8743.12, built on July 21, 2020
Runtime version: 11.0.7+10-b765.65 x86_64
- (5) JDK: jdk13.02

Use Project JDK (java version "13.0.2", path: /Library/Java/JavaVirtual...jdk-13.0.2.jdk/Contents/Home)

1.1.2 运行平台

任何平台下直接运行 `HttpServer.java` 即可, 所需的动态库及其他依赖已经一并打包到软件目录, 注意在运行之前检查一下 `classpath` 等路径问题, 因为运行这个 `java` 文件需要我写的其他类文件.

1.2 系统功能需求

题目:

编写一个 Web 服务器软件, 要求如下:

基本要求:

- ✧ 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);
- ✧ 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;
- ✧ 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。;
- ✧ 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:

- ✧ 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;
- ✧ 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);
- ✧ 在服务器端的屏幕上能够输出对每一个请求处理的结果;
- ✧ 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

本系统主要由以下四个模块构成:

- (1) 配置文件
- (2) 服务器
- (3) Request 分析

(4)Response 构建

1.4 系统实现

1.4.1 配置文件

★db.properties 配置文件

文件概述:这是一个配置文件,配置了服务器的主机地址和这个 Socket 运行服务的端口号,因为这个实验要规划一个 Socket,需要配置好主机地址和端口号

★GetProperties 类

文件概述:一个 static 类型的类,这个类在运行的时候处理 static 初始化块,找 db.properties 里面的主机地址和端口号信息,还找到了当前服务器的运行根目录.以后寻找服务器里面的文件就是 webroot(根目录)+url(请求的地址)

1.4.2 服务器

★概述

服务器概述:这个服务器是一个多线程的服务器,每当浏览器发起一个 Socket 连接请求的时候,这个时候服务器就可以建立一个线程,来处理这个 Socket 请求,总的来说就是一个 Socket 请求,服务器就建立一个线程,每个线程都建立一个 Socket 来接受浏览器发过来的 Socket 请求,连接建立了之后就可以进行通信.

★线程处理 Request 概述

初始化创建:首先创建两个 I/O 流,一个输入流,一个输出流

永真循环和异常处理:在 Socket 连接成功连接成功之后,线程并不会因为浏览器没有发送报文而被阻塞,每隔一个时间周期就会把输入流里面的内容全部提取出来交付到 Request 对象里面,这会分成两种情况.如果浏览器一直没有发送的话,那么就处理空报文.空报文会往上传递异常,处理之即可,只要别往 JVM 里面传即可.如果浏览器发送了请求,那么就到下一步.

Request 分析:Request 分析 HTTP 报文,提取出要处理的 url

Response 分析:Response 根据 url 构造 HTTP 响应报文

1.4.3 Request 类

★数据结构

private InputStream input;//输入流

private String uri;//这个 Request 的 uri

★构造函数

输入:一个输入流 input

将输入的 input 赋值给 this.input

★parse 函数

输入:无

输出:无

内部功能:首先先把传来的 byte[]转化成 String 类型的变量,接着找到 url.调用 parseUri 函数来获得 uri,然后把变量赋给 this.uri..

★parseUri 函数

输入:一个字符串,待处理的数据.

输出:一个字符串,处理得来的 Uri.

内部功能:找到第一个' '和第二个' '之间的那一部分,因为是报文的第一行就是 GET url

url 在第一个空格和第二个空格之间;如果找不到就返回 null.

★getTypes 函数

输入:无.

输出:一个字符串,代表申请的数据类型.

内部功能:根据 url 的最后部分来匹配请求的文件类型,使用 `endwith()` 方法再加上一个简单的分支结构即可.返回的字符串就是 HTTP 响应报文的 `content-type` 项.

1.4.4 Response 类

★数据结构

```
private final int BUFFER_SIZE = 1024; //最长发射数据量
```

```
Request request; //对应本 Response 的 Request 类
```

```
OutputStream output; //输出流
```

```
PrintStream writer; //写 HTTP 响应报文
```

★sendResource 函数

输入:响应类型 `format` 字符串.

输出:无.

内部功能:

1、寻找文件,根据我上面说过的,webroot+url 来寻找文件.

2、如果是没找到,就返回 404,报文的内容就是 "HTTP/1.1 404 File Not Found\r\n" + "Content-Type: text/html\r\n" + "Content-Length: 27\r\n" + "\r\n" + "<h1>404 File Not Found</h1>"; 如果找到了,先构造报文头 "HTTP/1.1 200 OK\r\n" + "Content-Type:" + formal + "\r\n" + "Content-Length:" + file.length() + "\r\n" + "\r\n" (注意 `formal` 的内容就是 1.4.3 第二部分返回的类型)

3、如果文件找到了,就以二进制流的方式输入报文后部.

4、输出比特流: `output.write(errorMessage.getBytes(StandardCharsets.UTF_8));`

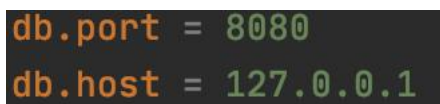
1.5 系统测试及结果说明

★基本信息

操作系统: MacOS Catalina 10.15.7;

浏览器: Safari 版本 14.0 (15610.1.28.1.9, 15610);

★基本参数设置



```
db.port = 8080
db.host = 127.0.0.1
```

图 1 - 1 基本参数设置

★访问服务器主页 <http://127.0.0.1:8080/index.html>, 如图 1-2:



图 1-2 服务器主页

★响应结果, 如下图 1-3:

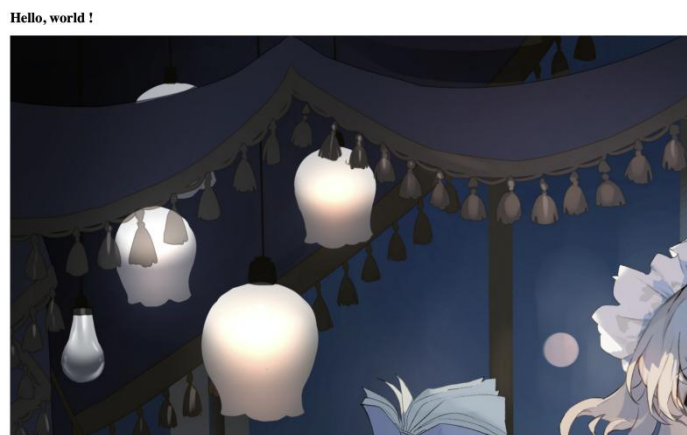


图 1-3 主页响应结果

★请求报文, 如下图 1-4:

```
GET /index.htm HTTP/1.1
Host: 127.0.0.1:8080
Upgrade-Insecure-Requests: 1
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/605.1.15 (KHTML, like Gecko) Version/14.0 Safari/605.1.15
Accept-Language: zh-cn
Accept-Encoding: gzip, deflate
Connection: keep-alive

GET /1.jpg HTTP/1.1
Host: 127.0.0.1:8080
Connection: keep-alive
Accept: image/png,image/svg+xml,image/*;q=0.8,video/*;q=0.8,*/*;q=0.5
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/605.1.15 (KHTML, like Gecko) Version/14.0 Safari/605.1.15
Referer: http://127.0.0.1:8080/index.html
Accept-Encoding: gzip, deflate
```

图 1-4 请求报文

★响应报文, 如下图 1-5:

```
HTTP/1.1 200 OK
Content-Type:text/html
Content-Length:174
```

图 1-5 响应报文

★ [index.html](http://127.0.0.1:8080/index.html), 如下图 1-6:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Title</title>
</head>
<body>
<h1>Hello, world !</h1>

</body>
</html>
```

图 1 - 6 index.html

★文件不存在响应结果，如下图 1-7:

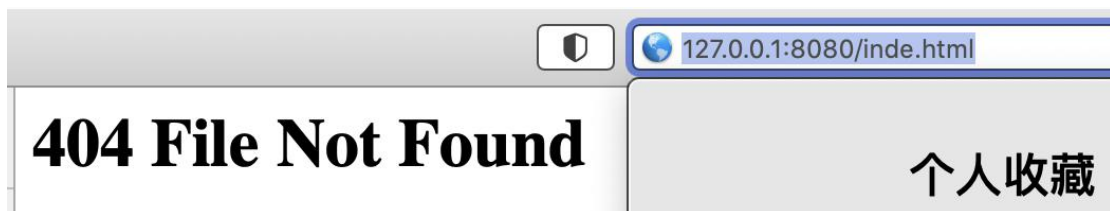


图 1 - 7 文件不存在 404 响应

★访问根目录,如下图 1-8:



图 1 - 8 403 响应

★多客户端并发，可从响应时间看，如下图 1 - 9:

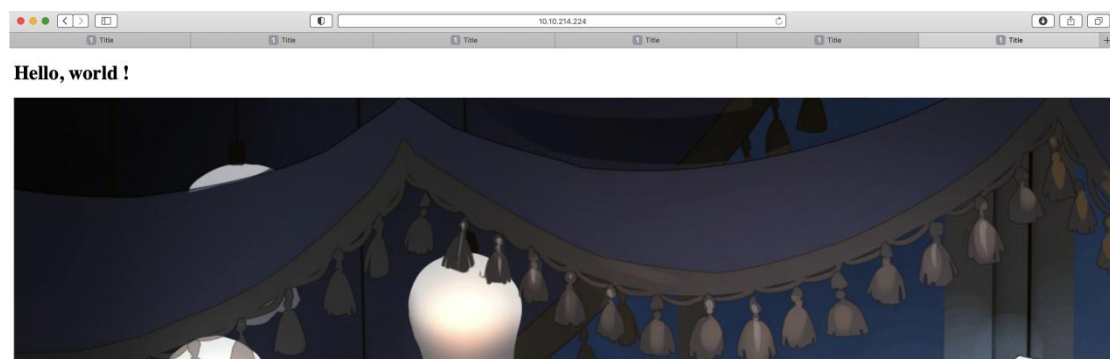
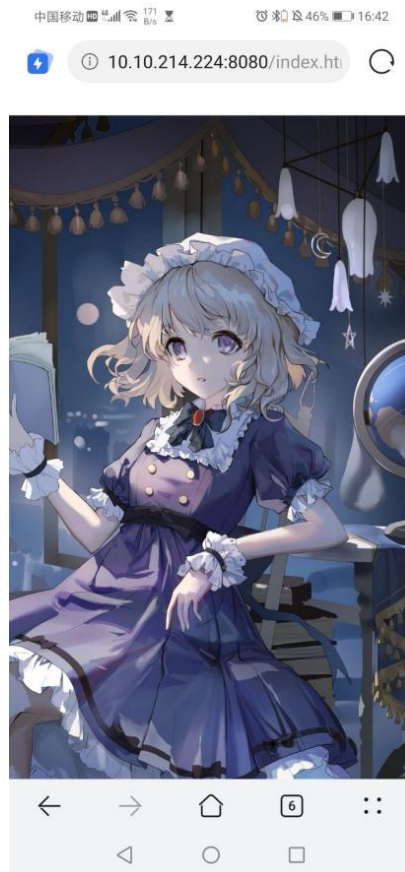


图 1 - 9 多客户端并发响应结果

★局域网访问（手机访问），如下图 1 - 10: 配置成内网地址了



/10.10.233.136

图 1 - 10 手机访问 index.html 和我手机的 IP 地址

1.6 其他问题



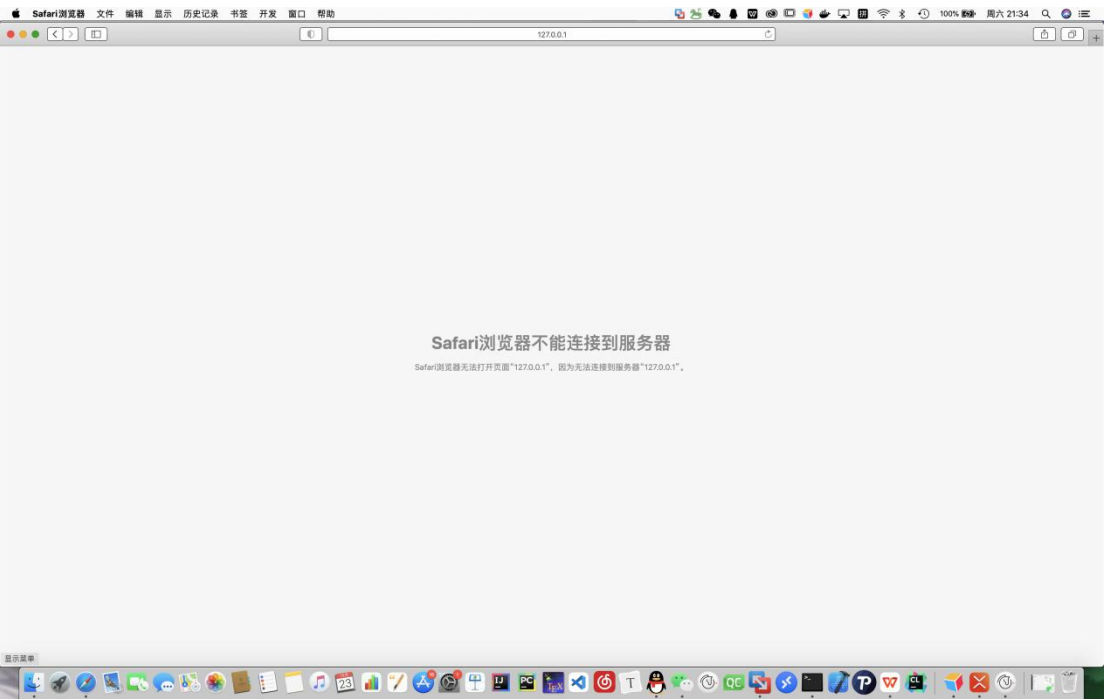
图 1 - 11 访问结果

星期三(10月20日)我发现:Socket 程序已经关闭,为什么还是可以显示 403 呢?



我的思考

今天(10月23日)发现这个关闭 Socket 的情况下已经不能打开了



实验二 数据可靠传输协议设计实验

2.1 环境

- (1) 操作系统: MacOS Catalina 10.15.7;
- (2) 处理器: 2.3 GHz 八核 Intel Core i9;
- (3) 物理内存: 16 GB 2667 MHz DDR4;
- (4) IDE: Microsoft Visual Studio Community 2019;
- (5) 依赖库: 老师所发模拟网络环境 netsimlib.lib。

2.2 实验要求

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, 分值为 50%。
- (2) 实现基于 SR 的可靠传输协议, 分值为 30%。
- (3) 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。分值 20%。

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) GBNRdtSender

★数据结构

int base;//发送窗口的 base

int nextseqnum;//发送窗口的 nextseqnum

bool waitingState;//是否处于等待 ACK 状态

Packet win[len];//发送窗口, 窗口大小为 8

int num_packet_win;//记录窗口中的报文个数

base 就是发送窗口第一个分组的序号,发送窗口里面有 len 个分组,分组的序号是 base+i,(i 是数组的下标,即下标为 0 的是序号 base,下标为 1 的是序号 base+1)

★bool GBNRdtSender::send(Message & message)

输入:应用层消息 Message.

输出:消息是否发送成功.

内部功能:发送消息:

(1)如果期望发送的下一个报文序号 $\text{nextseqnum} < \text{base} + \text{len}$ 的话,也就是说滑动窗口有位置给我们放新的分组,找到第一个空闲的位置,把存在里面的 Packet 元素进行调整,把检查和置 0,期待接收到的 seqnum 置为当前发送分组的标号,ack 标记为没收到.(就是把 win[num_packet_win](滑动窗口第一个空闲量)进行处理,求出这个分组的序号,也是标记窗口的这个位置发送的是什么分组),把消息放进窗口的这个位置里面,这个时候窗口的这个位置存在的 Packet 元素存放的是发出这个分组的内容,分组序号,以及 ack 的接受情况.

(2)计算出检查和,打开计时器,发送报文.

(3)报文发送后窗口分组元素加 1,并判断是不是已经满了.如果满了要做更改.

(4)如果一开始窗口就是满的,可以拒绝应用层的请求.

★void GBNRdtSender::receive(Packet &ackPkt)

输入:网络层分组 Packet.

输出:无

内部功能:接受 ACK.

(1)首先计算检验和,如果检验和没问题再继续下面的操作.

(2)记录下收到的 ack 序号,这个时候我们就用收到的 ack 数量减去 base+1 代表滑动窗口中有几个元素确认收到了,例如 base=1, ack1 丢失但是收到了 ack2, 也可以算 ack1 收到了.不存在没发送就收到这个 ack 的情况.

(3)然后判断一下滑动窗口里面是不是为空,判断的方法有两种:一个是收到的 ack 序号是不是等于最近一次发送的分组序号,还有就是看滑动窗口有没有元素.这里选择前者.

(4)处理一下发送方滑动窗口内元素数量和发送方滑动窗口内元素(向前移)

★void GBNRdtSender::timeoutHandler(int seqNum)

输入:超时的分组.

输出:无

内部功能:超时处理.

发送方定时器时间到,一般来说,定时器时间到一般都是第一个元素的时间到了,所以说这个时候,重新启用第一个元素的定时器.重发上一轮发送的报文,,然后重新发送窗口内的所有元素.

★bool GBNRdtSender::getWaitingState()

输入:无.

输出:当前滑动窗口是否满

内部功能:直接返回即可

(2) GBNRdtReceiver

★数据结构

int seq; //期待收到的下一个报文序号

Packet lastAckPkt; //上次发送的确认报文

★void GBNRdtReceiver::receive(Packet & packet)

输入:网络层分组 Packet.

输出:无

内部功能:收到 Sender 发出去的数据

(1)确认检验和是不是期望收到的报文序号,如果是的话,计算 ACK 的序号和检验和,发送 ACK.

(2)如果并不是,那就重新发送上一次发送的 ACK.

(3) GBN 协议验证

a) 执行结果，如下图 2 - 1:

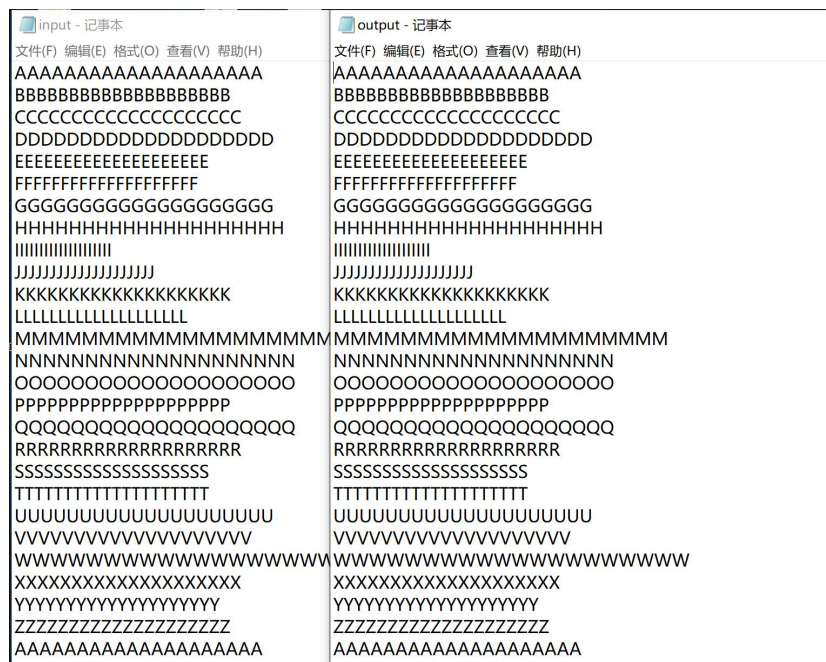


图 2 - 1 GBN 执行结果

b) 检查脚本检查结果如下图 2 - 2:

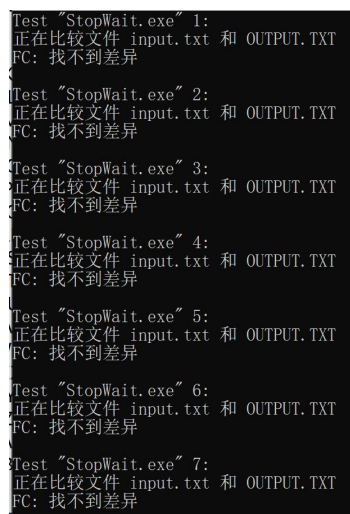


图 2 - 2 GBN 检查脚本检查结果

(4) GBN 结果分析

a) 滑动窗口变化，如下图 2 - 3: 就一直发送 ACK1

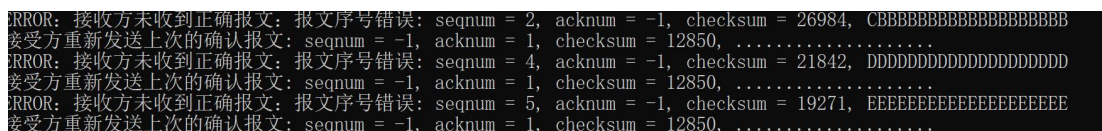


图 2 - 3 滑动窗口变化

b) 超时重传，如下图 2 - 4: (超时的话就发送 base)

```

ERROR: 接收方未收到正确报文: 报文序号错误: seqnum = 2, acknum = -1, checksum = 26984, CBBBBBBBBBBBBBBBBBBB
接受方重新发送上次的确认报文: seqnum = -1, acknum = 1, checksum = 12850, .....
ERROR: 接收方未收到正确报文: 报文序号错误: seqnum = 4, acknum = -1, checksum = 21842, DDDDDDDDDDDDDDDDDDD
接受方重新发送上次的确认报文: seqnum = -1, acknum = 1, checksum = 12850, .....
ERROR: 接收方未收到正确报文: 报文序号错误: seqnum = 5, acknum = -1, checksum = 19271, EEEEEEEEEEEEEEEEEEE
接受方重新发送上次的确认报文: seqnum = -1, acknum = 1, checksum = 12850, .....
发送方定时器时间到, 重发上一次发送的报文: seqnum = 2, acknum = -1, checksum = 26984, BBBBBBBBBBBBBBBBBBBB

```

图 2 - 4 超时重传情况

c) 综上, GBN 设计正确。

2.3.2 SR 协议的设计、验证及结果分析

(1) SRRdtSender

★数据结构

```

const int seqsize;

const int wndsize;//窗口长度

Packet* const sendBuf;//发送缓冲区, 避免反复构造析构

bool* const bufStatus;//保存 ACK 的接受状况

int base, nextSeqnum;//base 和期待的下一个分组序号

bool WaitingState;//状态

```

这个的数据结构状态就是这样子的,有一个长度为 seqsize 的一个数组,然后一个 wndsize 的滑动窗口在长度为 seqsize 的数组里面滑动

```

[0 1 2 3] 4 5 6 7
0 [1 2 3 4] 5 6 7
0 1 [2 3 4 5] 6 7
0 1 2 [3 4 5 6] 7
0 1 2 3 [4 5 6 7]
0] 1 2 3 4 [5 6 7

```

总之,元素就在这样的滑动窗口里面来回循环,就是一个长度为 seqsize 的数组,取其中连续 wndsize 个元素。

★bool SRRdtSender::send(Message & message)

输入:应用层消息 Message.

输出:消息是否发送成功.

内部功能:发送消息:

(1)如果期望发送的下一个报文序号 $nextseqnum < base + len/2$ 的话,也就是说滑动窗口有位置给我们放新的分组,找到第一个空闲的位置,把存在里面的 Packet 元素进行调整,把检查和置 0,期待接收到的 seqnum 置为当前发送分组的标号,ack 标记为没收到.(就是把 win[num_packet_win](滑动窗口第一个空闲量)进行处理,求出这个分组的序号,也是标记窗口的这个位置发送的是什么分组),把消息放进窗口的这个位置里面,这个时候窗口的这个位置存在的 Packet 元素存放的是发出这个分组的内容,分组序号,以及 ack 的接受情况.下一个发送的分组序号+1.

(2)计算出检查和,打开计时器,发送报文.

(3)报文发送后窗口分组元素加 1,并判断是不是已经满了.如果满了要做更改.

(4)如果一开始窗口就是满的,可以拒绝应用层的请求.

★void SRRdtSender::receive(Packet &ackPkt)

输入:网络层分组 Packet.

输出:无

内部功能:接受 ACK.

确认 ACK 的这一个部分比 GBN 复杂一点:

(1)首先还是经典检查检验和,检验和不对就报错.

(2)标记这个分组序号的 bufStatus 表示已经收到了这个分组的 ACK.

(3)如果是收到了 base 的 ACK,那么从前往后进行下面的操作,如果这个分组序号的 ACK 已经收到,那么滑动窗口里面的状态清空,base 往后移动,一直到遇到第一个没有收到 ACK 的分组序号.

//while 循环在遇到第一个没收到的分组就停止.

```
while (bufStatus[base] == true)
    {
        //移动 base
        bufStatus[base] = false;
        base = (base + 1) % seqsize;
    }
```

(4)这个程序的简单就体现在,我开辟一个足够大的空间来维护 ACK 收取的状态,滑动窗口的时候,就不需要做额外的处理,简单许多.

★void SRRdtSender::timeoutHandler(int seqNum)

输入:超时的分组.

输出:无

内部功能:超时处理.

(1)找到超时的 Packet 在滑动窗口里面的位置.

(2)重新发送之.

★bool SRRdtSender::getWaitingState()

输入:无.

输出:当前滑动窗口是否满

直接返回 WaitingState.

(2) SRRdtReceiver

★数据结构

const int wndsize;//窗口大小

const int seqsize;

Packet lastAckPkt;//上一个收到的 ACK

Packet* const recvBuf;//分组缓存区

```
bool* const bufStatus;//分组的状态
```

```
int base;//int nextSeqnum;
```

和上面是一样的

★void GBNRdtReceiver::receive(Packet & packet)

输入:网络层分组 Packet.

输出:无

内部功能:收到 Sender 发出去的数据

(1)首先判断检验和,不对就不理.

(2)接着判断是不是在滑动窗口内,如果不是的话就传 ACK(n),注意 n 是收到的元素的序列号.这种窗口设计的好处就是,收到的元素如果不再滑动窗口内,一定是在 base-N 到 base 这个范围内.

(3)如果在滑动窗口内部,那就标记收到了,接着看看收到的元素是不是 base 元素,如果是的话,滑动窗口可以以后移了!

```
while (bufStatus[base] == true)
{
    Message msg;
    memcpy(msg.data, recvBuf[base].payload, sizeof(recvBuf[base].payload));
    pns->deliverToAppLayer(RECEIVER, msg);
    pUtils->printPacket("递交到应用层的数据:", recvBuf[base]);
    bufStatus[base] = false;
    base = (base + 1) % seqsize;
}
```

(3) SR 协议验证:略

2.3.3 简单 TCP/IP 协议的设计、验证及结果分析

(1) TcpRdtSender (基于 GBNRdtSender)

★数据结构

相比 GBN, 增加 count 检测冗余情况, 如下:

```
int base;//发送窗口的 base
```

```
int nextseqnum;//发送窗口的 nextseqnum
```

```
bool waitingState;//是否处于等待 ACK 状态
```

```
Packet win[len];//发送窗口, 窗口大小为 4
```

```
int num_pac_win;//记录窗口中的报文个数
```

```
int count;//记录冗余 ACK 数目
```

```
int current_rcv_ack;//现在收到的 ACK 序号
```

```
int last_rcv_ack;//上一次收到的 ACK 序号
```

★bool TcpRdtSender::send(Message & message)

输入:应用层消息 Message.

输出:消息是否发送成功.

内部功能:发送消息:

和 GBN 的做法是一样的.

(1)如果期望发送的下一个报文序号 $\text{nextseqnum} < \text{base} + \text{len}$ 的话,也就是说滑动窗口有位置给我们放新的分组,找到第一个空闲的位置,把存在里面的 Packet 元素进行调整,把检查和置 0,期待接收到的 seqnum 置为当前发送分组的标号,ack 标记为没收到.(就是把 $\text{win}[\text{num_packet_win}]$ (滑动窗口第一个空闲量)进行处理,求出这个分组的序号,也是标记窗口的这个位置发送的是什么分组),把消息放进窗口的这个位置里面,这个时候窗口的这个位置存在的 Packet 元素存放的是发出这个分组的内容,分组序号,以及 ack 的接受情况.

(2)计算出检查和,打开计时器,发送报文.

(3)报文发送后窗口分组元素加 1,并判断是不是已经满了.如果满了要做更改.

(4)如果一开始窗口就是满的,可以拒绝应用层的请求.

★void TcpRdtSender::receive(Packet &ackPkt)

输入:网络层分组 Packet.

输出:无

内部功能:接受 ACK.

(1)首先计算检验和,如果检验和没问题再继续下面的操作.

(2)记录下收到的 ack 序号,这个时候我们就用收到的 ack 数量减去 $\text{base}+1$ 代表滑动窗口中有几个元素确认收到了,例如 $\text{base}=1$, ack1 丢失但是收到了 ack2, 也可以算 ack1 收到了.不存在没发送就收到这个 ack 的情况.还有一种情况就是当前收到的 ACK(current_ack 和上一次收到的 ACK 是一样的话就标记为冗余 ACK,当冗余 ACK 的数量为 4 的时候,就代表收到过 3 个冗余 ACK,那么就进行快速重传,如果这个 ACK 不是冗余的 ACK,则清空计数)

```
if (ackPkt.acknum == this->win[0].seqnum)//判断现在的序号和上一次的序号是否相同
{
    this->count++;//计数+1
    if (count == 4)//如果 ack 四次、则重传当前窗口的第一个报文
    {
        pns->stopTimer(SENDER, this->win[0].seqnum);
        pns->sendToNetworkLayer(RECEIVER, this->win[0]);//将第一个报文发送给接收方
        pUtils->printPacket("\n 冗余 ACK*4, 快速重传当前窗口第一个报文", win[0]);
        pns->startTimer(SENDER, Configuration::TIME_OUT, this->win[0].seqnum);
        printf("\n 冗余 ACK%d *4 \n", ackPkt.acknum);
        this->count = 0;
        return;
    }//冗余 ack 快速重传
}
```

(3)然后判断一下滑动窗口里面是不是为空,判断的方法有两种:一个是收到的 ack 序号是不是等于最近一次发送的分组序号,还有就是看滑动窗口有没有元素.这里选择前者.

(4)处理一下发送方滑动窗口内元素数量和发送方滑动窗口内元素(向前移)

★void TcpRdtSender::timeoutHandler(int seqNum)

输入:超时的分组.

输出:无

内部功能:超时处理.

TCP 采用快速重传, 配合 receiver 方法, 数据包超时时只重传最早未确认的包 (即 base) 即可。

★bool TcpRdtSender::getWaitingState()

输入:无.

输出:当前滑动窗口是否满

返回 WaitingState 即可.

(2) TCPRdtReceiver

和 GBN 完全相同

(3) TCP 验证

a) 运行情况, 如下图 2 - 9:

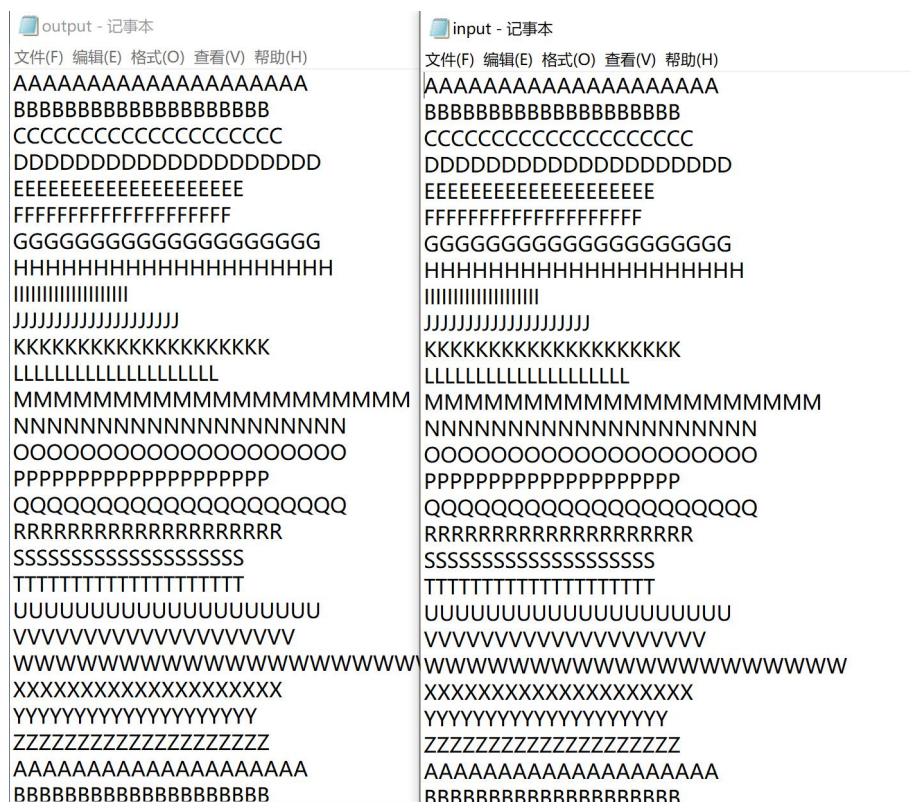


图 2 - 9 TCP 执行情况

b) 检查脚本检查如下图 2 - 10:

```
C:\Windows\system32\cmd.exe
Test "TCP.exe" 1:
正在比较文件 input.txt 和 OUTPUT.TXT
FC: 找不到差异

Test "TCP.exe" 2:
正在比较文件 input.txt 和 OUTPUT.TXT
FC: 找不到差异

Test "TCP.exe" 3:
正在比较文件 input.txt 和 OUTPUT.TXT
FC: 找不到差异

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
```

图 2 - 10 检查脚本检查 TCP 情况

(4) TCP 结果分析

滑动窗口变化和 GBN 相同，因此只给出超时重传以及冗余重传截图。

a) 超时重传，如下（只重传最早未确认的包）图 2 - 11:

```
the current window's rtt number is 0
发送方定时器超时，重新发送报文段: seqnum = 5, acknum = -1, checksum = 19271, EEEEEEEEEEEEEEEEEEE
接收方正确收到报文: seqnum = 5, acknum = -1, checksum = 19271, EEEEEEEEEEEEEEEEEEE
```

图 2 - 11 TCP 超时重传情况

b) 冗余重传，如下（收到连续三个冗余 ack）图 2 - 12:

```
冗余ACK*4, 快速重传当前窗口第一个报文: seqnum = 4, acknum = -1, checksum = 21842, DDDDDDDDDDDDDDDDDDD
冗余ACK *4
```

图 2 - 12 TCP 冗余超时情况

c) 综上，TCP 协议正确。

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

- (1) 操作系统: MacOS Catalina 10.15.7;
- (2) 处理器: 2.3 GHz 八核 Intel Core i9;
- (3) 物理内存: 16 GB 2667 MHz DDR4;
- (4) 软件: Cisco Packet Tracer 7.3.0838

3.2 实验要求

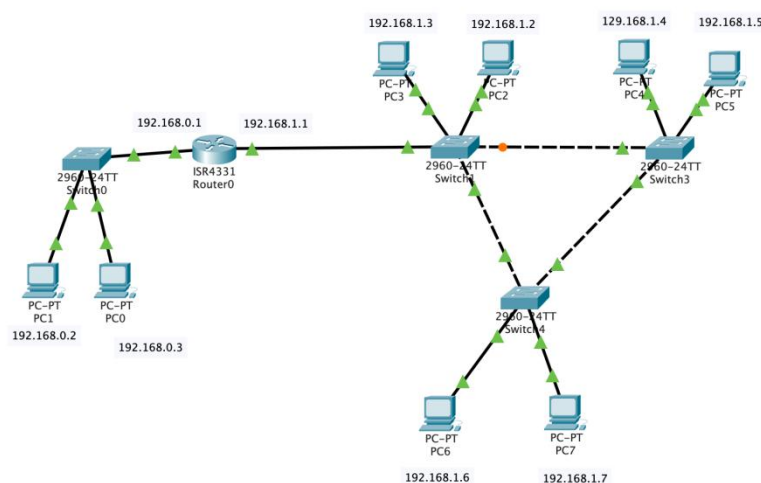
熟悉 Cisco Packet Tracer 仿真软件, 利用 Cisco Packet Tracer 仿真软件完成实验内容。

3.3 各部分的实验实现细节

第一部分

★第一部分 1-1 问题

- a) 将 PC1、PC2 设置在同一个网段, 子网地址是: 192.168.0.0/24;
- b) 将 PC3~PC8 设置在同一个网段, 子网地址是: 192.168.1.0/24;
- c) 为路由器配置端口地址, 使得两个子网内部的各 PC 机之间可以自由通信。



★第一部分 1-1 解决策略

分成两个子网,每个主机和路由器端口 IP 全部已经配置好.地址显示在旁边,刚好可以满足内部通信的需求.子网 1 的两个 PC 可以连接到 192.168.0.1,然后经过路由器的转发转发到网络号为 192.168.1.0 的网络中.路由器的作用就是使得两个网络之间的 PC 能够互通.

★第一部分 1-2 问题

- a) 将 PC1、PC2 设置在同一个网段, 子网地址是: 192.168.0.0/24;
- b) 将 PC3、PC5、PC7 设置在同一个网段, 子网地址是: 192.168.1.0/24;

- c) 将 PC4、PC6、PC8 设置在同一个网段，子网地址是：192.168.2.0/24；
- d) 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；
- e) 测试各 PC 之间的连通性，并结合所学理论知识进行分析；
- f) 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。

★第一部分 1-2 解决策略

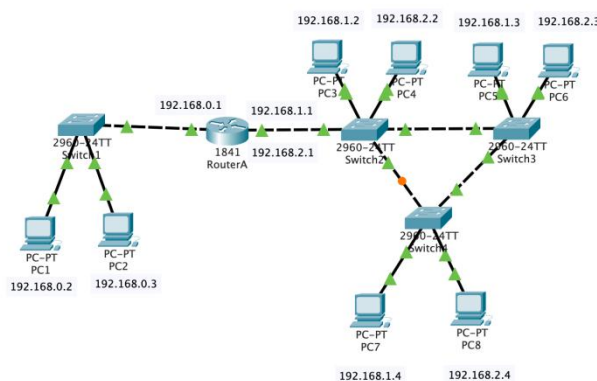
(1) 在基本内容 1 的基础上，将 PC4、PC6、PC8 从子网 192.168.1.0/24 分离，编入 192.168.2.0/24 子网段，所以对这三台 PC 重新配置 IP。PC4 配置为 192.168.2.2，PC6 配置为 192.168.2.3，PC8 配置为 192.168.2.4，三台 PC 子网掩码都为 255.255.255.0

(2) 在交换机 1 上添加一个 VLAN2，并且把 PC1 和 PC2 添加到 VLAN2 中。在交换机 2、3、4 中添加 VLAN3 和 VLAN4，并且把 PC3 添加到交换机 2 的 VLAN3 中，PC4 添加到交换机 2 的 VLAN4 中，PC5 添加到交换机 3 的 VLAN3 中，PC6 添加到交换机 3 的 VLAN4 中，PC7 添加到交换机 4 的 VLAN3 中，PC8 添加到交换机 4 的 VLAN4 中。因为上面的情况是：所有的相同子网的都在一块，在这个情况下相同的链路层交换机可能连着两个不同子网的 PC 主机，所以说要配置虚拟局域网 VLAN。因为有链路层交换机是互相相连的，在配置 VLAN 端口通讯的时候，要设置一个 TRUNK 端口来连接链路层交换机。

(3) 在三个子网段内部，PC1 和 PC2 能够自由通信，PC3、PC5、PC7 能够自由通信，PC4、PC6、PC8 能够自由通信，但三个网段之间无法通信，这是由于路由器只有一个端口 fa0/1，但需要接入两个子网。

(4) 路由器 vlan 配置：由于路由器只有两个端口，fa0/1 端口所管理的网络中有两个网段的 pc 机，所以要为该端口分配子端口，并为子端口分配 vlan，才能使两个 vlan 的主机可以通信。对路由器 A 的快速以太网接口 0/1，创建子接口 fa0/1.1，并将其划分到 vlan3 中，ip 地址设置为 192.168.1.1。创建子接口 fa0/1.2，将其划入 vlan4，ip 地址设置为 192.168.2.1。路由器的 fa0/0 接口创建一个子接口 fa0/0.1，划分到 vlan2，ip 设置为 192.168.0.1，pc1 和 pc2 的 vlan 划分完成。

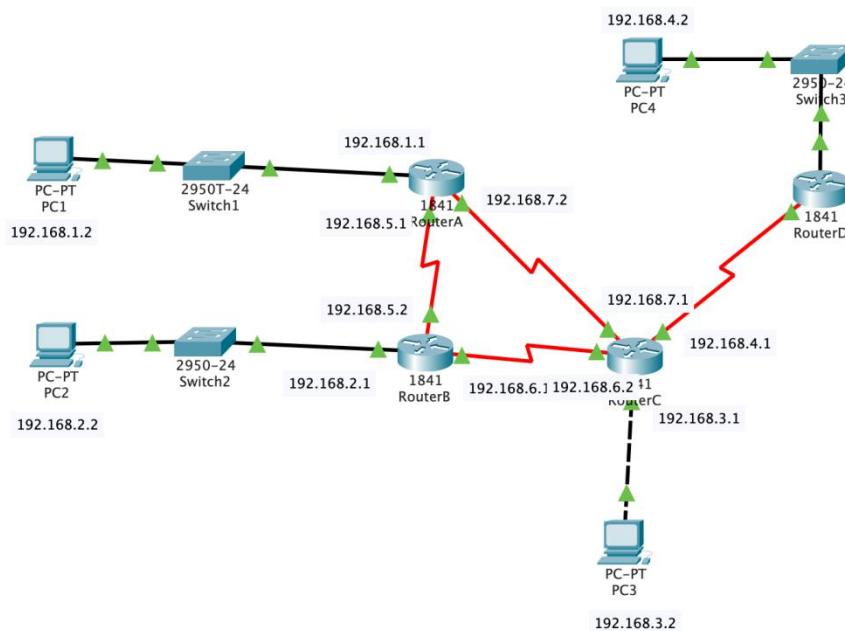
CPI 语句：进入子端口设置状态 interface type number.subinterface 在这个命令环境下配置子端口



第二部分

★第二部分 2-1 问题

- a) 将 PC1 设置在 192.168.1.0/24 网段;
- b) 将 PC2 设置在 192.168.2.0/24 网段;
- c) 将 PC3 设置在 192.168.3.0/24 网段;
- d) 将 PC4 设置在 192.168.4.0/24 网段
- e) 设置路由器端口的 IP 地址
- f) 在路由器上配置 RIP 协议, 使各 PC 机能互相访问



★第二部分 2-1 解决策略

主机和路由器端口的设置如上:

一般来说,路由器的每个端口一定都属于不同的网段,路由器如果端口和端口相连的情况下,这两个端口单独属于一个自己的网络.配置子网先配置主机的网络,然后再对路由器每一个端口进行配置,每个端口都与直接相连的节点具有相同的网络序号.假如说找不到与之对应的已知 IP 的主机与之相连,那么使用一个新的网络号.

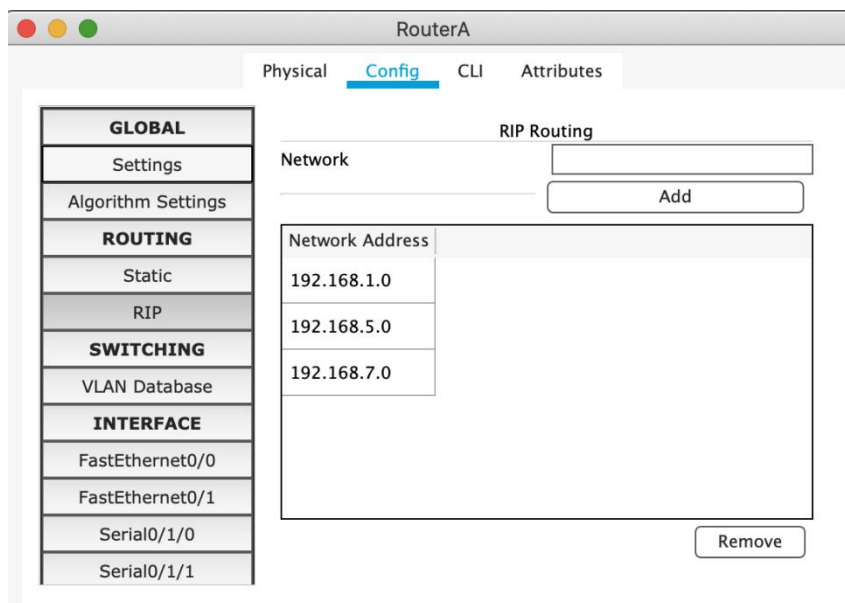
(1) 配置路由器 IP, 将四个路由器管理的 PC 机的四个子网分别进行配置, 四个子网分别处于 192.168.1.0、192.168.2.0、192.168.3.0、192.168.4.0 网段, 所以路由器 A 的左端口 IP 为 192.168.1.1, 路由器 B 的左端口 IP 为 192.168.2.1, 路由器 C 的下方端口 IP 为 192.168.3.1, 路由器 D 的上方端口 IP 为 192.168.4.1.

路由器之间相连的端口也需要配置网段, 选择 192.168.5.0、192.168.6.0、192.168.7.0、192.168.8.0 四个子网为路由器之间的四条链路进行 IP 分配.

(2) 配置 PC 机 IP 地址和网关, PC1-PC4 分别处于 192.168.1.0、192.168.2.0、192.168.3.0、192.168.4.0

网段，可将其 IP 地址分别设置为 192.168.1.2、192.168.2.2、192.168.3.2、192.168.4.2。每台 pc 的网关都应该是与它距离最近的路由器的 IP 端口的地址，设为 192.168.1.1、192.168.2.1、192.168.3.1、192.168.4.1，这分别是路由器 A、B、C、D 的 fa 端口地址。

(3) 配置路由器 RIP 协议，RIP 协议是用于自治系统内的动态路由协议，它只和与自己相连的路由器交换信息，所以每个路由器配置该协议时，只需要把自己每个端口的 IP 地址所在网段填上即可。以路由器 A 为例，它的 fa 端口 IP 为 192.168.1.1，两个 serial 端口 IP 分别为 192.168.5.1 和 192.168.7.1，所以应该配置这三个网段。选择：配置->路由配置->RIP，在方框中填入三个 IP 地址对应的网段，点击添加。



★第二部分 2-2 问题

- a) 将 PC1 设置在 192.168.1.0/24 网段;
- b) 将 PC2 设置在 192.168.2.0/24 网段;
- c) 将 PC3 设置在 192.168.3.0/24 网段;
- d) 将 PC4 设置在 192.168.4.0/24 网段
- e) 设置路由器端口的 IP 地址
- f) 在路由器上配置 OSPF 协议，使各 PC 机能互相访问

★第二部分 2-2 解决策略

除配置路由 OSPF 协议与 RIP 协议不同外，其他配置均与 RIP 协议相同。OSPF 协议是区别于 RIP 协议的另一种选路协议，同样可以用于以太网的通信，我们采用命令行来为路由器配置 OSPF 协议。以路由器 A 为例，任选一个数字作为进程号，为路由器配置 OSPF，将路由器端口的 IP 地址和子网掩码绑定到路由器上。

```
Router#conf t //进入全局配置模式
```

```
Router(config)#router ospf 1 //选择 ospf 协议
```

```
Router(config-router)#network 192.168.5.0 0.0.0.255 area 0
```

```
Router(config-router)#network 192.168.7.0 0.0.0.255 area 0
```

//0.0.0.255 是翻转掩码, 0 和 1 设置刚好和子网掩码相反

```
Router(config-router)#network 192.168.1.0 0.0.0.255 area 0
```

```
Router(config-router)#end
```

```
Router#copy run startup
```

这里选择了 1 号进程来运行 OSPF 协议,配置好运行 OSPF 协议的网段之后就可以运行之.

★第二部分 2-3 问题

- a) 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。
- b) 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。

★第二部分 2-3-1 解决策略

用命令行对路由器 A 进行配置。要使得 pc1 无法访问其他网段, 而且不能被其他网段访问, 应该在路由器 A 与交换机 0 相连的端口进行配置。先用 access-list 命令创建访问控制列表, 可选用 100 以内的数字作为 acl 编号, deny 表示屏蔽某网段的消息, permit 表示接受某网段的消息, 这里使用了 deny, 因为我们要屏蔽 pc1 的通信。命令中的 ip 地址是要屏蔽或接受的网段, 最后一个参数是子网掩码的反码。创建 acl 完成, 打开端口 fa0/0, 用 access-group 命令把 acl 绑定到路由器上, acl 就配置完成了。

```
Router#conf t
```

```
Router(config)#access-list 10 deny 192.168.1.0 0.0.0.255
```

```
Router1(config-if)#interface fa0/0
```

```
Router1(config-if)#ip access-group 10 in
```

★第二部分 2-3-2 解决策略

① pc1 不能访问 pc2, 但能访问其他 pc。第一次配置 acl 使只使用了 deny 指令, 所以只能屏蔽作用, 这里既需要屏蔽一部分消息, 又需要允许一部分消息通过, 所以还要使用 permit 指令。

② 仍然对路由器 A 进行配置, 如图 3-36, 使用 36 作为 acl 号, 当然也可以使用其他数字。首先 deny 来自 pc2, 也就是 192.168.2.0 网段的信息, 然后 permit 其他任何信息, acl 列表就创建好了, 然后绑定到端口 fa0/0 上。

```
Router#conf t
```

```
Router(config)#access-list 10 deny 192.168.2.0 0.0.0.255
```

```
Router1(config-if)#access-list 10 permit any
```

```
Router1(config-if)#interface fa0/0
```

```
Router1(config-if)#ip access-group 10 in
```


第三部分

★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

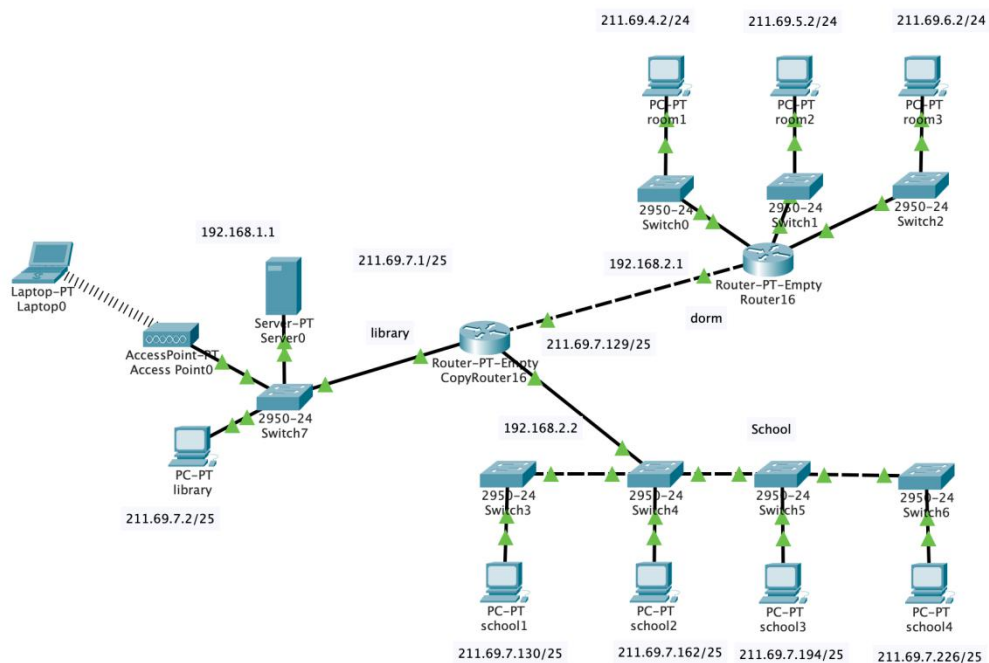
- 图书馆能够无线上网
- 学院之间可以相互访问
- 学生宿舍之间可以相互访问
- 学院和学生宿舍之间不能相互访问
- 学院和学生宿舍皆可访问图书馆。

★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。

★组网

每个宿舍 200 台主机, $128 \leq 200 \leq 256$, 所以说每个宿舍分长度为 8 的地址块, 学校有 4 个, 就分出去 3 个长度为 8 的地址块. 剩下 256 个地址, 一个长度为 7 的地址块分给了图书馆, 剩下那个长度为 7 的地址块分给每个学院:

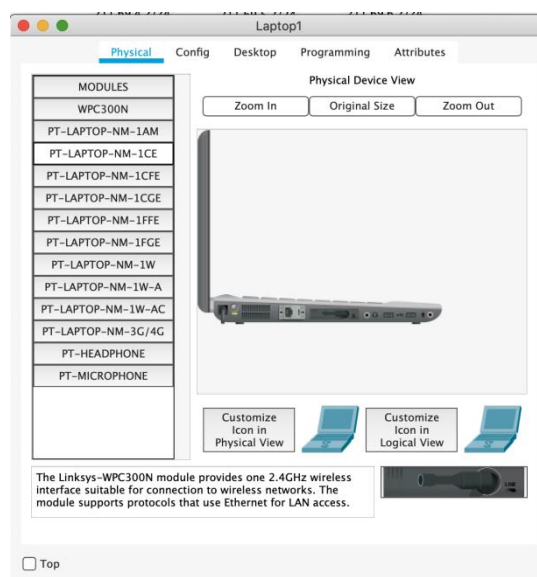


	IP 地址	网关	分配的 IP 数量
宿舍 1	211.69.4.0/24 211.69.4.2-211.69.4.255	211.69.4.1	254
宿舍 2	211.69.5.0/24 211.69.5.2-211.69.5.255	211.69.5.1	254
宿舍 3	211.69.6.0/24 211.69.6.2-211.69.6.255	211.69.6.1	254
图书馆	211.69.7.0/25 211.69.7.2-211.69.7.127	211.69.7.1	126
学院 1	211.69.7.128/27 211.69.7.130-211.69.7.159	211.69.7.129	30
学院 2	211.69.7.160/27 211.69.7.162-211.69.7.191	211.69.7.129	30
学院 3	211.69.7.192/27 211.69.7.194-211.69.7.223	211.69.7.129	30
学院 4	211.69.7.224/27 211.69.7.226-211.69.7.255	211.69.7.129	30

这一步组网确保学院,图书馆和宿舍之间能够互相互联。

★无线上网

这个是让 Laptop 插上无线网卡的图片



★DHCP 配置

这个是配置 DHCP 服务器:由快速以太网接口 f0/0 接入.

Pool Name	Default Gateway	DNS Server	Start IP Address	Subnet Mask	Max Users	TFTP Server	WLC Address
serverPool	211...	0.0.0.0	211...	255...	120	0.0.0.0	0.0.0.0

★RIP 配置

还是与之前那部分一样,在可视化的徒图形界面上配置 RIP:

Network Address
192.168.2.0
211.69.7.0

★ACL 配置

在与学院相连的那个端口配置 ACL:在与学院接入的端口上屏蔽去往 3 个宿舍网络的分组.

Router#conf t

Router(config)#**access-list 10 deny 211.69.4.0 0.0.0.255**

Router(config)#**access-list 10 deny 211.69.5.0 0.0.0.255**

Router(config)#**access-list 10 deny 211.69.6.0 0.0.0.255**

Router1(config-if)#**access-list 10 permit any**

Router1(config-if)#**interface fa0/0**

Router1(config-if)#**ip access-group 10 in**