《网 络 与 通 信》

实验指导书（V1.0）

钱权 袁方 编写

上海大学计算机工程与科学学院

2007 年 9 月

目 录

目 录 2
实验一：常用网络工具的使用 4
1.1【实验目的】 4
1.2【实验内容】 4
1.3【实验原理】 4
1.3.1 常用网络服务的配置 4
1.3.2 常用网络命令的使用 12
1.4【实验方式】 18
1.5【实验报告】 18
实验二、Socket通信实验 19
2.1【实验目的】 19
2.2【实验内容】 19
2.3【实验原理】 20
2.3.1 Socket通信原理 20
2.3.2 样例程序（Java语言） 29
2.4【实验方式】 32
2.5【实验报告】 32
实验三、数据包结构分析实验 33
3.1【实验目的】 33
3.2【实验内容】 33
3.3【实验原理】 34
3.3.1 网络监听基本原理 34
3.3.2 Sniffer软件使用 34
3.3.3 数据报文解码详解 36
3.4【实验方式】 40
3.5【实验报告】 40
实验四 网络层路由实验（静态路由实验） 41
4.1【实验目的】 41
4.2【实验原理】 41
4.2.1 Linux机器的静态路由配置 41
4.2.2 Cisco路由器静态路由配置（NetSim模拟） 41
4.3【实验内容】 42
4.3.1 配置Linux机器为 1 台IP路由器 42
4.3.2 设置Cisco路由器的静态路由 44
4.4【实验方式】 46
4.5【实验报告】 46
实验五 网络层路由实验（动态路由实验） 47
5.1【实验目的】 47
5.2【实验原理】 47
5.2.1 RIP协议 47
5.2.2 OSPF协议 48
5.3【实验内容】 50

5.3.1 RIP路由实验 50
5.3.2 OSPF路由实验 50
5.4【实验方式】 50
5.5【实验报告】 50
附件一、Sniffer软件使用.....51
附件二、NetSim软件使用 51

实验一：常用网络工具的使用

1.1【实验目的】

1、掌握 Windows 系统常用网络服务的配置方法

2、掌握常用的 TCP/IP 网络中网络测试和网络诊断命令的使用方法

1.2【实验内容】

1、使用 Windows 操作系统，了解 Telnet、FTP、WEB 服务等网络服务的配置方法；

2、使用 Windows 操作系统，掌握常用网络测试命令的使用方法。

1.3【实验原理】

本实验主要介绍网络服务的配置以及常用 TCP/IP 网络测试与网络故障诊断命令的使用。

1.3.1 常用网络服务的配置

1.3.1.1 TELNET 服务与客户端工具使用

Telnet 是 TCP/IP 协议簇中的一个虚拟终端协议，它允许连接到远程主机。通过使用 Telnet 命令，远程设备可以做为一个虚拟终端进行远程登录，还可以检查源站点和目的站点的应用层软件的可用性。如果我们能够使用 telnet 命令远程登陆，那说明网络通过了所有的测试，是正常连通的。telnet 运行在 OSI 参考模型的应用层，它利用 TCP 来保证正确和有序的在客户机和服务器之间传输数据。

Telnet 客户端使用：

Telnet 包括两个部分：客户端和服务器端。客户端就是 Telnet 客户机，而服务端是提供 Telnet 网络服务的系统。Telnet 客户机的作用是：

(1)建立一个网络与服务器间的 TCP 连接

(2)以方便的方式接收输入

(3)对某些标准的格式化输入作重新格式化后传送给服务器。

(4)以某些标准的格式化从服务器中接受输出

(5)重新格式化显示给自己的输出

Telnet 可发送除了"escape"的任何字符到远程主机上。因为"escape"字符在 Telnet 中是客户机的一个特殊的命令模式，它的默认值是"Ctrl-]"。但要注意不要与键盘上的 Esc 键混淆，我们可以设定"escape"为任意某个字符，只是对 Telnet 来说意味着该字符不可能再被传送到远程主机上，而 Esc 键是一非打印字符，Telnet 用它来删除远程系统中的命令。

可以仅仅键入 Telnet，后面不带机器字句。这种情况下所看到的是 Telnet>，这是告知 Telnet 在等待键入命令，比如键入问号"?"那么就得到一个有用的命令表。不同版本的 Telnet 程序可用的命令有所不同，需要具体试用。大多数的 Telnet 程序都配有如下命令：

```
telnet> ?
Commands may be abbreviated. Commands are:
close           close current connection
logout          forcibly logout remote user and close the connection
display         display operating parameters
mode            try to enter line or character mode ('mode ?' for more)
open            connect to a site
quit            exit telnet
send            transmit special characters ('send ?' for more)
set             set operating parameters ('set ?' for more)
unset           unset operating parameters ('unset ?' for more)
status          print status information
toggle          toggle operating parameters ('toggle ?' for more)
z               suspend telnet
?               print help information
!               invoke a subshell
environ         change environment variables ('environ ?' for more)
slc             change state of special charaters ('slc ?' for more)
telnet>
```

下面我们简单介绍各命令的功能。

- Close

终止当前已经建立的联接或正在进行的联接。自动将本地系统与远程系统切断。有时进入某个网络时由于某种原因会被锁住，远程主机系统不能识别任何本

地用户在键盘上键入的命令，甚至不能用 logout 命令退出 Telnet 状态，这时可以用^]键，进入 Telnet 的命令状态，然后用 close 命令切断当前的联接，重新开始新的登录。用 close 命令切断联接后，可用 o 或 open 加主机名再打开一个新的联接。

- Display

显示系统当前的操作参数。在 Telnet 的命令状态下，键入 display，按回车键，屏幕将显示当前系统的操作参数，例如，在紧急状态下是否发送中断字符，是否重新确认控制字符，以及^ E(回应)，^](进入命令状态)，^ C(中断)，^ U(删除一行)等键盘命令的含义。

- Mode

进入逐行方式(line)：用户每键入一行信息，本地系统向远端主机发送一次；或逐个字符方式(character)：用户每键入一个字符，本地系统向远端主机发送一次。

- open(或 o) 主机名

与指定的这台主机建立 Telnet 联接，同“telnet 主机名”命令的意义相同。在打开一个新的连接前，必须终止当前所有的联接。也就是说在同一窗口下一台本地机同一时间内只能与一台远程主机建立联接（在 Windows 或类 Windows 的图形界面环境下可打开多个窗口，建立与多台主机的联接）。这是 Telnet 的一个特性。

- Quit

退出 Telnet 应用进程，回到本地系统，任何 Telnet 命令不再起作用。

- Logout

强制关闭登录的用户帐号，并关闭连接，退回本地系统（它与许多系统下的 EXIT 具有相同功能）。

- Send

已经登录到某台主机后，可以通过 send 命令发送一些信息到远程系统上。关于 send 命令的详细信息可以在 Telnet 的命令状态下，通过键入 send ?命令获得。

- Set

设置所有可以用 display 命令显示的操作参数。例如，设置^ E 为启动本地回应开关命令，^]为进入 Telnet 命令状态的命令，^ U 为删除一行，用?显示帮助信息，等等。例如：“set echo ^ E”表示本地回应开关为^ E。关于 set 命令的详细信息可以在 Telnet 命令状态下，通过键入 set ?命令获得。

- unset

取消已设置的用 display 命令显示的操作参数。它与 set 命令功能刚好相反。

- Status

显示当前状态信息。该命令只有已经登录到某一台主机后才有效。

■ Toggle

激活某些操作参数，这些参数决定 Telnet 对事件的响应方式，例如，激活在发出中断命令后，系统自动排出全部存储结果(autofluh)；收到故障反馈后，自动映射(crmod)等等。关于 toggle 命令的详细信息可以在 Telnet 命令状态下，用 toggle ? 命令获得。

■ Z

暂时中止 Telnet 通信，使本地系统可以执行其它命令，例如回到本地系统，看看有没有新邮件到来等。一般用 fg 命令可以恢复原来的联接或用 open(或 o)命令建立新的联接。也有一些系统在执行 z 命令后，便退出 Telnet 状态。

■ ?

显示帮助信息，帮助用户了解系统可以提供哪些命令、每个命令的用法。

■ ！、environ、slc

这三个命令是 UNIX 系统下的命令，有些系统不提供，因此这里不作详细解释。

这里需要注意的是，虽然用 close 或 quit 命令都可关闭当前的 Telnet 联接，但是最好还是用 logout 退出远程系统，以确保系统工作正常。logout 方式退出可确保 Telnet 进程确实终止和用户所要的数据存盘。

Telnet 服务器端配置

下面都是使用 tlntadmn 命令来配置 Telnet 服务器端。使用 tlntadmn 需要注意的地方有：

1、要管理的计算机和在其上使用 tlntadmn 命令的计算机都必须在运行 Windows NT、Windows 2000、Windows XP 或 Windows Server 2003 家族成员。

2、要使用 tlntadmn 命令，您必须使用管理凭据登录到本地计算机。要管理远程计算机，您还必须为远程计算机提供管理凭据。使用具有本地计算机和远程计算机管理凭据的帐户登录到本地计算机即可做到这一点。如果不能使用这种方法，可使用 -u 和 -p 参数为远程计算机提供管理凭据。

管理运行 Telnet Server 的本地或远程计算机。如果使用时不带参数，tlntadmn 将显示本地服务器设置。

■ 管理运行 Telnet Server 的计算机

tlntadmn [\\RemoteServer] [start] [stop] [pause] [continue] [-u UserName-p Password]

参数说明：

\\ RemoteServer：指定要管理的远程服务器名称。如果没有指定服务器，则假定使用本地服务器。

Start：启动 Telnet Server。

Stop：停止 Telnet Server。

Pause：中断 Telnet Server。

Continue：恢复 Telnet Server。

-u UserName -p Password：指定要管理的远程服务器的管理凭据。如果您要管理远程服务器，但未使用管理凭据登录，则必须提供该参数。

/?：在命令提示符下显示帮助。

- 管理 Telnet 会话

tlntadmn [\\RemoteServer] [-s] [-k{SessionID | all}] [-m {SessionID | all} "Message"]

参数说明：

\\ RemoteServer：指定要管理的远程服务器名称。如果没有指定服务器，则假定使用本地服务器。

-s：显示活动的 Telnet 会话。

-k{SessionID | all}：终止会话。键入会话 ID 以终止特定会话，或者键入 all 终止所有会话。

-m {SessionID | all} "Message"：向一个或多个会话发送消息。键入会话 ID 以便将消息发送给特定会话，或者键入 all 将消息发送给所有会话。在引号内键入要发送的消息（即 "Message"）。

/?：在命令提示符下显示帮助。

注释

要在管理远程服务器时使用这些参数，您必须使用管理凭据登录到远程服务器。

- 在运行 Telnet Server 的计算机上设置连接的最大数量

tlntadmn [\\RemoteServer] config [maxconn=PositiveInteger] [-u UserName-p Password]

参数说明：

maxconn= PositiveInteger：设置连接的最大数量。必须使用小于 10,000,000 的正整数来指定该数。

- 在运行 Telnet Server 的计算机上设置失败登录尝试的最大次数

tlntadmn [\\RemoteServer] config [maxfail=PositiveInteger] [-u UserName-p

Password]

参数说明：

maxfail= PositiveInteger：设置允许用户执行的最大失败登录尝试次数。必须用一个小于 100 的正整数来指定该数。

- 在运行 Telnet Server 的计算机上设置操作模式

tlntadmn [\\RemoteServer] config [mode={console | stream}] [-u UserName-p Password]

参数说明：

mode={ console| stream}：指定操作模式。

- 在运行 Telnet Server 的计算机上设置 Telnet 端口

tlntadmn [\\RemoteServer] config [port=IntegerValue] [-u UserName-p Password]

参数说明：

port= IntegerValue：设置 Telnet 端口。必须使用小于 1,024 的整数指定端口。

- 在运行 Telnet Server 的计算机上设置身份验证方法

tlntadmn [\\RemoteServer] config [sec=[{+ | -}ntlm][{+ | -}passwd]] [-u UserName-p Password]

参数说明：

sec=[{+|-}ntlm][{+|-}passwd] ：指定是否使用 NTLM、密码或这两者对登录尝试进行身份验证。要使用特定类型的身份验证，请在该身份验证类型前键入加号 (+)。要防止使用特定类型的身份验证，请在该类身份验证之前键入减号 (-)。

NTLM 是两台计算机（一台或两台计算机在运行 Windows NT）之间的事务验证协议。另外，NTLM 是为没有加入到域中的计算机（如单机服务器和工作组）提供的身份验证协议。可以使用本地 Windows 用户名和密码或域帐户信息来访问 Telnet 服务器。

如果不使用 NTLM 身份验证选项，则用户名和密码将以明文方式发送到 Telnet 服务器上。如果使用 NTLM 身份验证，Telnet 客户会使用 Windows XP 安全环境进行身份验证（要首先登陆到服务器中，利用域用户名和密码），也不提示用户提供用户名和密码。用户名和密码是加密的。

如果利用 NTLM 身份验证连接 Telnet 服务器，则由于这种身份验证的限制使您将无法访问其他网络资源。

- 在运行 Telnet Server 的计算机上设置空闲会话超时

tlntadmn [\\RemoteServer] config [timeout=hh:mm:ss] [-u UserName-p Password]

参数说明：

timeout= hh :mm :ss：以小时、分钟和秒为单位设置超时时间段。

1.3.1.2 使用 TELNET 命令分析 HTTP 协议

使用 Telnet 命令可以用来分析很多应用层的协议，包括：HTTP、SMTP、FTP 等。下面以 HTTP 协议为例，了解分析的过程。

1. Start a DOS command prompt
2. From the command prompt type

```
telnet www.microsoft.com 80
```

3. Turn on localecho so you can see what you are doing.
    1. Type Ctrl+] (hold down the Ctrl key and press the right bracket)
    2. Type

```
set localecho
```

    3. Press Enter on a blank line
4. type
5. GET / HTTP/1.1
6. Host:www.microsoft.com
7. Press Enter twice

1.3.1.3 FTP 服务与 FTP 客户端工具使用

■ 使用 FTP 客户程序访问 FTP 服务

步骤：

(1) 进入 FTP 协议环境

$>ftp

(2) 建立和远程主机的连接

ftp>open 远程主机名或 IP 地址

此时需输入用户名和口令：

a、有名连接

使用真实的用户名和口令

b、匿名连接

使用 anonymous 作为用户名，使用 e-mail 地址作为口令

(3) 使用 FTP 操作命令

dir—查看当前目录

cd—改变当前工作目录

pwd—显示当前工作目录

binary—设置二进制传输方式

ascii—设置 ASCII 传输方式（默认方式）

get—单个文件下载

mget—多个文件下载

put—单个文件上传

mput—多个文件上传

?—帮助命令

(4) 关闭和远程主机的连接

ftp>close

(5) 退出 FTP 协议环境

ftp>bye

- 使用 Web 浏览器访问 FTP 服务

除了使用命令行外，也可以通过在 Web 浏览器的地址栏中输入 FTP URL 来使用 FTP，如 ftp://主机名或 IP 地址

- 使用第三方的 FTP 客户端软件

例如：cuteftppro， FlashFXP，flashGet 等第三方软件，自己查阅有关资料。

- windows 环境下 FTP 服务器的设置

熟悉使用 IIS 配置 FTP 服务器，也可以使用其他 FTP 服务器端软件，例如：Serv-U 软件。

1.3.2 常用网络命令的使用

1.3.2.1 IPCONFIG 命令

主要功能：显示所有当前的 TCP/IP 网络配置值。

发现和解决 TCP/IP 网络问题时，先检查出现问题的计算机上的 TCP/IP 配置。可以使用 ipconfig 命令获得主机配置信息，包括 IP 地址、子网掩码和默认网关。

- 对于 Windows 95 和 Windows 98 的客户机，请使用 winipcfg 命令而不是 ipconfig 命令。
- 使用带 /all 选项的 ipconfig 命令时，将给出所有接口的详细配置报告，包括任何已配置的串行端口。使用 ipconfig /all，可以将命令输出重定向到某个文件，并将输出粘贴到其他文档中。也可以用该输出确认网络上每台计算机的 TCP/IP 配置，或者进一步调查 TCP/IP 网络问题。
- 如果计算机配置的 IP 地址与现有的 IP 地址重复，则子网掩码显示为 0.0.0.0。

下面的范例是 ipconfig /all 命令输出，该计算机配置成使用 DHCP 服务器动态配置 TCP/IP，并使用 WINS 和 DNS 服务器解析名称。

- **Windows 2000 IP Configuration**
- **Node Type.. . . . . : Hybrid**
- **IP Routing Enabled.. . . . : No**
- **WINS Proxy Enabled.. . . . : No**
- **Ethernet adapter Local Area Connection:**
- **Host Name.. . . . . : corp1.microsoft.com**
- **DNS Servers . . . . . : 10.1.0.200**
- **Description. . . . . : 3Com 3C90x Ethernet Adapter**
- **Physical Address. . . . . : 00-60-08-3E-46-07**
- **DHCP Enabled.. . . . . : Yes**
- **Autoconfiguration Enabled.: Yes**
- **IP Address. . . . . : 192.168.0.112**
- **Subnet Mask. . . . . : 255.255.0.0**
- **Default Gateway. . . . . : 192.168.0.1**
- **DHCP Server. . . . . : 10.1.0.50**

- **Primary WINS Server. . . . : 10.1.0.101**
- **Secondary WINS Server. . . : 10.1.0.102**
- **Lease Obtained.. . . . . : Wednesday, September 02, 1998 10:32:13 AM**
- **Lease Expires.. . . . . : Friday, September 18, 1998 10:32:13 AM**
-

1.3.2.2 Ping 命令

主要功能：测试连接

Ping 命令有助于验证 IP 级的连通性。发现和解决问题时，可以使用 Ping 向目标主机名或 IP 地址发送 ICMP 回应请求。当需要验证主机能否连接到 TCP/IP 网络和网络资源时，也可以使用 Ping。另外还可以使用 Ping 隔离网络硬件问题和不兼容配置。

使用 Ping 时应该执行以下步骤：

- Ping 环回地址验证是否在本地计算机上安装 TCP/IP 以及配置是否正确。
- **ping 127.0.0.1**
- Ping 本地计算机的 IP 地址验证是否正确地添加到网络。
- **ping IP_address_of_local_host**
- Ping 默认网关的 IP 地址验证默认网关是否运行以及能否与本地网络的本地主机通讯。
- **ping IP_address_of_default_gateway**
- Ping 远程主机的 IP 地址验证能否通过路由器通讯。
- **ping IP_address_of_remote_host**

注意：Ping 命令用 Windows 套接字样式的名称解析将计算机名解析成 IP 地址，所以如果用地址成功，但是用名称 Ping 失败，则问题出在地址或名称解析上，而不是网络连通性的问题。

如果在任何点上都无法成功地使用 Ping，需要确认：

- 安装和配置 TCP/IP 之后重新启动计算机。
- “Internet 协议 (TCP/IP) 属性”对话框“常规”选项卡上的本地计算机的 IP 地址有效而且正确。
- 用 IP 路由，并且路由器之间的链路是可用的。

可以使用 Ping 命令的不同选项来指定要使用的数据包大小、要发送多少数据包、是否记录用过的路由、要使用的生存时间 (TTL) 值以及是否设置“不分段”标志。可以键入 ping -? 查看这些选项。

下例说明如何向 IP 地址 172.16.48.10 发送两个 Ping，每个都是 1,450 字节：

- **C:\>ping -n 2 -l 1450 172.16.48.10**
- **Pinging 172.16.48.10 with 1450 bytes of data:**
- **Reply from 172.16.48.10:bytes=1450 time<10ms TTL=32**
- **Reply from 172.16.48.10:bytes=1450 time<10ms TTL=32**
- **Ping statistics for 157.59.8.1:**
- **Packets:Sent = 2, Received = 2, Lost = 0 (0% loss),**
- **Approximate roundtrip times in milli-seconds:**
- **Minimum = 0ms, Maximum = 10ms, Average = 2ms**

默认情况下，在显示“请求超时”之前，Ping 等待 1,000 毫秒（1 秒）的时间让每个响应返回。如果通过 Ping 探测的远程系统经过长时间延迟的链路，如卫星链路，则响应可能会花更长的时间才能返回。可以使用 -w （等待）选项指定更长时间的超时。

1.3.2.3 ARP 命令

主要功能：显示和修改“地址解析协议 (ARP)”缓存中的项目

“地址解析协议 (ARP)”允许主机查找同一物理网络上的主机的 MAC 地址。可以使用 arp 命令查看和修改本地计算机上的 ARP 表项。arp 命令对于查看 ARP 缓存和解决地址解析问题非常有用。

1.3.2.4 Netstat 命令

主要功能：显示连接统计信息。

可以使用 netstat 命令显示协议统计信息和当前的 TCP/IP 连接。常用命令如下：

- netstat -a 命令将显示所有连接；
- netstat -r 显示路由表和活动连接；
- netstat -e 命令将显示 Ethernet 统计信息；
- netstat -s 显示每个协议的统计信息；
- netstat –n 以数字形式显示，而不能将地址和端口号转换成名称。

1.3.2.5 Tracert 命令

- **主要功能：跟踪网络连接**

Tracert（跟踪路由）是路由跟踪实用程序，用于确定 IP 数据报访问目标所采取的路径。Tracert 命令用 IP 生存时间(TTL)字段和 ICMP 错误消息来确定从一个主机到网络上其他主机的路由。

- Tracert 工作原理

通过向目标发送不同 IP 生存时间 (TTL) 值的“Internet 控制消息协议(ICMP)”回应数据包，Tracert 诊断程序确定到目标所采取的路由。要求路径上的每个路由器在转发数据包之前至少将数据包上的 TTL 递减 1。数据包上的 TTL 减为 0 时，路由器应该将“ICMP 已超时”的消息发回源系统。

Tracert 先发送 TTL 为 1 的回应数据包，并在随后的每次发送过程将 TTL 递增 1，直到目标响应或 TTL 达到最大值，从而确定路由。通过检查中间路由器发回的“ICMP 已超时”的消息确定路由。某些路由器不经询问直接丢弃 TTL 过期的数据包，这在 Tracert 实用程序中看不到。

Tracert 命令按顺序打印出返回“ICMP 已超时”消息的路径中的近端路由器接口列表。如果使用 -d 选项，则 Tracert 实用程序不在每个 IP 地址上查询 DNS。

在下例中，数据包必须通过两个路由器（10.0.0.1 和 192.168.0.1）才能到达主机 172.16.0.99。主机的默认网关是 10.0.0.1，192.168.0.0 网络上的路由器的 IP 地 址是 192.168.0.1。

- **C:\>tracert 172.16.0.99 -d**
- **Tracing route to 172.16.0.99 over a maximum of 30 hops**
- **1 2s 3s 2s 10,0.0,1**
- **2 75 ms 83 ms 88 ms 192.168.0.1**
- **3 73 ms 79 ms 93 ms 172.16.0.99**
- **Trace complete.**

- 用 tracert 解决问题

可以使用 tracert 命令确定数据包在网络上的停止位置。下例中，默认网关确定 192.168.10.99 主机没有有效路径。这可能是路由器配置的问题，或者是 192.168.10.0 网络不存在（错误的 IP 地址）。

- **C:\>tracert 192.168.10.99**
- **Tracing route to 192.168.10.99 over a maximum of 30 hops**
- **1 10.0.0.1 reportsestination net unreachable.**

- **Trace complete.**
  - Tracert 命令行选项

Tracert 命令支持多种选项，如下所示：

tracert [-d] [-h maximum_hops] [-j host-list] [-w timeout] target_name

选项描述：

-d 指定不将 IP 地址解析到主机名称。

-h maximum_hops 指定跃点数以跟踪到称为 target_name 的主机的路由。

-j host-list 指定 Tracert 实用程序数据包所采用路径中的路由器接口列表。

-w timeout 等待 timeout 为每次回复所指定的毫秒数。

target_name 目标主机的名称或 IP 地址。

1.3.2.6 pathping 路由跟踪

pathping 命令是一个路由跟踪工具，它将 ping 和 tracert 命令的功能和这两个工具所不能提供的其他信息结合起来。pathping 命令在一段时间内将数据包发送到到达最终目标的路径上的每个路由器，然后基于数据包的计算机结果从每个跃点返回。由于命令显示数据包在任何给定路由器或链接上丢失的程度，因此可以很容易地确定可能导致网络问题的路由器或链接。

默认的跃点数是 30，并且超时前的默认等待时间是 3 秒。默认时间是 250 毫秒，并且沿着路径对每个路由器进行查询的次数是 100，计算平均的丢包率。

Pathping 的命令选项如下表所示：

选项	名称	功能
-n	Hostnames	不将地址解析成主机名
-h	Maximum hops	搜索目标的最大跃点数
-g	Host-list	沿着路由列表释放源路由
-p	Period	在 ping 之间等待的毫秒数
-q	Num_queries	每个跃点的查询数
-w	Time-out	为每次回复所等待的毫秒数
-T	Layer 2 tag	将第 2 层优先级标记（例如，对于 IEEE 802.1p）连接到数据包并将它发送到路径中的每个网络设备。这有助于标识没有正确配置第 2 层优先级的网络设备。-T 开关用于测试服务质量（QoS）连通性
-R	RSVP isbase Che	检查以确定路径中的每个路由器是否支持“资源保留协议（RSVP）”，此协议允许主机为数据流保留一定量的带宽。 -R 开关用于测试服务质量（QoS）连通性。

典型的 pathping 报告分为两个部分：第一部分是通向目的地的线路上的每一个跃点的列表，第二部分是每一个跃点的统计，包括每一个跃点的数据包丢失

的数量。执行命令：pathping –n www.sina.com.cn

1.3.2.7 Nslookup 命令

主要功能：通过查询 DNS 服务器检查记录、域主机别名、域主机服务和操作系统信息。

nslookup 命令的功能是查询一台机器的 IP 地址和其对应的域名。它通常需要一台域名服务器来提供域名服务。如果用户已经设置好域名服务器，就可以用这个命令查看不同主机的 IP 地址对应的域名。

- **格式：nslookup [IP 地址或域名]**
- **在本地机上使用 nslookup 命令查看本机的域名服务器名称和 IP 地址。**
- **查看 www.163.com 的 IP，在提示符后输入要查询的 IP 地址或域名并回车即可。**
- **退出该命令，输入 exit 并回车即可。**

1.3.2.8 Net 系列命令

- **net view, net user, net use,net session,net share 一系列 Net 命令的使用**
- **显示当前工作组服务器列表 net view，当不带选项使用本命令时，它就会显示当前域或网络上的计算机上的列表。**
- **比如：查看这个 IP 上的共享资源，就可以**
- **C:\>net view 192.168.10.8**
- **查看在 192.168.10.8 的共享资源**
- **资源共享名 类型 用途 注释**
- **----------**
- **查看计算机上的用户帐号列表 net user**
- **查看网络链接 net use**

  例如：net use z: \\192.168.10.8\movie 将这个 IP 的 movie 共享目录映射为本地的 Z 盘
- **记录链接 net session**
- **例如：C:\>net session**
- **计算机 用户名 客户类型 打开空闲时间**
- **----------**
- **\\192.168.10.110 ROME Windows 2000 2195 0 00:03:12**

- **\\192.168.10.51 ROME Windows 2000 2195 0 00:00:39**
- **查看你机器的共享资源 net share**
- **手工删除共享**
- **net share c$ /d**
- **net share d$ /d**
- **net share ipc$ /d**
- **net share admin$ /d**
- **注意$后有空格。**
- **增加一个共享：**
- **c:\net share mymovie=e:\downloads\movie /users:1**
- **mymovie 共享成功。同时限制链接用户数为 1 人。**
- **网络信使服务**
- **net start messenger 开始信使服务**
- **net stop messenger 停止信使服务，也可以在面板－服务修改**
- **Net send 计算机名/IP * (广播) 传送内容，注意：不能跨网段**
- **在网络邻居上隐藏你的计算机**
- **net config server /hidden:yes**
- **net config server /hidden:no 则为开启**

1.4【实验方式】

- **对常用的 TCP/IP 网络测试和故障诊断命令，进行实验。观察机器的输出，并能解释输出的结果。如果出现错误，能分析出现错误的原因。**
- **尝试不同类型的网络服务的安装、配置和测试，观察测试的结果，并分析，并分析可能的错误原因。**

1.5【实验报告】

按照实验报告格式要求，提交实验报告。

实验二、Socket 通信实验

2.1【实验目的】

- 掌握 Socket 编程过程，编写简单的网络应用程序。
- 实验环境：Windows 9x/NT/2000/XP/2003；TCP/IP 协议
- 编程工具：Java、VC＋＋；C＃；VB 任选

2.2【实验内容】

利用你选择的任何一个编程语言，分别基于 TCP 和 UDP 编写一个简单的 Client/Server 网络应用程序。

程序要求：

- 客户程序(基本要求)：
  - 能通过用户名和口令登录
  - 向服务器发一文本文件或向服务器端发一段文本要求用户在通信完成后可自行退出
- 服务器程序：
  - 判断用户身份是否合法为合法用户 echo 数据
  - 要求在服务器控制台显示服务器所处状态信息和用户登录信息服务器程序可控制关闭连接和退出
- 其他要求：
  - 所用语言和开发工具可自行选择。可用 JAVA、VC、VB、Delphi 等。
  - 发送数据的文件格式和内容及访问界面可以自行设计，以方便用户、清晰美观为好。
  - 服务器可设计为重复服务器（一个时间只能和一个客户建立连接），也可设计为并发服务器（同一时间可有多个子进程和不同的客户建立连接）。

2.3【实验原理】

2.3.1 Socket 通信原理

2.3.1.1 什么是 Socket

Socket 接口是 TCP/IP 网络的 API，Socket 接口定义了许多函数或例程，程序员可以用它们来开发 TCP/IP 网络上的应用程序。要学 Internet 上的 TCP/IP 网络编程，必须理解 Socket 接口。

Socket 接口设计者最先是将接口放在 Unix 操作系统里面的。如果了解 Unix 系统的输入和输出的话，就很容易了解 Socket 了。网络的 Socket 数据传输是一种特殊的 I/O，Socket 也是一种文件描述符。Socket 也具有一个类似于打开文件的函数调用 Socket()，该函数返 回一个整型的 Socket 描述符，随后的连接建立、数据传输等操作都是通过该 Socket 实现的。常用的 Socket 类型有两种：流式 Socket （SOCK_STREAM）和数据报式 Socket（SOCK_DGRAM）。流式是一种面向连接的 Socket，针对于面向连接的 TCP 服务应用；数据 报式 Socket 是一种无连接的 Socket，对应于无连接的 UDP 服务应用。

2.3.1.2 Socket 建立

为了建立 Socket，程序可以调用 Socket 函数，该函数返回一个类似于文件描述符的句柄。socket 函数原型为：

int socket(int domain, int type, int protocol);

domain 指明所使用的协议族，通常为 PF_INET，表示互联网协议族（TCP/IP 协议族）；type 参数指定 socket 的类型： SOCK_STREAM 或 SOCK_DGRAM，Socket 接口还定义了原始 Socket（SOCK_RAW），允许程序使用低层协议；protocol 通常赋值"0"。 Socket()调用返回一个整型 socket 描述符，你可以在后面的调用使用它。

Socket 描述符是一个指向内部数据结构的指针，它指向描述符表入口。调用 Socket 函数时，socket 执行体将建立一个 Socket，实际上"建立一个 Socket"意味着为一个 Socket 数据结构分配存储空间。Socket 执行体为你管理描述符表。

两个网络程序之间的一个网络连接包括五种信息：通信协议、本地协议地址、本地主机端口、远端主机地址和远端协议端口。Socket 数据结构中包含这五种信息。

2.3.1.3 Socket 配置

通过 socket 调用返回一个 socket 描述符后，在使用 socket 进行网络传输以前，必须配置该 socket。面向连接的 socket 客户端通过 调用 Connect 函数在 socket 数据结构中保存本地和远端信息。无连接 socket 的客户端和服务端以及面向连接 socket 的服务端通过调用 bind 函数来配置本地信息。

Bind 函数将 socket 与本机上的一个端口相关联，随后你就可以在该端口监听服务请求。Bind 函数原型为：

int bind(int sockfd,struct sockaddr *my_addr, int addrlen);

Sockfd 是调用 socket 函数返回的 socket 描述符,my_addr 是一个指向包含有本机 IP 地址及端口号等信息的 sockaddr 类型的指针；addrlen 常被设置为 sizeof(struct sockaddr)。

struct sockaddr 结构类型是用来保存 socket 信息的：

```
struct sockaddr {
unsigned short sa_family; /* 地址族， AF_xxx */
char sa_data[14]; /* 14 字节的协议地址 */
};
```

sa_family 一般为 AF_INET，代表 Internet（TCP/IP）地址族；sa_data 则包含该 socket 的 IP 地址和端口号。

另外还有一种结构类型：

```
struct sockaddr_in {
    short int sin_family; /* 地址族 */
    unsigned short int sin_port; /* 端口号 */
    struct in_addr sin_addr; /* IP 地址 */
    unsigned char sin_zero[8]; /* 填充 0 以保持与 struct sockaddr 同样大小 */
    };
```

这个结构更方便使用。sin_zero 用来将 sockaddr_in 结构填充到与 struct sockaddr 同样的长度，可以用 bzero()或 memset()函数将其置为零。指向 sockaddr_in 的指针和指向 sockaddr 的指针可以相互转换，这意味着如果一个函数所需参数类型是 sockaddr 时，你可以在函数调用的时候将一个指向 sockaddr_in 的指针转换为指向 sockaddr 的指针；或者相反。

使用 bind 函数时，可以用下面的赋值实现自动获得本机 IP 地址和随机获取一个没有被占用的端口号：

my_addr.sin_port = 0; /* 系统随机选择一个未被使用的端口号 */

my_addr.sin_addr.s_addr = INADDR_ANY; /* 填入本机 IP 地址 */

通过将 my_addr.sin_port 置为 0，函数会自动为你选择一个未占用的端口来使用。同样，通过将 my_addr.sin_addr.s_addr 置为 INADDR_ANY，系统会自动填入本机 IP 地址。

注意在使用 bind 函数是需要将 sin_port 和 sin_addr 转换成为网络字节优先顺序；而 sin_addr 则不需要转换。

计算机数据存储有两种字节优先顺序：高位字节优先和低位字节优先。Internet 上数据以高位字节优先顺序在网络上传输，所以对于在内部是以低位字节优先方式存储数据的机器，在 Internet 上传输数据时就需要进行转换，否则就会出现数据不一致。

下面是几个字节顺序转换函数：

- htonl()：把 32 位值从主机字节序转换成网络字节序
- htons()：把 16 位值从主机字节序转换成网络字节序
- ntohl()：把 32 位值从网络字节序转换成主机字节序
- ntohs()：把 16 位值从网络字节序转换成主机字节序

Bind()函数在成功被调用时返回 0；出现错误时返回"-1"并将 errno 置为相应的错误号。需要注意的是，在调用 bind 函数时一般不要将端口号置为小于 1024 的值，因为 1 到 1024 是保留端口号，你可以选择大于 1024 中的任何一个没有被占用的端口号。

2.3.1.4 连接建立

面向连接的客户程序使用 Connect 函数来配置 socket 并与远端服务器建立一个 TCP 连接，其函数原型为：

int connect(int sockfd, struct sockaddr *serv_addr,int addrlen);

Sockfd 是 socket 函数返回的 socket 描述符；serv_addr 是包含远端主机 IP 地址和端口号的指针；addrlen 是远端地质结构的长度。 Connect 函数在出现错误时返回-1，并且设置 errno 为相应的错误码。进行客户端程序设计无须调用 bind()，因为这种情况下只需知道目的机器 的 IP 地址，而客户通过哪个端口与服务器建立连接并不需要关心，socket 执行体为你的程序自动选择一个未被占用的端口，并通知你的程序数据什么时候到 打断口。

Connect 函数启动和远端主机的直接连接。只有面向连接的客户程序使用 socket 时才需要将此 socket 与远端主机相连。无连接协议从不建立直接连接。面向连接的服务器也从不启动一个连接，它只是被动的在协议端口监听客户的请

求。

Listen 函数使 socket 处于被动的监听模式，并为该 socket 建立一个输入数据队列，将到达的服务请求保存在此队列中，直到程序处理它们。

int listen(int sockfd， int backlog);

Sockfd 是 Socket 系统调用返回的 socket 描述符；backlog 指定在请求队列中允许的最大请求数，进入的连接请求将在队列中等待 accept()它们（参考下文）。Backlog 对队列中等待 服务的请求的数目进行了限制，大多数系统缺省值为 20。如果一个服务请求到来时，输入队列已满，该 socket 将拒绝连接请求，客户将收到一个出错信息。

当出现错误时 listen 函数返回-1，并置相应的 errno 错误码。

accept()函数让服务器接收客户的连接请求。在建立好输入队列后，服务器就调用 accept 函数，然后睡眠并等待客户的连接请求。

int accept(int sockfd, void *addr, int *addrlen);

sockfd 是被监听的 socket 描述符，addr 通常是一个指向 sockaddr_in 变量的指针，该变量用来存放提出连接请求服务的主机的信息（某 台主机从某个端口发出该请求）；addrten 通常为一个指向值为 sizeof(struct sockaddr_in)的整型指针变量。出现错误时 accept 函数返回-1 并置相应的 errno 值。

首先，当 accept 函数监视的 socket 收到连接请求时，socket 执行体将建立一个新的 socket，执行体将这个新 socket 和请求连接进程的地址联系起来，收到服务请求的 初始 socket 仍可以继续在以前的 socket 上监听，同时可以在新的 socket 描述符上进行数据传输操作。

2.3.1.5 数据传输

Send()和 recv()这两个函数用于面向连接的 socket 上进行数据传输。

Send()函数原型为：

int send(int sockfd, const void *msg, int len, int flags);

Sockfd 是你想用来传输数据的 socket 描述符；msg 是一个指向要发送数据的指针；Len 是以字节为单位的数据的长度；flags 一般情况下置为 0（关于该参数的用法可参照 man 手册）。

Send()函数返回实际上发送出的字节数，可能会少于你希望发送的数据。在程序中应该将 send()的返回值与欲发送的字节数进行比较。当 send()返回值与 len 不匹配时，应该对这种情况进行处理。

char *msg = "Hello!";

int len, bytes_sent;

……

len = strlen(msg);

bytes_sent = send(sockfd, msg,len,0);

……

recv()函数原型为：

int recv(int sockfd,void *buf,int len,unsigned int flags);

Sockfd 是接受数据的 socket 描述符；buf 是存放接收数据的缓冲区；len 是缓冲的长度。Flags 也被置为 0。Recv()返回实际上接收的字节数，当出现错误时，返回-1 并置相应的 errno 值。

Sendto()和 recvfrom()用于在无连接的数据报 socket 方式下进行数据传输。由于本地 socket 并没有与远端机器建立连接，所以在发送数据时应指明目的地址。

sendto()函数原型为：

int sendto(int sockfd, const void *msg,int len,unsigned int flags,const struct sockaddr *to, int tolen);

该函数比 send()函数多了两个参数，to 表示目地机的 IP 地址和端口号信息，而 tolen 常常被赋值为 sizeof (struct sockaddr)。Sendto 函数也返回实际发送的数据字节长度或在出现发送错误时返回-1。

Recvfrom()函数原型为：

int recvfrom(int sockfd,void *buf,int len,unsigned int flags,struct sockaddr *from,int *fromlen);

from 是一个 struct sockaddr 类型的变量，该变量保存源机的 IP 地址及端口号。fromlen 常置为 sizeof (struct sockaddr)。当 recvfrom()返回时，fromlen 包含实际存入 from 中的数据字节数。Recvfrom()函数返回接收到的字节数或 当出现错误时返回-1，并置相应的 errno。

如果你对数据报 socket 调用了 connect()函数时，你也可以利用 send()和 recv()进行数据传输，但该 socket 仍然是数据报 socket，并且利用传输层的 UDP 服务。但在发送或接收数据报时，内核会自动为之加上目地和源地址信息。

2.3.1.6 结束传输

当所有的数据操作结束以后，你可以调用 close()函数来释放该 socket，从而停止在该 socket 上的任何数据操作：

close(sockfd);

你也可以调用 shutdown()函数来关闭该 socket。该函数允许你只停止在某个方向上的数据传输，而一个方向上的数据传输继续进行。如你可以关闭某 socket

的写操作而允许继续在该 socket 上接受数据，直至读入所有数据。

int shutdown(int sockfd,int how);

Sockfd 是需要关闭的 socket 的描述符。参数 how 允许为 shutdown 操作选择以下几种方式：

- 0-------不允许继续接收数据
- 1-------不允许继续发送数据
- 2-------不允许继续发送和接收数据，
- 均为允许则调用 close ()

shutdown 在操作成功时返回 0，在出现错误时返回-1 并置相应 errno。

2.3.1.7 Socket 阻塞与非阻塞

阻塞函数在完成其指定的任务以前不允许程序调用另一个函数。例如，程序执行一个读数据的函数调用时，在此函数完成读操作以前将不会执行下一程序语句。当 服务器运行到 accept 语句时，而没有客户连接服务请求到来，服务器就会停止在 accept 语句上等待连接服务请求的到来。这种情况称为阻塞（blocking）。而非阻塞操作则可以立即完成。比如，如果你希望服务器仅仅注意检查是否有客户在等待连接，有就接受连接，否则就继续做其他事情，则 可以通过将 Socket 设置为非阻塞方式来实现。非阻塞 socket 在没有客户在等待时就使 accept 调用立即返回。

```
#include <unistd.h>
#include <fcntl.h>
……
sockfd = socket(AF_INET,SOCK_STREAM,0);
fcntl(sockfd,F_SETFL,O_NONBLOCK);
……
```

通过设置 socket 为非阻塞方式，可以实现"轮询"若干 Socket。当企图从一个没有数据等待处理的非阻塞 Socket 读入数据时，函数将立即返 回，返回值为-1，并置 errno 值为 EWOULDBLOCK。但是这种"轮询"会使 CPU 处于忙等待方式，从而降低性能，浪费系统资源。而调用 select()会有效地解决这个问题，它允许你把进程本身挂起来，而同时使系统内核监听所要求的一组文件描述符的任何活动，只要确认在任何被监控的文件 描述符上出现活动，select()调用将返回指示该文件描述符已准备好的信息，从而实现了为进程选出随机的变化，而不必由进程本身对输入进行测试而浪费 CPU 开销。Select 函数原型为:

int select(int numfds,fd_set *readfds,fd_set *writefds，

fd_set *exceptfds,struct timeval *timeout);

其中 readfds、writefds、exceptfds 分别是被 select()监视的读、写和异常处理的文件描述符集合。如果你希望确定是否可以 从标准输入和某个 socket 描述符读取数据，你只需要将标准输入的文件描述符 0 和相应的 sockdtfd 加入到 readfds 集合中；numfds 的值 是需要检查的号码最高的文件描述符加 1，这个例子中 numfds 的值应为 sockfd+1；当 select 返回时，readfds 将被修改，指示某个文件 描述符已经准备被读取，你可以通过 FD_ISSSET()来测试。为了实现 fd_set 中对应的文件描述符的设置、复位和测试，它提供了一组宏：

FD_ZERO(fd_set *set)----清除一个文件描述符集；

FD_SET(int fd,fd_set *set)----将一个文件描述符加入文件描述符集中；

FD_CLR(int fd,fd_set *set)----将一个文件描述符从文件描述符集中清除；

FD_ISSET(int fd,fd_set *set)----试判断是否文件描述符被置位。

Timeout 参数是一个指向 struct timeval 类型的指针，它可以使 select()在等待 timeout 长时间后没有文件描述符准备好即返回。struct timeval 数据结构为：

```
struct timeval {
int tv_sec; /* seconds */
int tv_usec; /* microseconds */
};
```

2.3.1.8 Socket 编程实例（C 代码）

代码实例中的服务器通过 socket 连接向客户端发送字符串"Hello, you are connected!"。只要在服务器上运行该服务器软件，在客户端运行客户软件，客户端就会收到该字符串。

该服务器软件代码如下：

```
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <string.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <sys/socket.h>
#include <sys/wait.h>
```

```
#define SERVPORT 3333 /*服务器监听端口号 */
#define BACKLOG 10 /* 最大同时连接请求数 */
main()
{
    int sockfd,client_fd; /*sock_fd：监听 socket；client_fd：数据传输 socket */
    struct sockaddr_in my_addr; /* 本机地址信息 */
    struct sockaddr_in remote_addr; /* 客户端地址信息 */
    if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) == -1) {
       perror("socket 创建出错！"); exit(1);
    }
    my_addr.sin_family=AF_INET;
    my_addr.sin_port=htons(SERVPORT);
    my_addr.sin_addr.s_addr = INADDR_ANY;
    bzero(&(my_addr.sin_zero),8);
    if (bind(sockfd, (struct sockaddr *)&my_addr, sizeof(struct sockaddr)) == -1) {
        perror("bind 出错！");
        exit(1);
    }
    if (listen(sockfd, BACKLOG) == -1) {
        perror("listen 出错！");
        exit(1);
    }
    while(1) {
       sin_size = sizeof(struct sockaddr_in);
       if ((client_fd = accept(sockfd, (struct sockaddr *)&remote_addr, &sin_size)) == -1) {
             perror("accept 出错");
             continue;
       }
       printf("received a connection from %s\n", inet_ntoa(remote_addr.sin_addr));
       if (!fork()) { /* 子进程代码段 */
           if (send(client_fd, "Hello, you are connected!\n", 26, 0) == -1)
           perror("send 出错！");
           close(client_fd);
           exit(0);
       }
       close(client_fd);
       }
    }
}
```

服务器的工作流程是这样的：首先调用 socket 函数创建一个 Socket，然后调用 bind 函数将其与本机地址以及一个本地端口号绑定，然后调用 listen 在相应的 socket 上监听，当 accpet 接收到一个连接服务请求时，将生成一个新的 socket。服务器显示该客户机的 IP 地址，并通过 新的 socket 向客户端发送字符串"Hello，you are connected!"。最后关闭该 socket。

代码实例中的 fork()函数生成一个子进程来处理数据传输部分，fork()语句对于子进程返回的值为 0。所以包含 fork 函数的 if 语句是子进程代码部分，它与 if 语句后面的父进程代码部分是并发执行的。

客户端程序代码如下：

```
#include<stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <string.h>
#include <netdb.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <sys/socket.h>
#define SERVPORT 3333
#define MAXDATASIZE 100 /*每次最大数据传输量  */
main(int argc, char *argv[]){
    int sockfd, recvbytes;
    char buf[MAXDATASIZE];
    struct hostent *host;
    struct sockaddr_in serv_addr;
    if (argc < 2) {
fprintf(stderr,"Please enter the server's hostname!\n");
exit(1);
}
    if((host=gethostbyname(argv[1]))==NULL) {
herror("gethostbyname 出错！ ");
exit(1);
}
    if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) == -1){
perror("socket 创建出错！ ");
exit(1);
}
    serv_addr.sin_family=AF_INET;
    serv_addr.sin_port=htons(SERVPORT);
    serv_addr.sin_addr = *((struct in_addr *)host->h_addr);
    bzero(&(serv_addr.sin_zero),8);
    if (connect(sockfd, (struct sockaddr *)&serv_addr, \
       sizeof(struct sockaddr)) == -1) {
perror("connect 出错！ ");
exit(1);
}
    if ((recvbytes=recv(sockfd, buf, MAXDATASIZE, 0)) ==-1) {
perror("recv 出错！ ");
```

```
exit(1);
}
    buf[recvbytes] = '\0';
    printf("Received: %s",buf);
    close(sockfd);
}
```

客户端程序首先通过服务器域名获得服务器的 IP 地址，然后创建一个 socket，调用 connect 函数与服务器建立连接，连接成功之后接收从服务器发送过来的数据，最后关闭 socket。

函数 gethostbyname()是完成域名转换的。由于 IP 地址难以记忆和读写，所以为了方便，人们常常用域名来表示主机，这就需要进行域名和 IP 地址的转换。函数原型为：

```
struct hostent *gethostbyname(const char *name);
```

函数返回为 hosten 的结构类型，它的定义如下：

```
struct hostent {
    char *h_name; /*  主机的官方域名  */
    char **h_aliases; /*  一个以 NULL 结尾的主机别名数组  */
    int h_addrtype; /*  返回的地址类型，在 Internet 环境下为 AF-INET */
    int h_length; /*  地址的字节长度  */
    char **h_addr_list; /*  一个以 0 结尾的数组，包含该主机的所有地址*/
};
#define h_addr h_addr_list[0] /*在 h-addr-list 中的第一个地址*/
```

当 gethostname()调用成功时，返回指向 struct hosten 的指针，当调用失败时返回-1。当调用 gethostbyname 时，你不能使用 perror()函数来输出错误信息，而应该使用 herror()函数来输出。

无连接的客户/服务器程序的在原理上和连接的客户/服务器是一样的，两者的区别在于无连接的客户/服务器中的客户一般不需要建立连接，而且在发送接收数据时，需要指定远端机的地址。

2.3.2 样例程序（Java 语言）

2.3.2.1 TCP 方式

- **客户端代码（TCP 通信）：**

```
//TCPClient.java
```

```
import java.io.*;
import java.net.*;
class TCPClient {
public static void main(String argv[]) throws Exception
{
    String sentence;
    String modifiedSentence;
    BufferedReader inFromUser =
        new BufferedReader(new InputStreamReader(System.in));
    Socket clientSocket = new Socket("hostname", 6789);
    DataOutputStream outToServer =
        new DataOutputStream(clientSocket.getOutputStream());
    BufferedReader inFromServer =
        new BufferedReader(new
    InputStreamReader(clientSocket.getInputStream()));
    sentence = inFromUser.readLine();
    outToServer.writeBytes(sentence + '\n');
    modifiedSentence = inFromServer.readLine();
    System.out.println("FROM SERVER: " + modifiedSentence);
    clientSocket.close();
}
```

- **服务器端代码（TCP 通信）：**

```
//TCPServer.java

import java.io.*;
import java.net.*;
class TCPServer {
    public static void main(String argv[]) throws Exception
    {
        String clientSentence;
        String capitalizedSentence;
        ServerSocket welcomeSocket = new ServerSocket(6789);
        while(true) {
            Socket connectionSocket = welcomeSocket.accept();
            BufferedReader inFromClient =
                new BufferedReader(new
                    InputStreamReader(connectionSocket.getInputStream()));
            DataOutputStream outToClient =
            new DataOutputStream(connectionSocket.getOutputStream());
            clientSentence = inFromClient.readLine();
            capitalizedSentence = clientSentence.toUpperCase() + '\n';
            outToClient.writeBytes(capitalizedSentence);
        }
```

```
    }
}
```

2.3.2.2 UDP 方式

- **基于 UDP 实现客户端向服务器端传输任意一个字符串的客户端程序**

```
//UDPClient.java

import java.io.*;
import java.net.*;
class UDPClient {
    public static void main(String args[]) throws Exception
    {
        BufferedReader inFromUser =
            new BufferedReader(new InputStreamReader(System.in));
        DatagramSocket clientSocket = new DatagramSocket();
        InetAddress IPAddress = InetAddress.getByName("hostname");
        byte[] sendData = new byte[1024];
        byte[] receiveData = new byte[1024];
        String sentence = inFromUser.readLine();
        sendData = sentence.getBytes(); DatagramPacket sendPacket =
            new DatagramPacket(sendData, sendData.length, IPAddress, 9876);
        clientSocket.send(sendPacket);
        DatagramPacket receivePacket =
            new DatagramPacket(receiveData, receiveData.length);
        clientSocket.receive(receivePacket);
        String modifiedSentence =
            new String(receivePacket.getData());
        System.out.println("FROM SERVER:" + modifiedSentence);
        clientSocket.close();
    }
}
```

- **基于 UDP 实现服务器将收到的字符串变换成大写后传送回客户端程序**

```
//UDPServer.java

import java.io.*;
import java.net.*;
class UDPServer {
    public static void main(String args[]) throws Exception
    {
        DatagramSocket serverSocket = new DatagramSocket(9876);
        byte[] receiveData = new byte[1024];
```

```
            byte[] sendData = new byte[1024];
            while(true)
            {
                DatagramPacket receivePacket =
                    new DatagramPacket(receiveData, receiveData.length);
                serverSocket.receive(receivePacket);
                String sentence = new String(receivePacket.getData());
                InetAddress IPAddress = receivePacket.getAddress();
                int port = receivePacket.getPort();
                String capitalizedSentence = sentence.toUpperCase();
                sendData = capitalizedSentence.getBytes();
                DatagramPacket sendPacket =
                    new DatagramPacket(sendData, sendData.length, IPAddress, port);
                serverSocket.send(sendPacket);
            }
        }
}
```

2.4【实验方式】

每位同学上机编程实验，由学生自行查阅资料完成为主，实验指导教师辅导。

2.5【实验报告】

- 系统概述：运行环境、编译、使用方法、实现环境、程序文件列表等；
- 主要数据结构；
- 主要算法描述；
- 遇到的问题及解决方法；
- 用户使用手册；

实验三、数据包结构分析实验

3.1【实验目的】

- 了解 Sniffer 的工作原理，掌握 Sniffer 抓包、记录和分析数据包的方法；
- 在这个实验中，你将使用抓包软件捕获数据包，并通过数据包分析每一层协议。
- 实验环境：
  - Windows 9x/NT/2000/XP/2003；
  - TCP/IP 协议；
  - Sniffer 软件

3.2【实验内容】

- 练习安装 Sniffer
- 配置 Sniffer 及定义捕获规则
- 对捕获报文进行解析
- 练习 Packet Generator 的使用
- 具体实验内容包括：
  - 能够指定需要侦听的网卡（考虑一台机器上多张网卡的情况）
  - 能够侦听所有进出本主机的数据包，解析显示数据包（ ICMP 、IP 、TCP 、 UDP 等）各个字段。比如，对 IP 头而言，需要显示 版本、头长度、服务类型、数据包长度、标识、 DF/MF 标志、段内偏移、生存期、协议类型、源目的 IP 地址、选项内容、数据内容。要求显示数据的实际含义（例如用 ASCII 表示）；
  - 能够侦听来源于指定 IP 地址的数据包，能够侦听指定目的 IP 地址的数据包，显示接收到的 TCP 和 UDP 数据包的全部实际内容。需要考虑一个 TCP 或 UDP 包划分为多个 IP 包传输的情况；
  - 能够根据指定的协议类型来过虑包，例如，只侦听 ICMP 包，或只侦听 ICMP 和 UDP 包。
  - 功能验证手段：在运行 Sniffer 的同时，执行标准的 Ping 、 Telnet 和浏览网页等操作，检查 Sniffier 能否返回预期的结果。
  - 数据包保存：可以保存选中的包，保存文件要有可读性。

3.3【实验原理】

3.3.1 网络监听基本原理

数据在网络上是以很小的被称为“帧”或“包”的协议数据单元（PDU）方式传输的。以数据链路层的“帧”为例，不同的“帧”由多个对应不同信息功能的部分组成。帧的类型与格式根据通信双方数据链路层所使用的协议确定，由网络驱动程序按照一定规则生成，通过网络接口卡发送到网络中，通过网络传送至目的主机。在正常情况下，网络接口卡读入一帧并进行检查，如果帧中所携带的目的 MAC 地址和自己的物理地址一致或者是广播地址时，网络接口卡通过产生一个硬件中断引起操作系统注意，然后将帧中所包含的数据传送给系统处理，否则就将该帧丢弃。但如果某个网卡被设置为“混杂”模式，则该网卡将接收所有在网络中传输的帧，这就形成了监听。

3.3.2 Sniffer 软件使用

Sniffer 软件有很多，常用的有 Sniffer Pro、Iris、NetXray、Ethereal 等，本实验以 Sniffer Pro 为主来介绍该软件的基本使用方法。Sniffer 软件的基本功能有：

- 捕获网络流量进行详细分析
- 利用专家分析系统诊断问题
- 实时监控网络活动
- 收集网络利用率和错误等

在进行流量捕获之前首先选择网络适配器，确定从计算机的哪个网络适配器上接收数据。位置：File->select settings 选择网络适配器后才能正常工作。该软件安装在 Windows 98 操作系统上，Sniffer 可以选择拨号适配器对窄带拨号进行操作。如果安装了 EnterNet500 等 PPPOE 软件还可以选择虚拟出的 PPPOE 网卡。对于安装在 Windows 2000/XP 上则无上述功能，这和操作系统有关。

3.3.2.1 报文捕获分析

- 捕获面板：报文捕获功能可以在报文捕获面板中进行完成。
- 捕获过程报文统计：在捕获过程中可以通过查看下面面板查看捕获报文的数量和缓冲区的利用率。

- 捕获报文查看：Sniffer 软件提供了强大的分析能力和解码功能。对于捕获的报文提供了一个 Expert 专家分析系统进行分析，还有解码

选项及图形和表格的统计信息。

- 解码分析：是对捕获报文进行解码的显示。对于解码主要要求分析人员对协议比较熟悉，这样才能看懂解析出来的报文。
- 过滤规则设置：按照过滤器设置的过滤规则进行数据的捕获或显示。在菜单上的位置分别为 Capture->Define Filter 和 Display->Define Filter。过滤器可以根据物理地址或 IP 地址和协议选择进行组合筛选。
- 设置捕获条件
  - 基本捕获条件（两种）
    - 链路层捕获，按源 MAC 和目的 MAC 地址进行捕获，输入方式为十六进制连续输入，如：00E0FC123456。
    - IP 层捕获，按源 IP 和目的 IP 进行捕获。输入方式为点间隔方式，如：10.107.1.1。如果选择 IP 层捕获条件则 ARP 等报文将被过滤掉。
  - 高级捕获条件：在“Advance”页面下，你可以编辑你的协议捕获条件。在协议选择树中你可以选择你需要捕获的协议条件，如果什么都不选，则表示忽略该条件，捕获所有协议。在捕获帧长度条件下，你可以捕获，等于、小于、大于某个值的报文。在错误帧是否捕获栏，你可以选择当网络上有如下错误时是否捕获。在保存过滤规则条件按钮“Profiles”，你可以将你当前设置的过滤规则，进行保存，在捕获主面板中，你可以选择你保存的捕获条件。
  - 任意捕获条件：在 Data Pattern 下，你可以编辑任意捕获条件，用这种方法可以实现复杂的报文过滤，但很多时候是得不偿失，有时截获的报文本就不多，还不如自己看看来得快。

3.3.2.2 报文发送

- 编辑报文发送

Sniffer 软件报文发送功能就比较弱发送前，你需要先编辑报文发送的内容。点击发送报文编辑按钮。首先要指定数据帧发送的长度，然后从链路层开始，一个一个将报文填充完成。

- 捕获编辑报文发送

将捕获到的报文直接转换成发送报文，然后修改也可。选中某个捕获的报文，用鼠标右键激活菜单，选择“Send Current Packet”，这时你就会发现，该报文的内容已经被原封不动的送到“发送编辑窗口”中了。这时，你在修修改改，就比

你全部填充报文省事多了。

发送模式有两种：连续发送和定量发送。可以设置发送间隔，如果为 0，则以最快的速度进行发送。

3.3.3 数据报文解码详解

数据报文分层、以太报文结构、IP 协议、ARP 协议等的解码分析做了简单的描述，目的在于介绍 Sniffer 软件在协议分析中的功能作用并通过解码分析对协议进一步了解。

对于层次型网络结构而言，每一层都由众多协议组成，如应用层协议 Telnet、FTP 和 Email，传输层协议 TCP 和 UDP，网络层协议 IP、ICMP 和 IGMP 等。在 Sniffer 的解码表中用 DLC 表示链路层，用 IP 表示网络层，用 UDP 表示传输层，用 NETB 表示应用层。另外，Sniffer 会在捕获报文的时候自动记录捕获时间，在解码时显示出来。

如上图所示在 Sniffer 的解码表中分别对每一个层次协议进行解码分析。链路层对应“DLC”；网络层对应“IP”；传输层对应“UDP”；应用层对对应的是“NETB”等高层协议。Sniffer 可以针对众多协议进行详细结构化解码分析。并利用树形结构良好的表现出来。

3.3.3.1 以太报文结构

■ **EthernetII 以太网帧结构**

Ethernet_II 以太网帧类型报文结构为：目的 MAC 地址（6bytes）＋源 MAC 地址＋（6bytes）上层协议类型（2bytes）＋数据字段（46-1500bytes）＋校验（4bytes）。

Sniffer 会在捕获报文的时候自动记录捕获的时间，在解码显示时显示出来，

在分析问题时提供了很好的时间记录。

源目的 MAC 地址在解码框中可以将前 3 字节代表厂商的字段翻译出来，方便定位问题，例如网络上 2 台设备 IP 地址设置冲突，可以通过解码翻译出厂商信息方便的将故障设备找到，如 00e0fc 为华为，010042 为 Cisco 等等。如果需要查看详细的 MAC 地址用鼠标在解码框中点击此 MAC 地址，在下面的表格中会突出显示该地址的 16 进制编码。

IP 网络来说 Ethertype 字段承载的时上层协议的类型主要包括 0x800 为 IP 协议，0x806 为 ARP 协议。

■ **IEEE802.3 以太网报文结构**

上图也给出了 IEEE802.3SNAP 帧结构，与 EthernetII 不通点是目的和源地址后面的字段代表的不是上层协议类型而是报文长度。并多了 LLC 子层。

3.3.3.2 IP 报文结构

IP 报文结构为 IP 协议头＋载荷，其中对 IP 协议头部的分析，时分析 IP 报文的主要内容之一，关于 IP 报文详细信息请参考相关资料。这里给出了 IP 协议头部的一个结构。

- 版本：4——IPv4
- 首部长度：单位为 4 字节，最大 60 字节
- TOS：IP 优先级字段
- 总长度：单位字节，最大 65535 字节
- 标识：IP 报文标识字段
- 标志：占 3 比特，只用到低位的两个比特
  - MF（More Fragment）
    - MF=1，后面还有分片的数据包
    - MF=0，分片数据包的最后一个
  - DF（Don't Fragment）
    - DF=1，不允许分片
    - DF=0，允许分片
- 段偏移：分片后的分组在原分组中的相对位置，总共 13 比特，单位为 8 字节
- 寿命：TTL（Time To Live）丢弃 TTL=0 的报文
- 协议：携带的是何种协议报文
  - 1：ICMP

- 6：TCP
- 17：UDP
- 89：OSPF

- 头部检验和：对 IP 协议首部的校验和
- 源 IP 地址：IP 报文的源地址
- 目的 IP 地址：IP 报文的目的地址

上图对 Sniffer 对 IP 协议首部的解码分析结构，和 IP 首部各个字段相对应，并给出了各个字段值所表示含义的英文解释。如上图报文协议（Protocol）字段的编码为 0x11，通过 Sniffer 解码分析转换为十进制的 17，代表 UDP 协议。其他字段的解码含义可以与此类似，只要对协议理解的比较清楚对解码内容的理解将会变的很容易。

3.3.3.3 ARP 报文结构

ARP 分组具有如下的一些字段：

HTYPE（硬件类型）。这是一个 16 比特字段，用来定义运行 ARP 的网络的类型。每一个局域网基于其类型被指派给一个整数。例如，以太网是类型 1。ARP 可使用在任何网络上。

PTYPE（协议类型）。这是一个 16 比特字段，用来定义协议的类型。例如，对 IPv4 协议，这个字段的值是 0800。ARP 可用于任何高层协议。

HLEN（硬件长度）。这是一个 8 比特字段，用来定义以字节为单位的物理地址的长度。例如，对以太网这个值是 6。

PLEN（协议长度）。这是一个 8 比特字段，用来定义以字节为单位的逻辑地址的长度。例如，对 IPv4 协议这个值是 4。

OPER（操作）。这是一个 16 比特字段，用来定义分组的类型。已定义了两种类型：ARP 请求（1），ARP 回答（2）。

SHA（发送站硬件地址）。这是一个可变长度字段，用来定义发送站的物理地址的长度。例如，对以太网这个字段是 6 字节长。

SPA（发送站协议地址）。这是一个可变长度字段，用来定义发送站的逻辑（例如，IP）地址的长度。对于 IP 协议，这个字段是 4 字节长。

THA（目标硬件地址）。这是一个可变长度字段，用来定义目标的物理地址的长度。例如，对以太网这个字段是 6 字节长。对于 ARP 请求报文，这个字段是全 0，因为发送站不知道目标的物理地址。

TPA（目标协议地址）。这是一个可变长度字段，用来定义目标的逻辑地址（例如，IP 地址）的长度。对于 IPv4 协议，这个字段是 4 字节长。

上面为通过 Sniffer 解码的 ARP 请求和应答报文的结构。

3.3.3.3 传输层和应用层报文结构

学生自己查阅课本和其它参考资料，自己学习。

3.4【实验方式】

每位同学上机实验，实验指导教师指导。

3.5【实验报告】

- 学会使用 Sniffer 捕获数据包，根据所设计的具体应用，对所捕获的数据包进行协议分析。利用 Sniffer 的“数据包生成器”向网络中发送一个数据包
- 在实验报告中写出数据包和分析结果。

实验四 网络层路由实验（静态路由实验）

4.1【实验目的】

- 将一台 Linux PC 机器配置为 1 台 IP 路由器（实验目的）
  - 熟悉如何将一个多接口的计算机配置为 1 台路由器；
  - 如何在 Linux PC 机器上设置静态路由；
- Cisco 的静态路由配置（实验目的）
  - 掌握 Cisco 路由器 IOS 的命令模式和切换方法；
  - 学会配置 Cisco 路由器的 IP 接口
  - 学会设置 Cisco 路由器的静态路由表条目

4.2【实验原理】

4.2.1 Linux 机器的静态路由配置

请自行查阅相关书书籍。

4.2.2 Cisco 路由器静态路由配置（NetSim 模拟）

- **Boson NetSim 模拟软件**

Boson NetSim 模拟软件是 Boson 公司开发的一个针对 Cisco 认证的路由器配置模拟软件，主要模拟 Cisco 路由器的命令行配置。软件还提供一个拓扑设计功能，使用者可以使用软件提供的缺省拓扑或自己设计的拓扑进行网络配置实验。

Boson Netsim 软件的使用帮助，请参考附件。

- **静态路由的配置**

静态路由只有一条配置命令，主要参数是目的 IP 地址、掩码、下一跳地址（或发送接口）。配置静态路由的关键是要能根据网络拓扑结构正确设置配置命令中的目的 IP 地址和下一跳地址（或发送接口）。

Cisco 路由器的静态路由配置是在全局配置模式下使用如下命令进行设置：

ip route *prefix mask* {*nexthop-address* | *interface*} [*distance*]

如果要删除一条已设置的静态路由，则在全局配置模式下使用下面这条命令：

no ip route *prefix mask* {*nexthop-address* | *interface*} [*distance*]

命令中的参数含义如下：

- *prefix* 和 *mask* 分别是目的网络的 IP 地址和掩码，均为点分十进制格式。
- *nexthop-address* 和 *interface* 分别是到目的地的下一跳路由器 IP 地址（点分十进制格式）和到目的地的本路由器发送接口（格式为 *type slot/port*）。这两个参数在配置时二选一。这两种配置方法在路径方面没有区别，但是用发送接口配置的静态路由是作为直连的静态路由输入路由表中的，而用下一跳地址配置的静态路由则是作为一种非直连的静态路由输入路由表中的。
- *distance* 是一个可选参数，用来确定首选的路由源。管理距离值越小，路由选择方法的可信度越高，越优先选用。管理距离

在全局配置模式下使用 **show ip route** 命令，可在路由器的路由选择表中查看静态配置的路由。

4.3【实验内容】

4.3.1 配置 Linux 机器为 1 台 IP 路由器

4.3.1.1 启动 Linux 的 IP 转发功能

- **命令行方式（临时性的）**

在 Linux 系统中，当文件/proc/sys/net/ipv4/ip_forward 中包含一个 1 时，则启动 IP 转发功能；当包含 0 时，则禁用 IP 转发功能。命令方式如下：

echo “1” > /proc/sys/net/ipv4/ip_forward （启用 IP 转发）

#echo “0” > /proc/sys/net/ipv4/ip_forward (禁用 IP 转发)

- **文件操作方式（永久保存）**

上述方式，系统重新引导后，所做的改变会立即消失。永久改变需要改变配置文件/etc/sysctl.conf，在该文件中若加入：

net.ipv4.ip_forward =1 (启用 IP 转发)

net.ipv4.ip_forward =0 (禁用 IP 转发)

4.3.1.2 设置 Linux PC 机器的静态路由表条目

网络拓扑结构为：从左向右用一条链路连接起来，分别是 PC1、以太网交换

机 1（10.0.1.0/24 网络）、PC2（两个接口左边 Eth0、右边 Eth1，作为路由器使用）、以太网交换机 2（10.0.2.0/24 网络）、Router1（两个接口，左边 Eth0，右边 Eth1）、以太网交换机 3（10.0.3.0/24 网络）、PC4。每个接口的 IP 分配，见下表。

Linux PC	**以太网接口 Eth0**	**以太网接口 Eth1**
PC1	10.0.1.11/24	禁用
PC2	10.0.1.21/24	10.0.2.22/24
PC4	10.0.3.41/24	禁用
Router1	10.0.2.1/24	10.0.3.1/24

根据教师给定的网络拓扑结构图，分别配置 PC1 和 PC4 静态路由表，PC2 经过配置后作为一台路由器使用（包括启动转发和配置接口 IP）。

要求学生掌握的命令有：

- 追加路由条目命令：

//向路由表中添加一条某网络的路由表条目，该网络前缀由 IP 地址 netaddress 和网络掩码 netmask 确定，下一跳地址由 IP 地址 gw_address 或接口 iface 来确定。

- route add –net netaddress netmask mask gw gw_address
- route add –net netaddress netmask mask dev iface

//向路由表中添加一条某主机的路由表条目，该主机的 IP 地址为 hostaddress，下一跳地址由 IP 地址 gw_address 或接口 iface 来确定。

- route add –host hostaddress gw gw_address
- route add –host hostaddress dev iface

//设置默认路由为 IP 地址 gw_address。

- route add default gw gw_address
- 从路由表删除路由的命令：
- route del –net netaddress netmask mask gw gw_address
- route del –host hostaddress gw gw_address
- route del default gw gw_address

显示路由表的高速缓存：

- route –C
- netstat –rn (显示 PC 机器上的路由表项)
- 考核方式：学生需要将最后形成的高速缓存中的路由表条目，记录下来，以判断学生的掌握情况。

4.3.2 设置 Cisco 路由器的静态路由

4.3.2.1 切换 Cisco 路由器的命令模式

- **用户 EXEC 模式：**

该模式下，提示符为“Router>”，需要了解该模式下可以实用的命令，输入“?”。

- **特权 EXEC 模式：**

查看 Cisco 路由器的系统参数，必需进入特权 EXEC 模式，输入命令：

Router1> enable

Password:

Router1#

- **全局配置模式：**

若需要修改系统范围内的配置参数，必需进入全局配置模式。输入命令：

Router1# configure terminal

Router1(config)#

- **接口配置模式：**

若要修改网络接口，需要进入接口配置模式。输入命令：

Router1(Config)#interface Ethernet 0/0

Router1(config-if)#

- **返回：**
  - Exit 命令：层层返回，即退回到上一个命令层次。
  - End 命令：从任何模式直接退回到特权 EXEC 模式。
  - Disable 命令：从特权 EXEC 模式返回到用户 EXEC 模式。即
    - Router1＃disable
    - Router1>
  - Logout 命令：从用户 EXEC 模式终止控制台会话，输入 logout。

4.3.2.2 配置 Cisco 路由器的 IP 接口

- **实验要求：给 Cisco 路由器的特定接口（Ethernet0/5），设置 IP 地址（10.0.3.1／255.255.255.0）。**

示例：

router1>enable

router1#configure terminal

Router1(config)#no ip routing

Router1(config)#ip routing

Router1(config)#interface Ethernet 0/0

Router1(config)#ip address 10.0.2.1 255.255.255.0

Router1(config)#no shutdown

检查配置发生的变化：

Router1＃show interfaces

Router1# show running-config

4.3.2.3 设置 Cisco 路由器的静态路由表

■ **实验准备：**

特权 IOS 模式下：

- 显示路由表的内容：

show ip route

- 清除所有路由表条目：

clear ip route *

- 显示路由高速缓存：

show ip cache

全局配置模式下：

- 启用路由高速缓存：

ip route -Cache

- 禁用路由高速缓存：

no ip route -Cache

- 添加路由

ip route destination mask gw_address

ip route destination mask Iface

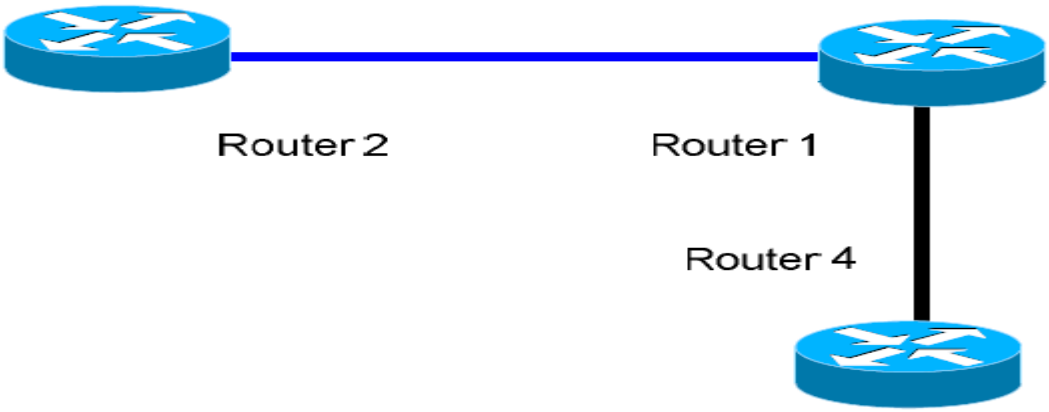
- 从路由表中删除某个路由条目

no ip route destination mask gw_address

no ip route destination mask Iface

■ **实验内容：**

在给定网络拓扑结构的情况下，为路由器设置静态路由，并将最后的路由表结果保存下来。



	Router1	Router2	Router4
Interface Ethernet 0	10.1.1.1 255.255.255.0	10.1.1.2 255.255.255.0	
Interface Serial 0	12.5.10.1 255.255.255.0		12.5.10.2 255.255.255.0

使用Netsim的Static Routing实验部分（Stand Alone Lab，Lab12 Static Routes）。

- **实验报告：**

将 Show ip route 显示的路由表结果保存下来，提交。

4.4【实验方式】

每位同学上机实验，实验指导教师指导。

4.5【实验报告】

- **LINUX 静态路由部分（选做题、可额外加分）**

1、写出 Linux 机器 PC1 或 PC4 的的静态路由表项。

2、若机房条件不具备，完成 CISCO 静态路由也可。

- **CISCO 静态路由部分：**

1、每台路由器上的静态路由表项。

2、使用 ping 进行连通性测试的结果。

实验五 网络层路由实验（动态路由实验）

5.1【实验目的】

- 加深理解目前较广泛使用的域内路由协议 RIP 和 OSPF。
- 掌握在 Cisco 路由器上配置 RIP 和 OSPF 路由协议。

5.2【实验原理】

5.2.1 RIP 协议

RIP 协议的全称是路由选择信息协议（Routing Information Protocol）。它是一个基于距离向量路由选择（Distance Vector Routing，D-V）的内部路由选择协议。RIP 协议虽然没有其他路由选择协议功能强大，但它简单易用，并有广泛的应用。在小型网络的互联设计中，RIP 协议还是相当单一的设计。

RIP 协议当前存在两个版本：版本 1（RIP）和版本 2（RIPv2）。版本 1 的标准是 RFC 1058，它是一个有类别路由选择协议。

- **启动 RIP 进程**

缺省情况下，路由器不运行 RIP 协议。关闭 RIP 进程后，所有与该进程相关的配置信息（包括接口上配置的 RIP 参数）也同时失效。

Cisco 路由器的缺省 RIP 版本操作是所有运行 RIP 协议的接口接收 RIPv1 和 RIPv2 的消息，但只发送 RIPv1 消息。

表 5-1 显示了 Cisco 路由器所提供的 RIP 协议启动配置命令。

表 5-1 启动 RIP 进程的配置命令

命令	描述
router rip	在全局配置模式下启动 RIP 进程，进入路由器配置模式。
no router rip	在全局配置模式下关闭 RIP 进程。

- **指定每个需要运行 RIP 协议的主网络**

启动 RIP 进程后，必须将本路由器接口直连的网络根据需要指定为 RIP 协议工作的网络，使得路由器的接口在该网络上可以收发 RIP 协议消息。

表 5-2 显示了 Cisco 路由器所提供的指定每个需要运行 RIP 协议的主网络的配置命令。

表 5-2 指定每个需要运行 RIP 协议的主网络的配置命令

命令	描述
network *network-address*	在路由器配置模式下指定运行 RIP 的一个主网络。
no network *network-address*	在路由器配置模式下删除运行 RIP 的一个主网络。

要注意的是，无论路由器运行的是 RIP 还是 RIPv2，network 命令所配置的网络参数 network-address 都是一个 A 类、B 类或 C 类的主网络。

一旦在路由器上指定了运行 RIP 协议的一个主网络，也就意味着该路由器接口直连的网络中所有属于这个主网络的子网都运行 RIP 协议，对应的接口上都要发布和接收 RIP 协议消息。

■ **检验 RIP 协议的配置和运行结果**

在配置完 RIP 协议后，可以使用表 2-3 所示的显示命令检验路由器上的 RIP 协议配置和运行结果。

其中，**show ip protocols** 命令可以显示 RIP 协议的定时器参数、每个接口上接收和发送的 RIP 消息版本以及运行 RIP 协议的网络。

表 5-3　检验 RIP 协议配置和运行结果的命令

命令	描述
show running-config	显示本路由器当前的所有配置信息。
show ip protocols	显示所有被启用路由选择协议进程的运行信息。
show ip route	显示本路由器的路由选择表信息。

5.2.2 OSPF 协议

■ **OSPF 路由器 ID**

OSPF 路由器 ID 是一个标识该路由器的 IP 地址。

在一台路由器上的 OSPF 进程启动运行时，如果该路由器配置有一个环回（loopback）接口，那么 OSPF 将越过所有物理接口的 IP 地址，优先选用 loopback 接口的 IP 地址。如果路由器具有多个带 IP 地址的 loopback 接口，那么 OSPF 将选用数值最高的 loopback 地址作为路由器 ID。

如果路由器上没有配置 loopback 接口，OSPF 则从所有已经有效（up）的物理接口的 IP 地址中选择一个数值最高的作为路由器 ID。

■ **OSPF 区域**

OSPF 区域（Area）是一组逻辑上的 OSPF 路由器和链路，它可以有效地把一个 OSPF 域分割成几个子域。

在一个区域内的 OSPF 路由器不需要了解他们所在区域外部地拓扑细节，因此可以减少维护在链路状态数据库中的链路状态数量，处理较少的 LSA 通告，

也就降低了对路由器的内存和 CPU 的消耗。同时，划分区域后，大量的 LSA 洪泛将被限制在一个区域内部，因此也减轻了网络链路上的开销。

OSPF 区域通过一个 32 位的区域 ID（Area ID）来识别。区域 ID 可以表示成一个十进制的数字，也可以表示成一个点分十进制的数字。这两种表示方式在 Cisco 路由器中都可以使用，通常取决于使用的方便性。

使用单个区域设计一个小型的 OSPF 网络是非常合理的。

■ OSPF 协议的基本配置

● 启动 OSPF 进程

缺省情况下，路由器不运行 OSPF 协议。关闭 OSPF 进程后，所有与该进程相关的配置信息（包括接口上配置的 OSPF 参数）也同时失效。

Cisco 路由器在全局配置模式下使用如下命令启动 OSPF 协议：

router ospf *process-id*

参数 *process-id* 是在路由器上运行的 OSPF 进程的进程 ID，可以是任何正整数，并且仅在配置它的路由器内有意义。Cisco IOS 软件允许在一台路由器上运行多个 OSPF 进程，但是每个 OSPF 进程单独维护自己的数据库，这会占用较多的路由器资源，因此并不鼓励这么做。

● 指定需要运行 OSPFP 协议的接口和它们所在的区域

启动 OSPF 进程后，指定需要运行 OSPFP 协议的接口和它们所在的区域。

Cisco 路由器在路由器配置模式下使用如下命令配置指定需要运行 OSPFP 协议的接口和它们所在的区域：

network *address wildcard-mask* **area** *area-id*

参数对（*address*，*wildcard-mask*）表示的是一个地址范围。参数 *area-id* 是参数对（*address*，*wildcard-mask*）所指定的地址范围所在的 OSPF 区域 ID。在单区域的 OSPF 环境中，这个区域 ID 不必一定是区域 0。

● 检验 OSPF 协议的配置和运行结果

在配置完 OSPF 协议后，可以使用表 3-2 所示的显示命令检验路由器上的 OSPF 协议配置和运行结果。

表 3-2 检验 OSPF 协议配置和运行结果的命令

命令	描述
show running-config	显示本路由器当前的所有配置信息。
show ip ospf neighbor	显示本路由器的 OSPF 邻居信息。
show ip ospf database	显示本路由器上的 OSPF 链路状态数据库信息。
show ip route	显示本路由器的路由选择表信息。

5.3【实验内容】

完成 RIP 实验和 OSPF 实验，并完成实验报告。

5.3.1 RIP 路由实验

使用Netsim的RIP Routing实验部分（Stand Alone Lab，Lab13 RIP）。

5.3.2 OSPF 路由实验

使用Netsim的OSPF Routing实验部分（Stand Alone Lab，Lab37 OSPF Routes）。

5.4【实验方式】

每位同学上机实验，实验指导教师指导。

5.5【实验报告】

- **RIP 实验部分**

1、写出每台路由器上的 RIP 路由表项。

2、使用 ping 进行连通性测试的结果。

- **OSPF 实验部分**

1、每台路由器上的 OSPF 路由表项。

2、使用 ping 进行连通性测试的结果。

附件一、Sniffer 软件使用

参阅：《Sniffer 使用指导》

附件二、NetSim 软件使用

参阅：《NetSim 使用指导》