**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 CS2207 姓名 瞿明睿 学号 U202215561

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 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](#_Toc185789216)

[**1.1 环境** 1](#_Toc185789217)

[**1.2 系统功能需求** 1](#_Toc185789218)

[**1.3 系统设计** 2](#_Toc185789219)

[**1.4 系统实现** 2](#_Toc185789220)

[**1.5 系统测试及结果说明** 4](#_Toc185789221)

[**1.6 其它需要说明的问题** 5](#_Toc185789222)

[实验二 数据可靠传输协议设计实验 7](#_Toc185789223)

[**2.1 环境** 7](#_Toc185789224)

[**2.2 实验要求** 7](#_Toc185789225)

[**2.3 协议的设计、验证及结果分析** 7](#_Toc185789226)

[**2.4 其它需要说明的问题** 17](#_Toc185789227)

[实验三 基于CPT的组网实验 18](#_Toc185789228)

[**3.1 环境** 18](#_Toc185789229)

[**3.2 实验要求** 18](#_Toc185789230)

[**3.3 基本部分实验步骤说明及结果分析** 19](#_Toc185789231)

[**3.4 综合部分实验设计、实验步骤及结果分析** 25](#_Toc185789232)

[**3.5** **其它需要说明的问题** 29](#_Toc185789233)

[心得体会与建议 30](#_Toc185789234)

[**4.1 心得体会** 30](#_Toc185789235)

[**4.2 建议** 30](#_Toc185789236)

# 实验一 Socket编程实验

## **1.1 环境**

### **1.1.1 开发平台**

* CPU:11900KF
* GPU:2080TI
* RAM:DDR4 32G

### **1.1.2 运行平台**

* Windows 10专业版
* Cisco Packet Tracer 7.0

## **1.2 系统功能需求**

本次实验是编写一个 Web 服务器软件

基本要求如下：

1.可配置性：能够配置 Web 服务器的监听地址、监听端口和主目录，且配置不写在代码中，避免每次配置重编译代码。

2.单线程处理请求：当客户（浏览器）连接时创建连接套接字，单线程处理请求。

3.接收并处理请求：从连接套接字接收 http 请求报文，确定用户请求的网页文件。

4.获取并发送文件：从服务器文件系统获取请求文件，创建 http 响应报文，经 TCP 连接向浏览器发送，使浏览器正确显示网页内容。

高级要求如下：

1.多媒体传输：能够传输包含多媒体（如图片）的网页给客户端，并在客户端正确显示。

2.请求来源输出：在服务器端屏幕输出请求的来源，包括 IP 地址、端口号和 HTTP 请求命令行。

3.处理结果输出：在服务器端屏幕输出对每个请求处理的结果。

4.错误处理：对于无法定位文件的请求，根据错误原因给出相应错误提示，并具备异常情况处理能力。

## **1.3 系统设计**

我们计划实现分成三个部分：网络配置以及管理、套接字创建以及初始化和连接建立与数据处理。

对于网络配置管理：服务器程序启动时，通过读取保存文件中的配置信息（监听地址、监听端口和主目录）。需要处理配置文件不存在和不完整的情况。

对于套接字创建与初始化：创建服务器套接字，指定地址族和套接字类型。设置服务器套接字的地址和端口，进行IP 地址转换，套接字与指定地址和端口绑定。设置套接字为监听状态，等待客户端连接请求。

连接处理与数据传输：接受客户端连接请求，当有连接到来时，接受连接并创建会话套接字。将会话套接字设置为非阻塞模式，以便在等待数据时不会阻塞程序执行。接收客户端请求数据，并根据接收结果进行处理。针对接受结果使用函数处理。

## **1.4 系统实现**

不妨构建一个流程图如下

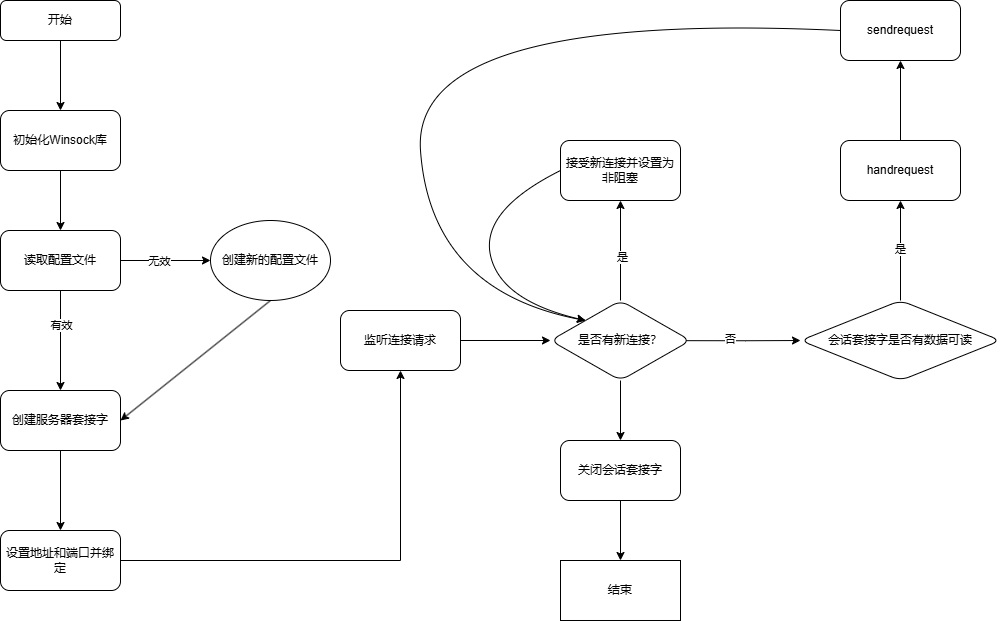


图1.1 套接字SERVER流程图

* 初始化阶段

程序自启动节点起始，首先执行“初始化 Winsock 库”操作，调用WSAStartup函数以完成Winsock库的初始化工作，同时检查版本信息并处理可能出现的错误。

继而进行配置文件的读取，通过readConfigFile函数从配置文件中提取服务器的监听地址、监听端口和主目录等关键参数，为后续设置服务器套接字和处理文件请求做准备。函数内部必须妥善处理文件打开失败的情况，并输出相应的错误信息；对于配置文件中的每一行，需精确解析键值对，去除首尾空格后存入map容器，若解析过程中遇到格式错误的行，亦应输出错误提示，但不影响程序继续尝试读取其他行。

对配置的有效性进行判断，若配置文件读取失败或缺少必要参数，则程序流程直接跳转至结束节点；若配置有效，则继续执行后续操作。。

* 套接字准备阶段

构建服务器端套接字，通过调用socket(AF\_INET, SOCK\_STREAM, 0)函数，以实现服务器端套接字的创建。在此过程中，地址族参数被指定为AF\_INET，该参数适用于Internet地址族；套接字类型参数被设定为SOCK\_STREAM，以支持基于TCP协议的可靠、面向连接的数据传输服务；协议参数则设置为0，以便系统自动选择最为适宜的协议。

套接字创建完成后，进行地址和端口的配置，并执行绑定操作。具体而言，通过调用bind(srvSocket, (LPSOCKADDR)&serverAddr, sizeof(serverAddr))函数，将服务器端套接字srvSocket与预先设定的地址和端口进行绑定。

绑定操作成功后，服务器端套接字将进入监听状态，以接受连接请求。这一过程通过调用listen(srvSocket, 5)函数实现，其中参数5代表服务器端套接字的最大等待连接数。若监听状态设置成功，将输出服务器监听信息，即通过std::cout << "Server is listening on " << address << ":" << port << std::endl;语句，向用户通告服务器当前的监听地址和端口。

* 主循环处理阶段

在服务器套接字进入监听模式后，程序随即进入其主循环。在此循环中，通过调用select函数以等待套接字事件的发生（select(0, &rfds, &wfds, NULL, NULL);）。一旦FD\_ISSET(srvSocket, &rfds)条件成立，即表明服务器套接字存在可读事件，意味着有新的客户端连接请求到达。此时，通过调用accept(srvSocket, (LPSOCKADDR)&clientAddr, &addrLen)函数来接受客户端的连接请求。成功建立会话套接字sessionSocket后，利用ioctlsocket(sessionSocket, FIONBIO, &blockMode)函数将该会话套接字配置为非阻塞模式，以确保在数据等待期间不会导致程序执行的阻塞。

将已建立的会话套接字纳入fd\_set集合中，以便后续进行数据事件的监听。通过recv(sessionSocket, recvBuf, sizeof(recvBuf), 0)函数接收数据，并将接收到的数据存储于recvBuf缓冲区中。根据recv函数的返回值进行相应的处理：若返回值大于0，说明数据接收成功，数据长度由返回值给出，此时可调用handleRequest函数来处理客户端请求，并将recvBuf、接收数据长度以及主目录路径作为参数传递给handleRequest函数；若返回值为0，表明客户端已断开连接，此时应输出提示信息std::cout << "Client disconnected." << std::endl;，关闭会话套接字，将sessionSocket设置为INVALID\_SOCKET，并继续下一轮循环以等待新的事件。

* 连接关闭阶段：

在客户端连接终止时，应适时关闭会话套接字（closesocket(sessionSocket);），以释放与该会话套接字相关的系统资源，防止资源泄露。套接字关闭后，将sessionSocket设置为INVALID\_SOCKET，确保程序能够准确识别该套接字已处于关闭状态，从而继续监听新的连接请求或处理其他会话套接字的事件。在程序终止前，应关闭服务器套接字并调用WSACleanup函数以清理Winsock库资源，确保所有资源得到妥善释放。

## **1.5 系统测试及结果说明**

本次实验使用的机器硬件配置：CPU:11900KF GPU:2080TI RAM: 32G Windows 10 专业版

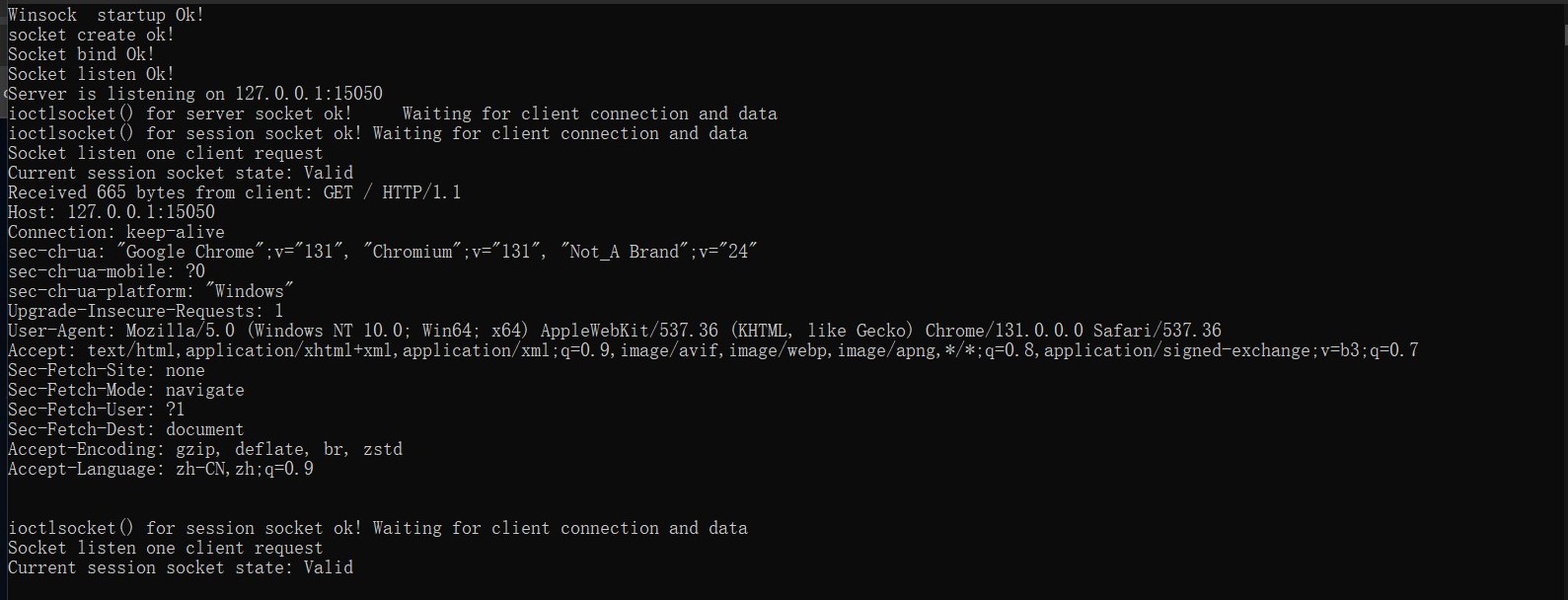


图1.2 localhost访问

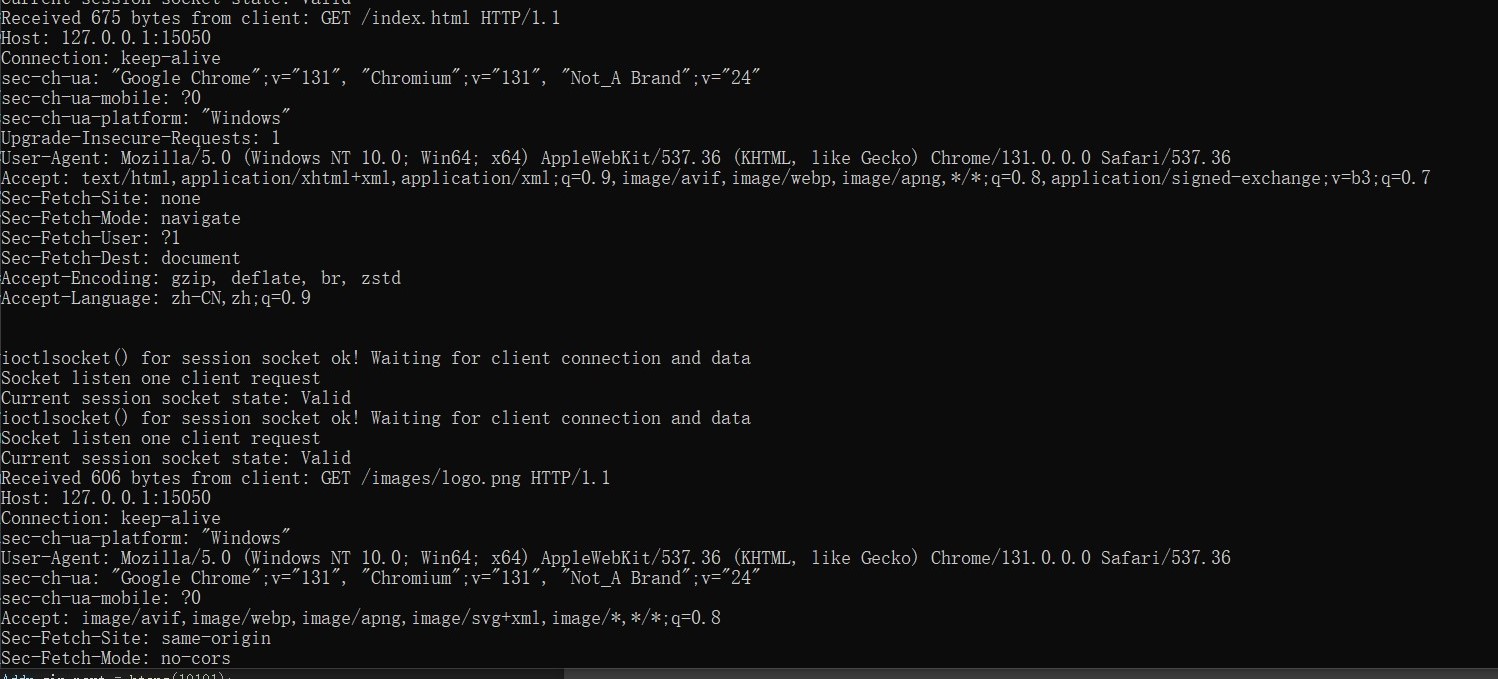


图1.3 GET index.html

这里我们使用Whistle工具请求结果检查200 OK状态码和404 NOT FOUND状态码

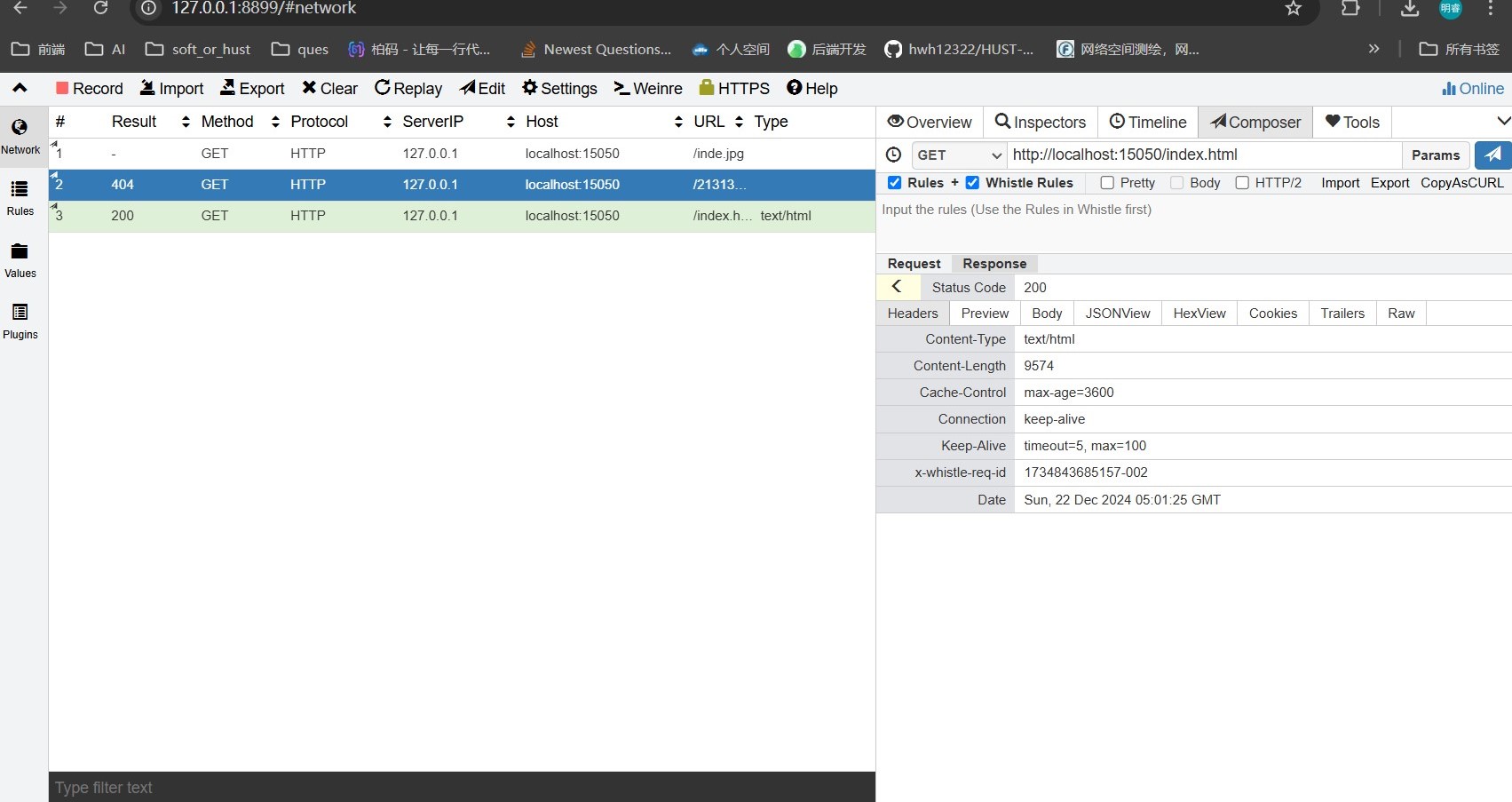


图1.4 Whistle抓包返回状态码示意图

我们依次请求localhost的15050端口和server监听一致，并请求一个乱序的不存在的文件和index.html，可以发现200、404状态码成功发送。

## **1.6 其它需要说明的问题**

1. URL的中文问题

在这里特地说明URL解析处理过程也就是handlerequest函数部分：在handleRequest函数中，解析客户端请求数据。首先从请求数据中提取请求行requestLine = request.substr(0, request.find("\r\n"))，然后从请求行中解析出请求的文件路径filePath = requestLine.substr(firstSpace + 1, secondSpace - firstSpace - 1)，拼接主目录路径得到完整的文件路径filePath = root\_dir + filePath，以便在服务器文件系统中查找文件。

根据文件路径在服务器文件系统中查找文件，若文件存在，根据文件扩展名确定Content-Type（.html对应text/html,.css对应text/css），以二进制模式打开文件，获取文件大小，构建 HTTP 响应头（包括状态码200 OK、Content-Type、Content-Length、Cache-Control、Connection等），先发送响应头，然后循环读取文件内容并发送，最后发送剩余部分，最后关闭文件。当文件不存在时，构建404 Not Found响应string response = "HTTP/1.1 404 Not Found\r\n\r\n";

std::string requestLine = request.substr(0, request.find("\r\n")); filePath = requestLine.substr(firstSpace + 1, secondSpace - firstSpace - 1)这两句代码解析了URL，但是实际上请求求的中文URL使用什么编码，服务器在这里没有做任何处理，可能就无法识别，而且这种路径编码也可能受限制，这里实验一下

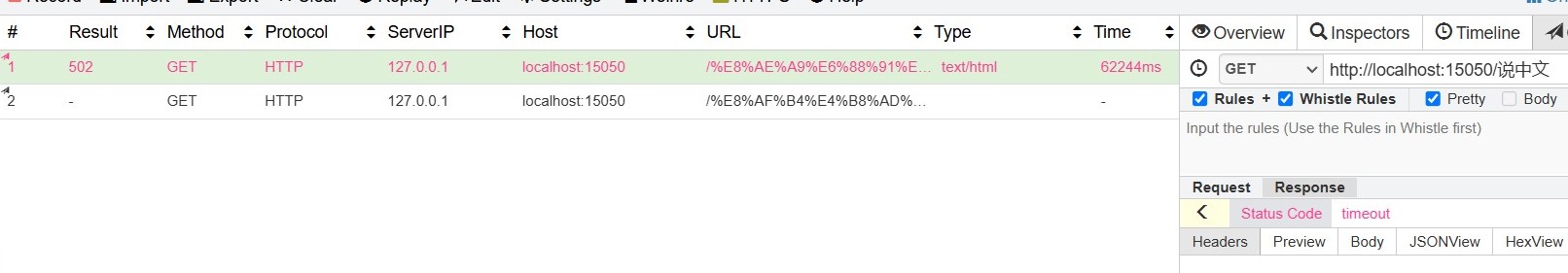


图1.5 转义演示

可以发现这里Whistle帮我将中文转义成为字符乱码，这样就难以处理本地的文件。当我保存在本地的文件是中文名称，我需要判断这个请求的URL解析出的结果到底是什么，有可能是直接的英文字符直接寻找文件，但是也有可能是转义的其他字符，可能需要哦中间在转义成为原来的字符段做匹配才能正常识别到。

2.多线程问题

通过创建多个线程，实现服务器同时处理多个客户端连接请求，提高服务器的并发处理能力，提升系统性能和响应速度，每个线程独立处理一个客户端的通信逻辑，避免单个客户端长时间占用服务器资源导致其他客户端等待。

在接受客户端连接请求并成功创建会话套接字后，创建一个新线程来处理该客户端的请求。

但是实际上我们的思路就是使用操作系统的能力，线程函数内部实现与当前客户端的完整通信逻辑，包括接收客户端请求（使用recv函数）、处理请求（如解析请求、查找文件、构建响应，与单线程处理中的逻辑类似）和发送响应（使用send函数）

std::mutex fileMutex;

void ThreadFunction(SOCKET sessionSocket, const std::string& root\_dir) {

其他实现

std::string filePath = root\_dir + requestedFilePath;

std::ifstream file;

{

std::lock\_guard<std::mutex> guard(fileMutex); // 使用互斥锁保护文件访问

file.open(filePath, std::ios::binary);

}

// 根据文件是否打开成功进行后续处理，如发送响应等

//...

}

实际上这只是一个很简单的模拟，因为实际上问要处理线程之间的数据同步和资源共享问题、主线程与子线程交互等，过于复杂就放弃了

# 实验二 数据可靠传输协议设计实验

## **2.1 环境**

本次实验使用的机器硬件配置：

* CPU:11900KF
* GPU:2080TI
* RAM:DDR4 32G

本次实验使用的系统软件组件：

* Windows 10专业版
* Cisco Packet Tracer 7.0

## **2.2 实验要求**

可靠运输层协议实验仅考虑单向传输，即发送方发送数据报文，接收方接收报文并给出确认报文。

* 实现基于GBN（Go-Back-N）的可靠传输协议。
* 实现基于SR（Selective Repeat）的可靠传输协议。
* 在实现GBN协议基础上，根据TCP的可靠数据传输机制实现简化版TCP协议。

TCP要求包括报文段格式、接收方缓冲区大小与 GBN 协议保持一致；报文段序号按报文段为单位编号；使用单一超时计时器，无需估算RTT动态调整定时器Timeout参数；支持快速重传和超时重传，重传最早发送且未被确认的报文段；确认号为收到的最后一个报文段序号；暂不考虑流量控制和拥塞控制。

实现具体协议时，需指定编码报文序号的二进制位数以及窗口大小，且报文段序号必须按指定二进制位数编码。

基于 C++ 语言实现。

## **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**

GBN发送方实现：

#define N 4

#define Seqlenth 8

private:

int base;

int expectSequenceNumberSend; // 下一个发送序号

int init\_flag = 1;

bool waitingState; // 是否处于等待Ack的状态

Packet packetWaitingAck[Seqlenth]; //已发送并等待Ack的数据包

* 状态变量维护

通过expectSequenceNumberSend变量记录下一个预期发送的序列号，初始值设定为0，以保证发送顺序的有序性。waitingState标志位用于指示发送方是否处于等待确认状态，初始设置为false。

引入base变量以表示发送窗口的基序号，初始值设定为0。packetWaitingAck数组用于存储已发送但尚未收到确认的数据包，通过序号取余操作实现数据包在数组中的循环存储与管理。

* 发送流程

发送前需检查waitingState标志位，观察窗口状态（是否已满或正在等待确认），若未准备好则返回false。

当窗口未满时，构建待发送数据包：设置序列号为expectSequenceNumberSend，确认号为-1，并将应用层消息message封装入数据包。若当前发送的是基序号数据包，则启动定时器，通过模拟网络环境发送数据包，并递增expectSequenceNumberSend。

当expectSequenceNumberSend达到窗口容量上限时，将waitingState标志位置为true，暂停发送新数据包，等待确认。

* 确认处理

在接收到ACK后，若检验校验和且确认序号大于等于base，则认为该ACK有效。更新base为确认序号加1，滑动窗口，并将已确认数据包对应的packetWaitingAck数组元素序号置为-1。

* 超时处理

定时器超时后，停止当前定时器（并重启timer）。从base开始，循环重传packetWaitingAck数组中未被确认的数据包（其中expectSequenceNumberSend>seq>= base），打印重传信息，以确保数据的可靠传输。

GBN接收方实现：

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

* 状态变量与初始化

expectSequenceNumberRcvd记录期待接收的下一个报文序号，初始化为 0，保证按序接收数据。lastAckPkt用于存储上次发送的确认报文，初始状态下确认号为-1，校验和为0，序号为-1，有效载荷填充默认字符，并计算校验和，为后续确认操作提供初始状态。

* 接收流程

接收数据包后，计算校验和，校验和正确且数据包序号等于expectSequenceNumberRcvd时，视为正确接收。打印接收正确信息，提取数据包中的消息并通过模拟网络环境递交给应用层。

更新lastAckPkt的确认序号为接收到的数据包序号，重新计算校验和，发送确认报文，递增expectSequenceNumberRcvd，准备接收下一个序号的数据包。

* 错误处理

校验和错误或数据包序号不匹配时，分别打印相应错误信息，并重新发送上次的确认报文。

生成GBN.exe，检验实现完整性和正确性，如下三图：

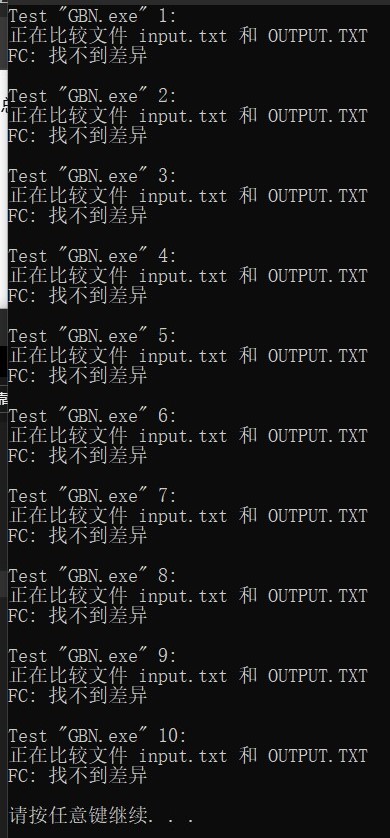


图2.1 GBN验证

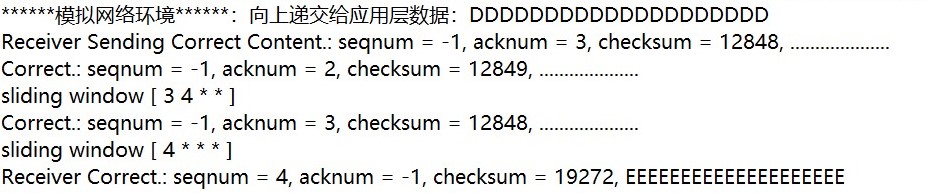


图2.2 GBN滑动窗口

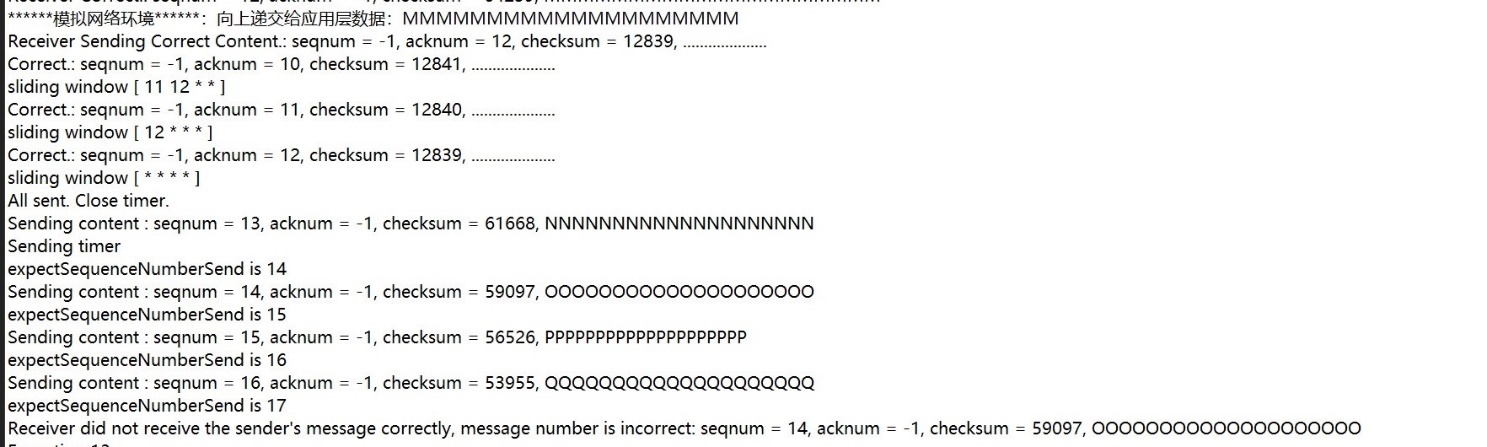


图2.3 GBN超时重发

可以看到GBN实现了超时重发，滑动窗口。

### **2.3.2 SR协议的设计、验证及结果分析**

SR发送方实现：

* 状态变量与初始化

基序号（base）指代接收窗口的起始序号，初始设定为0，用以界定接收窗口的起始边界。NextSeqNum代表接收窗口中下一个预期接收的序列号，初始设定为base+N，从而确定接收窗口的范围。

lastAckPkt变量用于保存上一次发送的确认报文信息，其初始状态下的确认号设为-1，校验和设为0，序号设为-1，有效载荷填充默认字符（'.'），并计算校验和，为后续的确认操作提供初始状态。packetWaitingFlags数组用于标识接收窗口内各个序列号对应的数据包是否已被接收，初始设定为false，以便于快速判断数据包的接收状态。ReceivedPacket数组则用于存储那些已经接收但可能处于乱序状态的数据包，等待按序递交给应用层。

* 发送流程

发送前检查waitingState，若为true，表示窗口已满或正在等待确认，拒绝发送新消息，直接返回false，避免网络拥塞和数据混乱。

当窗口未满时，构建待发送数据包：设置序号为expectSequenceNumberSend，确认号为 - 1，计算并设置校验和，从应用层消息message复制有效载荷到数据包。然后打印发送内容信息，启动定时器，通过模拟网络环境发送数据包，递增expectSequenceNumberSend。同时，标记该数据包对应的ACKFlags为false，表示未收到确认。

当expectSequenceNumberSend达到base + N（窗口已满）时，将waitingState置为true，暂停发送新数据包，等待确认。

* 确认处理

接收确认包后，计算校验和，校验和正确时，根据确认序号进行不同处理：

若确认序号等于base，表示基序号数据包已确认，滑动窗口：更新base，将已确认数据包对应的packetWaitingAck数组元素序号置为 - 1，标记ACKFlags为true，关闭定时器，根据ACKFlags数组连续为true的情况滑动窗口，将连续已确认的数据包标记清除，并更新base。若窗口内所有数据包都已确认，将waitingState置为false，表示可继续发送新数据。

若确认序号大于base且对应数据包未被标记为已确认（!ACKFlags[ackPkt.acknum % Seqlenth]），表示该数据包已被选择性确认，标记ACKFlags为true，关闭对应定时器，更新相关状态。

若确认序号不满足上述条件，视为错误确认，打印错误信息，继续等待正确确认。

* 超时处理

定时器超时后，停止当前定时器并重新启动。获取超时的数据包序号，打印重传信息，通过模拟网络环境重传该数据包，确保未被确认的数据包能够在超时情况下重新发送，保证数据的可靠传输。

SR接收方实现：

* **状**态变量与初始化

基序号（base）指代接收窗口的起始序号，初始设定为0，用以界定接收窗口的起始边界。NextSeqNum代表接收窗口中下一个预期接收的序列号，初始设定为base+N，从而确定接收窗口的范围。

lastAckPkt变量用于保存上一次发送的确认报文信息，其初始状态下的确认号设为-1，校验和设为0，序号设为-1，有效载荷填充默认字符（'.'），并计算校验和，为后续的确认操作提供初始状态。packetWaitingFlags数组用于标识接收窗口内各个序列号对应的数据包是否已被接收，初始设定为false，以便于快速判断数据包的接收状态。ReceivedPacket数组则用于存储那些已经接收但可能处于乱序状态的数据包，等待按序递交给应用层。

* 接收流程

在接收到数据包之后，首先进行校验和的计算。若校验和无误，则依据数据包的序号执行相应的处理：

若数据包的序号与基序号（base）相等，表明接收窗口的基序号数据包已正确抵达。此时，执行以下步骤：

输出接收成功的相关信息，从数据包中提取消息内容，并通过模拟网络环境将其传递至应用层。

更新lastAckPkt的确认序号至接收到的数据包序号，并重新计算校验和，随后发送确认报文。

将packetWaitingFlags标记为true，表示该数据包已被接收，并将其存入ReceivedPacket数组中。同时，清除该数据包的确认号（ReceivedPacket[packet.seqnum % Seqlenth].acknum = -1;）。

检查packetWaitingFlags中从base开始连续为true的数据包，按序号递交给应用层，并释放缓冲区（通过更新base和NextSeqNum，并将对应packetWaitingFlags标记为false）。

若数据包序号大于base且小于NextSeqNum，说明数据包位于接收窗口内但非基序号，此时执行以下操作：

记录缓存信息，将数据包存入ReceivedPacket数组，并将packetWaitingFlags标记为true。

更新lastAckPkt的确认序号至接收到的数据包序号，并重新计算校验和，随后发送确认报文。

若数据包序号大于等于base - N且小于base，表明该数据包为已确认但可能因网络延迟而重新到达的旧数据包，此时应重新发送确认报文。

若数据包序号不符合上述任一条件，则视为错误序号的数据包，输出错误信息，并提示期望的序号范围。

若校验和计算结果错误，则直接输出数据校验错误信息，并使接收方继续等待正确数据包的到来。

* 滑动窗口操作

在传输控制协议（TCP）中，发送方的滑动窗口机制通过基础序号（base）和预期发送序号（expectSequenceNumberSend）进行控制。基础序号标识窗口最左端已确认数据包的序号，而预期发送序号则表示窗口最右端已发送但尚未得到确认的数据包的下一个序号。当接收到对应于基础序号的累积确认（ACK）或选择性确认（SACK），并且窗口内所有连续数据包均已被确认时，基础序号将向前滑动，从而增加窗口内可发送新数据包的空间。若定时器超时，则仅需重传对应超时序号的数据包，窗口位置保持不变，直至收到确认后再进行相应的滑动。

在TCP协议中，接收方的滑动窗口机制由基础序号（base）和下一个期望接收序号（NextSeqNum）所界定。基础序号标识窗口最左端已按序接收并处理的数据包的序号，而下一个期望接收序号则表示窗口最右端期望接收数据包的下一个序号。当按序接收到对应于基础序号的数据包，并处理完所有连续已接收的数据包后，基础序号将向前滑动。若接收到窗口内但非基础序号的数据包时，接收方将该数据包缓存，不滑动窗口，等待基础序号的数据包到达后，按序处理并滑动窗口，以确保数据包能够按序递交给应用层。

生成SR.exe，检验SR协议的支持性，如下三图：

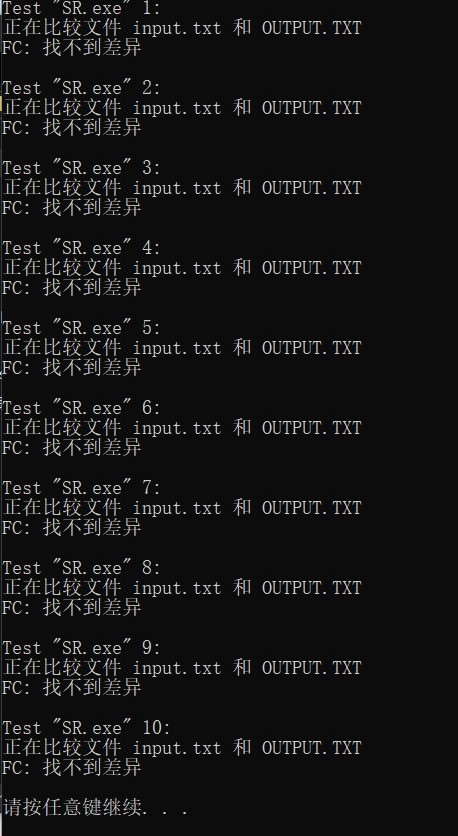


图 2.4 SR验证

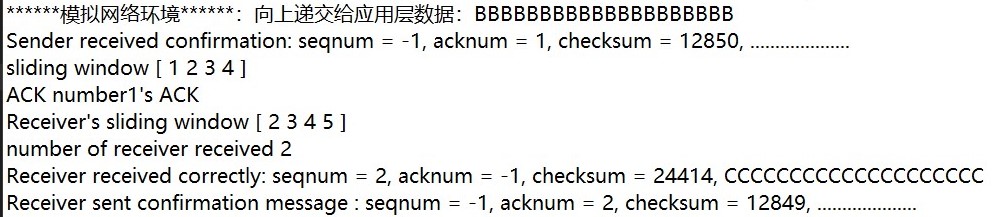


图 2.5 SR滑动窗口

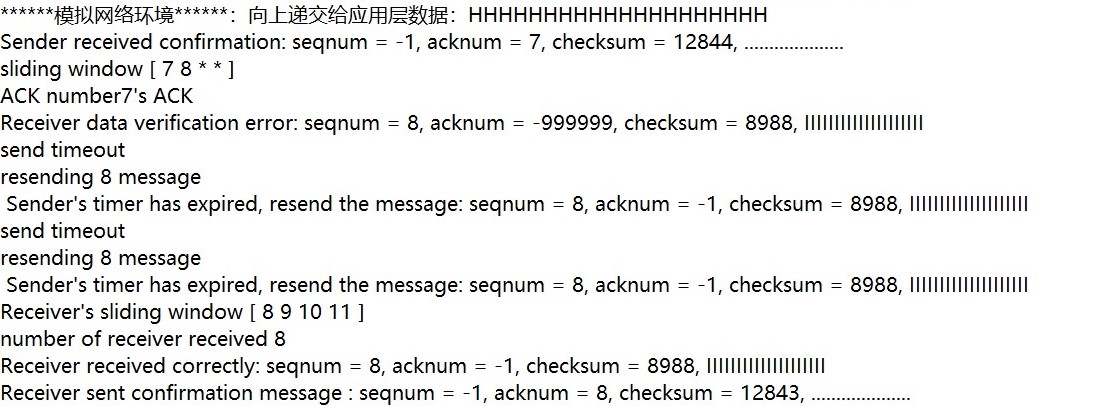


图 2.6 SR超时重传

可以看到SR实现窗口滑动、超时重传。

### **2.3.3 简单TCP/IP协议的设计、验证及结果分析**

TCP/IP发送方实现：

* 状态变量与初始化

变量base代表发送窗口的基序号，初始值设为0，用于界定已确认数据包的范围，是滑动窗口操作和重传控制的核心变量。expectSequenceNumberSend记录下一个预期发送的序号，初始值设为0，以确保发送顺序的有序性。waitingState变量用于标识发送方是否处于等待确认状态，初始值设为false，以控制数据发送的节奏。

packetWaitingAck数组用于存储已发送但等待确认的数据包，其长度与窗口大小相关，通过序号取余操作实现数据包在数组中的循环存储与管理。引入init\_flag变量用于初始化packetWaitingAck数组，确保初始状态下数据包序号的正确性。lastack变量用于记录上一次收到的确认序号，ACK\_count变量用于统计连续收到相同确认序号的次数，用于实现快速重传机制。

* 发送流程

发送前检查waitingState，true表窗口已满或正在等待确认，拒发新消息则返回false。

当init\_flag为 1 时，初始化packetWaitingAck数组中所有数据包的序号为 - 1，表示未使用状态，然后将init\_flag置为 0。

当窗口未满时，构建待发送数据包：设置序号为expectSequenceNumberSend，确认号为 - 1，计算并设置校验和，从应用层消息message复制有效载荷到数据包。然后打印发送内容信息，若当前发送的是基序号数据包，启动定时器，通过模拟网络环境发送数据包，递增expectSequenceNumberSend。

当窗口已满时，将waitingState置为true，暂停发送新数据包，等待确认。

* 确认处理

接收ACK后，校验和正确且确认序号大于等于base时，视为有效确认，执行以下操作：

更新base为确认序号加 1，滑动窗口：将已确认数据包对应的packetWaitingAck数组元素序号置为-1。

根据base与expectSequenceNumberSend关系决定定时器操作：若两者相等，表示所有数据包已发送且被确认，关闭定时器，将waitingState置为false；否则，关闭当前基序号对应的定时器，重新启动新基序号的定时器，并将waitingState置为false，表示继续等待后续确认。

若校验和正确但确认序号不满足上述条件，检查是否为连续收到的相同确认序号

若是则递增ACK\_count，当ACK\_count达到3时，触发快速重传机制：停止并重新启动对应序号定时器，重传基序号数据包。

若不是，更新lastack为当前确认序号，重置ACK\_count为 1。

若校验和错误，打印 ACK 错误信息，继续等待正确确认。

* 超时处理

定时器超时后，停止当前定时器并重新启动。获取超时的数据包序号，通过模拟网络环境重传该数据包，确保未被确认的数据包能够在超时情况下重新发送，保证可靠传输。

TCP/IP接收方实现：

* 状态变量与初始化

expectSequenceNumberRcvd记录期待接收的下一个报文序号，初始值设为 0，以保证按序接收数据。lastAckPkt用于存储上次发送的确认报文，初始状态下确认号为 - 1，校验和为 0，序号为 - 1，有效载荷填充默认字符（'.'），并计算校验和，为后续确认操作提供初始状态。

* 接收流程

接收数据包后，计算校验和，校验和正确且数据包序号等于expectSequenceNumberRcvd时，视为正确接收，执行以下操作：

打印接收正确信息，提取数据包中的消息并通过模拟网络环境递交给应用层。

更新lastAckPkt的确认序号为接收到的数据包序号，重新计算校验和，发送确认报文，递增expectSequenceNumberRcvd，准备接收下一个序号的数据包。

若校验和错误或数据包序号不匹配，分别打印相应错误信息（数据校验错误或报文序号不对），并重新发送上次的确认报文，维持接收方的稳定性和可靠性，促使发送方重传数据或调整发送策略。

滑动窗口操作

发送方的滑动窗口通过base和expectSequenceNumberSend控制，base表示窗口最左端已确认数据包的序号，expectSequenceNumberSend表示窗口最右端已发送但未确认数据包的下一个序号。当接收到有效确认时，base向前滑动，窗口内可发送空间增加；当定时器超时或触发快速重传机制时，根据情况重传相应数据包，窗口位置根据重传结果和后续确认进行调整。

接收方通过expectSequenceNumberRcvd维持接收窗口，按序接收，只有序号等于该值的数据包才被视为有效接收，接收后递增，实现接收窗口的有序滑动，确保数据按序递。

重传机制实现

发送方采用单一超时计时器，当定时器超时，重传最早发送且未被确认的报文段，确保数据的可靠性。同时，引入快速重传机制，当连续收到3个相同的冗余确认时，立即重传下一个未被确认的报文段（ackPkt.acknum+1），无需等待定时器超时，提高了数据传输效率，减少了因等待超时导致的延迟。

接收方在接收到错误序号的数据包或校验和错误时，通过重新发送上次的确认报文，促使发送方进行重传操作。

生成TCP.exe，校验简易TCP的协议实现，如下三图：

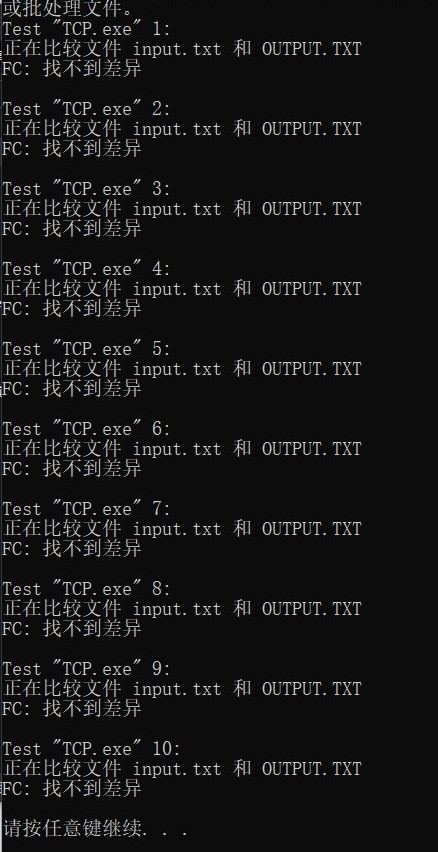


图 2.7 简易TCP验证

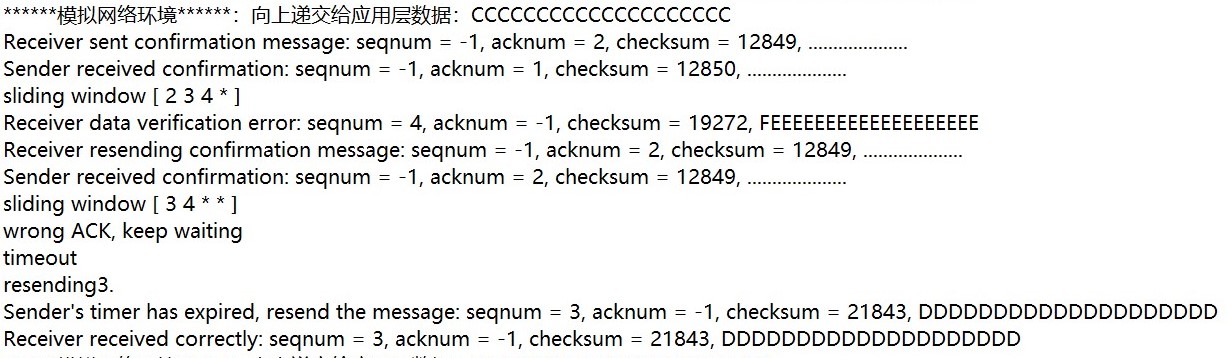


图 2.8 TCP滑动窗口

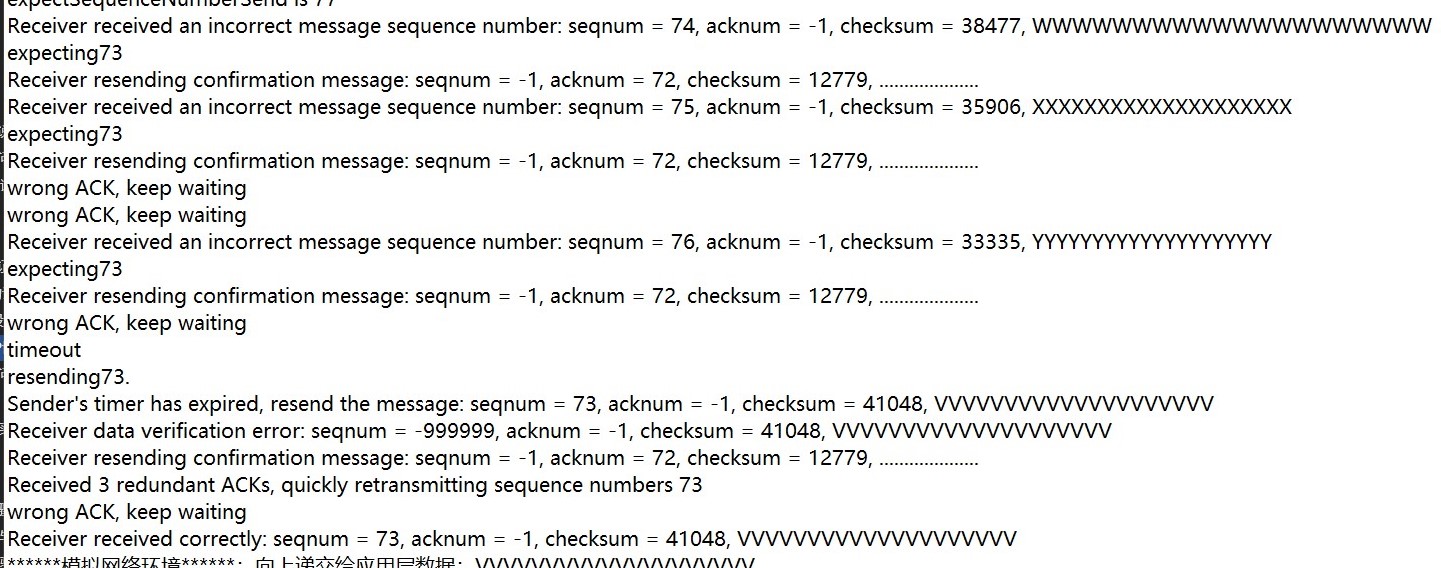


图 2.9 TCP快速重传

如上所示，实现“字节流”的窗口滑动、三次ACK快速重传。

## **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**

本次实验使用的机器硬件配置：

* CPU:11900KF
* GPU:2080TI
* RAM:DDR4 32G

本次实验使用的系统软件组件：

* Windows 10专业版
* Cisco Packet Tracer 7.0

## **3.2 实验要求**

1. IP 地址规划与 Vlan 分配实验

绘制网络拓扑图1.1，将PC1、PC2设置在192.168.0.0/24网段，PC3-PC8设置在 192.168.1.0/24网段，并配置路由器实现两个子网间 PC 机的自由通信。

绘制拓扑图1.1，将PC1、PC2设置在192.168.0.0网段，PC3、PC5、PC7设置在 192.168.1.0/24网段，PC4、PC6、PC8设置在192.168.2.0/24网段；配置交换机1-4，使PC1、PC2属于Vlan2，PC3、PC5、PC7属于Vlan3，PC4、PC6、PC8属于Vlan4；测试各PC间连通性并结合理论分析，最后配置路由器实现图上各PC机自由通信，并详细说明路由器配置过程。

1. 路由配置实验

绘制网络拓扑图1.2，将PC1-PC4分别设置在不同网段（192.168.1.0/24、192.168.2.0/24、192.168.3.0/24、192.168.4.0/24），设置路由器端口IP地址，并在路由器上配置RIP协议实现各PC机互相访问。

基于拓扑图1.2，将PC1-PC4分别设置在不同网段（192.168.1.0/24、192.168.2.0/24、192.168.3.0/24、192.168.4.0/24），设置路由器端口IP地址，并在路由器上配置OSPF协议实现各PC机互相访问。

在上述基础上，对路由器 A 进行访问控制配置，实现 PC1 无法访问其它 PC 且不能被其它 PC 机访问，以及 PC1 不能访问 PC2 但能访问其它 PC 机的功能。

1. 综合部分实验

某学校申请了 211.69.4.0/22 地址块，包含4个学院（各20台主机）、1个图书馆（100 台主机）、3个学生宿舍（各200台主机），设计满足图书馆能无线上网、学院间可互相访问、学生宿舍间可互相访问、学院和学生宿舍间不能互相访问但皆可访问图书馆的网络拓扑结构。依据理论知识，对全网 IP 地址进行合理规划分配。

## **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**

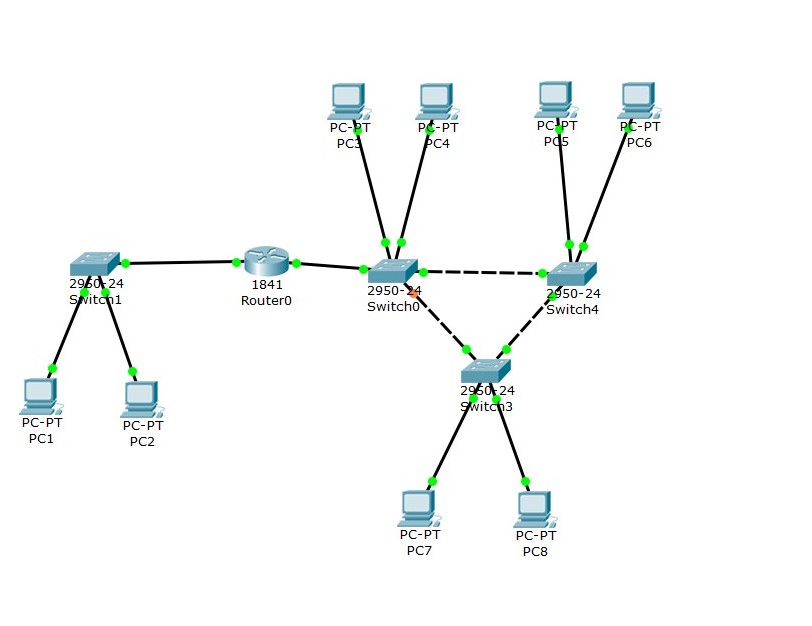


图 3.1 拓扑图

完成实验进行如下操作

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；

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。

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；

在CPT软件中，单击我们的设备即可进入配置页面，我们按计划划分以下IP地址：

PC1：192.168.0.1、PC2: 192.168.0.3、PC3：192.168.1.1、PC4：192.168.2.1、PC5: 192.168.1.1、PC6：192.168.2.1、PC7：192.168.1.1、PC8: 192.168.2.1

Router按子网安排。例子如下图：

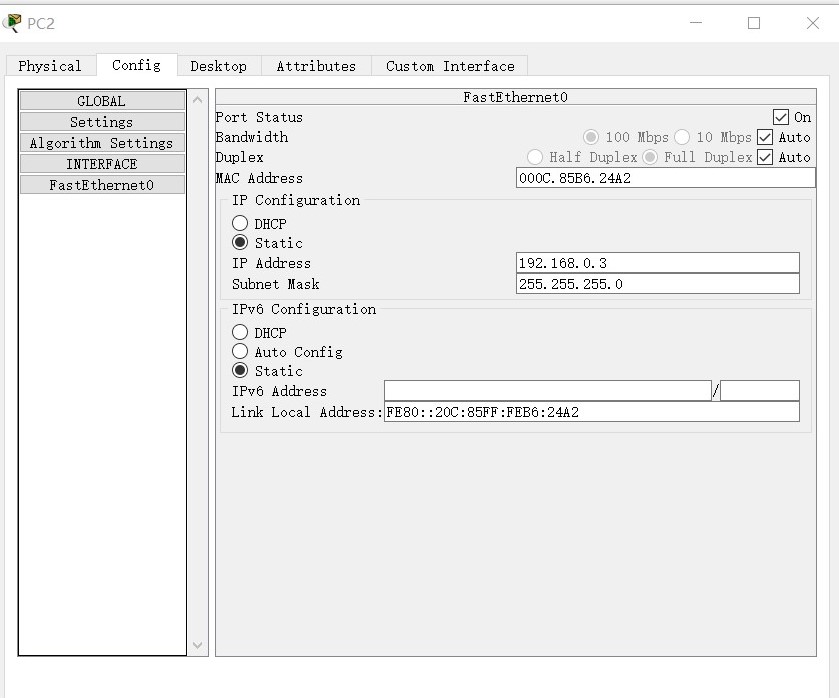


图 3.2 PC2地址分配样例

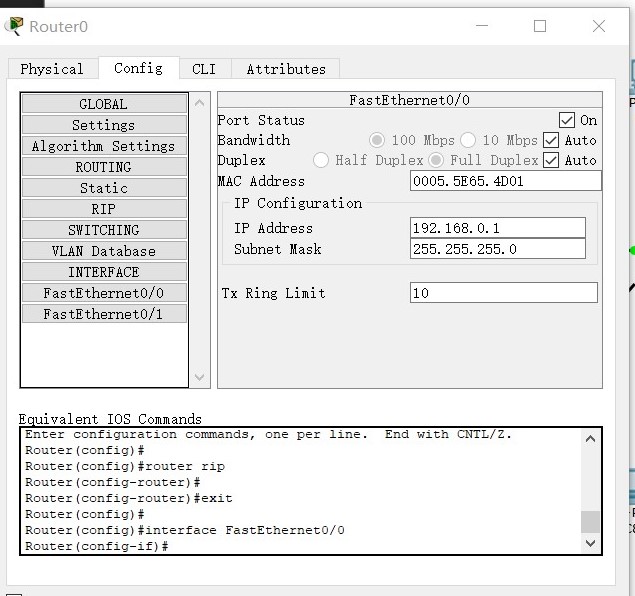


图 3.3 路由分配地址样例



图3.4 PC1分配参数样例

我们配置好网关，子网掩码和接口配置就可以检测子网连通性如下图：

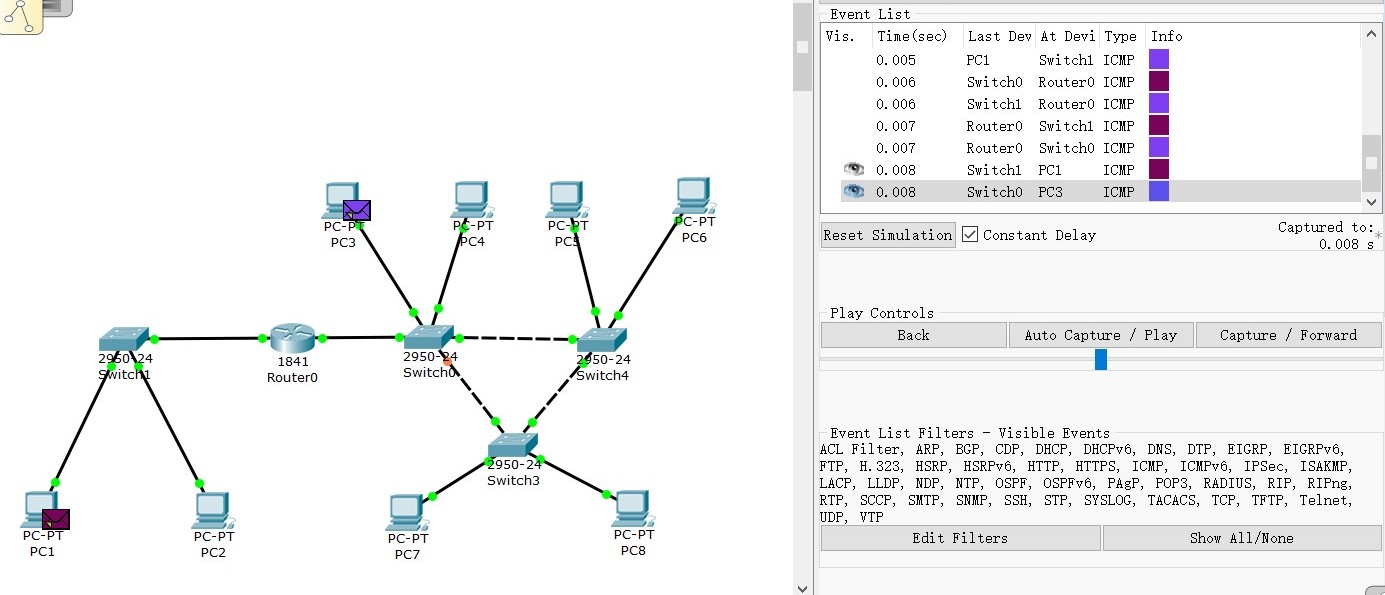


图3.5 连通性检测

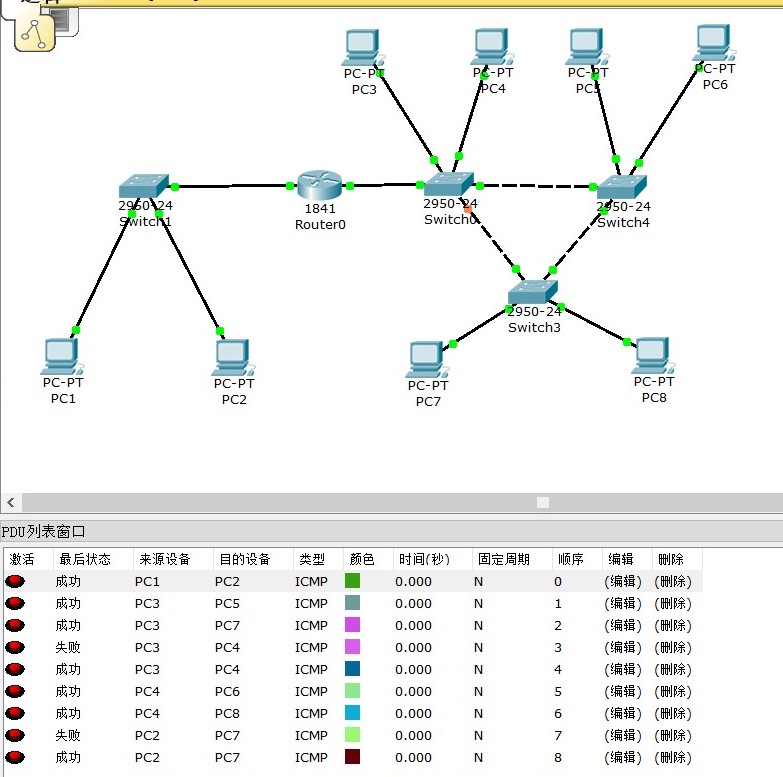


图3.6 连通性检测

这里我们可以看出，不同子网之间互相访问第一次ping是不通的因为这个时候路由和交换机表项并不全，这是后续需要完成的OSPF和RIP需要解决的维护问题，正如我们实验要求的那样，正确得到了子网内主机互相联通。此外并没有直接使用实验1基础要求一样的设定。

VLAN安排可以用图形化的方式，如下图交换机的设定

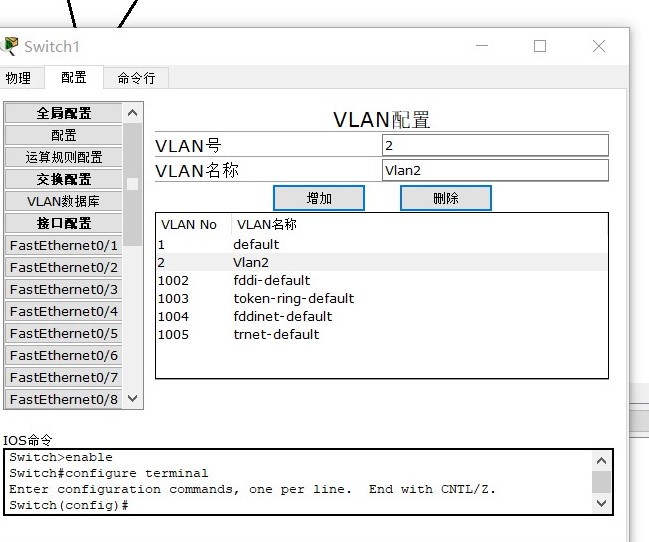


图3.7 交换机vlan设定示意图

建立好vlan还需要对交换机的端口设定属于是哪一个vlan，上图所示的交换机分配如下图：

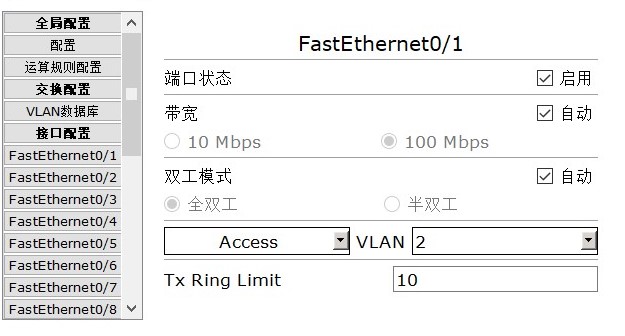


图3.8 vlan绑定端口

对于交换机与pc相连的链路，链路类型应该选择为access类型，access链路的默认vlan为vlan1，我们应该把它改成我们添加的vlan，此外在其他的交换机上我们也要加入其他的vlan并规定好我们的PC机遇交换机链路的access到底属于哪一个虚拟局域。

建立好我们的vlan体系后，我们可以实验，结果如下：

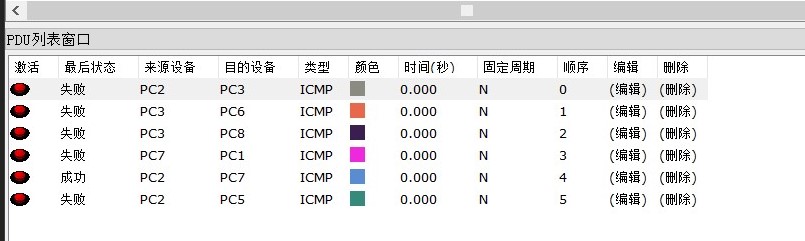


图3.9 vlan检测图

如上图所示不同vlan之间是不可以ping通的，但是同一个vlan之间可以。

我们得到这样的结果可以这样想在192.168.0.0与192.168.1.0网段间，数据包的传输归因于路由器端口已配置IP地址，路由器端口IP地址就是默认网关。当数据包从第一个网段发出时，路由器能够识别并转发至第二个网段但是当192.168.2.0网段没有默认网关，而路由器的两个端口已被其他两个网段占用，导致192.168.2.0网段没有网关可以用，无法与其他子网进行数据交换，路由器将忽略该网段发出的数据包。

在交换机上实施VLAN划分后，各主机被分配至不同的VLAN中，每个VLAN在逻辑上构成一个整体，不受物理位置和子网划分的限制。若未在路由器上进行VLAN配置，VLAN内部的主机间虽可通信，但与其他VLAN的主机则无法实现数据交换，相当于VLAN处于隔离状态。

### **3.3.2 路由配置实验的步骤及结果分析**

构建如下拓扑图

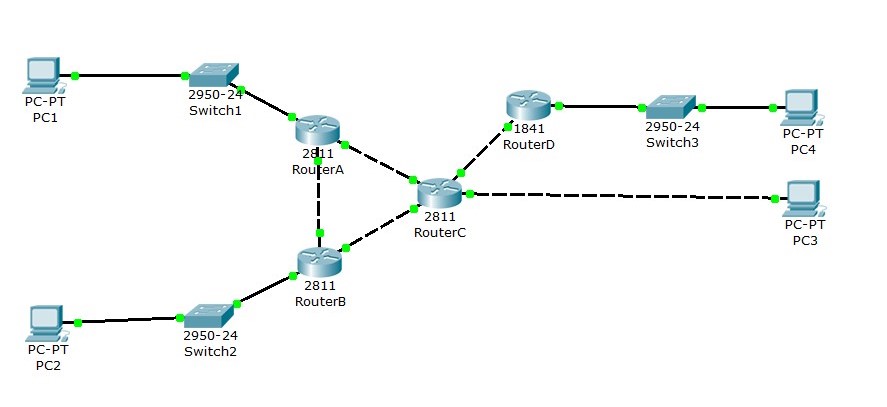


图3.10 路由设定拓扑图

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，所以路由器与pc机或者交换机连接的端口IP应该配置为对应网段的IP。

先给PC机IP：192.168.1.1、192.168.2.1、192.168.3.1、192.168.4.1，在思考如何给路由IP。

在路由器之间的内网安排其他的网段这里可以自由发挥，那么我们就可以这样做，在路由器的配置接口，分配好我们的接口IP，以routerA为例，FastEthernet0/0, FastEthernet0/1, FastEthernet1/0分别有MAC地址、子网掩码和IP地址，分别给成192.168.1.1、192.168.5.1和192.168.6.1。

我们配置RIP协议在配置页面即可实现，将需要的子网网段填入即可。

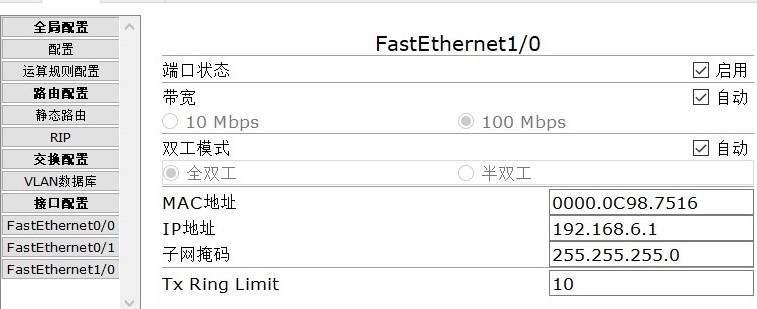


图3.11 路由接口其一配置页面



图3.12 路由RIP配置

最后我们使用PDU去检测如下图

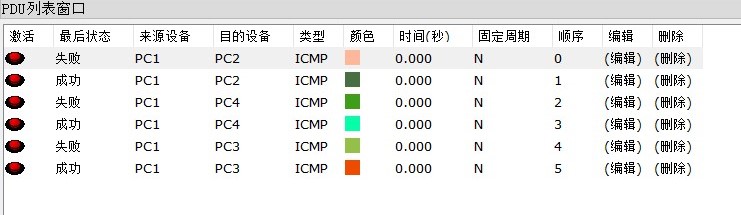


图3.13 RIP检测结果

正如上图所示，PC1去PING另外的主机，总是需要经过一次失败的过程，这就是路由器路由算法在不断更新的结果。

对于OSPF协议的实现是通过命令行实现的,这里截图观察同一个路由器使用OSPF的逻辑

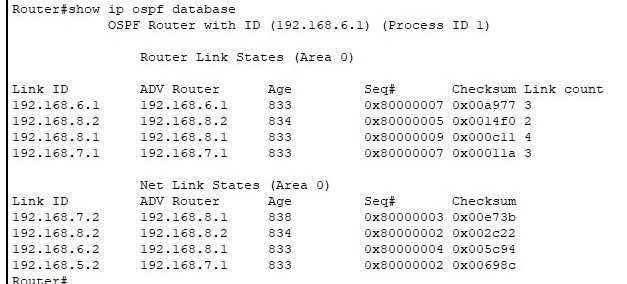


图3.14 OSPF配置检测

任选一个数字作为进程号为路由器配置OSPF，network开头的命令将路由器端口的IP地址和子网掩码绑定到路由器上。这里database就是OSPF所在路由器中形成的全网链路状态b表，确实是全图的很同容易分析出来。连通性测试和RIP没有太大差别就不做赘述。

连通性限制的实验这里是通过access-list的方式实现的，如下图对于PC1相邻的路由器配置如下

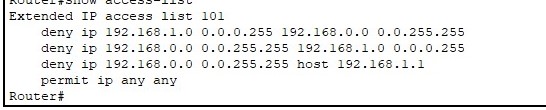


图3.15 ACL表项

第一条规则拒绝源IP属于192.168.1.0/24网段的主机向目的 IP 地址在192.168.0.0/16网段的主机发送的IP数据包。

第二条规则拒绝源IP在192.168.0.0/16网段的主机向目的 IP 地址属于192.168.1.0/24网段的主机发送的IP数据包

第三条规则拒绝源IP在192.168.0.0/16网段的主机向特定的目的 IP 地址（也就是主机地址192.168.1.1）发送IP数据包，这里host关键字明确指定了后面的 IP 是一个主机地址。

我们对PC1连通性进行实验，如下图：



图3.16 PC1限制后的连通性

通过分析上面的规则我们不难发现，ACL列表限制了数据包，那么当我们加上第一条规则就可以限制PC1访问外界的主机，同时，我们的第二条规则限制了外界主机对PC1的访问权利，加上第三条规则稍作修改我们甚至可以实现PC1是不可以PING自己的。

这样的话三个问题我们都顺利解决

## **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**

每个学生宿舍需要容纳200台主机，2^8=256>200，因此主机位需要8位，子网掩码为这里就是/24

三个学生宿舍共需要3个子网，2^2=4>3，从211.69.4.0/22地址块中借2位来划分子网

子网划分如下：

学生宿舍 1：211.69.4.0/24，可用 IP 地址范围：211.69.4.1 - 211.69.4.254

学生宿舍 2：211.69.5.0/24，可用 IP 地址范围：211.69.5.1 - 211.69.5.254

学生宿舍 3：211.69.6.0/24，可用 IP 地址范围：211.69.6.1 - 211.69.6.254

图书馆需要容纳100台主机，2^7=128>100，主机位需要7位，子网掩码为/25。

图书馆子网：211.69.7.128/25，可用IP地址范围：211.69.7.129 - 211.69.7.254

为每个接入层交换机配置 VLAN，将连接到同一子网的端口划分到同一 VLAN

在路由器上配置如下 ACL 规则：

* 允许学生宿舍子网（211.69.4.0/24、211.69.5.0/24、211.69.6.0/24）访问图书馆子网（211.69.7.128/25）。
* 允许学院子网（211.69.7.0/27、211.69.7.32/27、211.69.7.64/27、211.69.7.96/27）访问图书馆子网。
* 拒绝学生宿舍子网访问学院子网，拒绝学院子网访问学生宿舍子网。

对图书馆的无线接入点进行配置，设置 SSID 和加密方式。

将无线接入点连接到图书馆的交换机，并配置相应的 VLAN，使其能够访问图书馆子网

为每个学院、学生宿舍和图书馆的 PC 配置 IP 地址、子网掩码、默认网关和 DNS 服务器地址。

在图书馆的笔记本（Laptop - Pt）上搜索并连接到无线接入点，配置无线网卡的 IP 地址相关参数

### **3.4.2 实验步骤**

首先是PC的IP配置，这里设计部分已给出原理，不多做赘述，给出学院和宿舍的IP配置图

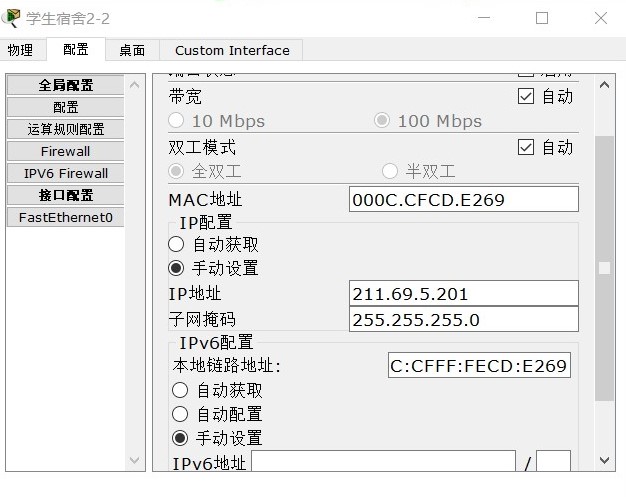


图3.17学生宿舍其一配置图

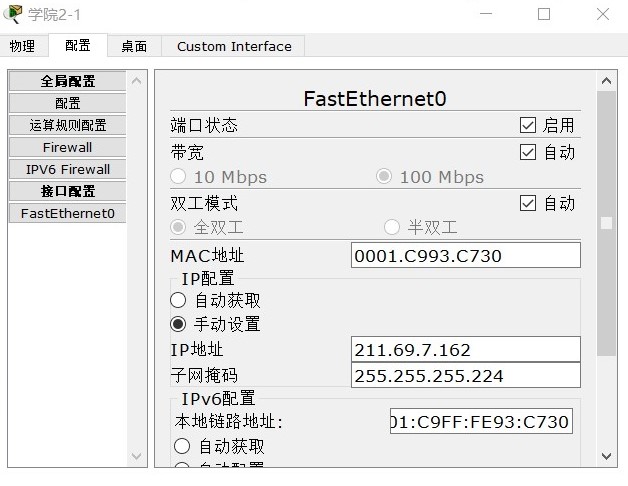


图3.18 学院其一配置如图

和以上处理相同，我们依次完成对于设备的网关设定、IP设定，服务器的相关网络配置，这部分的处理和操作和前面两个实验的基础操作一致，不多做赘述。

这里需要不同的子网互相通信，这里我们就用RIP方便添加和维护，对两个路由器添加RIP配置



图3.19 路由器0 RIP维护



图3.20 路由器1 RIP维护

学院和宿舍之间的限制我们可以通过子网AS之间的边缘路由器的ACL实现，如下如图

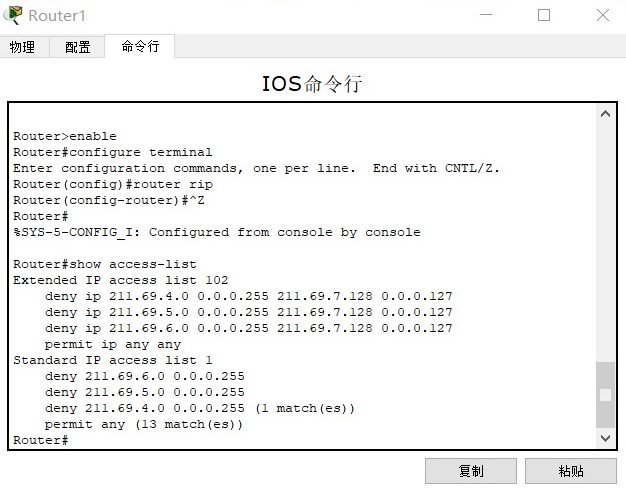


图3.21 ACL限制

这样我们就能实现学院和宿舍之间的约束。

这里对于笔记本的设置如下

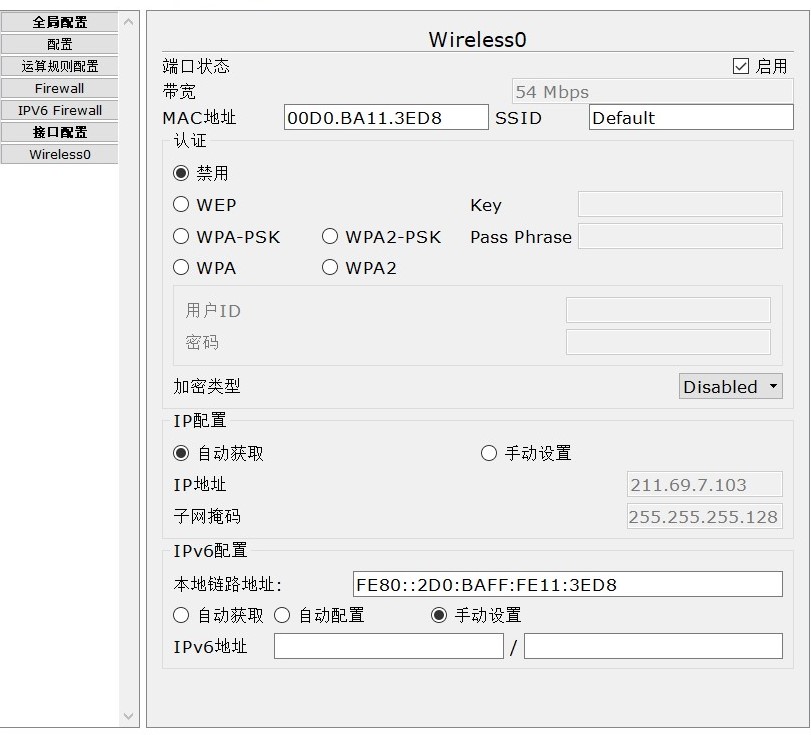


图3.22 无线网卡设定

这里我们采用DHCP协议也是方便我们的调整。

### **3.4.3 结果分析**

完成过后使用PDU列表进行连通性测试



图3.23 PDU列表

在学生宿舍内PC互相ping通，学生宿舍间可互相访问。

学院内的PC之间互相ping通，学院间可互相访问。

学生宿舍的PC ping学院内的PC，结果ping 不通，学生宿舍和学院间不能互相访问。

学生宿舍和学院的PC上ping图书馆内PC，能ping通，学生宿舍和学院皆可访问图书馆。

图书馆的笔记本上尝试访问网络资源，能正常访问，验证了图书馆的无线上网功能

完成了任务目标。

## **其它需要说明的问题**

无

# 心得体会与建议

## **4.1 心得体会**

在做Web服务器软件实验的时候，掌握了Socket编程的技术，能用各种套接字函数轻松搭建服务器和客户端之间的通信桥梁。我通过实现基于不同协议的可靠数据传输，深入理解了数据在网络中的传输过程和保障机制，学会了怎么处理丢包、乱序等网络问题，确保数据准确无误地传递。

对HTTP协议有了深入的理解，能够正确解析客户端请求，并根据请求内容从服务器文件系统中准确获取资源，构建并发送符合协议规范的响应报文。这个过程让我对Web服务器的工作原理有了更清晰的认识。

通过设计和实现GBN、SR和简化版TCP协议，我对可靠数据传输的原理和机制有了全面而深入的了解。我理解了滑动窗口、序列号、确认机制等关键概念在保证数据可靠性方面的重要作用，并能根据不同协议的特点，准确实现数据的发送、接收、确认和重传逻辑。

在处理网络环境中的丢包、报文损坏和乱序等问题时，我学会了用各种技术手段确保协议的稳定性和可靠性。比如，在GBN协议中，通过合理设置窗口大小和基序号，有效控制数据的发送和重传；在SR协议中，利用选择性重传机制，提高了数据传输效率；在简化版TCP协议中，实现了快速重传和超时重传等功能，进一步增强了协议对网络异常情况的适应能力。

基于CPT的组网实验让我学会了根据网络规划合理分配IP地址，深入理解了子网划分的原理和方法，能准确为不同子网和设备配置合适的IP地址，确保网络的互联互通。同时，熟练掌握了路由器和交换机的配置命令，能通过图形化界面和命令行方式对设备进行详细配置，包括接口设置、路由协议配置（比如RIP、OSPF）、VLAN划分以及访问控制列表（ACL）的设置等。

## **4.2 建议**

报告和之前一样写一个就行好吗（我要累死了）