

**计算机网络课程设计报告**



学 号 201706062412

姓 名 林英琮

班 级 数媒1702

提交日期 2019.1.16

**一、课程的目的和任务**

本课程要求学生在“计算机网络原理”理论学习基础上，验证和加深对计算机网络概念的理解。通过本课程的实践，培养学生独立思考、综合分析和动手实践的能力。

通过本课程的学习，达到如下目标：

(1). 理解计算机网络体系结构和工作原理，掌握常用的网络命令，能够对命令的功能进行解释，分析命令执行结果，得到合理有效结论。

(2). 掌握Packet Tracer软件的操作方法，能够使用该软件模拟组网、配置交换机、路由器，能够按照实验方案实施仿真实验，采集和整理数据。

(3). 具备网络编程能力，能够设计抓包程序获取数据包，结合相关协议对数据包各个字段的含义进行分析、处理和解释，获取有效结论。

**二、实验内容**

1. **任务一**

(1) 写出常用网络命令ipconfig, ping, netstat, tracert, arp, telnet的功能；

(2) 在windows环境下使用上述网络命令进行网络状态监测和跟踪，给出相应的截图和对结果的解释。（22分）

要求写出以上各条网络命令的基本使用方法并给出使用结果截图，说明其工作原理。

Ipconfig 、ping、netstat和arp各3分，tracert、telnet5分，如网络命令的使用说明是来自百度百科（即大篇幅抄袭）或对命令使用结果没有任何说明，酌情扣分。

**实验内容：**

**-> 功能简介：**

ipconfig: 于获取本机当前所有的TCP/IP的设置情况、刷新动态主机配置协议（DHCP）和域名系统（DNS）设置，包括IP地址、子网掩码、默认网关、DNS服务器等信息。

ping: 是一个使用频率极高的ICMP协议的程序，用于确定本地主机是否能与另一台主机交换（发送与接收）数据包、检查本机到某个网络是否连通。

可以获取延时、大致丢包率、TTL等信息。

netstat: 用于监控TCP/IP网络，可以显示与IP,TCP,UDP和ICMP协议相关的统计数据、路由表、实际的网络连接以及每一个网络接口设备的状态信息。

一般用于检验本机各端口的连接情况。

tracert: 是跟踪路由路径的一个实用程序，用于跟踪数据包访问目标网络，记录所走过的路径。

arp: 地址解析协议，用于确定对应IP地址的网卡物理地址、查询本机当前ARP缓存中IP地址和MAC地址的对应关系。

此外，也可以人工方式输入静态的网卡物理/IP地址对，对常用主句进行这项工作可以减少网络上的信息量。

telnet: 可以检查远程机器或本地机器上某端口是否打开，也可以让本地机器作为虚拟终端对远程机器进行操作。

**-> 实际操作分析**

本机为有线连接小米路由器mini，连接L2TP协议的中国移动校园网。

1. ipconfig:

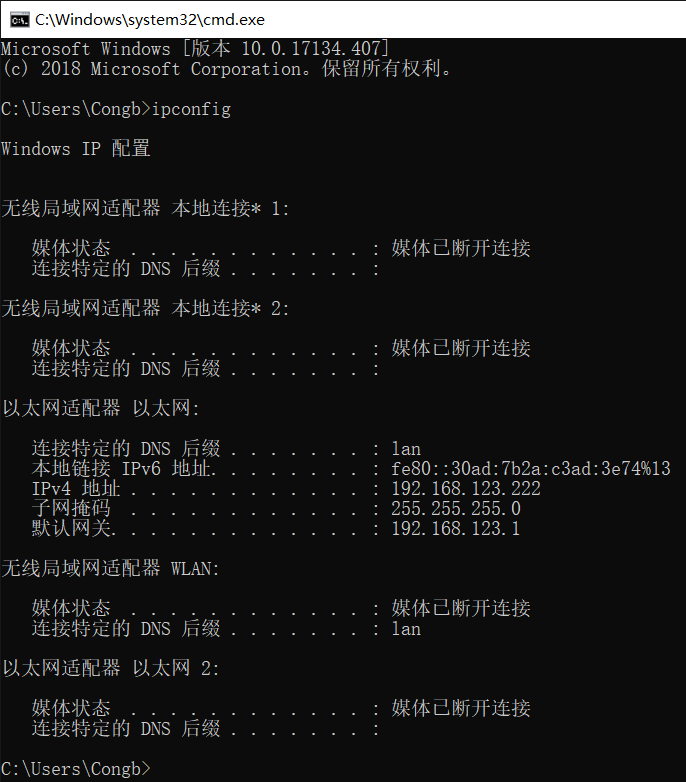


图1.1

如图所示，显示了本机各网络的TCP/IP配置情况。

可见：当前配置为：IPv6地址：fe80::30ad:7b2a:c3ad:3e74%13，IPv4地址：192.168.123.222，子网掩码：255.255.255.0，默认网关：192.168.123.1。

ping:

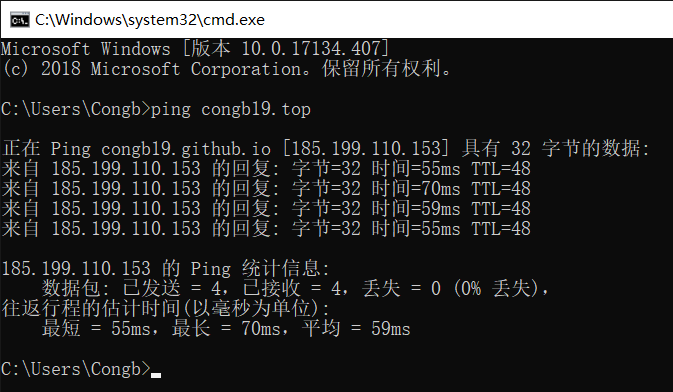


图1.2

如图所示，显示了本机到网络[185.199.110.153]的连通情况。

发送了32字节的数据包，平均延时为59ms，没有丢包，可见，本机到网络[185.199.110.153]连通顺畅。

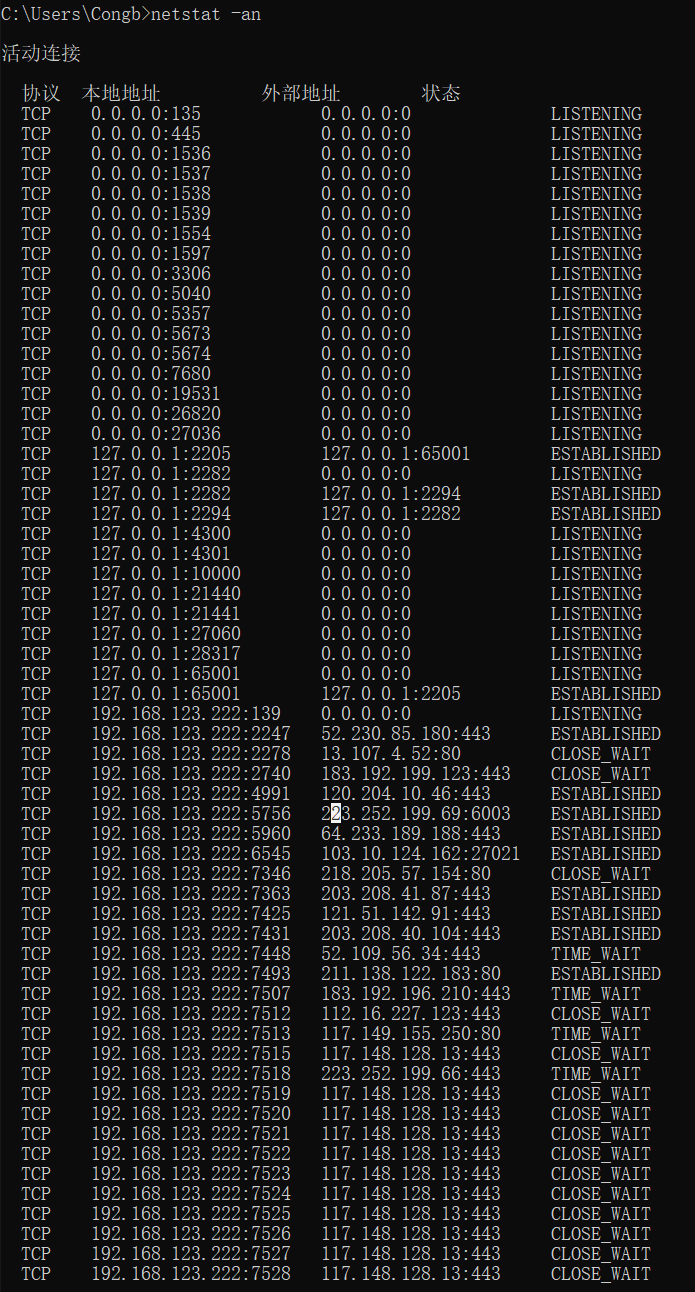
netstat: 

图1.3

如图所示，显示了路由表、实际的网络连接以及每一个网络接口设备的状态信息，非常详细，包括了协议、本地地址、外部地址和当前状态。

tracert:

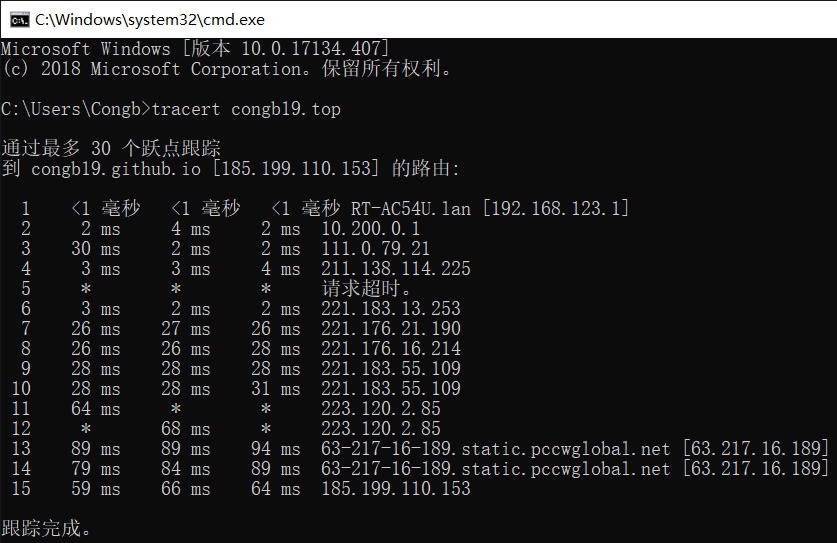


图1.4

发送了一个TTL为1的数据包，跟踪它访问目标网络的过程，记录了访问到每一个节点的延时和地址，知道跟踪完成。

如图所示，通过15个节点的跟踪，访问到了网络[185.199.110.153]，跟踪完成。

arp:

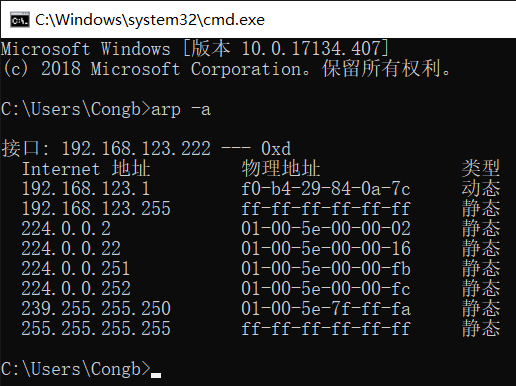


图1.5

如图，用arp -a查询到了本机中ARP缓存的信息，并显示。

包括Ip地址和MAC地址的对应关系和类型。

用arp -d等其他操作还可以对本机ARP缓存进行清除等操作。

telnet:

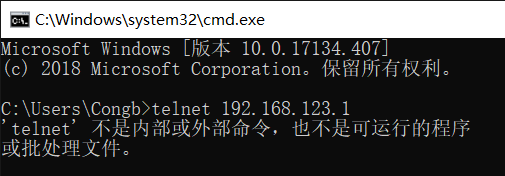


图1.6

本机似乎不支持telnet，在服务里，也没有找到开启telnet的选项

图1.7

故无法测试。

1. **任务二**

(1) 安装packet tracer，在packet tracer仿真环境下，熟悉交换机命令、交换机初始化配置；（38 分）

(2) 在交换机上实现VLAN配置；

要求：创建三个VLAN，给出拓扑，查看VLAN信息

(3) 基于Console控制台登录配置路由器，学习路由器配置相关命令；

(4) 基于packet tracer构建网络环境，分别进行静态路由配置和基于RIP的动态路由配置。

注：使用方法参考实验指导书

要求：静态路由配置拓扑中至少4个路由器；RIP动态路由配置中源站和目的站之间设置两条跳数不同的路径，通过RIP配置后查看选择的是哪条路径。 节点包括主机、路由器、交换机等都要根据个人名字首字母加学号(林英琮，201706062412)的后两位(L12)进行命名。

要求写出相应的步骤，给出截图和文字说明。

交换机和路由器基本配置命令行截图是否正确。（6分）

VLAN主要看其拓扑图以及配置截图或命令行截图是否正确。（10分）

静态路由配置主要看其各个路由器路由表配置是否正确。（12分）

RIP路由主要看其各个路由器路由表配置是否正确，路由跟踪后选路是否正确。（10分）

**实验内容：**

1. 安装packet tracer。在packet tracer仿真环境下，熟悉交换机命令、交换机初始化配置。

交换机基本配置-拓扑图：

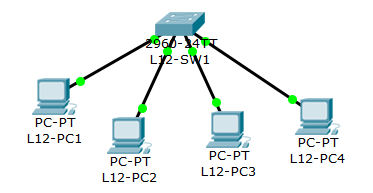


图2.1

配置了交换机主机名（L12-SW1）、加密使能密码（1234）、虚拟终端口令（5678）及超时时间（5分钟）、禁止名称解析服务。

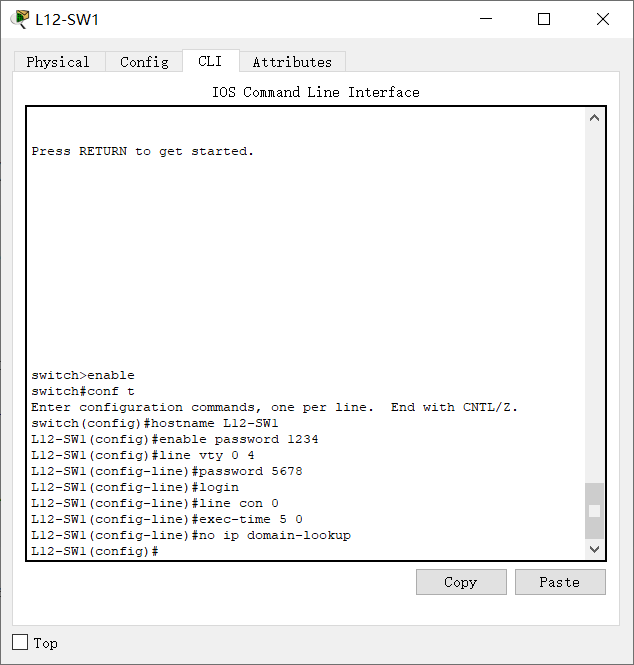


图2.2

配置了交换机管理 IP 地址（192.168.1.1）、子网掩码（255.255.255.0）、默认网关（192.168.1.254）。

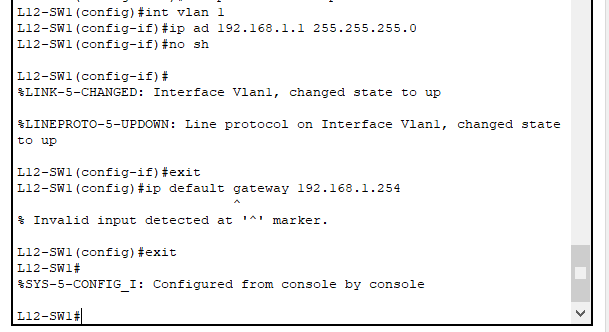


图2.3

3. 配置交换机端口速度（100Mbps）、端口双工方式（半双工）。

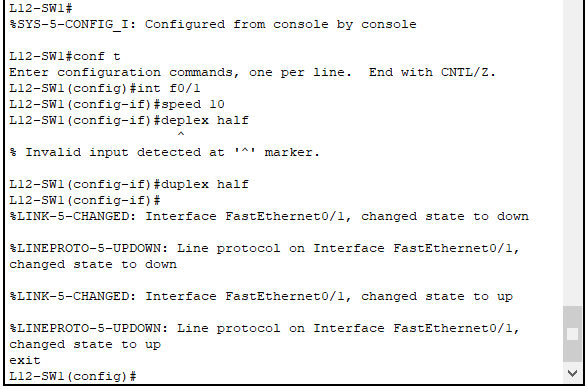


图2.4

1. 在交换机上实现VLAN配置；

要求：创建三个VLAN，给出拓扑，查看VLAN信息

VLAN拓扑图：

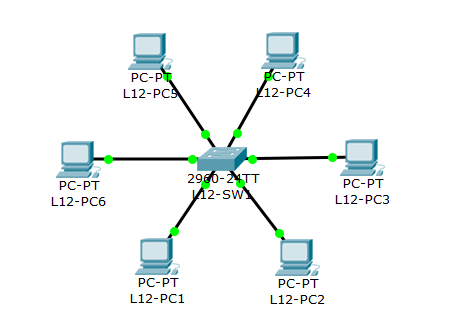


图2.5

新创建三个vlan: (vlanPEAR, vlanLEMON, vlanAPPLE)

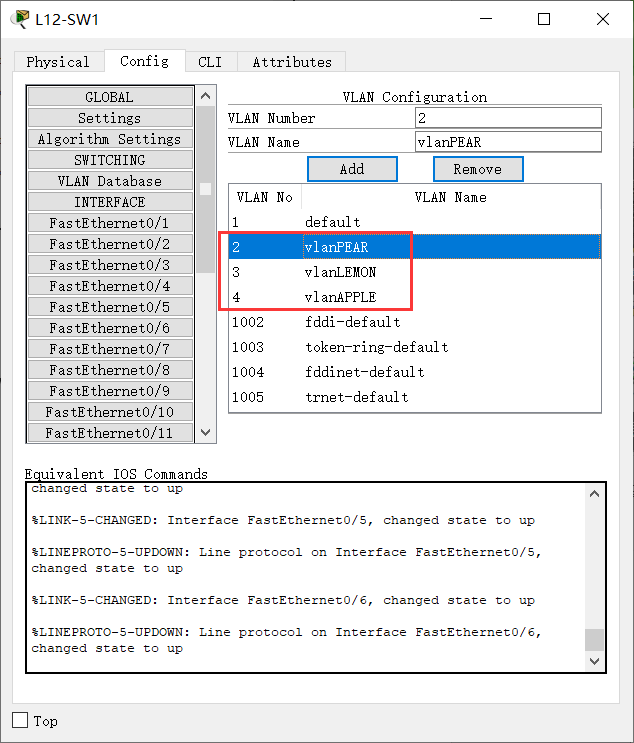


图2.6

设置六台主机的IP地址、子网掩码：

L12-PC1如图

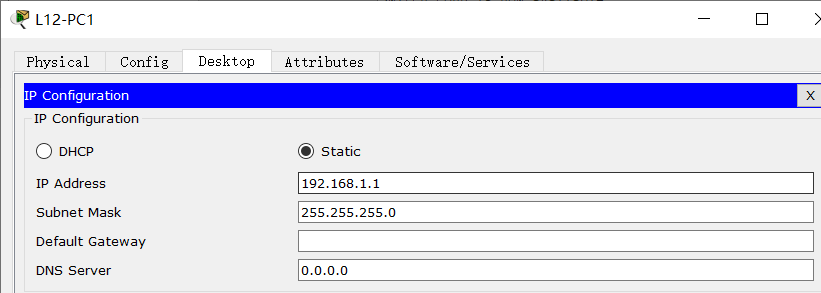


图2.7

后面五台IP依次为192.168.1.2，192.168.1.3，192.168.1.4，192.168.1.5，192.168.1.6。

把1-6号端口划分到各VLAN中：



图2.7

查看VLAN信息：

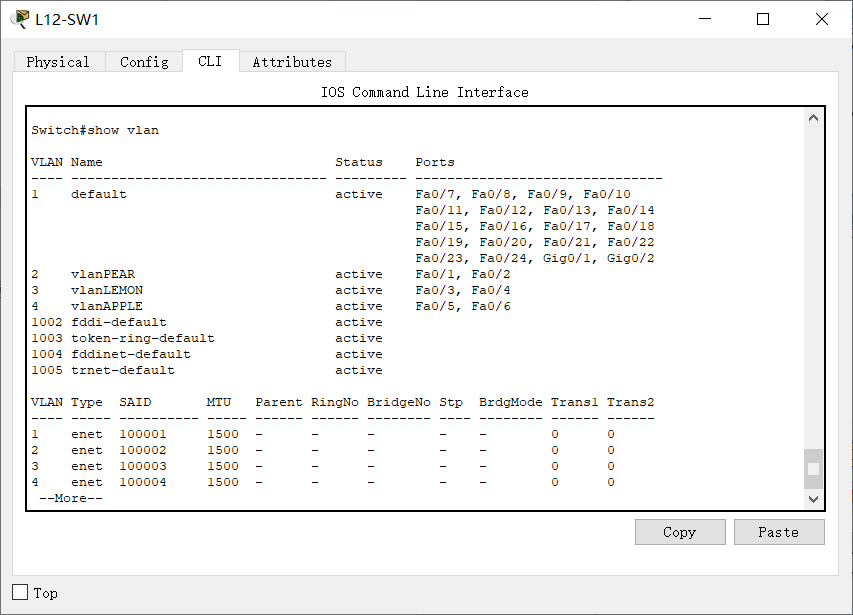


图2.8

测试VLAN：

PC1 ping PC2:

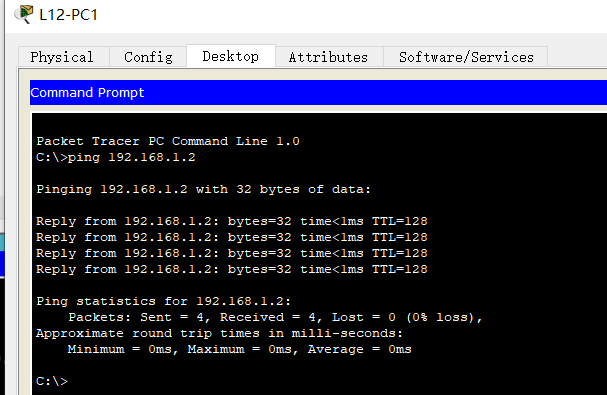


图2.9

成功ping通，表示PC1和PC2确实在一个VLAN内。(都在vlanPEAR中)。

PC3 ping PC5：

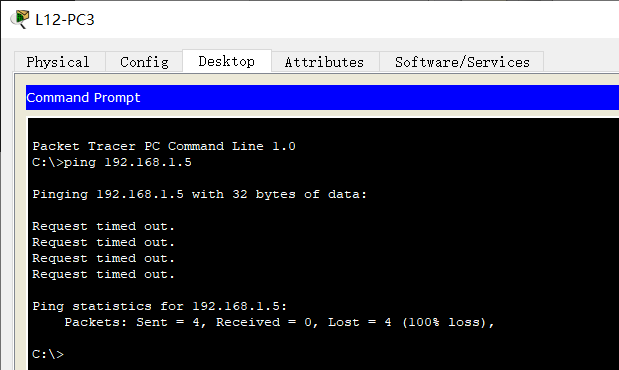


图2.10

未能ping通，表示PC3和PC5在不同的VLAN中。(PC3在vlanLEMON中，PC5在vlanAPPLE中)。

1. 基于Console控制台登录配置路由器，学习路由器配置相关命令；

路由器基本配置-拓扑图：

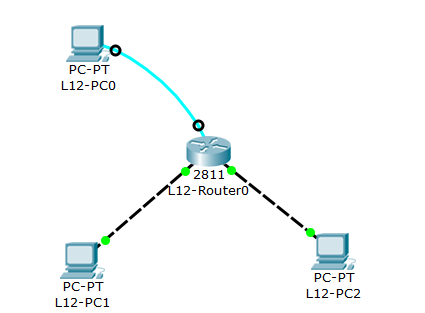


图2.3.1

配置用户名 和密码：

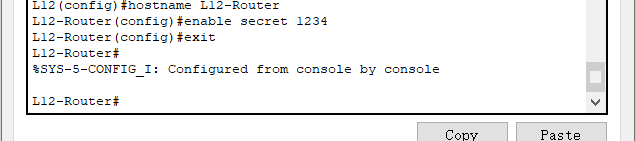


图2.3.2

配置各个端口的IP地址，并开启端口：

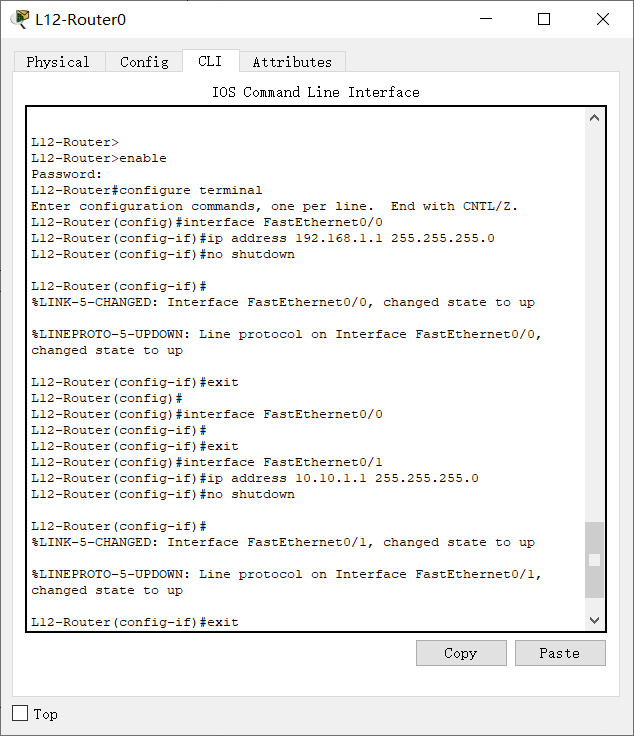


图2.3.3

测试：PC1的ipconfig和ping PC2：

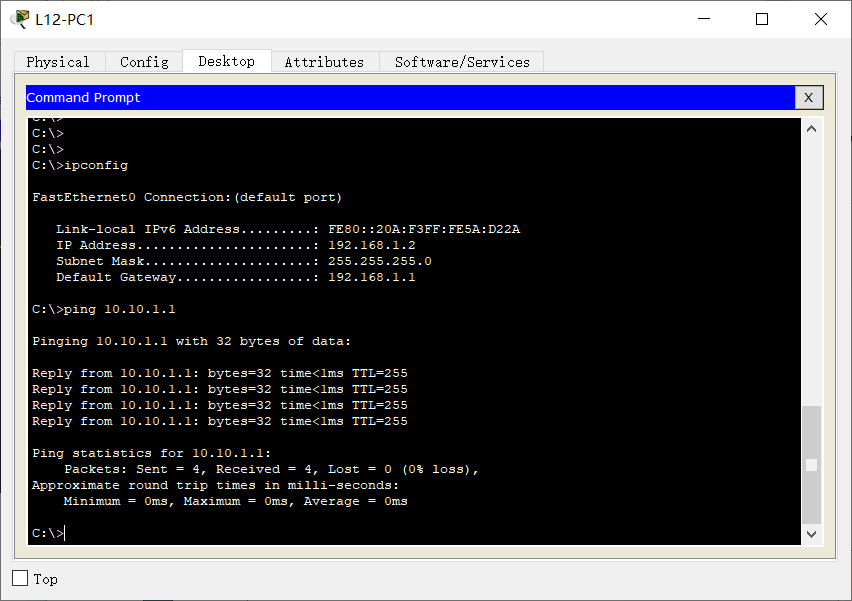


图2.3.4

PC1到PC2成功ping通，说明实验成功。

1. 基于packet tracer构建网络环境，分别进行静态路由配置和基于RIP的动态路由配置。

-> 静态路由配置：

拓扑图：

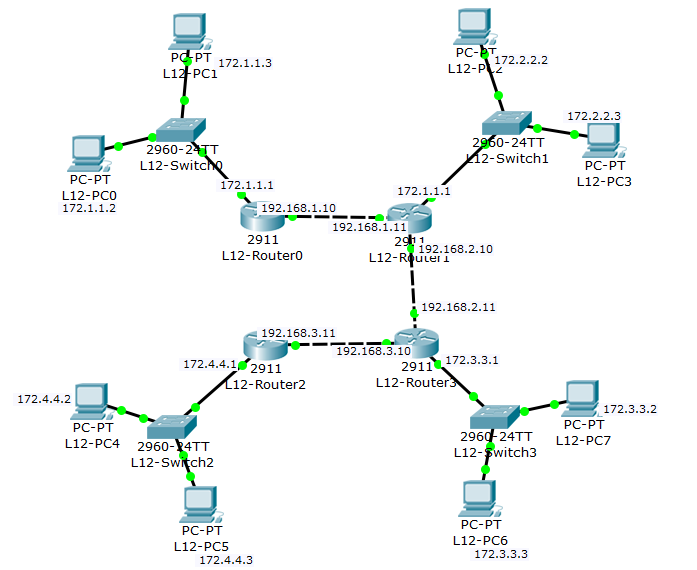


图2.4.1

如下配置路由表：

L12-Router0：

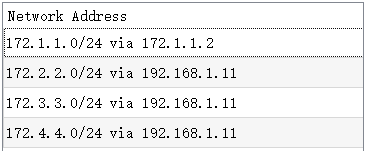


图2.4.2

L12-Router1：

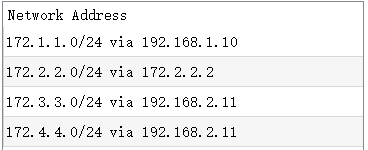


图2.4.3

L12-Router2：

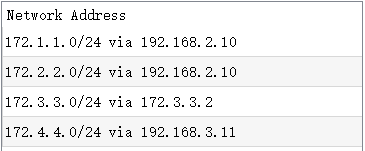


图2.4.4

L12-Router3：

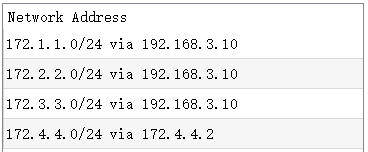


图2.4.5

测试：PC1 ping PC2、PC7、PC4：

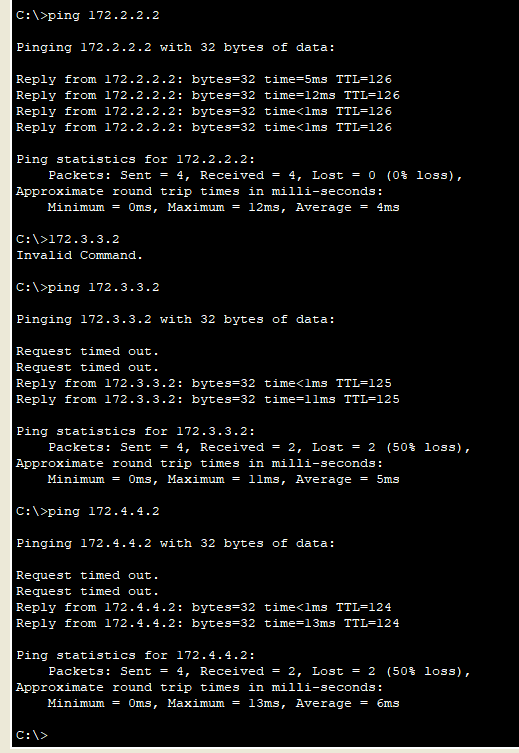


图2.4.6

全部ping通，表示静态路由配置完成。

-> RIP动态路由配置

拓扑图：

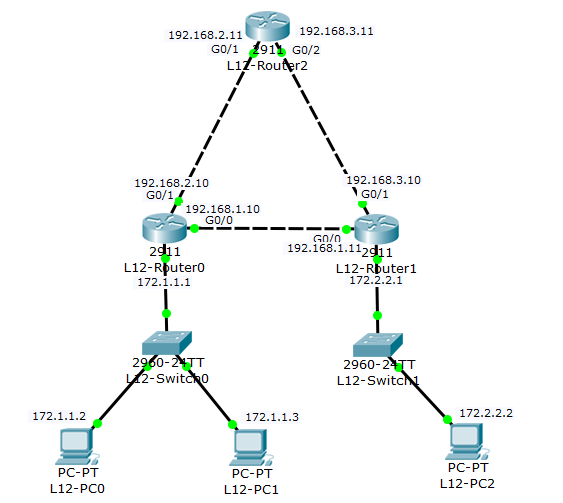


图2.4.7

L12-Router0路由表：

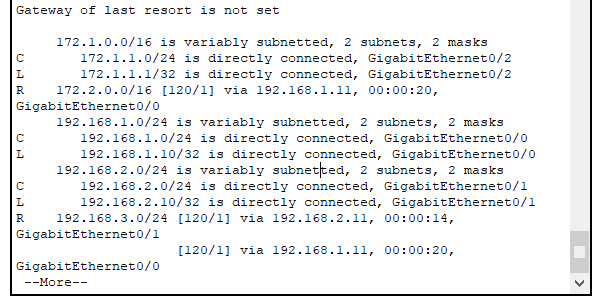


图2.4.8

L12-Router1路由表：

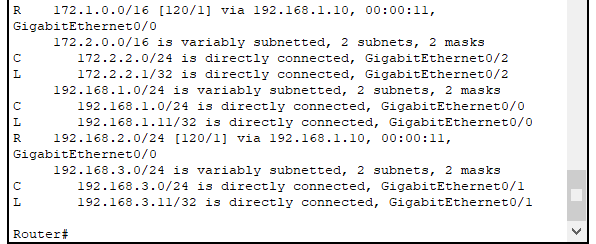


图2.4.9

L12-Router2路由表：

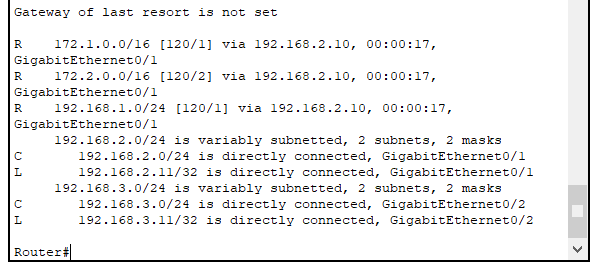


图2.4.10

测试：PC0 跟踪PC2

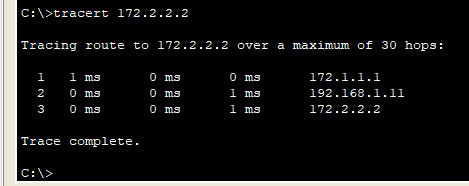


图2.4.11

发现：选择的是最短路径。

1. **任务三：网络编程**

编程要求：捕获本机网卡的IP包，对捕获的IP包进行解析。要求必须输出以下字段：版本号、总长度、标志位、片偏移、协议、源地址和目的地址。

要求有详细的说明文档，包括程序的设计思想、工作流程、关键问题、程序注释和对捕获包的解析截图。

编程语言不作要求，可使用自己熟悉的C、C++、java或C#等。（40 分）

给出IP包头的各个字段信息（10分）

说明自己使用的实验软件及操作系统（5分）

程序的设计思想（5分）

程序流程图（5分）

程序注释（5分）

实验结果与分析（5分）

实验总结与心得体会（5分）

**实验内容：**

项目代码分享于<https://github.com/Congb19/Capture_IP_Package>。

1. 源代码：

main.cpp

|  |
| --- |
| //  // Created by Congb on 2019/1/12.  //  #define \_WINSOCK\_DEPRECATED\_NO\_WARNINGS  #define \_CRT\_SECURE\_NO\_WARNINGS  #include <iostream>  #include <winsock2.h>  #include <windows.h>  #include <string.h>  #include <mstcpip.h>  #include <string>  #pragma comment(lib,"Ws2\_32.lib")  using namespace std;  struct IP\_head //IP首部  {  BYTE version\_len; // IP协议版本和IP首部长度。高4位为版本，低4位为首部的长度(单位为4B)  BYTE ser\_type; // 服务类型  WORD wPacketLen; // IP包总长度。包括首部，单位为byte。[Big endian]  WORD identification; // 标识，一般每个IP包的序号递增。[Big endian]  union  {  WORD flag; // 标志  WORD flagof; // 分段偏移  };  BYTE TTL; // 生存时间  BYTE Protocol\_Type; // 协议类型，见PROTOCOL\_TYPE定义  WORD Head\_checksum; // IP首部校验和[Big endian]  DWORD Source\_ip; // 源地址  DWORD Destination\_ip; // 目的地址  BYTE Options; // 选项  };  int cnt; // 记录序号  int Get\_IPdata(char \*buf, int len)  {  int n = len;  if (n >= sizeof(IP\_head))  {  IP\_head iphead;  iphead = \*(IP\_head\*)buf;  cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl ;  cout << "第 " << cnt++ << " 个IP数据包：" << endl;  cout << "—————————————————————————————————" << endl ;  cout << "版 本:" << "ipv"<< (iphead.version\_len >> 4) << endl;  cout << "首部长度:" << ((iphead.version\_len & 0x0F) << 2) << endl;//单位为4字节  cout << "服务类型:Priority: " << (iphead.ser\_type >> 5) << ",Service: " << ((iphead.ser\_type >> 1) & 0x0f) << endl;  cout << "总 长 度:" << ntohs(iphead.wPacketLen) << endl; //网络字节序转为主机字节序  cout << "标 识:" << ntohs(iphead.identification) << endl;  cout << "标 志:" << "DF=" << ((iphead.flag >> 14) & 0x01) << ",MF=" << ((iphead.flag >> 13) & 0x01) << endl;  cout << "片 偏 移:" << (iphead.flagof & 0x1fff) << endl;  cout << "生存时间:" << (int)iphead.TTL << endl;  cout << "协议类型:" << int(iphead.Protocol\_Type) << endl;  cout << "校 验 和:" << ntohs(iphead.Head\_checksum) << endl;  cout << "源 地 址:" << inet\_ntoa(\*(in\_addr\*)&iphead.Source\_ip) << endl;  cout << "目的地址:" << inet\_ntoa(\*(in\_addr\*)&iphead.Destination\_ip) << endl;  cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl << endl;  }  return 0;  }  void AutoWSACleanup()  {  ::WSACleanup();  }  int main()  {  int n;  WSADATA wd; // 用来存储被WSAStartup函数调用后返回的Windows Sockets数据  n = WSAStartup(MAKEWORD(2, 2), &wd); // WSAStartup：进行相应的socket库绑定  if (n)  {  cout << "WSAStartup函数错误！" << endl;  return -1;  }  atexit(AutoWSACleanup); // atexit函数是在正常程序退出时调用的登记函数  // WSACleanup()功能是终止Winsock 2 DLL(Ws2\_32.dll) 的使用  // 创建SOCKET  SOCKET sock = socket(AF\_INET, SOCK\_RAW, IPPROTO\_IP);  if (sock == INVALID\_SOCKET)  {  cout << WSAGetLastError(); // 该函数返回上次发生的网络错误  return 0;  }  // 获取本机地址  char name[128];  if (-1 == gethostname(name, sizeof(name))) // gethostname()返回本地主机的标准主机名。  {  closesocket(sock); // 关闭一个套接口  cout << WSAGetLastError();  return 0;  }  struct hostent \* pHostent;  pHostent = gethostbyname(name); // gethostbyname()返回对应于给定主机名的包含主机名字和地址信息的hostent结构的指针  // 绑定本地地址到SOCKET句柄  sockaddr\_in addr;  addr.sin\_family = AF\_INET;  addr.sin\_addr = \*(in\_addr\*)pHostent->h\_addr\_list[0]; //IP  addr.sin\_port = 8888; //端口，IP层端口可随意填  if (SOCKET\_ERROR == bind(sock, (sockaddr \*)&addr, sizeof(addr)))  {  closesocket(sock);  cout << WSAGetLastError();  return 0;  }  // 设置该SOCKET为接收所有流经绑定的IP的网卡的所有数据，包括接收和发送的数据包  u\_long sioarg = 1;  DWORD wt = 0;  if (SOCKET\_ERROR == WSAIoctl(sock, SIO\_RCVALL, &sioarg, sizeof(sioarg), NULL, 0, &wt, NULL, NULL))// WSAIoctl()控制一个套接口的模式  {  closesocket(sock);  cout << WSAGetLastError();  return 0;  }  // 只需要接收数据，因此设置为阻塞IO，使用最简单的IO模型  u\_long bioarg = 0;  if (SOCKET\_ERROR == ioctlsocket(sock, FIONBIO, &bioarg))// ioctlsocket()功能是控制套接口的模式  {  closesocket(sock);  cout << WSAGetLastError();  return 0;  }  // 开始接收数据  // 因为前面已经设置为阻塞IO，recv在接收到数据前不会返回。  cnt = 1;  char buf[65535];  int len = 0;  string temp;  do  {  len = recv(sock, buf, sizeof(buf), 0);// 用recv函数从TCP连接的另一端接收数据  if (len > 0)  {  Get\_IPdata(buf, len);  }  cout << endl << "输入回车获取下一个数据包";  getline(cin, temp);  cout << endl;  } while (len > 0);  closesocket(sock);  system("pause");  } |

1. IP包头的各个字段信息：

Version:版本号

Header Length:IP包头长度

Type of service:服务类型

Total Length:IP包总长

Identifier:标识符

Flags:标记

Fragment Offset:片偏移

Time to Live:生存时间

Protocol:协议

Header Checksum:头部校验

Source:源地址

Destination:目的地址

Options:可选项

Padding:填充

1. 使用的实验软件及操作系统：

程序编译软件：JetBrains CLion 2018.3.2

操作系统：Windows 10 专业版 64位 (10.0, 版本17763)

（做完任务一后本机进行了系统升级，从17134到了17763，因此截图中的系统版本不同）

1. 程序的设计思想：
2. 程序流程图：

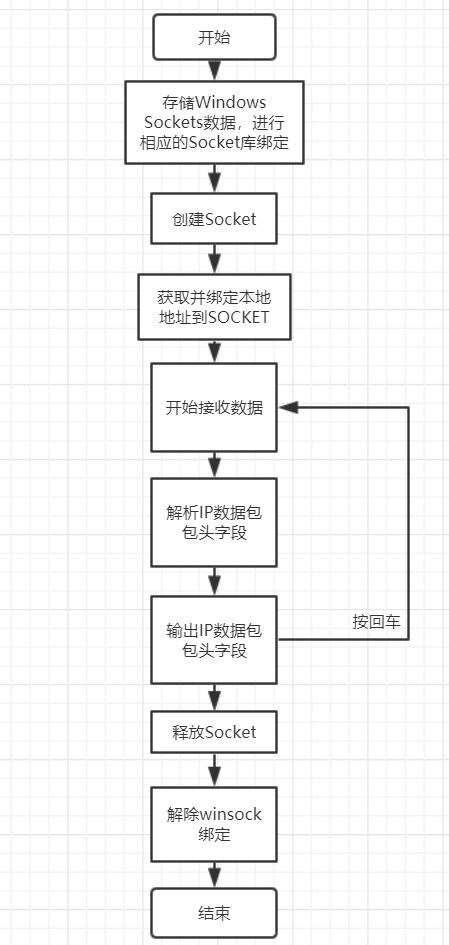


图3.5.1

1. 程序注释：

见源代码的//注释部分。

1. 实验结果与分析：

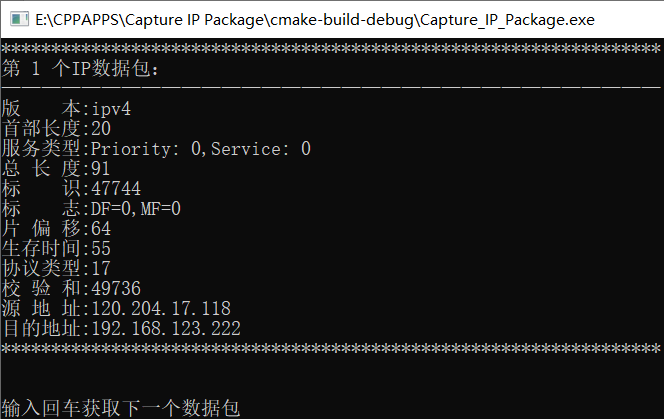


图3.5.2

结果分析如图所示，可以得知捕获的这个数据包的信息如下：

版 本:ipv4

首部长度:20

服务类型:Priority: 0,Service: 0

总 长 度:64

标 识:1015

标 志:DF=0,MF=0

片 偏 移:0

生存时间:128

协议类型:17

校 验 和:0

源 地 址:192.168.123.222

目的地址:103.10.124.163

1. 实验总结与心得体会：

**三、课程设计总结体会**