目錄

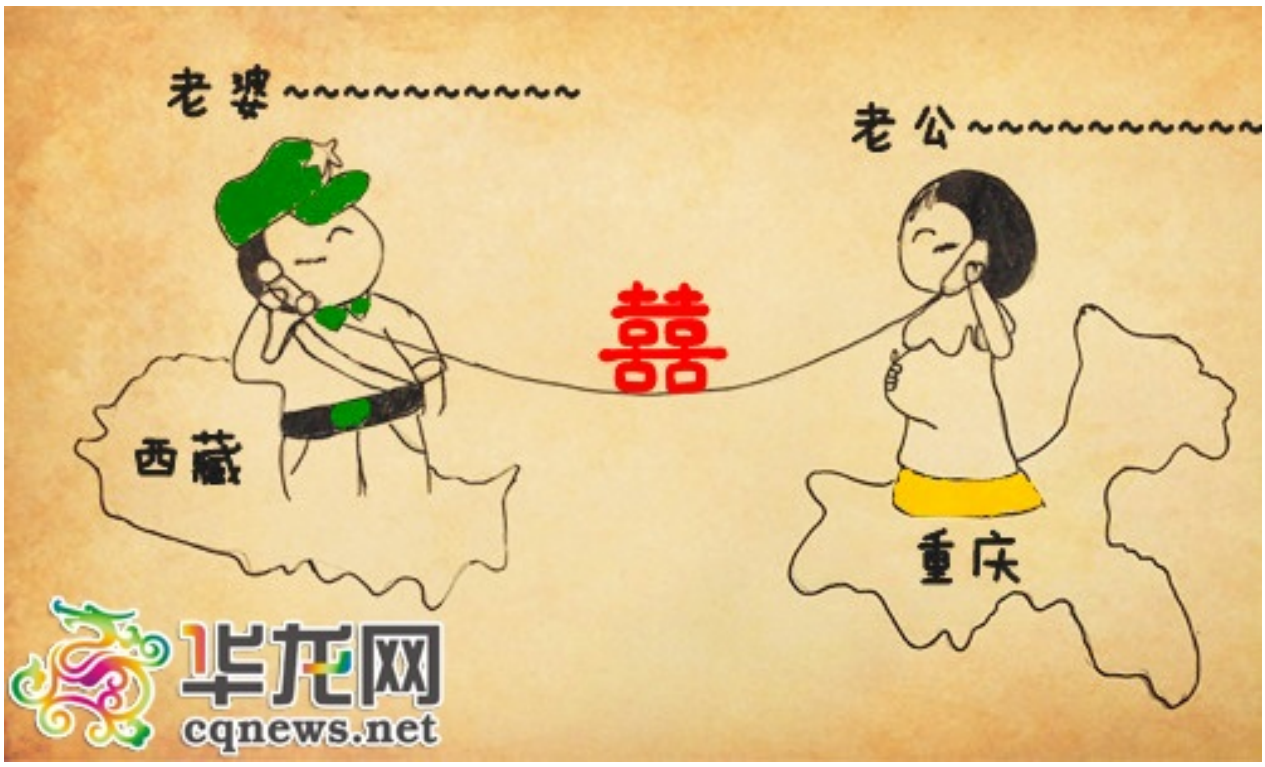
课程介绍

今日知识点

1. tcp/ip协议介绍
2. ip地址的分类
3. 端口、端口号
4. (重点)socket
5. (重点)udp通信
6. (重点、难点)udp应用-echo服务器、聊天室、多线程模拟QQ聊天

网络通信概述

1. 什么是网络



说明

- 网络就是一种辅助双方或者多方能够连接在一起的工具
- 如果没有网络可想 `单机` 的世界是多么的孤单

单机游戏（不能和远在他乡的朋友一起玩）

2. 使用网络的目的

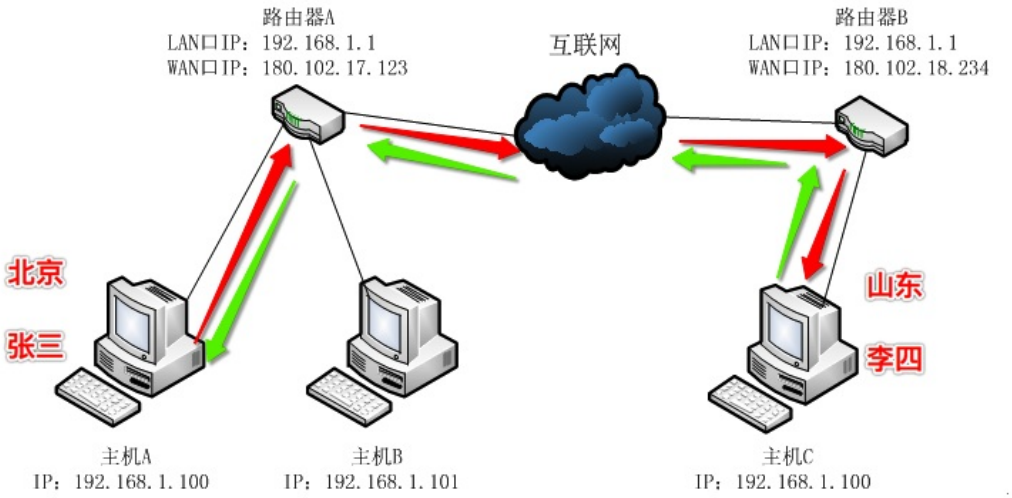
就是为了联通多方然后进行通信用的，即把数据从一方传递给另外一方

前面的学习编写的程序都是单机的，即不能和其他电脑上的程序进行通信

为了让在不同的电脑上运行的软件，之间能够互相传递数据，就需要借助网络的功能

小总结

- 使用网络能够把多方链接在一起，然后可以进行数据传递
- 所谓的网络编程就是，让在不同的电脑上的软件能够进行数据传递，即进程之间的通信



tcp/ip简介

作为新时代标杆的我们，已经离不开手机、离不开网络，对于互联网大家可能耳熟能详，但是计算机网络的出现比互联网要早很多

1. 什么是协议

有的说英语，有的说中文，有的说德语，说同一种语言的人可以交流，不同的语言之间就不行了

为了解决不同种族人之间的语言沟通障碍，现规定国际通用语言是英语，这就是一个规定，这就是协议

2. 计算机网络沟通用什么

现在的生活中，不同的计算机只需要能够联网（有线无线都可以）那么就可以相互进行传递数据

那么不同种类之间的计算机到底是怎么进行数据传递的呢?

就像说不同语言的人沟通一样，只要有一种大家都认可都遵守的协议即可，那么这个计算机都遵守的网络通信协议叫做 `TCP/IP协议`

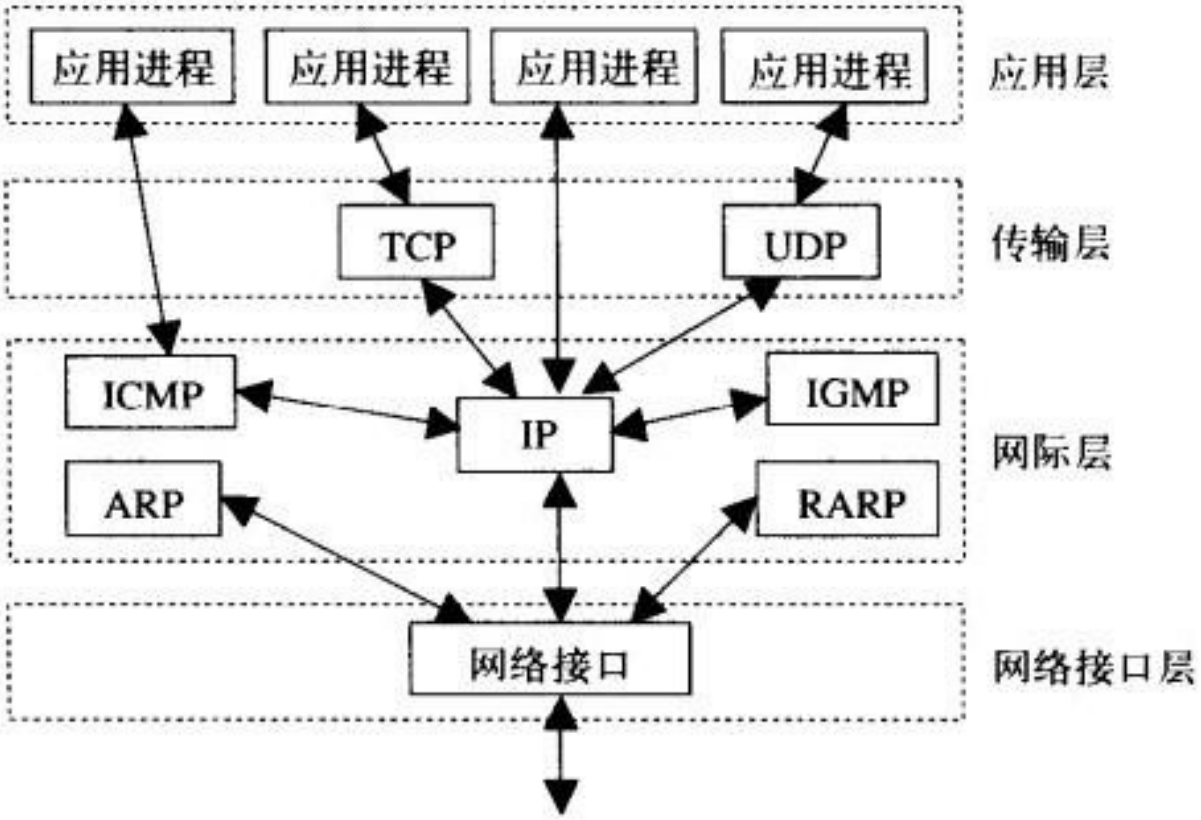
3. TCP/IP协议(族)

早期的计算机网络，都是由各厂商自己规定一套协议，IBM、Apple和Microsoft都有各自的网络协议，互不兼容

为了把全世界的所有不同类型的计算机都连接起来，就必须规定一套全球通用的协议，为了实现互联网这个目标，互联网协议簇（Internet Protocol Suite）就是通用协议标准。

因为互联网协议包含了上百种协议标准，但是最重要的两个协议是TCP和IP协议，所以，大家把互联网的协议简称TCP/IP协议

常用的网络协议如下图所示：



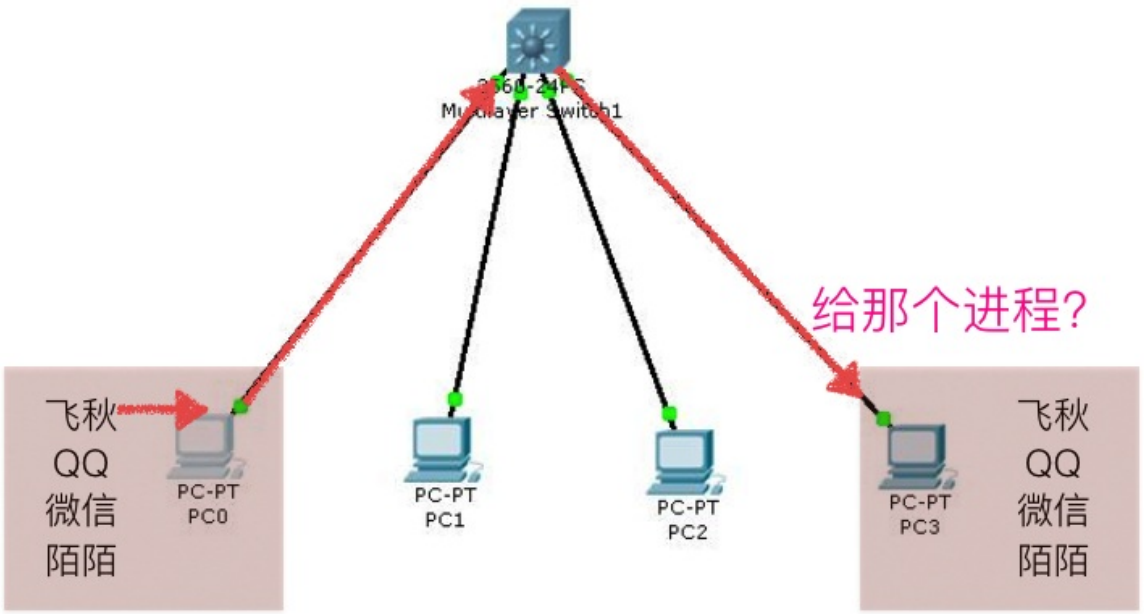
TCP/IP协议族中各协议之间的关系

说明：

网际层也称为：网络层
网络接口层也称为：链路层

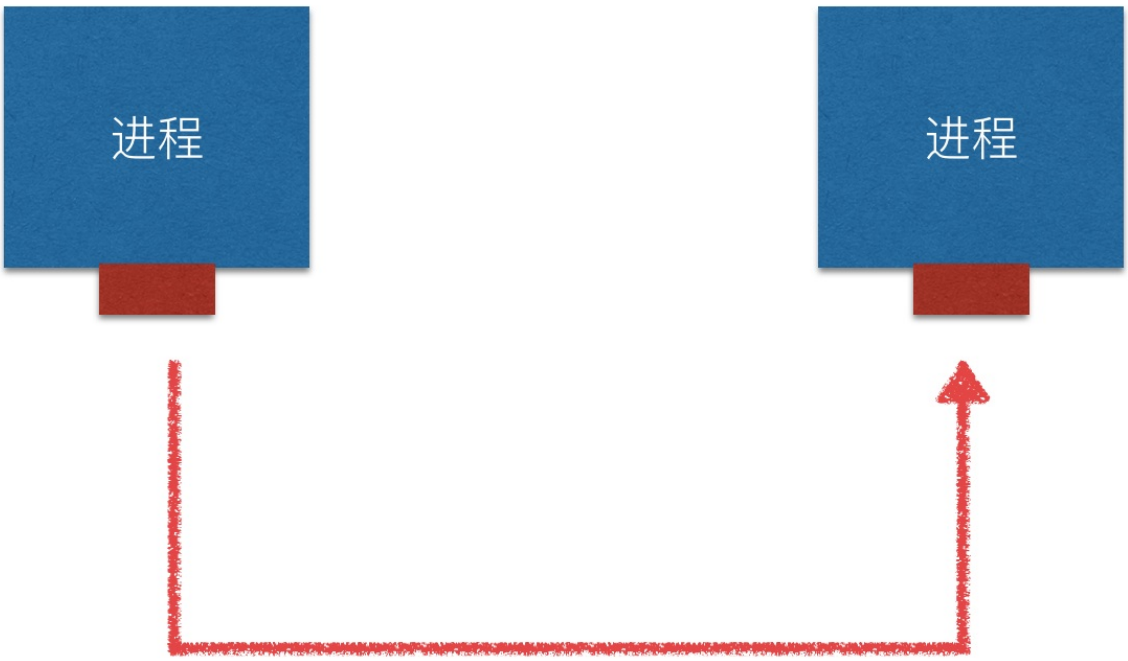
端口

1. 什么是端口



那么TCP/IP协议中的端口指的是什么呢?

端口就好一个房子的门，是出入这间房子的必经之路。



如果一个进程需要收发网络数据，那么就需要有这样的端口

在linux系统中，端口可以有65536（2的16次方）个之多！

既然有这么多，操作系统为了统一管理，所以进行了编号，这就是 `端口号`

2. 端口号

端口是通过端口号来标记的，端口号只有整数，范围是从0到65535

3. 端口是怎样分配的

端口号不是随意使用的，而是按照一定的规定进行分配。

端口的分类标准有好几种，我们这里不做详细讲解，只介绍一下知名端口和动态端口

3.1 知名端口（Well Known Ports）

知名端口是众所周知的端口号，范围从0到1023

```
80端口分配给HTTP服务
21端口分配给FTP服务
```

可以理解为，一些常用的功能使用的号码是估计的，好比 电话号码110、10086、10010一样

一般情况下，如果一个程序需要使用知名端口的需要有root权限

3.2 动态端口（Dynamic Ports）

动态端口的范围是从1024到65535

之所以称为动态端口，是因为它一般不固定分配某种服务，而是动态分配。

动态分配是指当一个系统进程或应用程序进程需要网络通信时，它向主机申请一个端口，主机从可用的端口号中分配一个供它使用。

当这个进程关闭时，同时也就释放了所占用的端口号。

3.3 怎样查看端口？

用“netstat －an”查看端口状态

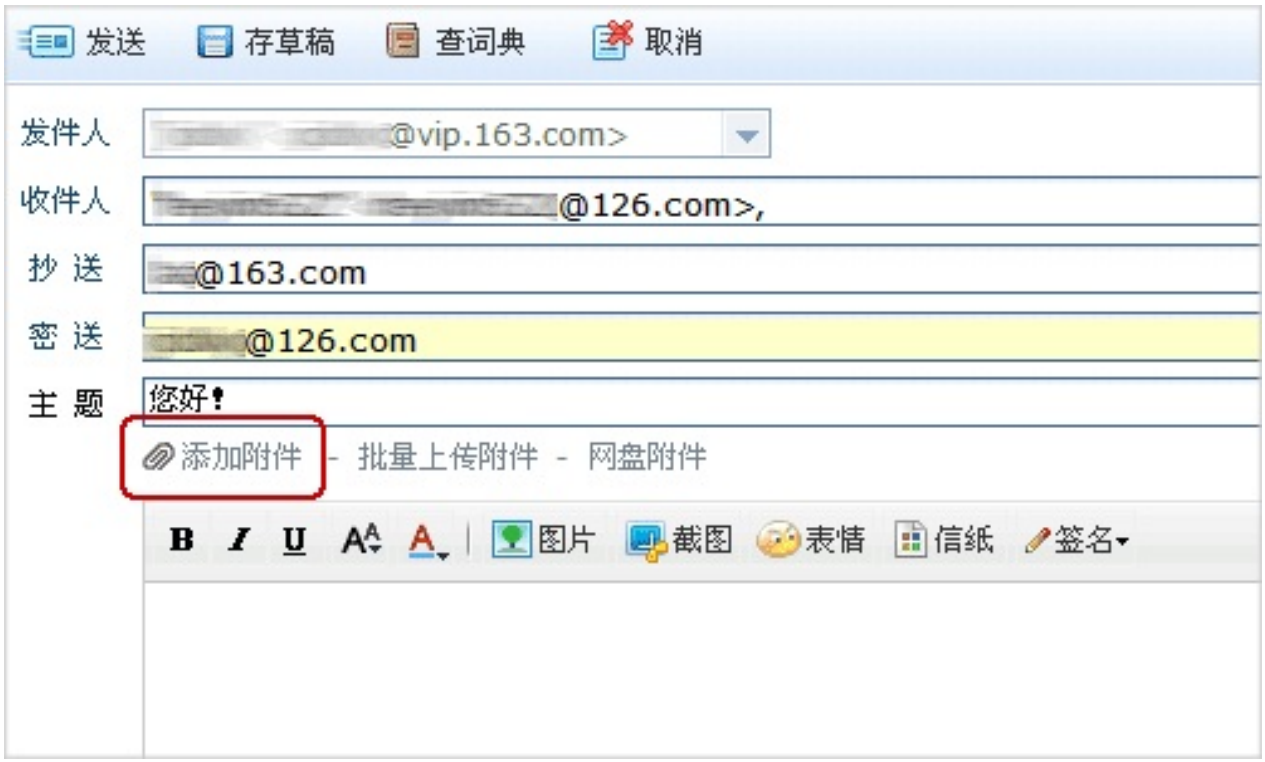
4. 小总结

端口有什么用呢？ 我们知道，一台拥有IP地址的主机可以提供许多服务，比如HTTP（万维网服务）、FTP（文件传输）、SMTP（电子邮件）等，这些服务完全可以通过1个IP地址来实现。那么，主机是怎样区分不同的网络服务呢？显然不能只靠IP地址，因为IP地址与网络服务的关系是一对多的关系。实际上是通过“IP地址+端口号”来区分不同的服务的。 需要注意的是，端口并不是一一对应的。比如你的电脑作为客户机访问一台WWW服务器时，WWW服务器使用“80”端口与你的电脑通信，但你的电脑则可能使用“3457”这样的端口。

ip地址

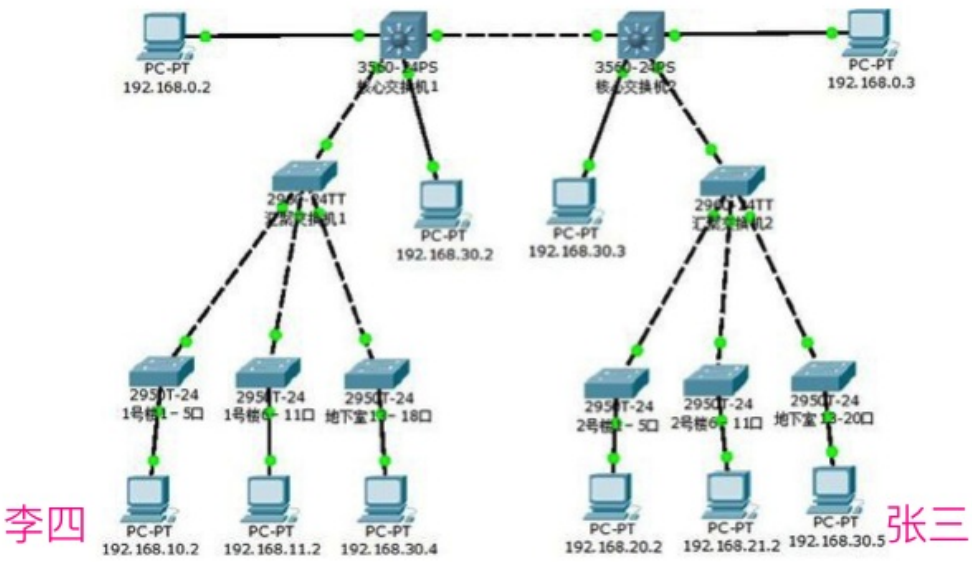
1. 什么是地址


圆通速递
YTO EXPRESS
VIP
始发地
DEPARTURE
省
Province
市(县)
City
区(镇)
Town
邮政编码
POSTAL CODE
收件人姓名
TO
单位名称
COMPANY NAME
收件地址
ADDRESS
城市 CITY
重量
WEIGHT
千克
KG
体积
VOLUME
付款方式
MEANS OF PAYMENT
现金
CASH
协议结算
AGREEMENT
资费¥
CHARGE
加急费¥
HASTY FEE
包装费¥
PACKAGING FEE
费用总计¥
TOTAL
数量
AMOUNT
易碎
FRAGILE
加急
URGENT
其他
OTHERS
寄件人签名:
POKED UP BY (SIGNATURE)
收件人签名:
RECEIVER'S SIGNATURE
代收人签名:
AUTHORIZED SIGNATURE
证件号:
ID NO.
备注 REMARK



地址就是用来标记地点的

2. ip地址的作用



怎么传过去?

to:张三
content：来吃晚饭

ip地址：用来在网络中标记一台电脑的一串数字，比如192.168.1.1；在本地局域网上是惟一的。

3. ip地址的分类

每一个IP地址包括两部分：网络地址和主机地址

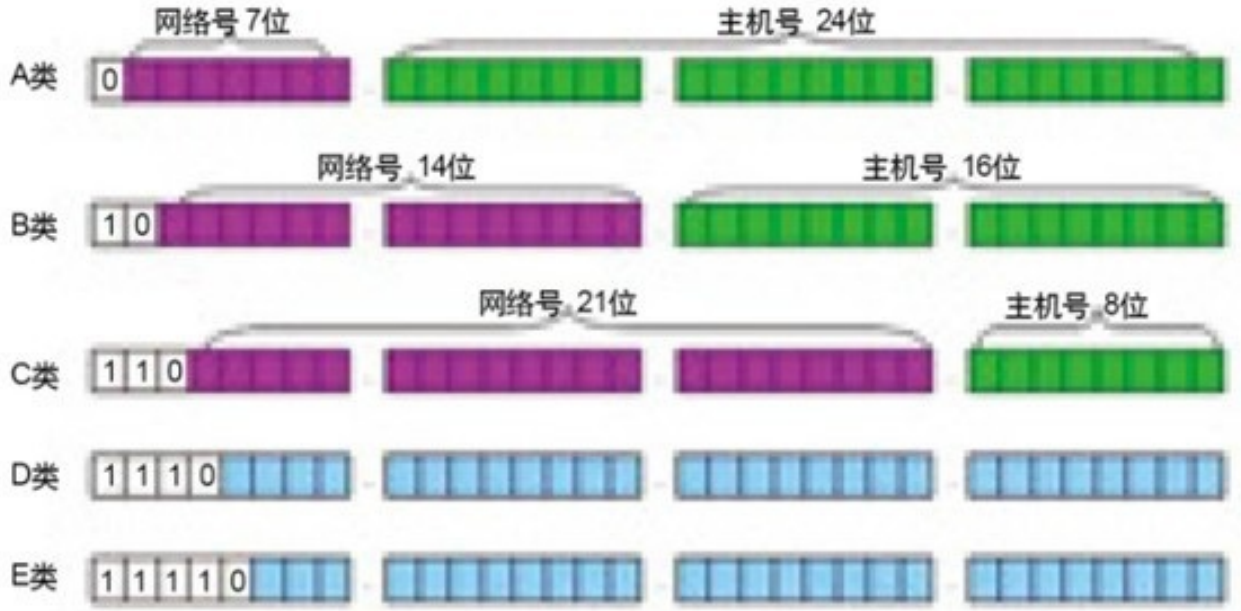


图1 IP地址的类别

3.1 A类IP地址

一个A类IP地址由1字节的网络地址和3字节主机地址组成，网络地址的最高位必须是“0”，

地址范围1.0.0.1-126.255.255.254

二进制表示为：00000001 00000000 00000000 00000001 - 01111110 11111111 11111111 11111110

可用的A类网络有126个，每个网络能容纳1677214个主机

3.2 B类IP地址

一个B类IP地址由2个字节的网络地址和2个字节的主机地址组成，网络地址的最高位必须是“10”，

地址范围128.1.0.1-191.255.255.254

二进制表示为：10000000 00000001 00000000 00000001 - 10111111 11111111 11111111 11111110

可用的B类网络有16384个，每个网络能容纳65534主机

3.3 C类IP地址

一个C类IP地址由3字节的网络地址和1字节的主机地址组成，网络地址的最高位必须是“110”

范围192.0.1.1-223.255.255.254

二进制表示为: 11000000 00000000 00000001 00000001 - 11011111 11111111 11111110 11111110

C类网络可达2097152个，每个网络能容纳254个主机

3.4 D类地址用于多点广播

D类IP地址第一个字节以“1110”开始，它是一个专门保留的地址。

它并不指向特定的网络，目前这一类地址被用在多点广播（Multicast）中

多点广播地址用来一次寻址一组计算机

地址范围224.0.0.1-239.255.255.254

3.5 E类IP地址

以“1111”开始，为将来使用保留

E类地址保留，仅作实验和开发用

3.6 私有ip

在这么多网络IP中，国际规定有一部分IP地址是用于我们的局域网使用，也就

是属于私网IP，不在公网中使用的，它们的范围是：

```
10.0.0.0～10.255.255.255

172.16.0.0～172.31.255.255
```

```
192.168.0.0～192.168.255.255
```

3.7 注意

IP地址127．0．0．1~127．255．255．255用于回路测试，

如：127.0.0.1可以代表本机IP地址，用 `http://127.0.0.1` 就可以测试本机中配置的Web服务器。

子网掩码

要想理解什么是子网掩码，就不能不了解IP地址的构成。互联网是由许多小型网络构成的，每个网络上都有许多主机，这样便构成了一个有层次的结构。IP地址在设计时就考虑到地址分配的层次特点，将每个IP地址都分割成网络号和主机号两部分，以便于IP地址的寻址操作。

IP地址的网络号和主机号各是多少位呢?

如果不指定，就不知道哪些位是网络号、哪些是主机号，这就需要通过子网掩码来实现。

子网掩码不能单独存在，它必须结合IP地址一起使用。

子网掩码只有一个作用，**就是将某个IP地址划分成网络地址和主机地址两部分子网掩码的设定必须遵循一定的规则**。

与IP地址相同，子网掩码的长度也是32位，

- 左边是网络位，用二进制数字“1”表示；
- 右边是主机位，用二进制数字“0”表示。

假设IP地址为“192.168.1.1”子网掩码为“255.255.255.0”。

```
其中，“1”有24个，代表与此相对应的IP地址左边24位是网络号；

“0”有8个，代表与此相对应的IP地址右边8位是主机号。

这样，子网掩码就确定了一个IP地址的32位二进制数字中哪些是网络号、哪些是主机号

这对于采用TCP/IP协议的网络来说非常重要，只有通过子网掩码，才能表明一台主机所
```

最常用的两种子网掩码

子网掩码是“255.255.255.0”的网络：

```
最后面一个数字可以在0~255范围内任意变化，因此可以提供256个IP地址。
但是实际可用的IP地址数量是256-2，即254个，因为主机号不能全是“0”或全是“1”。
```

主机号全为0，表示网络号

主机号全为1，表示网络广播

注意

如果将子网掩码设置过大，也就是说子网范围扩大，那么，根据子网寻径规则，很可能发往和本地主机不在同一子网内的目标主机的数据，会因为错误的判断而认为目标主机是在同一子网内，那么，数据包将在本子网内循环，直到超时并抛弃，使数据不能正确到达目标主机，导致网络传输错误；如果将子网掩码设置得过小，那么就会将本来属于同一子网内的机器之间的通信当做是跨子网传输，数据包都交给缺省网关处理，这样势必增加缺省网关(文章下方有解释)的负担，造成网络效率下降。因此，子网掩码应该根据网络的规模进行设置。如果一个网络的规模不超过254台电脑，采用“255.255.255.0”作为子网掩码就可以了，现在大多数局域网都不会超过这个数字，因此“255.255.255.0”是最常用的IP地址子网掩码；假如在一所大学具有1500多台电脑，这种规模的局域网可以使用“255.255.0.0”。

socket简介

1.本地的进程间通信（IPC）有很多种方式，例如

- 队列
- 同步（互斥锁、条件变量等）

以上通信方式都是在一台机器上不同进程之间的通信方式，那么问题来了

网络中进程之间如何通信?

2. 网络中进程之间如何通信

首要解决的问题是如何唯一标识一个进程，否则通信无从谈起！

在本地可以通过进程PID来唯一标识一个进程，但是在网络中这是行不通的。

其实TCP/IP协议族已经帮我们解决了这个问题，网络层的“ip地址”可以唯一标识网络中的主机，而传输层的“协议+端口”可以唯一标识主机中的应用程序（进程）。

这样利用 `ip地址，协议，端口` 就可以标识网络的进程了，网络中的进程通信就可以利用这个标志与其它进程进行交互

3. 什么是socket

socket(简称 `套接字`) 是进程间通信的一种方式，它与其他进程间通信的一个主要不同是：

它能实现不同主机间的进程间通信，我们网络上各种各样的服务大多都是基于 Socket 来完成通信的

例如我们每天浏览网页、QQ 聊天、收发 email 等等

4. 创建socket

在 Python 中 使用socket 模块的函数 socket 就可以完成：

```
socket.socket(AddressFamily, Type)
```

说明：

函数 socket.socket 创建一个 socket，返回该 socket 的描述符，该函数带有两个参数：

- Address Family：可以选择 AF_INET（用于 Internet 进程间通信） 或者 AF_UNIX（用于同一台机器进程间通信）,实际工作中常用AF_INET
- Type：套接字类型，可以是 SOCK_STREAM（流式套接字，主要用于 TCP 协议）或者 SOCK_DGRAM（数据报套接字，主要用于 UDP 协议）

创建一个tcp socket（tcp套接字）

```
import socket

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

print 'Socket Created'
```

创建一个udp socket（udp套接字）

```
import socket

s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

print 'Socket Created'
```

UDP介绍

UDP --- 用户数据报协议，是一个无连接的简单的面向数据报的运输层协议。UDP不提供可靠性，它只是把应用程序传给IP层的数据报发送出去，但是并不能保证它们能到达目的地。由于UDP在传输数据报前不用在客户和服务器之间建立一个连接，且没有超时重发等机制，故而传输速度很快。

UDP是一种面向无连接的协议，每个数据报都是一个独立的信息，包括完整的源地址或目的地址，它在网络上以任何可能的路径传往目的地，因此能否到达目的地，到达目的地的时间以及内容的正确性都是不能被保证的。

UDP特点：

UDP是面向无连接的通讯协议，UDP数据包括目的端口号和源端口号信息，由于通讯不需要连接，所以可以实现广播发送。 UDP传输数据时有大小限制，每个被传输的数据报必须限定在64KB之内。 UDP是一个不可靠的协议，发送方所发送的数据报并不一定以相同的次序到达接收方。

【适用情况】

UDP是面向消息的协议，通信时不需要建立连接，数据的传输自然是不可靠的，UDP一般用于多点通信和实时的数据业务，比如

- 语音广播
- 视频
- QQ
- TFTP(简单文件传送)
- SNMP（简单网络管理协议）
- RIP（路由信息协议，如报告股票市场，航空信息）
- DNS(域名解释)

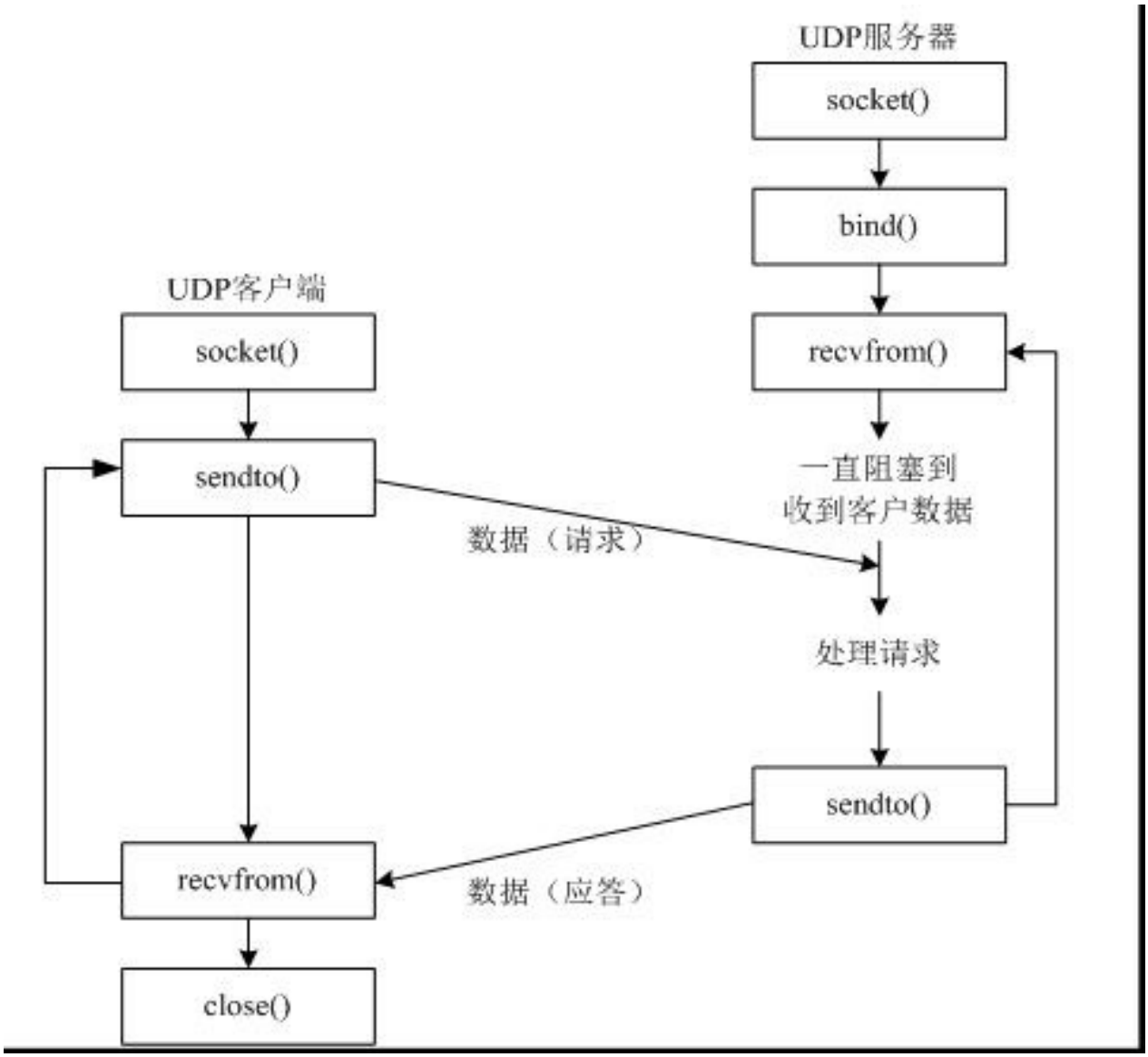
注重速度流畅

UDP操作简单，而且仅需要较少的监护，因此通常用于局域网高可靠性的分散系统中client/server应用程序。例如视频会议系统，并不要求音频视频数据绝对的正确，只要保证连贯性就可以了，这种情况下显然使用UDP会更合理一些。

udp网络程序-发送数据

创建一个udp客户端程序的流程是简单，具体步骤如下：

1. 创建客户端套接字
2. 发送/接收数据
3. 关闭套接字



代码如下：

```
#coding=utf-8
```

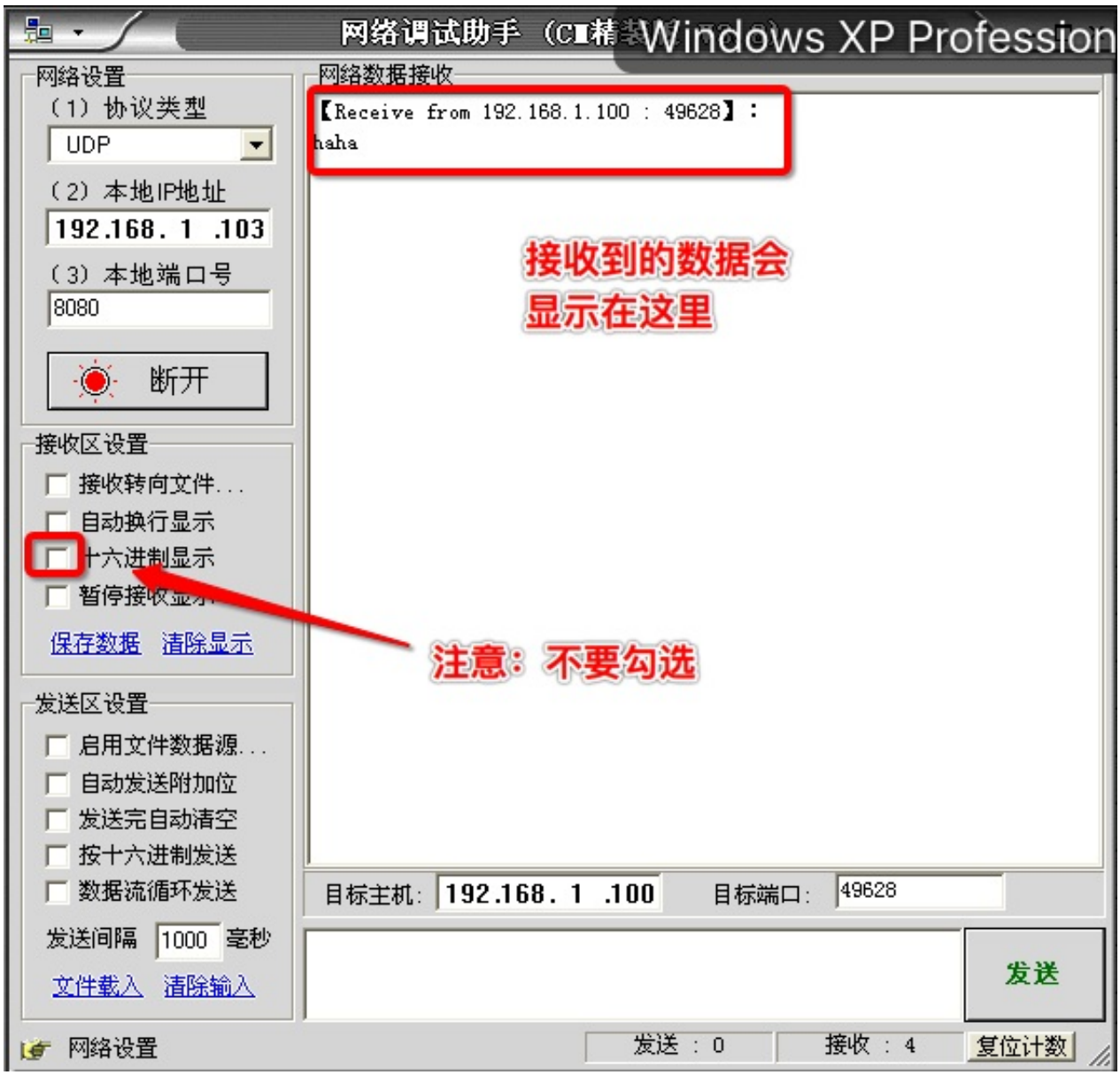
```python
from socket import *

#1. 创建套接字
udpSocket = socket(AF_INET, SOCK_DGRAM)

#2. 准备接收方的地址
sendAddr = ('192.168.1.103', 8080)

#3. 从键盘获取数据
sendData = raw_input("请输入要发送的数据:")

#4. 发送数据到指定的电脑上
udpSocket.sendto(sendData, sendAddr)

#5. 关闭套接字
udpSocket.close()
```

运行现象：

在Ubuntu中运行脚本：

```
[dongGe@localhost 02-网络编程$ python network_test.py
请输入要发送的数据:haha
```

在windows中运行“网络调试助手”：


网络调试助手
Windows XP Profession
网络设置
（1）协议类型
UDP
（2）本地IP地址
192.168. 1 .103
（3）本地端口号
8080
断开
接收区设置
接收转向文件...
自动换行显示
十六进制显示
暂停接收显示
保存数据
清除显示
发送区设置
启用文件数据源...
自动发送附加位
发送完自动清空
按十六进制发送
数据流循环发送
发送间隔
1000
毫秒
文件载入
清除输入
网络数据接收
【Receive from 192.168.1.100 : 49628】:
haha
接收到的数据会
显示在这里
注意：不要勾选
目标主机:
192.168. 1 .100
目标端口:
49628
发送
网络设置
发送 : 0
接收 : 4
复位计数

udp网络程序-发送、接收数据

1. 创建udp网络程序-接收数据

```
#coding=utf-8

from socket import *

#1. 创建套接字
udpSocket = socket(AF_INET, SOCK_DGRAM)

#2. 准备接收方的地址
sendAddr = ('192.168.1.103', 8080)

#3. 从键盘获取数据
sendData = raw_input("请输入要发送的数据:")

#4. 发送数据到指定的电脑上
udpSocket.sendto(sendData, sendAddr)

#5. 等待接收对方发送的数据
recvData = udpSocket.recvfrom(1024) # 1024表示本次接收的最大字节数

#6. 显示对方发送的数据
print(recvData)

#7. 关闭套接字
udpSocket.close()
```

python脚本：

```
[dongGe@localhost 02-网络编程$ python network_test.py
请输入要发送的数据:fasdfsadf
('thank you', ('192.168.1.103', 8080))
```

网络调试助手截图：


网络调试助手 (C■精装版 V3.8)
网络设置
(1) 协议类型
UDP
(2) 本地IP地址
192.168. 1 .103
(3) 本地端口号
8080
断开
接收区设置
接收转向文件...
自动换行显示
十六进制显示
暂停接收显示
保存数据
清除显示
发送区设置
启用文件数据源...
自动发送附加位
发送完自动清空
按十六进制发送
数据流循环发送
发送间隔
1000
毫秒
文件载入
清除输入
网络数据接收
【Receive from 192.168.1.100 : 65214】:
fasdfsadf
目标主机:
192.168. 1 .100
目标端口:
65214
thank you
输入发送的数据
发送
发送
发送结束!
发送 : 15
接收 : 48
复位计数

udp网络程序-端口问题

会变的端口号

重新运行多次脚本，然后在“网络调试助手”中，看到的现象如下：

说明：

- 每重新运行一次网络程序，上图中红圈中的数字，不一样的原因在于，这个数字标识这个网络程序，当重新运行时，如果没有确定到底用哪个，系统默认会随机分配

- 记住一点：这个网络程序在运行的过程中，这个就唯一标识这个程序，所以如果其他电脑上的网络程序如果想要向此程序发送数据，那么就需要向这个数字（即端口）标识的程序发送即可

udp绑定信息

1. 绑定信息

还记得在上一节课中，如果一个网络程序在每次运行的时候端口是随机变化的么?

一般情况下，在一天电脑上运行的网络程序有很多，而各自用的端口号很多情况下不知道，为了不与其他的网络程序占用同一个端口号，往往在编程中，udp的端口号一般不绑定

但是如果需要做成一个服务器端的程序的话，是需要绑定的，想想看这又是为什么呢?

如果报警电话每天都在变，想必世界就会乱了，所以一般服务性的程序，往往需要一个固定的端口号，这就是所谓的端口绑定

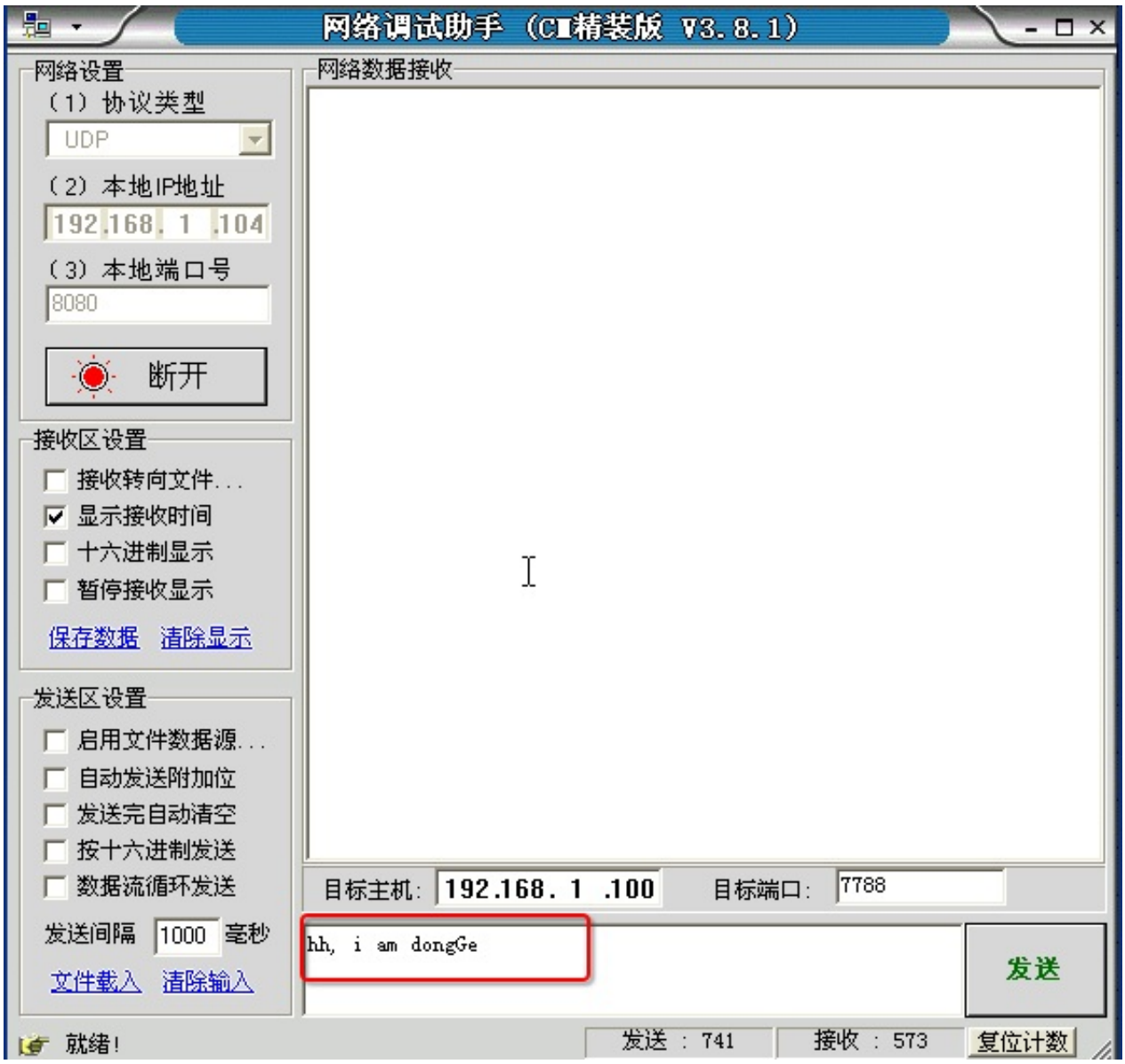
2. 绑定示例

```
#coding=utf-8

from socket import *

#1. 创建套接字
udpSocket = socket(AF_INET, SOCK_DGRAM)
```

```
#2. 绑定本地的相关信息，如果一个网络程序不绑定，则系统会随机分配
bindAddr = ('', 7788) # ip地址和端口号，ip一般不用写，表示本机的任何一个
udpSocket.bind(bindAddr)

#3. 等待接收对方发送的数据
recvData = udpSocket.recvfrom(1024) # 1024表示本次接收的最大字节数

#4. 显示接收到的数据
print recvData

#5. 关闭套接字
udpSocket.close()
```

运行结果：

测试端



本程序

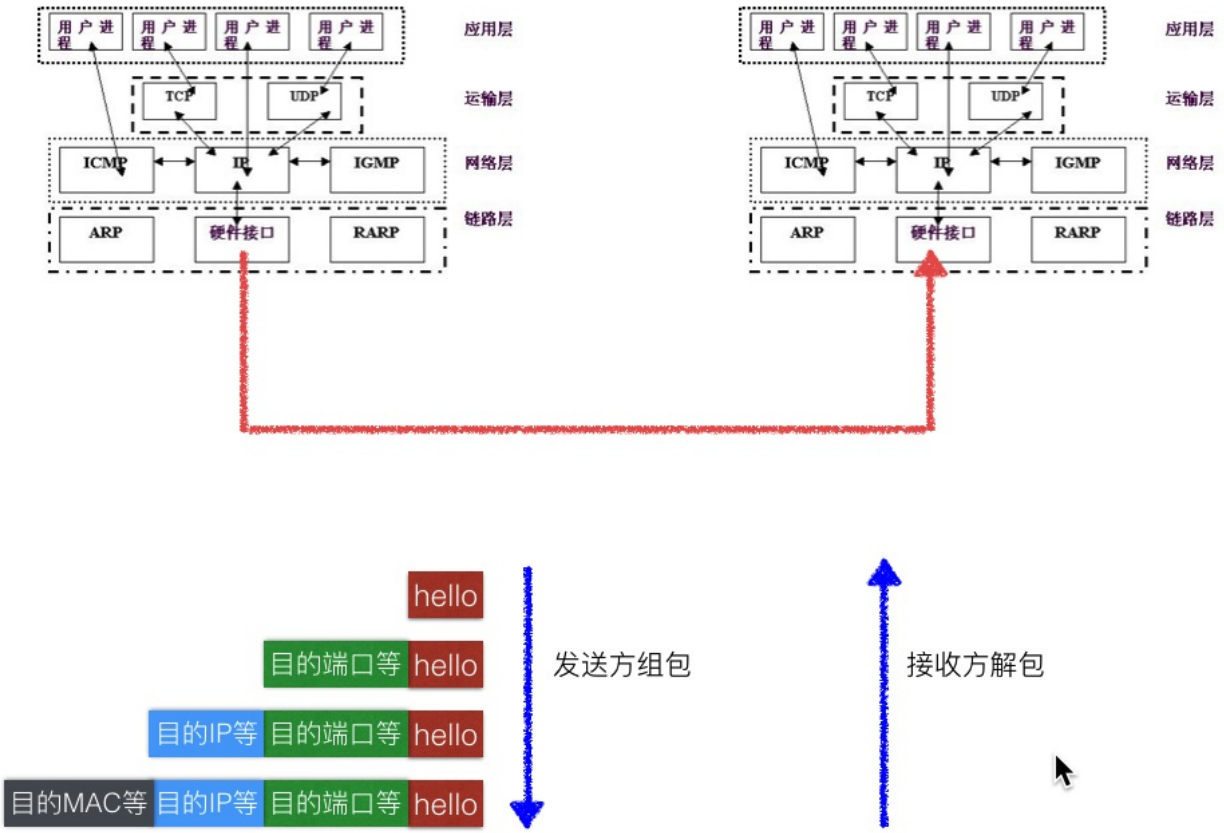
3. 总结

- 一个udp网络程序，可以不绑定，此时操作系统会随机进行分配一个端

口，如果重新运行次程序端口可能会发生变化

- 一个udp网络程序，也可以绑定信息（ip地址，端口号），如果绑定成功，那么操作系统用这个端口号来进行区别收到的网络数据是否是此进程的

udp网络通信过程


用户进程
用户进程
用户进程
用户进程
应用层
TCP
UDP
运输层
ICMP
IP
IGMP
网络层
ARP
硬件接口
RARP
链路层
用户进程
用户进程
用户进程
用户进程
应用层
TCP
UDP
运输层
ICMP
IP
IGMP
网络层
ARP
硬件接口
RARP
链路层

hello
目的端口等
hello
目的IP等
目的端口等
hello
目的MAC等
目的IP等
目的端口等
hello
发送方组包
接收方解包

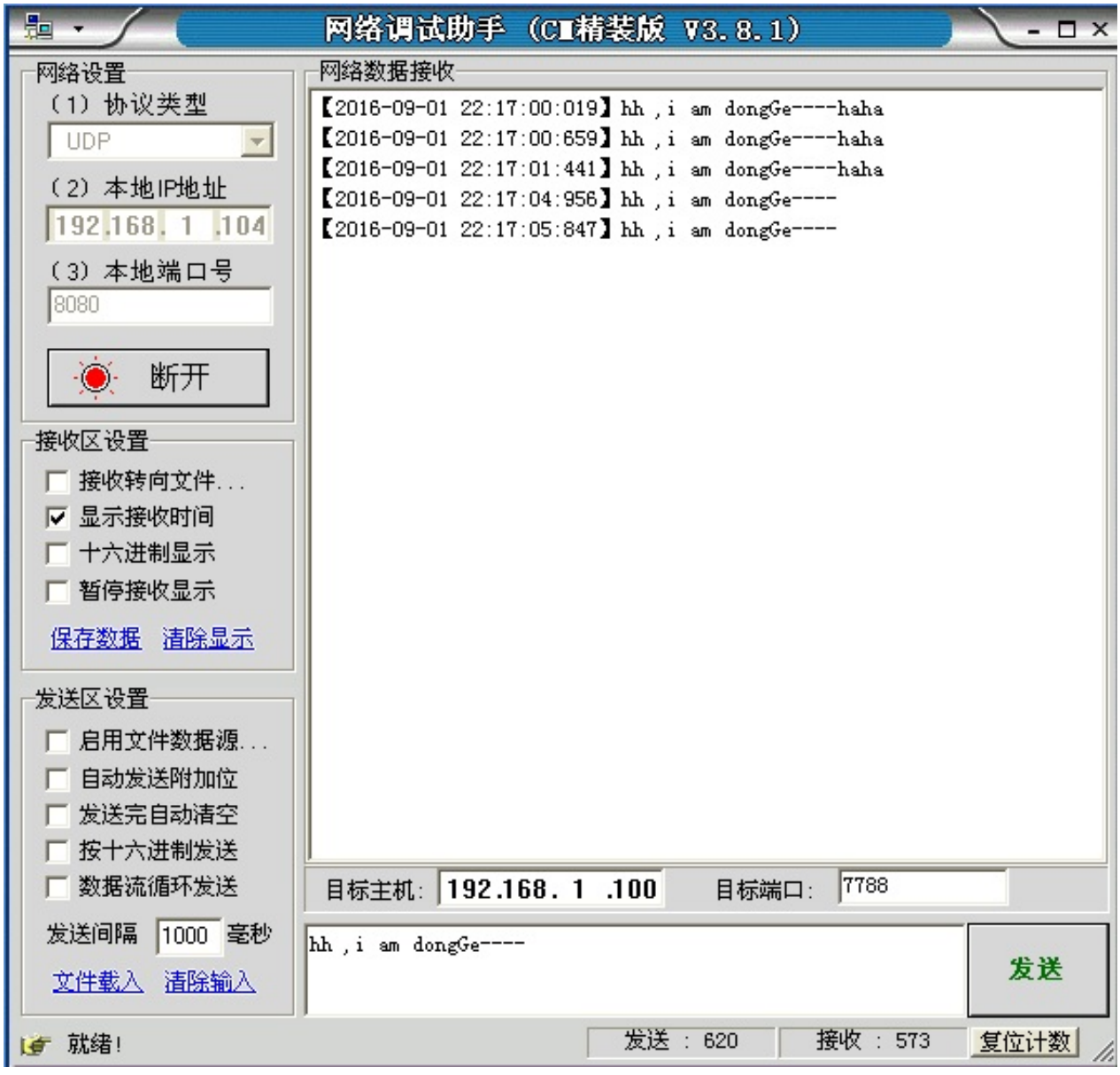
发送手机方

购买手机方

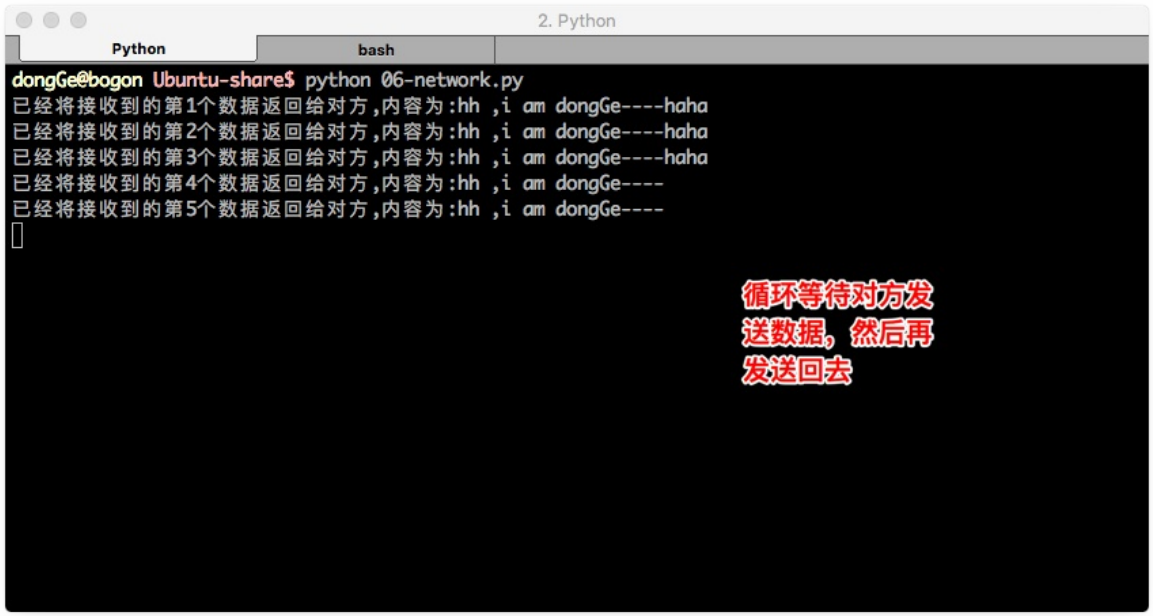
udp应用：echo服务器

1. 运行现象

测试端



echo服务器端



2. 参考代码

```
#coding=utf-8

from socket import *

#1. 创建套接字
udpSocket = socket(AF_INET, SOCK_DGRAM)

#2. 绑定本地的相关信息
bindAddr = ('', 7788) # ip地址和端口号，ip一般不用写，表示本机的任何一个
udpSocket.bind(bindAddr)

num = 1
while True:

    #3. 等待接收对方发送的数据
    recvData = udpSocket.recvfrom(1024) # 1024表示本次接收的最大字节

    #4. 将接收到的数据再发送给对方
    udpSocket.sendto(recvData[0], recvData[1])
```

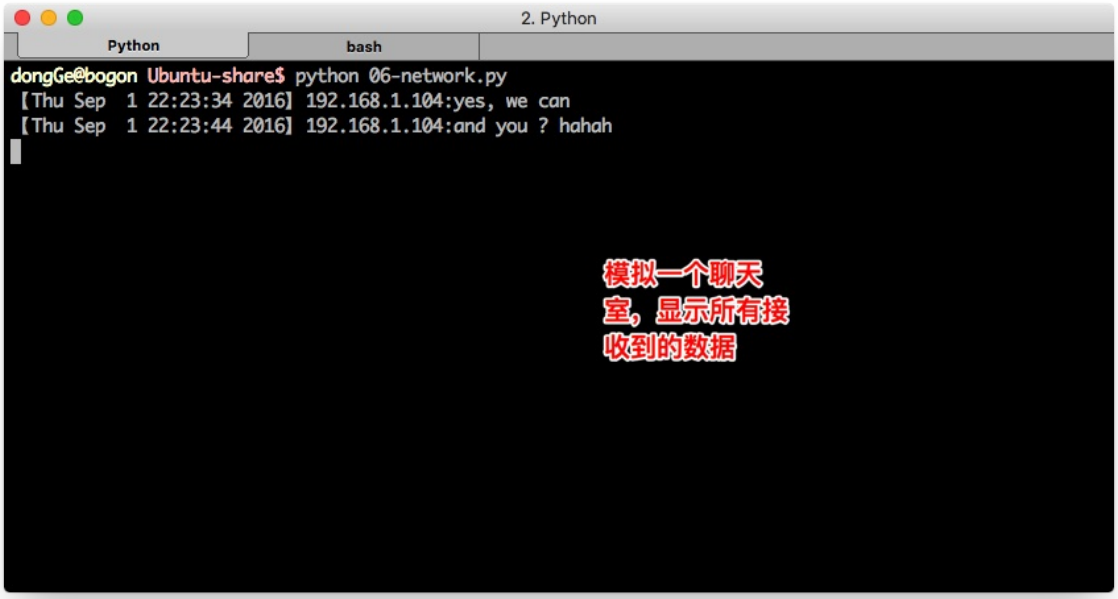
```
    #5. 统计信息
    print('已经将接收到的第%d个数据返回给对方,内容为:%s'%(num,recvData
    num+=1


#5. 关闭套接字
udpSocket.close()
```

udp应用：聊天室

1. 运行现象

测试端

聊天室端



2. 参考代码

```
#coding=utf-8

from socket import *
from time import ctime

#1. 创建套接字
udpSocket = socket(AF_INET, SOCK_DGRAM)

#2. 绑定本地的相关信息
bindAddr = ('', 7788) # ip地址和端口号，ip一般不用写，表示本机的任何一个
udpSocket.bind(bindAddr)

while True:

    #3. 等待接收对方发送的数据
    recvData = udpSocket.recvfrom(1024) # 1024表示本次接收的最大字节

    #4. 打印信息
    print('【%s】%s:%s'%(ctime(),recvData[1][0],recvData[0]))
```

```
#5. 关闭套接字
udpSocket.close()
```

udp总结

1. udp是TCP/IP协议族中的一种协议能够完成不同机器上的程序间的数据通信

2. udp服务器、客户端

- udp的服务器和客户端的区分：往往是通过 `请求服务` 和 `提供服务` 来进行区分
- 请求服务的一方称为：客户端
- 提供服务的一方称为：服务器

3. udp绑定问题

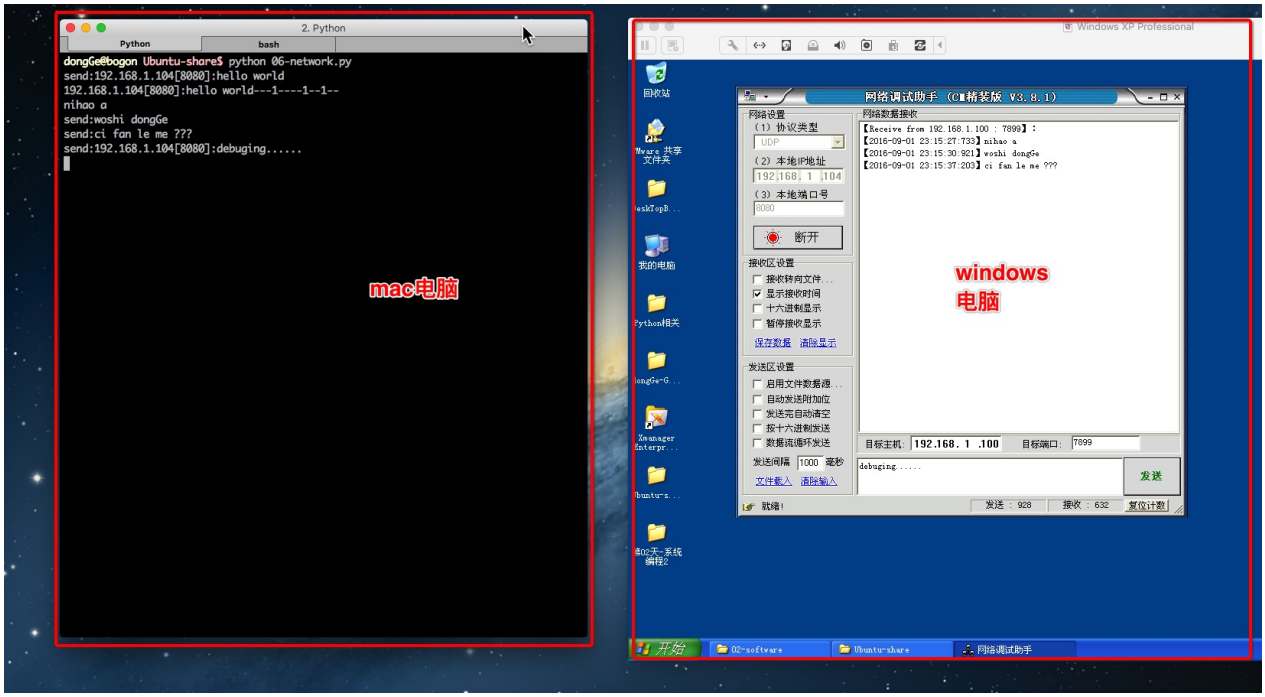
- 一般情况下，服务器端，需要绑定端口，目的是为了让其他的客户端能够正确发送到此进程
- 客户端，一般不需要绑定，而是让操作系统随机分配，这样就不会因为需要绑定的端口被占用而导致程序无法运行的情况

udp综合作业-模拟QQ

1. 任务要求：

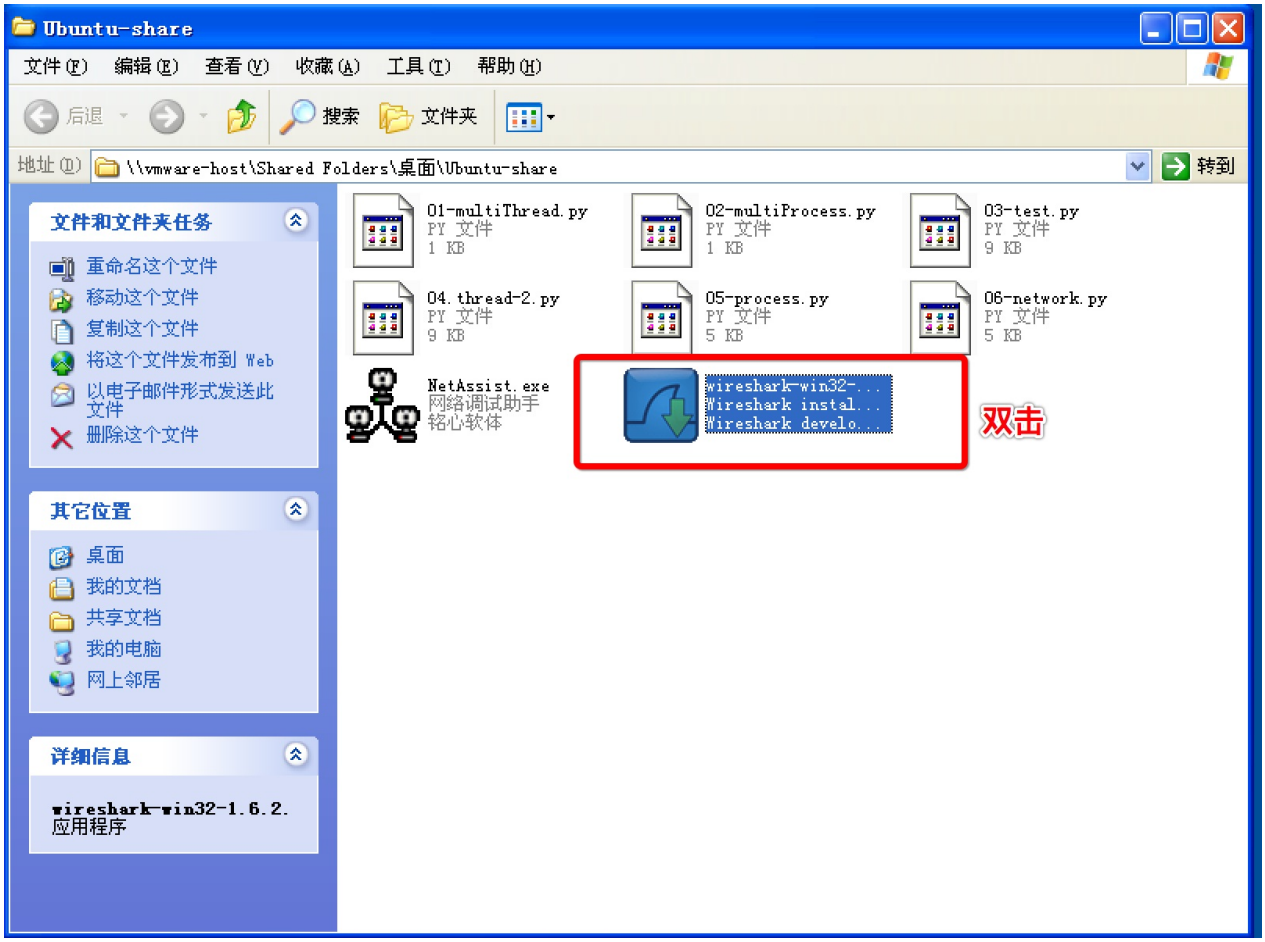
- 使用多线程完成一个全双工的QQ聊天程序

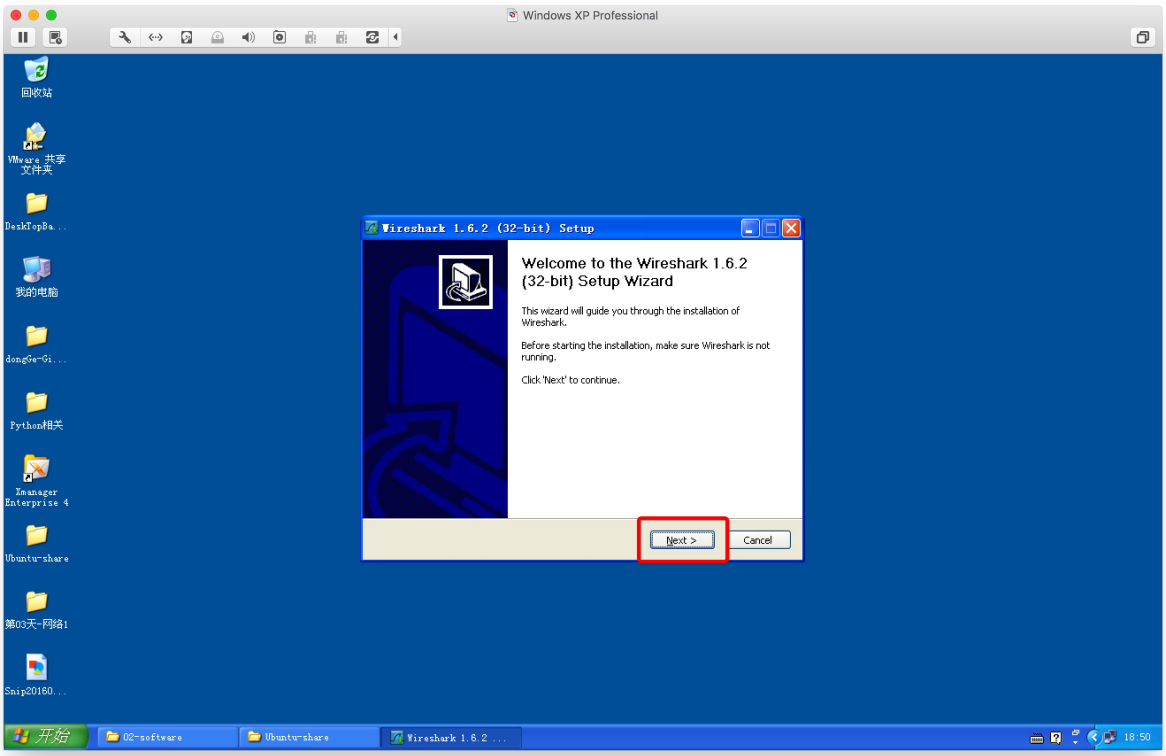
2. 运行现象如下

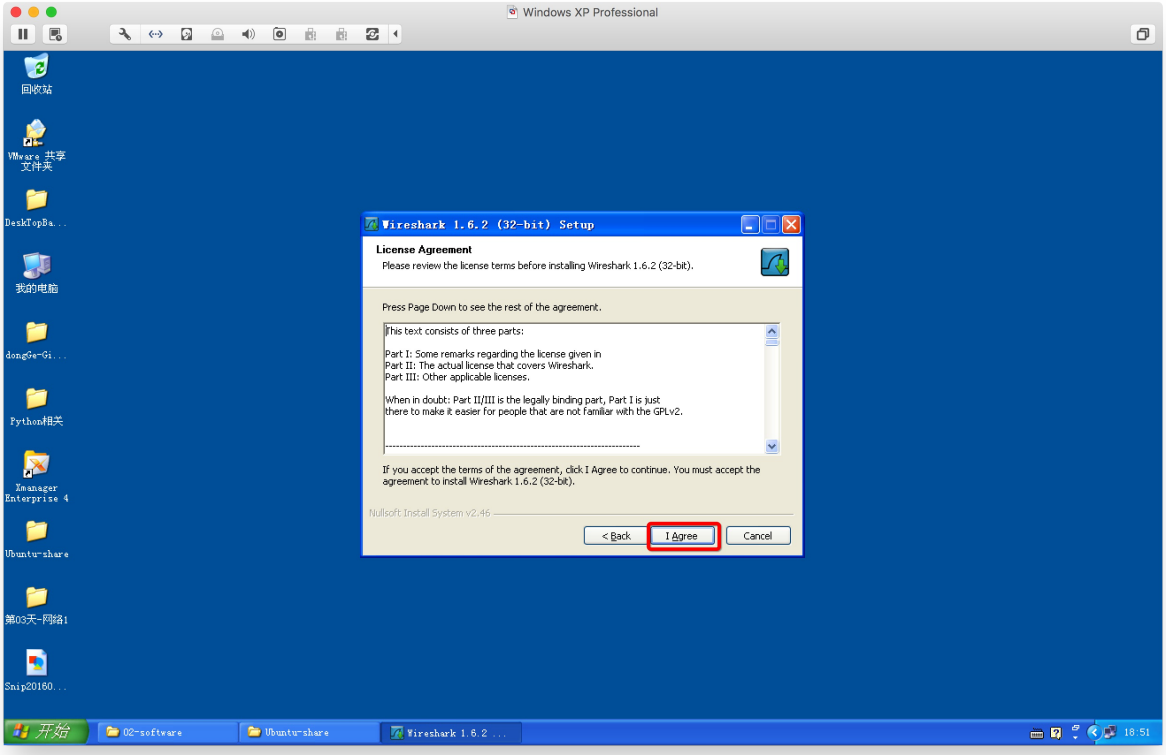


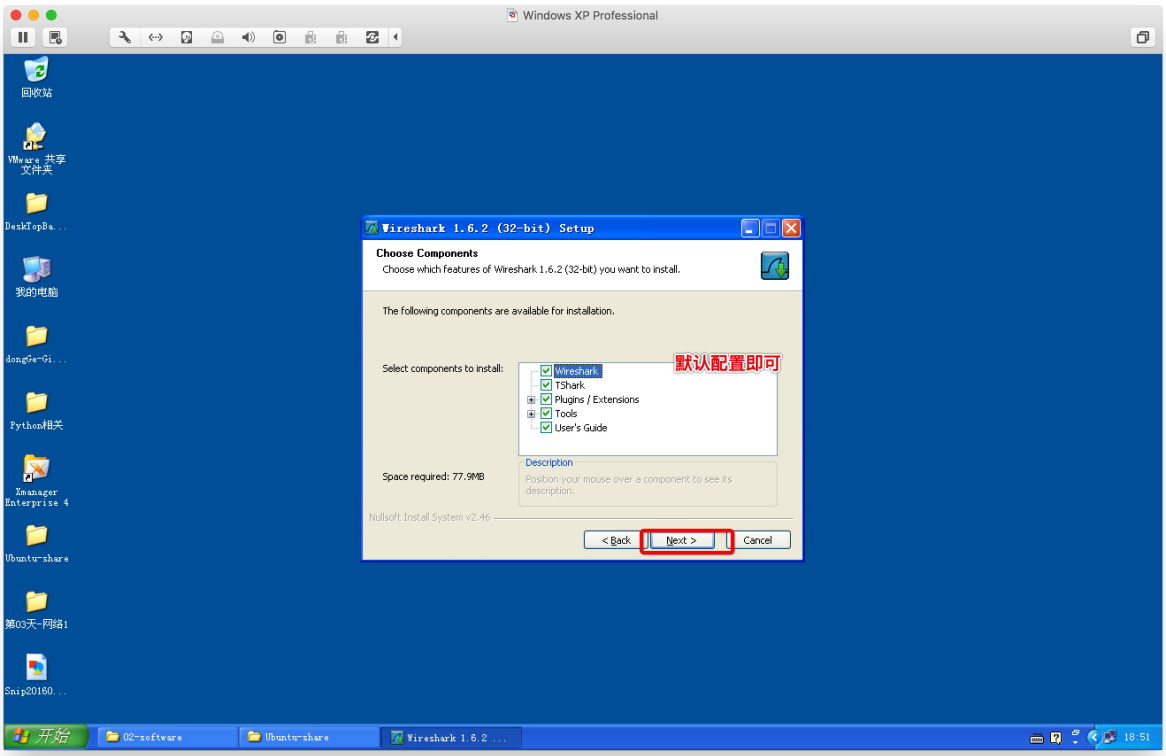
wireshark抓包工具使用

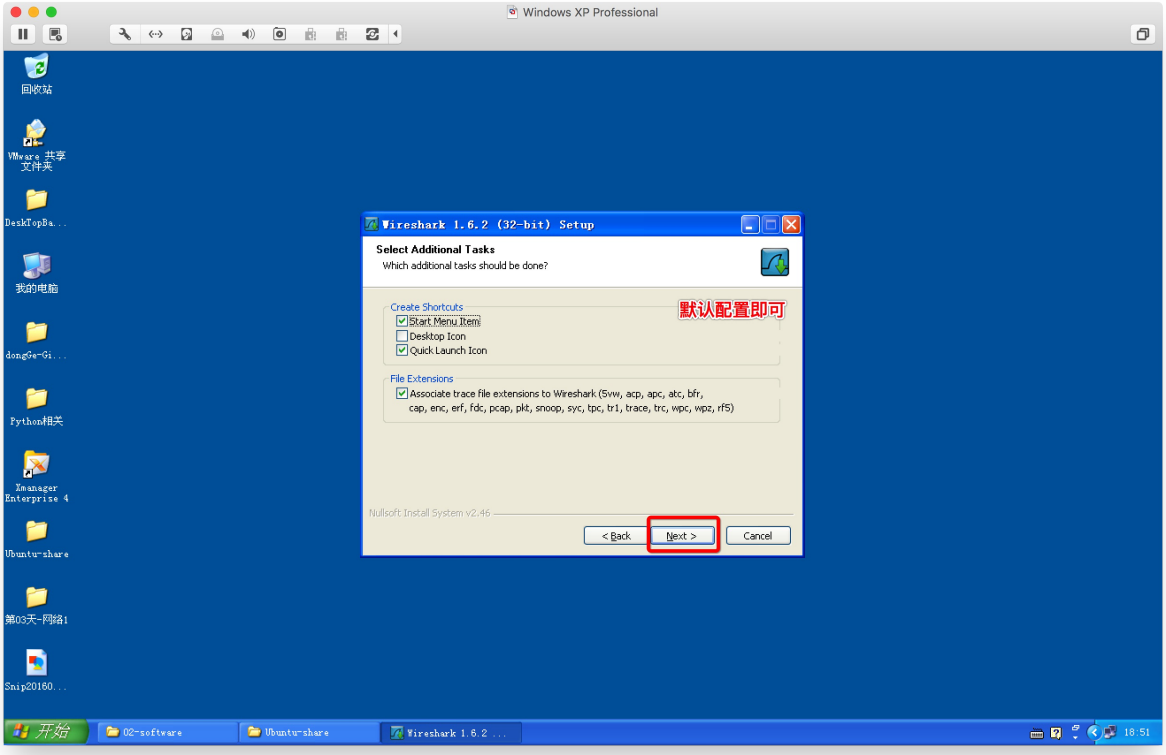
1. 安装wireshark

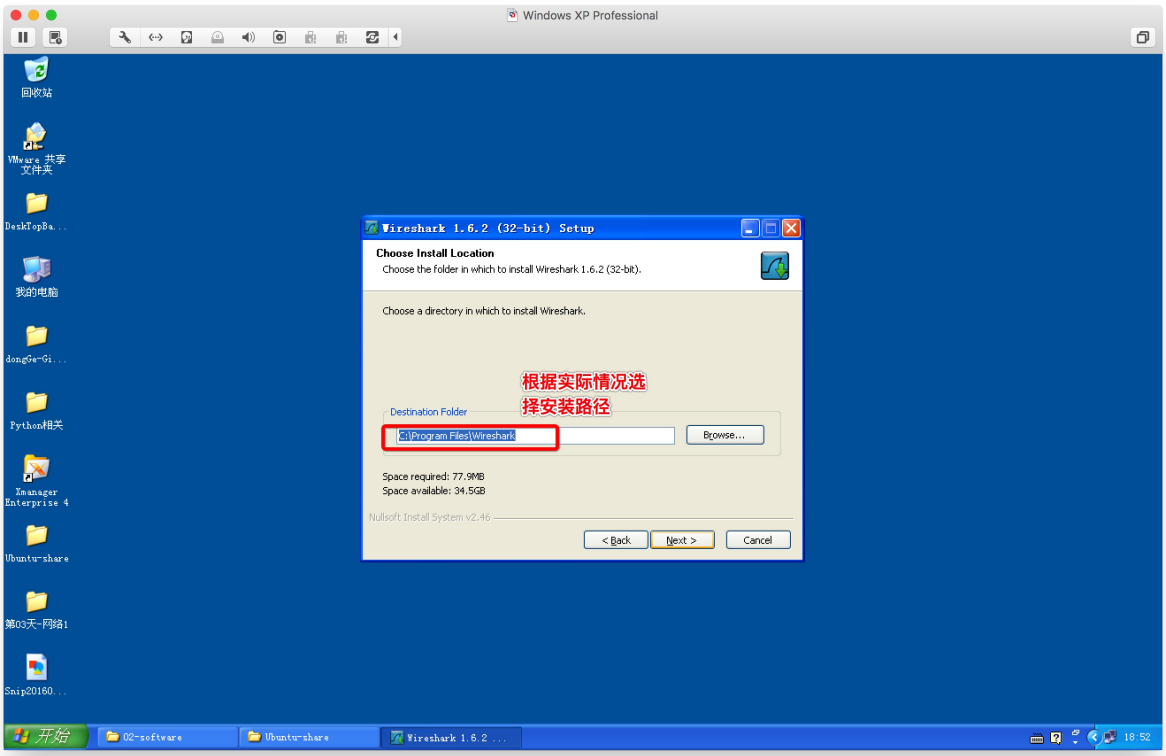


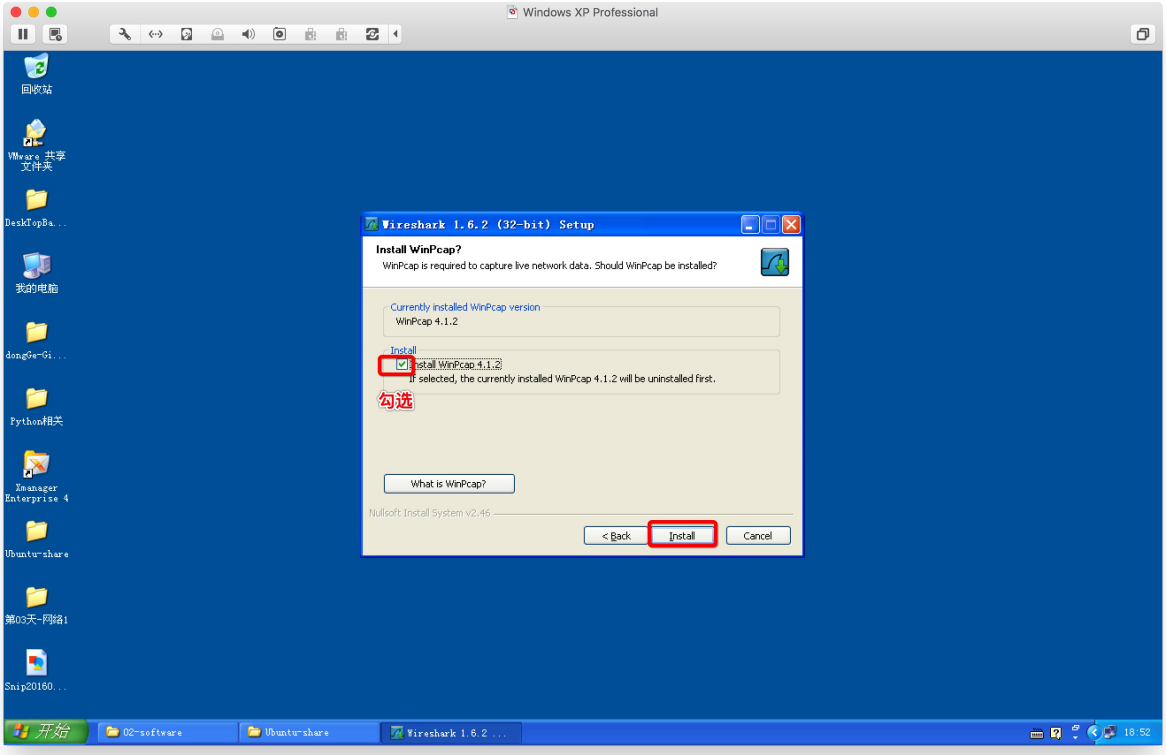

Windows XP Professional
Wireshark 1.6.2 (32-bit) Setup
Welcome to the Wireshark 1.6.2 (32-bit) Setup Wizard
This wizard will guide you through the installation of Wireshark.
Before starting the installation, make sure Wireshark is not running.
Click 'Next' to continue.
Next >
Cancel

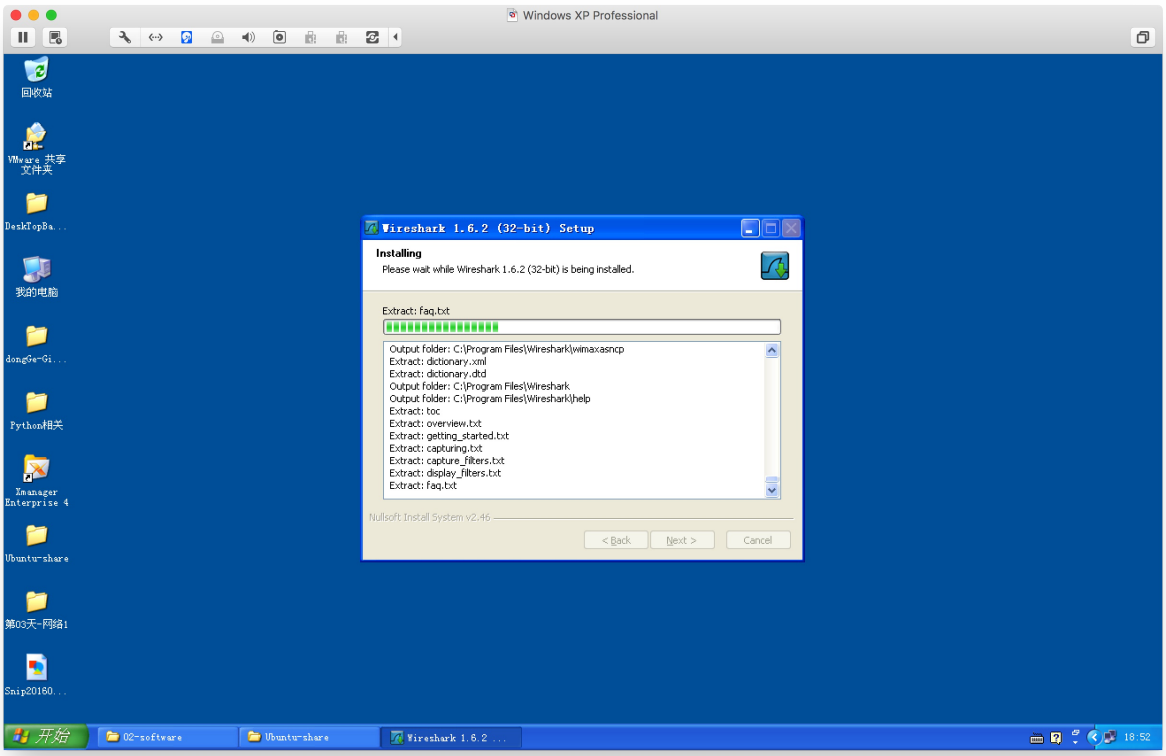

Windows XP Professional
Wireshark 1.6.2 (32-bit) Setup
License Agreement
Please review the license terms before installing Wireshark 1.6.2 (32-bit).
Press Page Down to see the rest of the agreement.
This text consists of three parts:
Part I: Some remarks regarding the license given in
Part II: The actual license that covers Wireshark.
Part III: Other applicable licenses.
When in doubt: Part II/III is the legally binding part, Part I is just there to make it easier for people that are not familiar with the GPLv2.
If you accept the terms of the agreement, click I Agree to continue. You must accept the agreement to install Wireshark 1.6.2 (32-bit).
Nullsoft Install System v2.46
< Back
I Agree
Cancel

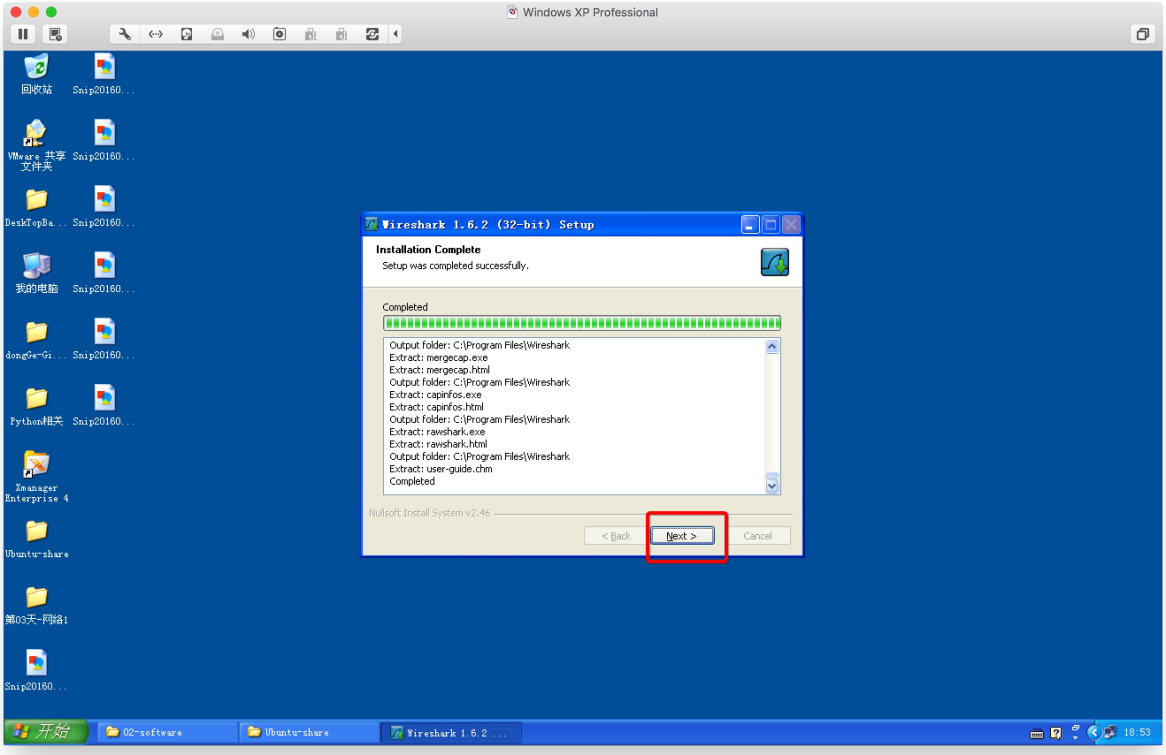

Windows XP Professional
Wireshark 1.6.2 (32-bit) Setup
Choose Components
Choose which features of Wireshark 1.6.2 (32-bit) you want to install.
The following components are available for installation.
Select components to install:
Wireshark
TShark
Plugins / Extensions
Tools
User's Guide
默认配置即可
Description
Position your mouse over a component to see its description.
Space required: 77.9MB
Nullsoft Install System v2.46
< Back
Next >
Cancel

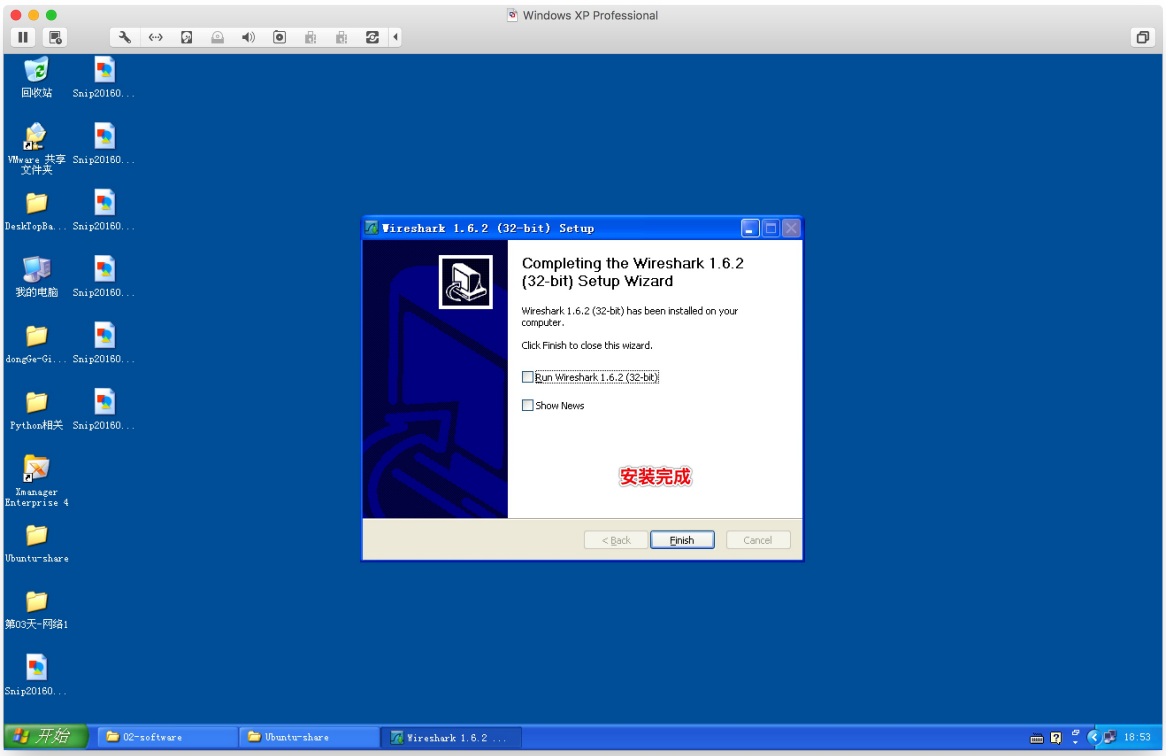

Windows XP Professional
Wireshark 1.6.2 (32-bit) Setup
Select Additional Tasks
Which additional tasks should be done?
Create Shortcuts
Start Menu Item
Desktop Icon
Quick Launch Icon
默认配置即可
File Extensions
Associate trace file extensions to Wireshark (5vw, acp, apc, atc, bfr, cap, enc, erf, fdc, pcap, pkt, snoop, syc, tpc, tr1, trace, trc, wpc, wpz, rf5)
Nullsoft Install System v2.46
< Back
Next >
Cancel


Windows XP Professional
Wireshark 1.6.2 (32-bit) Setup
Choose Install Location
Choose the folder in which to install Wireshark 1.6.2 (32-bit).
Choose a directory in which to install Wireshark.
根据实际情况选
择安装路径
Destination Folder
C:\Program Files\Wireshark
Browse...
Space required: 77.9MB
Space available: 34.5GB
Nullsoft Install System v2.46
< Back
Next >
Cancel

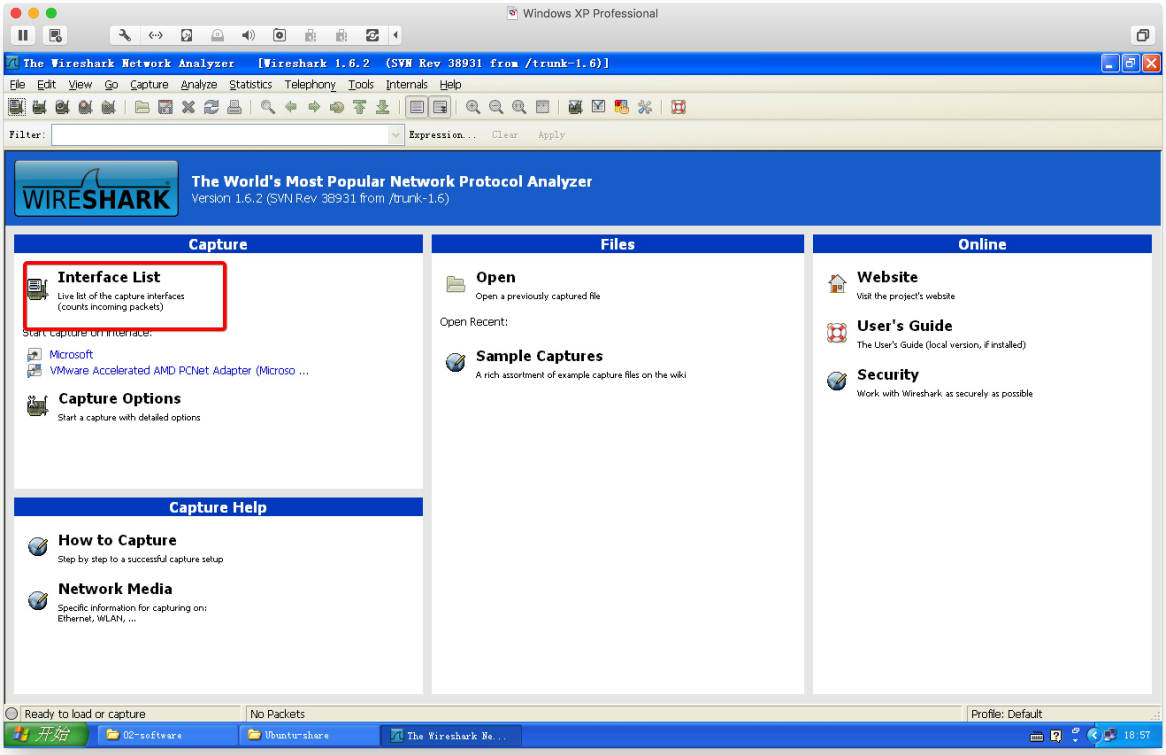

Windows XP Professional
Wireshark 1.6.2 (32-bit) Setup
Install WinPcap?
WinPcap is required to capture live network data. Should WinPcap be installed?
Currently installed WinPcap version
WinPcap 4.1.2
Install
Install WinPcap 4.1.2
If selected, the currently installed WinPcap 4.1.2 will be uninstalled first.
勾选
What is WinPcap?
Nullsoft Install System v2.46
< Back
Install
Cancel

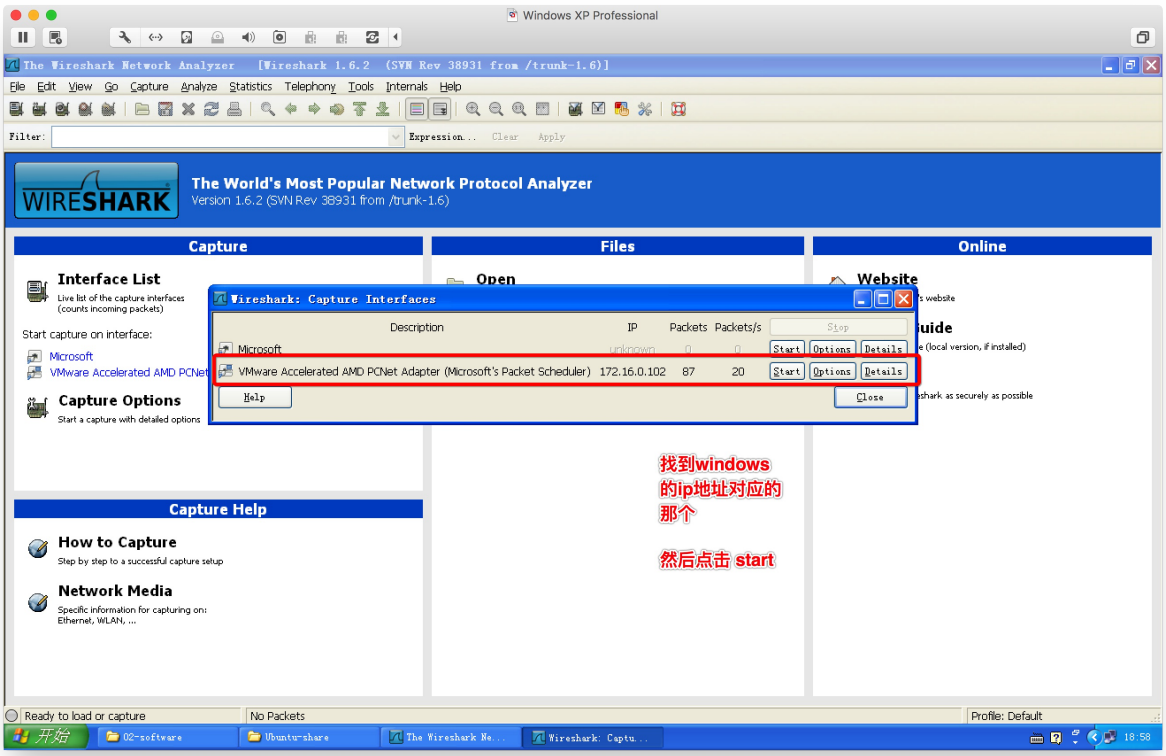

Windows XP Professional
Wireshark 1.6.2 (32-bit) Setup
Installing
Please wait while Wireshark 1.6.2 (32-bit) is being installed.
Extract: faq.txt
Output folder: C:\Program Files\Wireshark\wimaxasncp
Extract: dictionary.xml
Extract: dictionary.dtd
Output folder: C:\Program Files\Wireshark
Output folder: C:\Program Files\Wireshark\help
Extract: toc
Extract: overview.txt
Extract: getting_started.txt
Extract: capturing.txt
Extract: capture_filters.txt
Extract: display_filters.txt
Extract: faq.txt
Nullsoft Install System v2.46
< Back
Next >
Cancel
开始
18:52

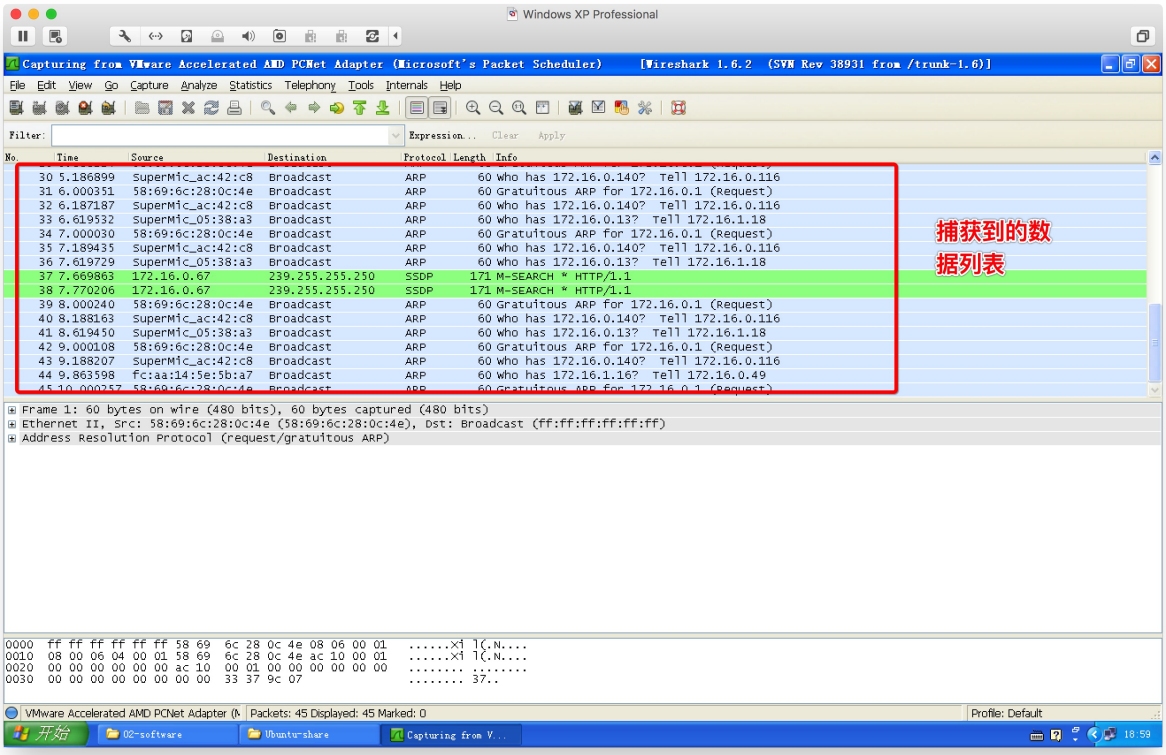

Windows XP Professional
Wireshark 1.6.2 (32-bit) Setup
Installation Complete
Setup was completed successfully.
Completed
Output folder: C:\Program Files\Wireshark
Extract: mergecap.exe
Extract: mergecap.html
Output folder: C:\Program Files\Wireshark
Extract: capinfos.exe
Extract: capinfos.html
Output folder: C:\Program Files\Wireshark
Extract: rawshark.exe
Extract: rawshark.html
Output folder: C:\Program Files\Wireshark
Extract: user-guide.chm
Completed
Nullsoft Install System v2.46
< Back
Next >
Cancel
开始
18:53

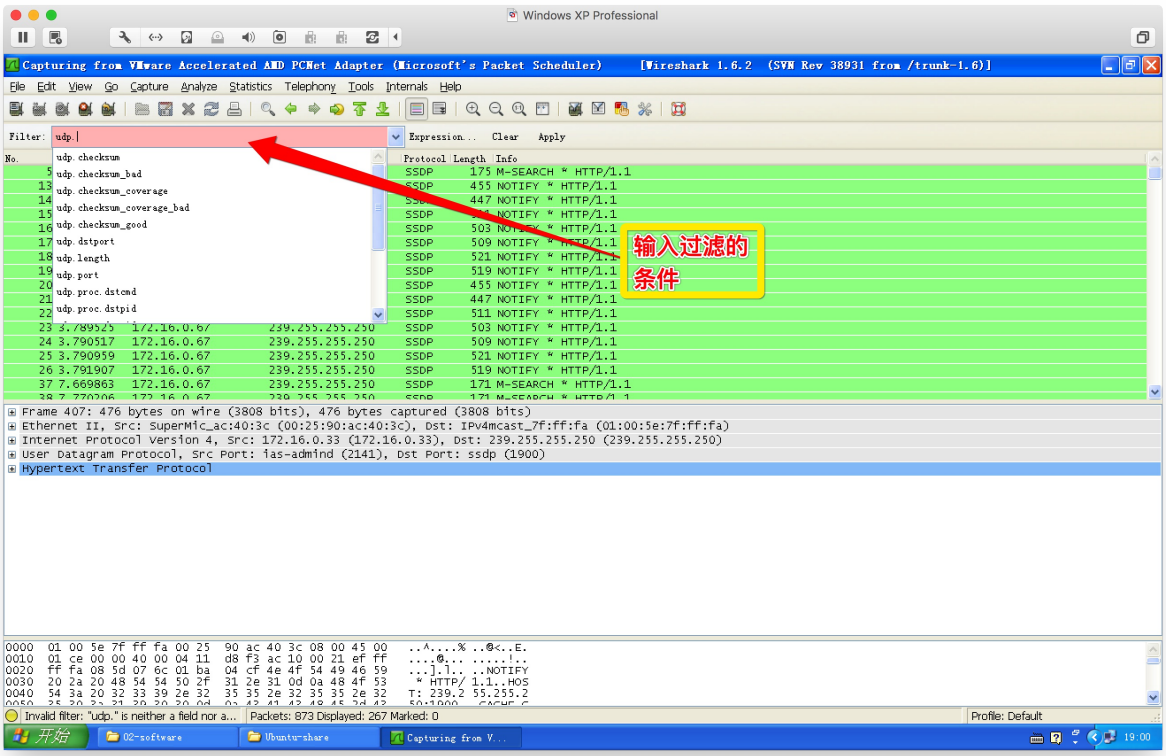


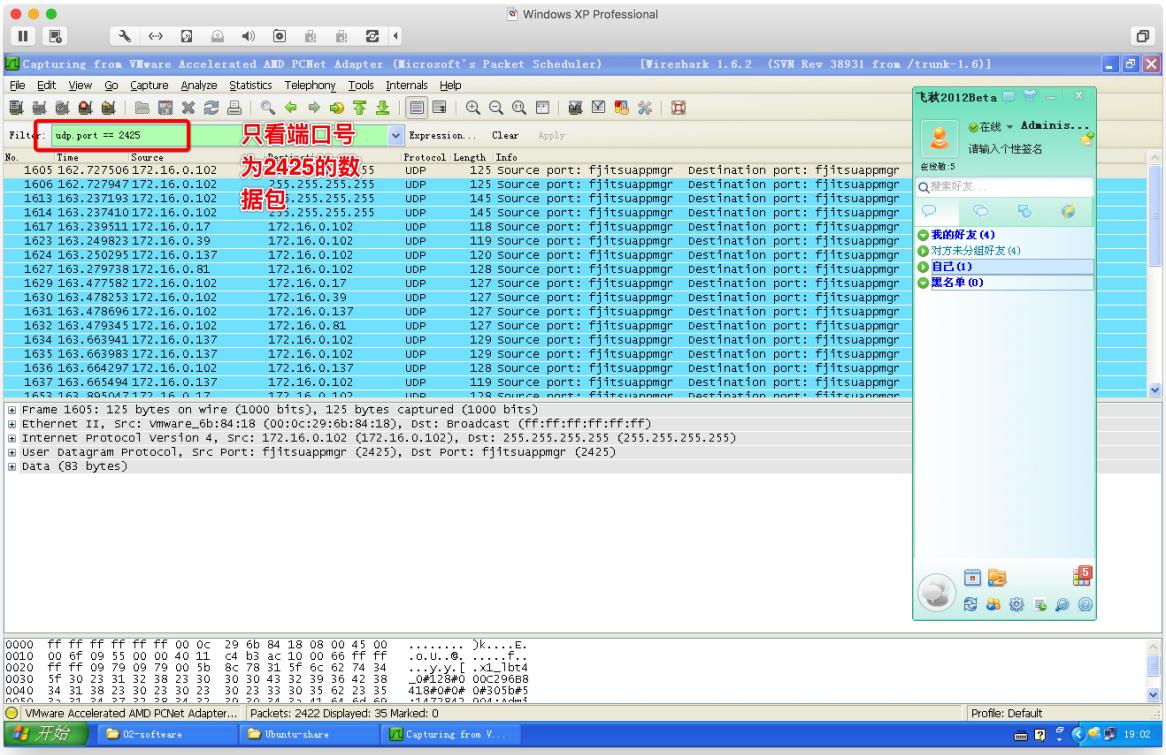
2. wireshark的使用

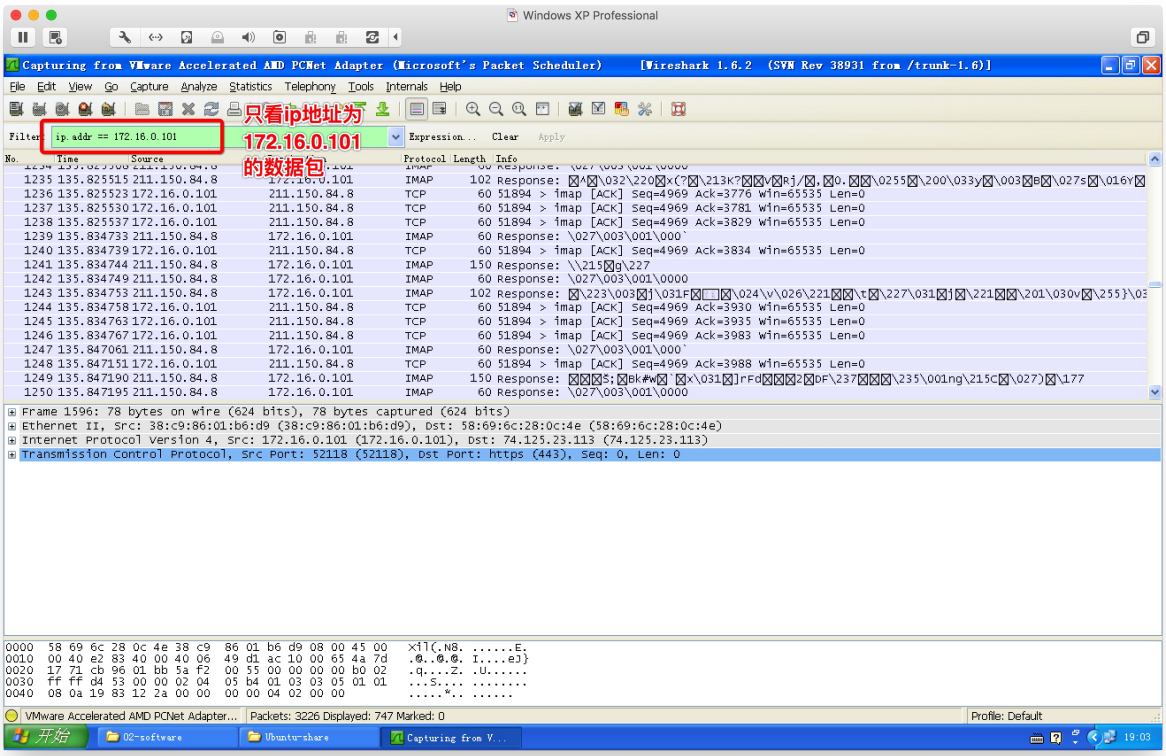


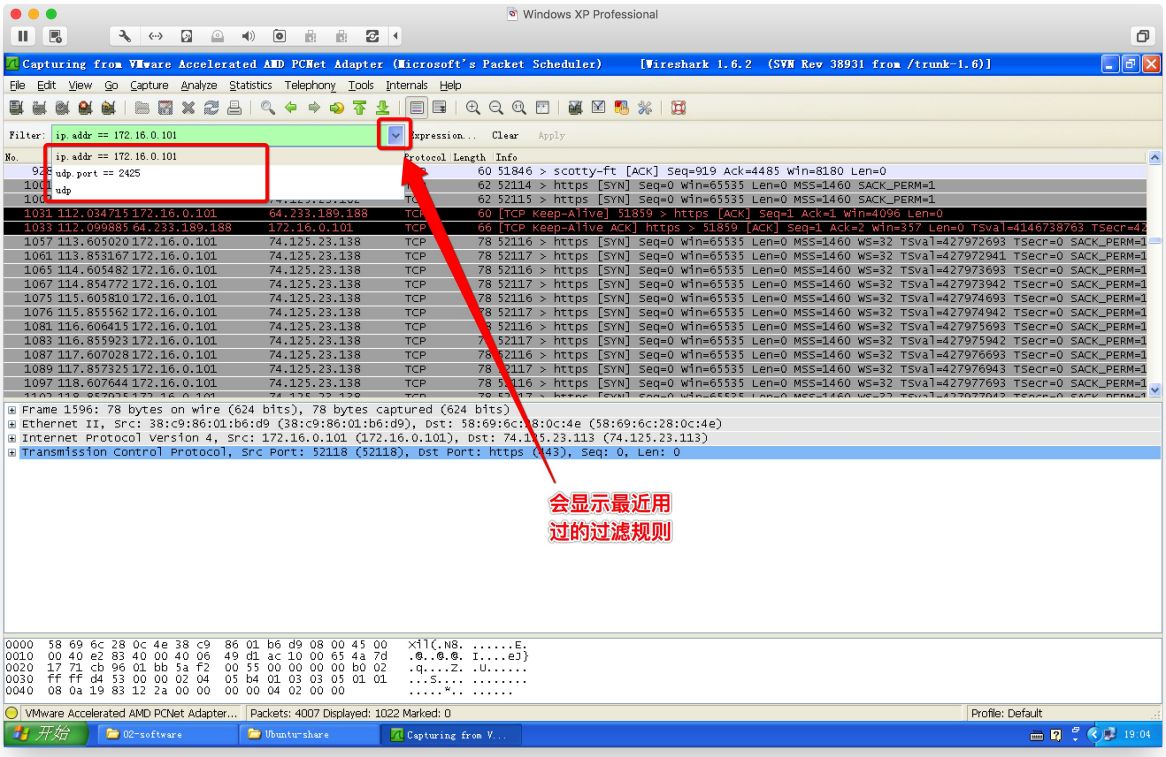

Windows XP Professional
The Wireshark Network Analyzer [Wireshark 1.6.2 (SVN Rev 38931 from /trunk-1.6)]
The World's Most Popular Network Protocol Analyzer
Version 1.6.2 (SVN Rev 38931 from /trunk-1.6)
Capture
Files
Online
Interface List
Capture Options
Capture Help
How to Capture
Network Media
Wireshark: Capture Interfaces
VMware Accelerated AMD PCNet Adapter (Microsoft's Packet Scheduler) 172.16.0.102
找到windows的ip地址对应的那个
然后点击 start

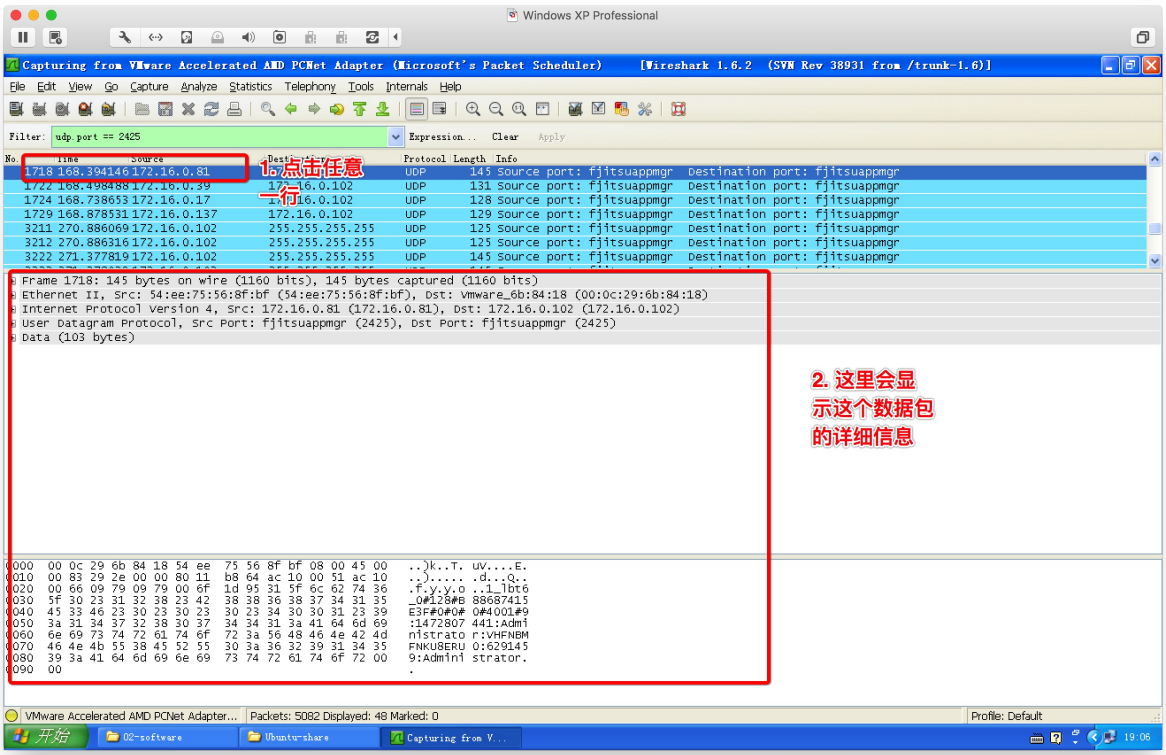

Windows XP Professional
Capturing from VMware Accelerated AMD PCNet Adapter (Microsoft's Packet Scheduler) [Wireshark 1.6.2 (SVN Rev 38931 from /trunk-1.6)]
捕获到的数据列表

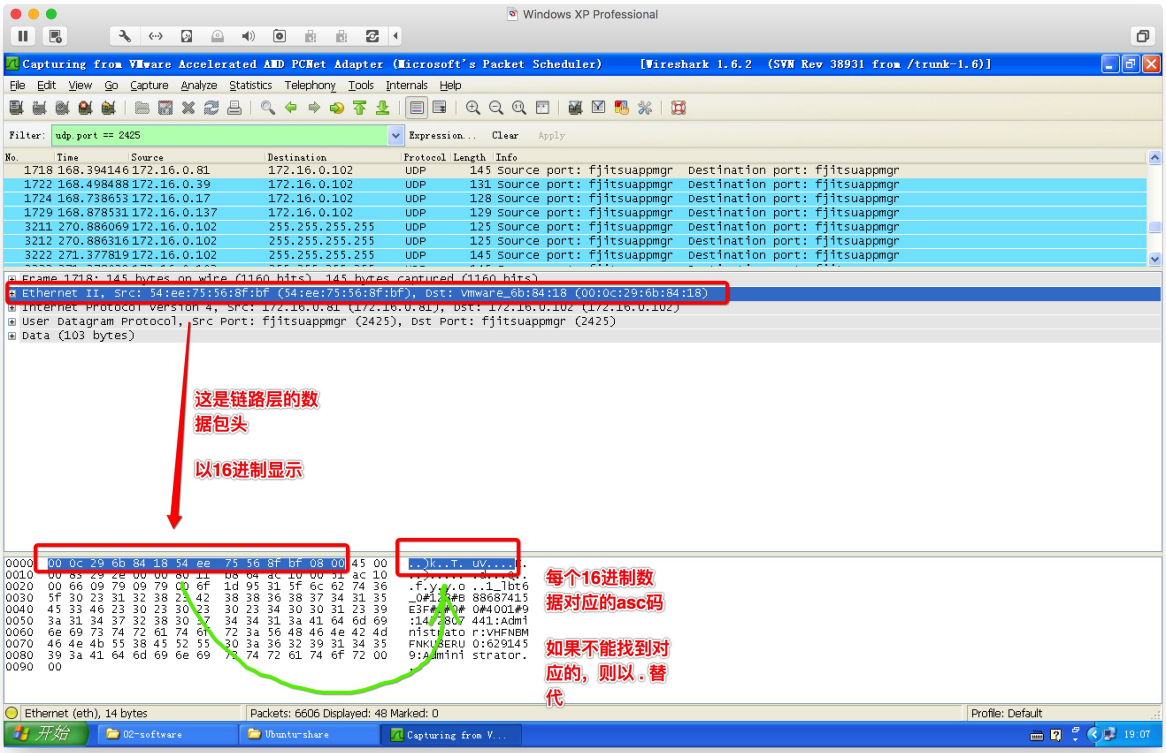

Windows XP Professional
Capturing from VMware Accelerated AMD PCNet Adapter (Microsoft's Packet Scheduler) [Wireshark 1.6.2 (SVN Rev 38931 from /trunk-1.6)]
File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Filter: udp.
udp.checksum
udp.checksum_bad
udp.checksum_coverage
udp.checksum_coverage_bad
udp.checksum_good
udp.dstport
udp.length
udp.port
udp.proc.dstcmd
udp.proc.dstpid
输入过滤的条件
Frame 407: 476 bytes on wire (3808 bits), 476 bytes captured (3808 bits)
Ethernet II, Src: SuperMic_ac:40:3c (00:25:90:ac:40:3c), Dst: IPv4mcast_7f:ff:fa (01:00:5e:7f:ff:fa)
Internet Protocol Version 4, Src: 172.16.0.33 (172.16.0.33), Dst: 239.255.255.250 (239.255.255.250)
User Datagram Protocol, Src Port: ias-admind (2141), Dst Port: ssdp (1900)
Hypertext Transfer Protocol

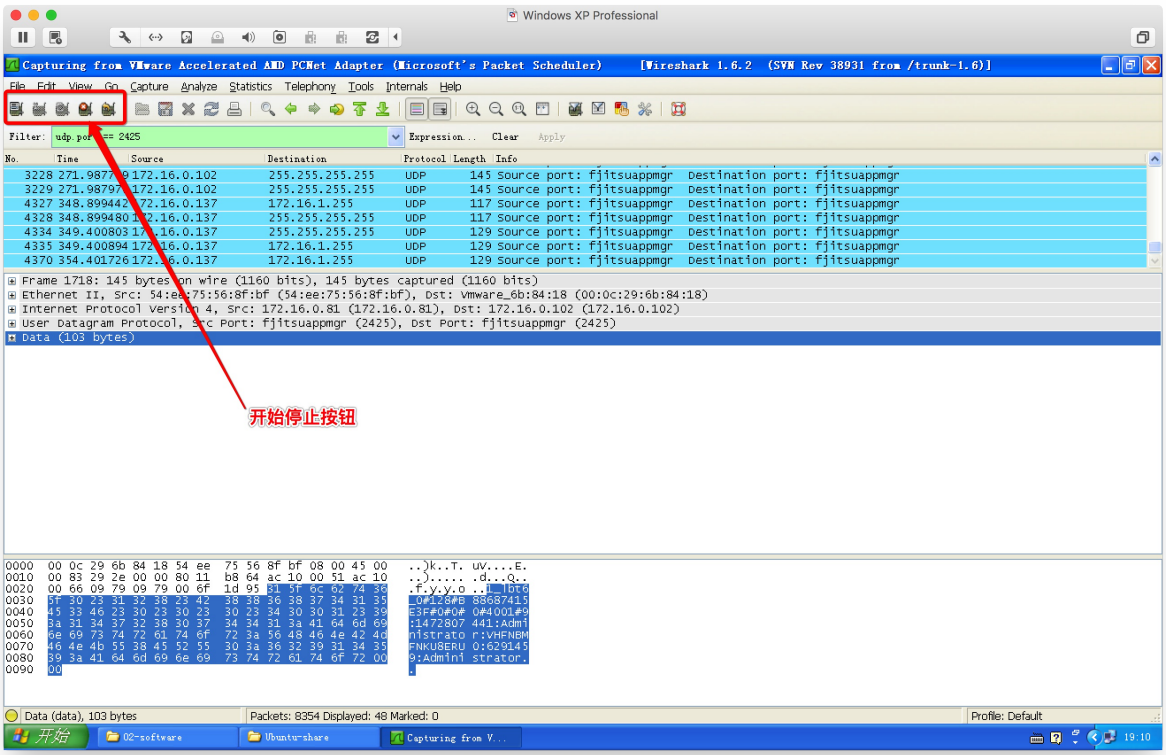

Windows XP Professional
Capturing from VMware Accelerated AMD PCNet Adapter (Microsoft's Packet Scheduler) [Wireshark 1.6.2 (SVN Rev 38931 from /trunk-1.6)]
File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Filter: udp.port == 2425
只看端口号为2425的数据包
飞秋2012Beta
我的好友 (4)
对方未分组好友 (4)
自己 (1)
黑名单 (0)
Frame 1605: 125 bytes on wire (1000 bits), 125 bytes captured (1000 bits)
Ethernet II, Src: Vmware_6b:84:18 (00:0c:29:6b:84:18), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
Internet Protocol Version 4, Src: 172.16.0.102 (172.16.0.102), Dst: 255.255.255.255 (255.255.255.255)
User Datagram Protocol, Src Port: fjitsuappmgr (2425), Dst Port: fjitsuappmgr (2425)
Data (83 bytes)


只看ip地址为
172.16.0.101
的数据包

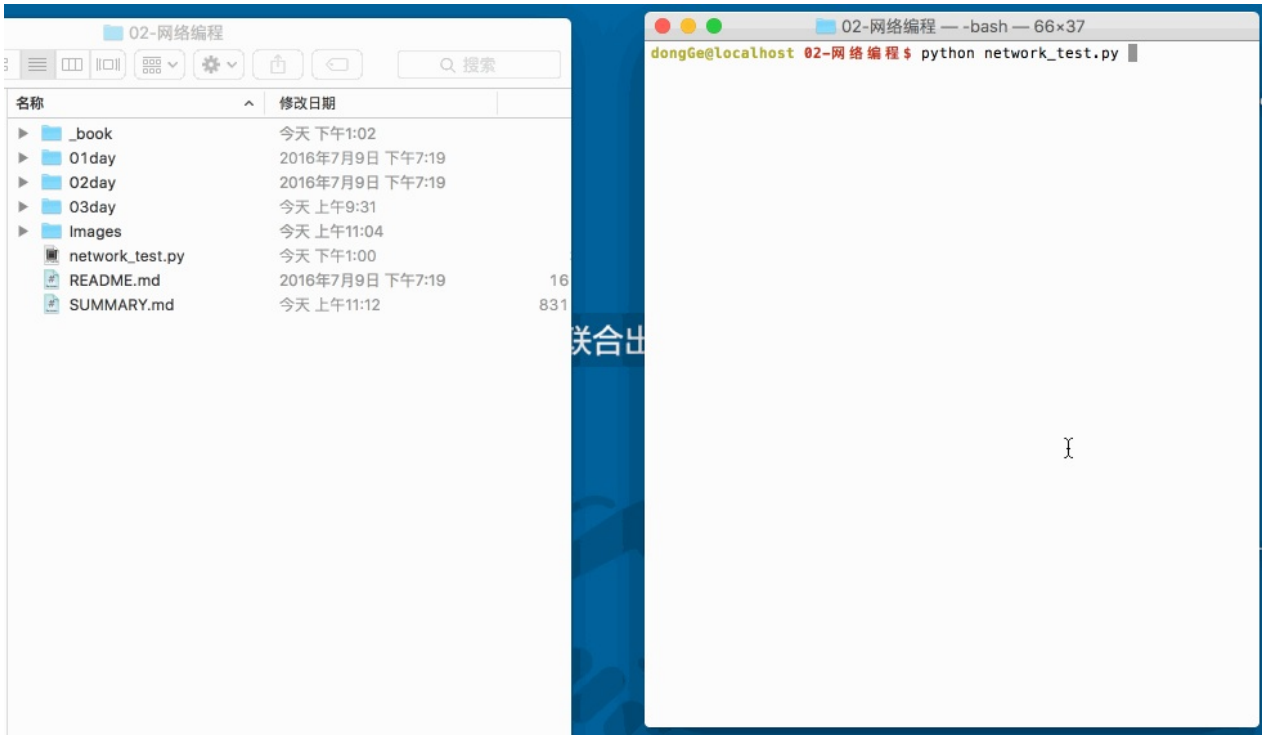

会显示最近用
过的过滤规则


Windows XP Professional
Capturing from VMware Accelerated AMD PCNet Adapter (Microsoft's Packet Scheduler) [Wireshark 1.6.2 (SVN Rev 38931 from /trunk-1.6)]
1. 点击任意一行
2. 这里会显示这个数据包的详细信息


Windows XP Professional
Capturing from VMware Accelerated AMD PCNet Adapter (Microsoft's Packet Scheduler) [Wireshark 1.6.2 (SVN Rev 38931 from /trunk-1.6)]
这是链路层的数据包头
以16进制显示
每个16进制数据对应的asc码
如果不能找到对应的，则以 . 替代


Windows XP Professional
Capturing from VMware Accelerated AMD PCNet Adapter (Microsoft's Packet Scheduler) [Wireshark 1.6.2 (SVN Rev 38931 from /trunk-1.6)]
File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Filter: udp.port == 2425
Expression... Clear Apply
No. Time Source Destination Protocol Length Info
3228 271.987779 172.16.0.102 255.255.255.255 UDP 145 Source port: fjitsuappmgr Destination port: fjitsuappmgr
3229 271.987971 172.16.0.102 255.255.255.255 UDP 145 Source port: fjitsuappmgr Destination port: fjitsuappmgr
4327 348.899442 172.16.0.137 172.16.1.255 UDP 117 Source port: fjitsuappmgr Destination port: fjitsuappmgr
4328 348.899480 172.16.0.137 255.255.255.255 UDP 117 Source port: fjitsuappmgr Destination port: fjitsuappmgr
4334 349.400803 172.16.0.137 255.255.255.255 UDP 129 Source port: fjitsuappmgr Destination port: fjitsuappmgr
4335 349.400894 172.16.0.137 172.16.1.255 UDP 129 Source port: fjitsuappmgr Destination port: fjitsuappmgr
4370 354.401726 172.16.0.137 172.16.1.255 UDP 129 Source port: fjitsuappmgr Destination port: fjitsuappmgr
Frame 1718: 145 bytes on wire (1160 bits), 145 bytes captured (1160 bits)
Ethernet II, Src: 54:ee:75:56:8f:bf (54:ee:75:56:8f:bf), Dst: vmware_6b:84:18 (00:0c:29:6b:84:18)
Internet Protocol Version 4, Src: 172.16.0.81 (172.16.0.81), Dst: 172.16.0.102 (172.16.0.102)
User Datagram Protocol, Src Port: fjitsuappmgr (2425), Dst Port: fjitsuappmgr (2425)
Data (103 bytes)
开始停止按钮
0000 00 0c 29 6b 84 18 54 ee 75 56 8f bf 08 00 45 00
0010 00 83 29 2e 00 00 80 11 b8 64 ac 10 00 51 ac 10
0020 00 66 09 79 09 79 00 6f 1d 95 31 5f 6c 62 74 36
0030 5f 30 23 31 32 38 23 42 38 38 36 38 37 34 31 35
0040 45 33 46 23 30 23 30 23 30 23 34 30 30 31 23 30
0050 3a 31 34 37 32 38 30 37 34 34 31 3a 41 64 6d 69
0060 6e 69 73 74 72 61 74 6f 72 3a 56 48 46 4e 42 4d
0070 46 4e 4b 55 38 45 52 55 30 3a 36 32 39 31 34 35
0080 39 3a 41 64 6d 69 6e 69 73 74 72 61 74 6f 72 00
0090 00
Data (data), 103 bytes
Packets: 8354 Displayed: 48 Marked: 0
Profile: Default
开始
02-software
Ubuntu-share
Capturing from V...
19:10

TFTP下载演示


02-网络编程
搜索
名称
修改日期
_book
今天 下午1:02
01day
2016年7月9日 下午7:19
02day
2016年7月9日 下午7:19
03day
今天 上午9:31
Images
今天 上午11:04
network_test.py
今天 下午1:00
README.md
2016年7月9日 下午7:19
16
SUMMARY.md
今天 上午11:12
831
02-网络编程 — -bash — 66×37
dongGe@localhost 02-网络编程$ python network_test.py

应用：TFTP客户端

1. TFTP协议介绍

TFTP（Trivial File Transfer Protocol,简单文件传输协议）

是TCP/IP协议族中的一个用来在客户端与服务器之间进行简单文件传输的协议

特点：

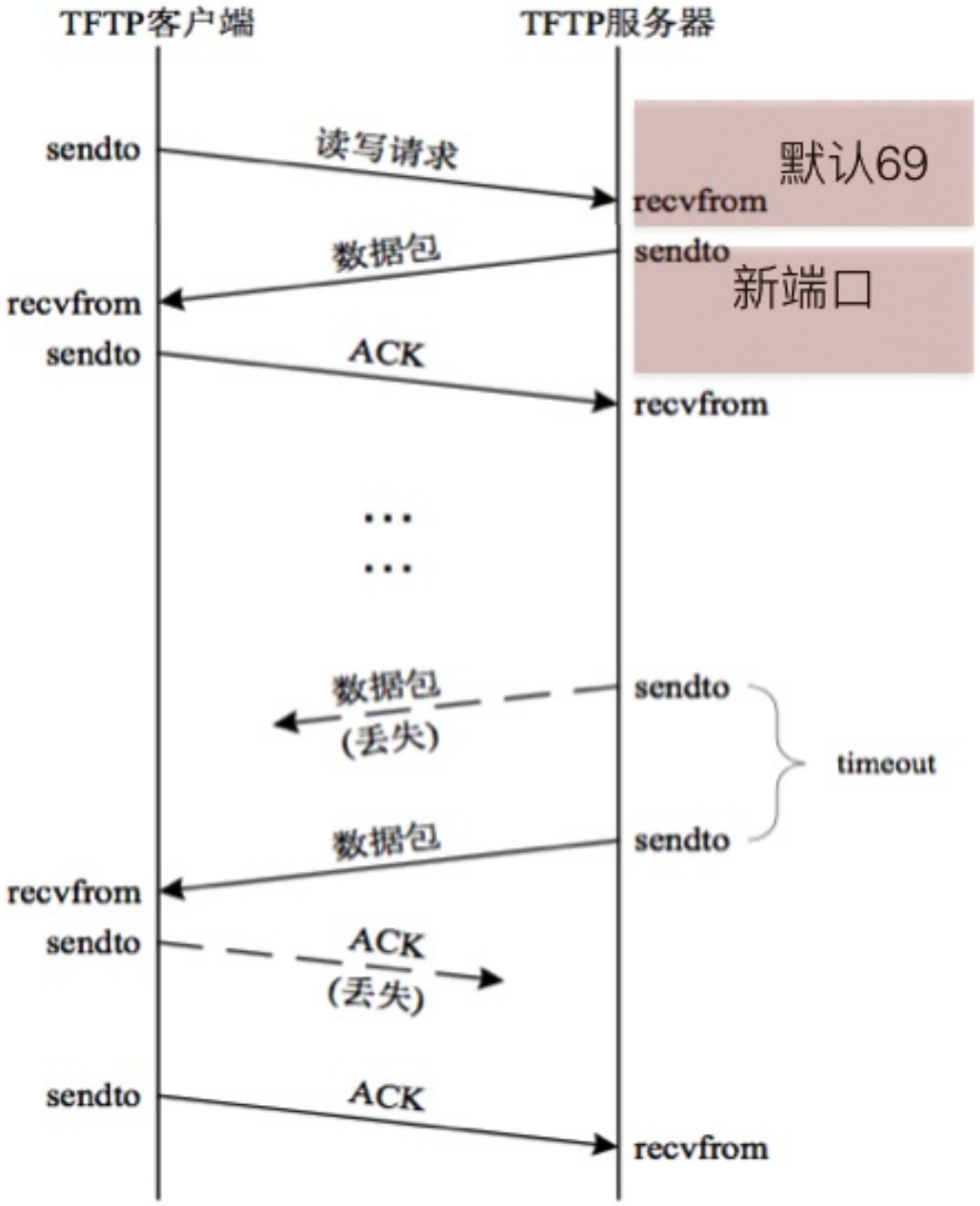
- 简单
- 占用资源小
- 适合传递小文件
- 适合在局域网进行传递
- 端口号为69
- 基于UDP实现

2. TFTP下载过程

TFTP服务器默认监听69号端口

当客户端发送“下载”请求（即读请求）时，需要向服务器的69端口发送

服务器若批准此请求,则使用一个新的、临时的 端口进行数据传输



当服务器找到需要现在的文件后，会立刻打开文件，把文件中的数据通过TFTP协议发送给客户端

如果文件的总大小较大（比如3M），那么服务器分多次发送，每次会从文件中读取512个字节的数据发送过来

因为发送的次数有可能会很多，所以为了让客户端对接收到的数据进行排序，所以在服务器发送那512个字节数据的时候，会多发2个字节的数据，用来存放序号，并且放在512个字节数据的前面，序号是从1开始的

因为需要从服务器上下载文件时，文件可能不存在，那么此时服务器就会发送一个错误的信息过来，为了区分服务发送的是文件内容还是错误的提示信息，所以又用了2个字节 来表示这个数据包的功能（称为操作码），并且在序号的前面

操作码	功能
1	读请求，即下载
2	写请求，即上传
3	表示数据包，即DATA
4	确认码，即ACK
5	错误

因为udp的数据包不安全，即发送方发送是否成功不能确定，所以TFTP协议中规定，为了让服务器知道客户端已经接收到了刚刚发送的那个数据包，所以当客户端接收到一个数据包的时候需要向服务器进行发送确认信息，即发送收到了，这样的包成为ACK(应答包)

为了标记数据已经发送完毕，所以规定，当客户端接收到的数据小于516（2字节操作码+2个字节的序号+512字节数据）时，就意味着服务器发送完毕了

TFTP数据包的格式如下:

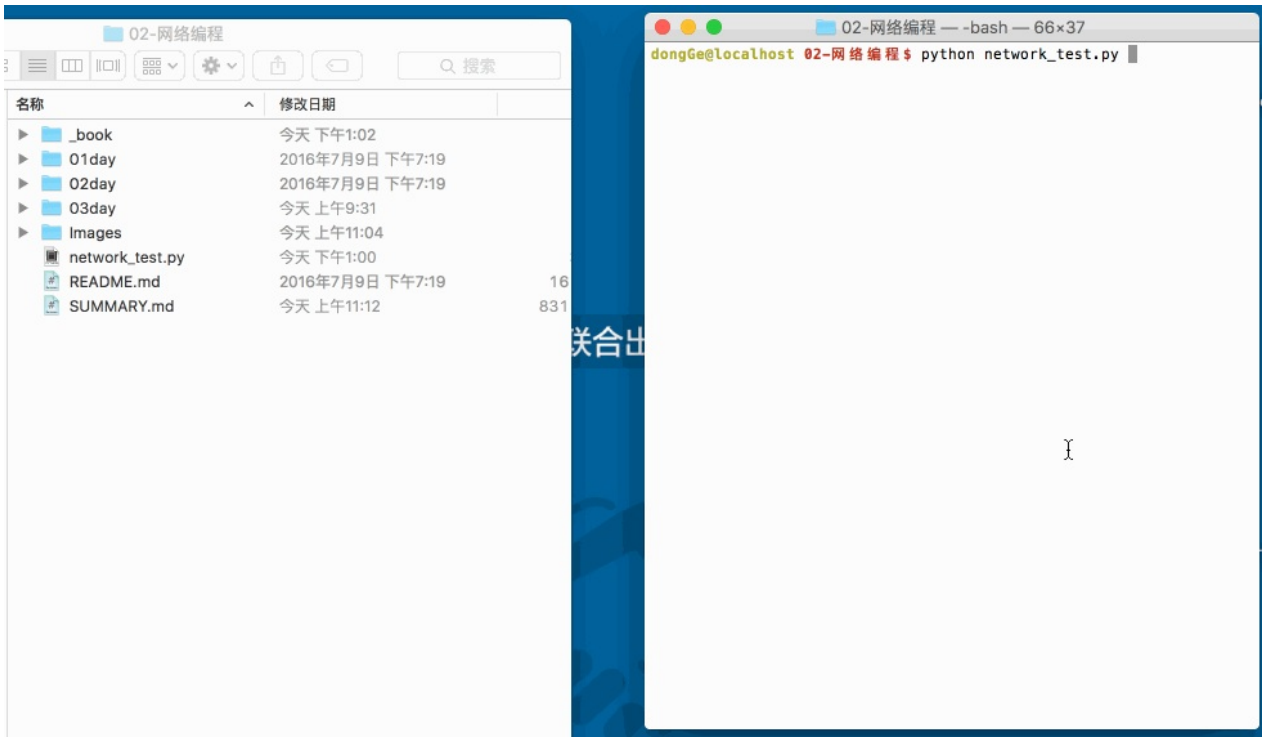
2. 参考代码如下：

```
#coding=utf-8

from socket import *
import struct
import sys

if len(sys.argv) != 2:
    print('-'*30)
    print("tips:")
    print("python xxxx.py 192.168.1.1")
    print('-'*30)
    exit()
else:
    ip = sys.argv[1]

# 创建udp套接字
udpSocket = socket(AF_INET, SOCK_DGRAM)

#构造下载请求数据
cmd_buf = struct.pack("!H8sb5sb",1,"test.jpg",0,"octet",0)
```

```
#发送下载文件请求数据到指定服务器
sendAddr = (ip, 69)
udpSocket.sendto(cmd_buf, sendAddr)

p_num = 0

recvFile = ''

while True:
    recvData,recvAddr = udpSocket.recvfrom(1024)

    recvDataLen = len(recvData)

    # print recvAddr # for test

    # print len(recvData) # for test

    cmdTuple = struct.unpack("!HH", recvData[:4])

    # print cmdTuple # for test

    cmd = cmdTuple[0]
    currentPackNum = cmdTuple[1]

    if cmd == 3: #是否为数据包

        # 如果是第一次接收到数据，那么就创建文件
        if currentPackNum == 1:
            recvFile = open("test.jpg", "a")

        # 包编号是否和上次相等
        if p_num+1 == currentPackNum:
            recvFile.write(recvData[4:]);
            p_num +=1
            print '(%d)次接收到的数据'%(p_num)

            ackBuf = struct.pack("!HH",4,p_num)
```

```
            udpSocket.sendto(ackBuf, recvAddr)
        # 如果收到的数据小于516则认为出错
        if recvDataLen<516:
            recvFile.close()
            print '已经成功下载！！！'
            break

    elif cmd == 5: #是否为错误应答
        print "error num:%d"%currentPackNum
        break

udpSocket.close()
```

运行现象：



udp广播

现实生活中的广播

网络编程中的广播

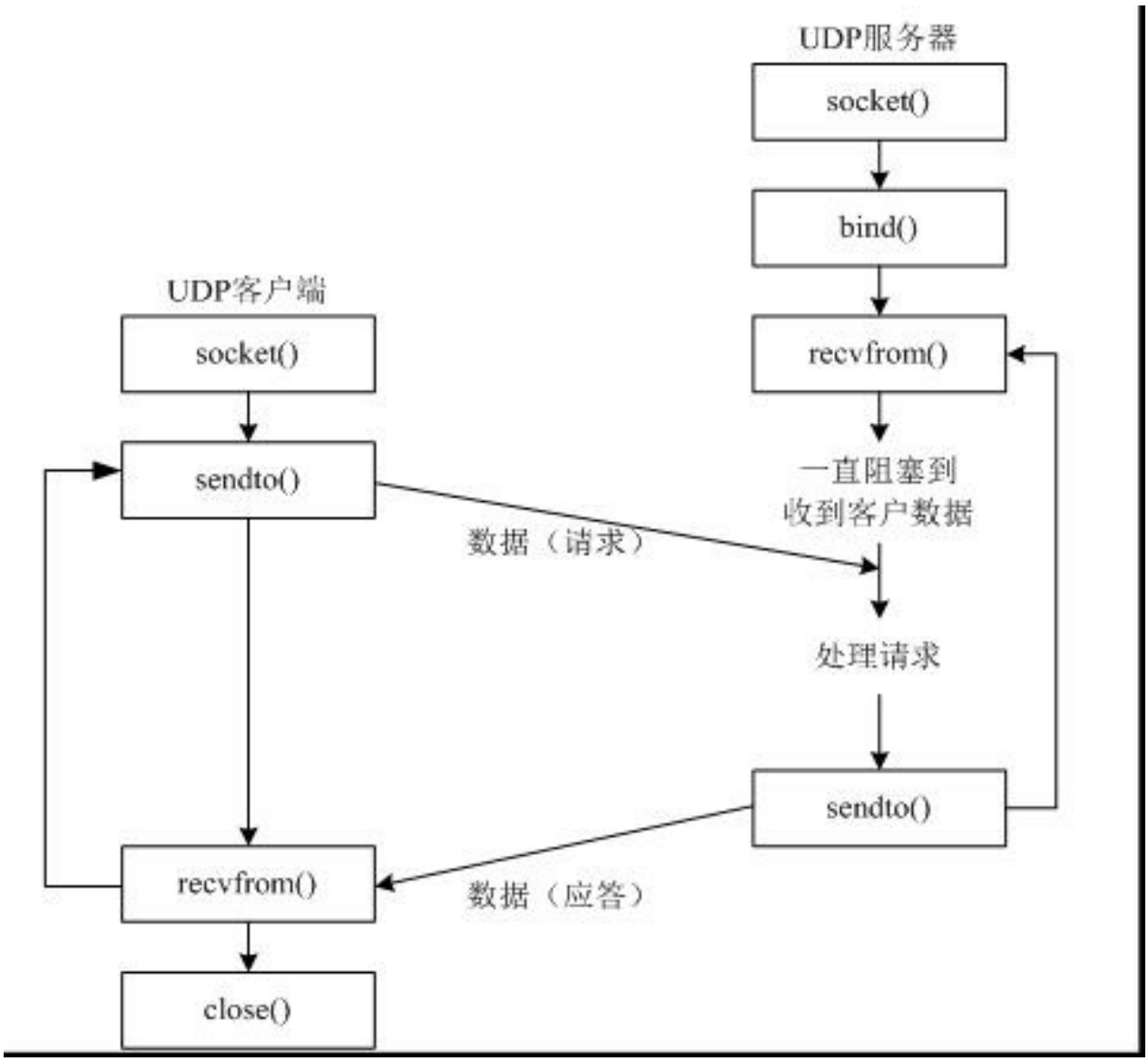
```
#coding=utf-8

import socket, sys

dest = ('<broadcast>', 7788)

# 创建udp套接字
s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
```

```
# 对这个需要发送广播数据的套接字进行修改设置，否则不能发送广播数据
s.setsockopt(socket.SOL_SOCKET, socket.SO_BROADCAST,1)

# 以广播的形式发送数据到本网络的所有电脑中
s.sendto("Hi", dest)

print "等待对方回复（按ctrl+c退出）"

while True:
    (buf, address) = s.recvfrom(2048)
    print "Received from %s: %s" % (address, buf)
```

tcp相关介绍

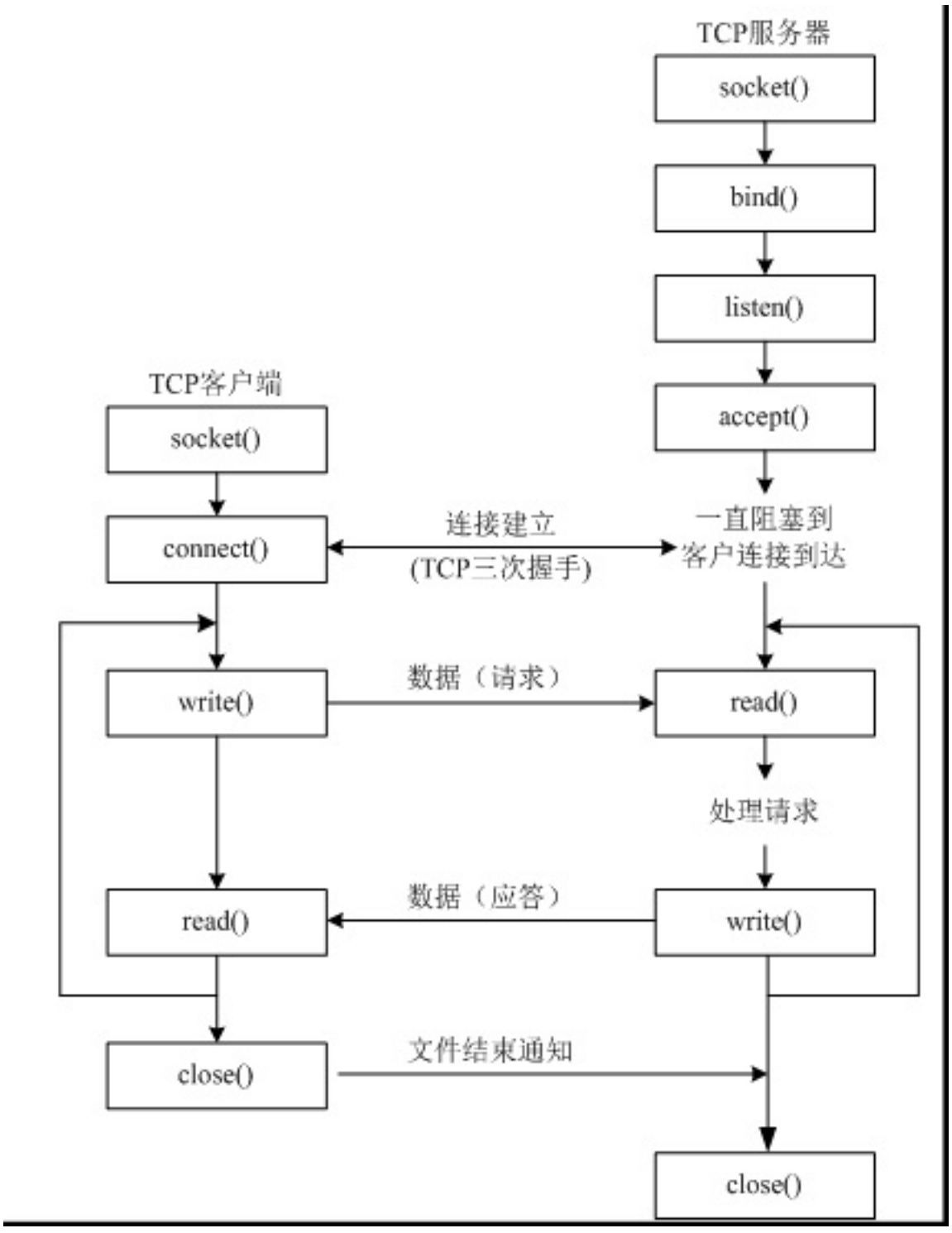
udp通信模型

udp通信模型中，在通信开始之前，不需要建立相关的链接，只需要发送数据即可，类似于生活中，"写信""



tcp通信模型

udp通信模型中，在通信开始之前，一定要先建立相关的链接，才能发送数据，类似于生活中，"打电话""


TCP服务器
socket()
bind()
listen()
accept()
一直阻塞到
客户连接到达
TCP客户端
socket()
connect()
连接建立
(TCP三次握手)
write()
数据（请求）
read()
处理请求
read()
数据（应答）
write()
close()
文件结束通知
close()

tcp服务器

生活中的电话机

如果想让别人能更够打通咱们的电话获取相应服务的话，需要做一下几件事情：

1. 买个手机
2. 插上手机卡
3. 设计手机为正常接听状态（即能够响铃）
4. 静静的等着别人拨打

tcp服务器

如同上面的电话机过程一样，在程序中，如果想要完成一个tcp服务器的功能，需要的流程如下：

1. socket创建一个套接字
2. bind绑定ip和port
3. listen使套接字变为可以被动链接
4. accept等待客户端的链接
5. recv/send接收发送数据

一个很简单的tcp服务器如下：

```
#coding=utf-8
from socket import *

# 创建socket
tcpSerSocket = socket(AF_INET, SOCK_STREAM)

# 绑定本地信息
address = ('', 7788)
```

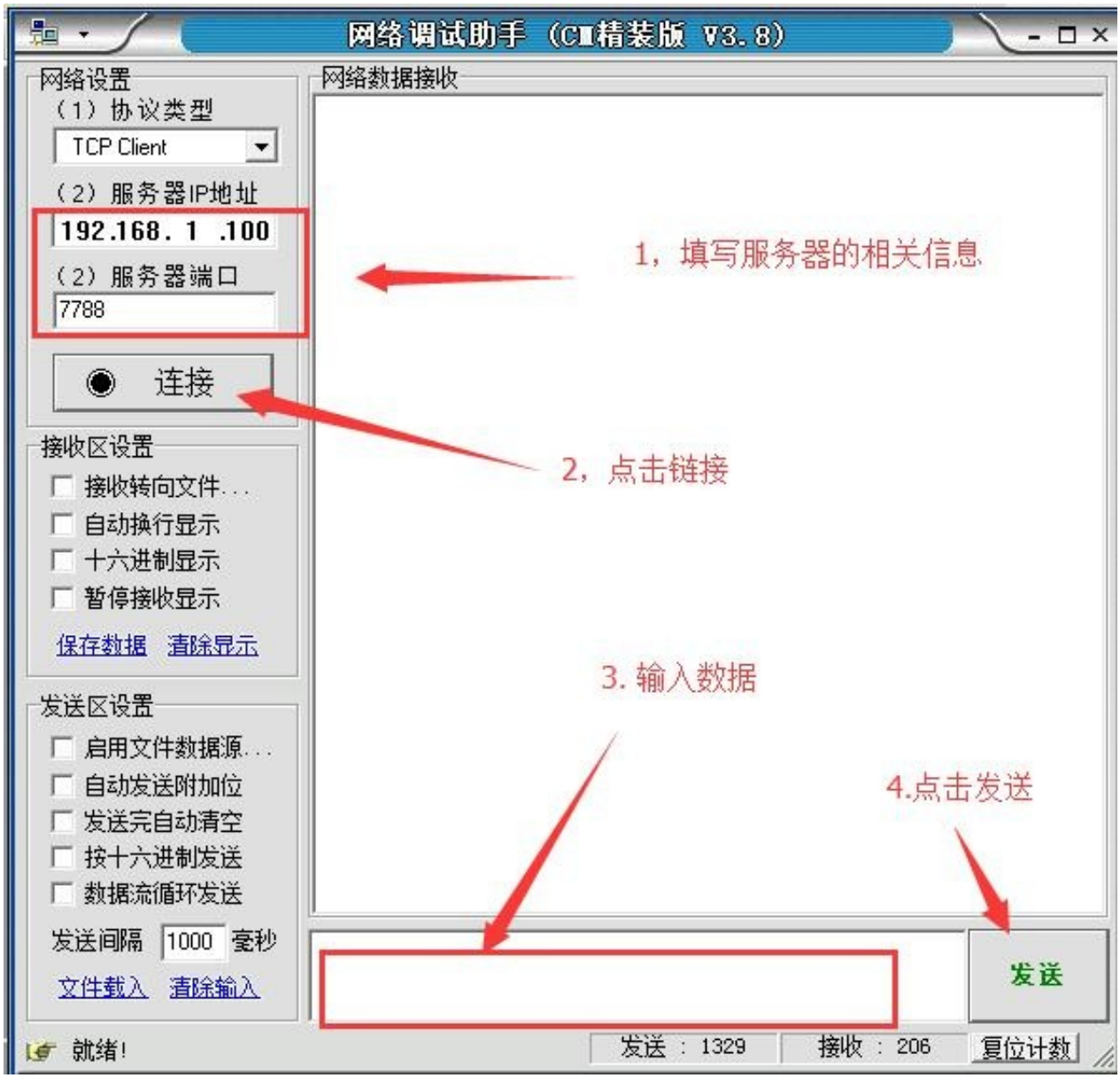
```
tcpSerSocket.bind(address)

# 使用socket创建的套接字默认的属性是主动的，使用listen将其变为被动的，这样
tcpSerSocket.listen(5)

# 如果有新的客户端来链接服务器，那么就产生一个新的套接字专门为这个客户端服务
# newSocket用来为这个客户端服务
# tcpSerSocket就可以省下来专门等待其他新客户端的链接
newSocket, clientAddr = tcpSerSocket.accept()

# 接收对方发送过来的数据，最大接收1024个字节
recvData = newSocket.recv(1024)
print '接收到的数据为:',recvData

# 发送一些数据到客户端
newSocket.send("thank you !")

# 关闭为这个客户端服务的套接字，只要关闭了，就意味着为不能再为这个客户端服务
newSocket.close()

# 关闭监听套接字，只要这个套接字关闭了，就意味着整个程序不能再接收任何新的客
tcpSerSocket.close()
```

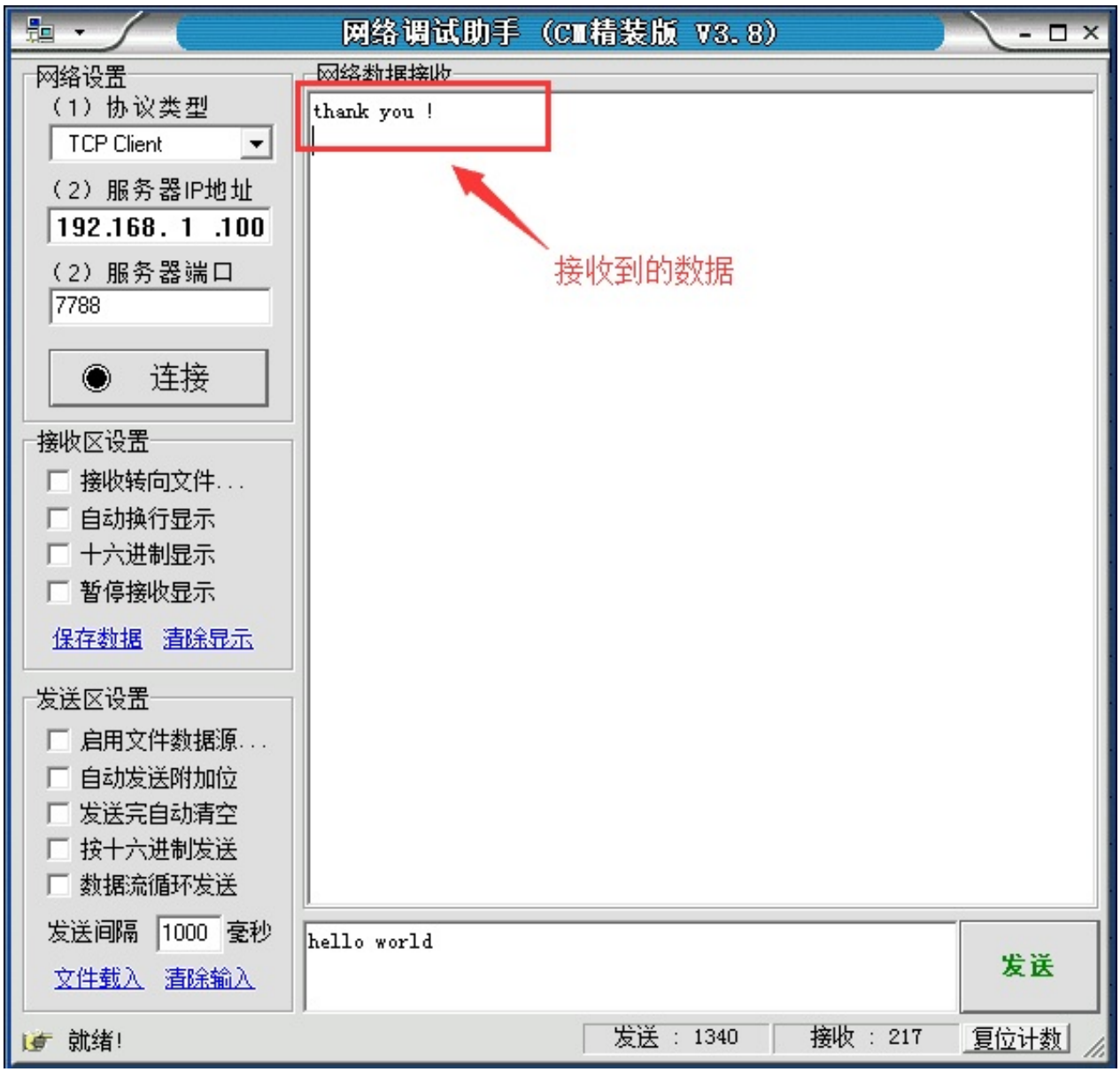
运行流程：

<1>tcp服务器

```
[dongGe@localhost 02-网络编程$ python network_test.py
接收到的数据为: hello world
```

<2>网络调试助手：


网络调试助手（CM精装版 V3.8）
网络设置
（1）协议类型
TCP Client
（2）服务器IP地址
192.168. 1 .100
（2）服务器端口
7788
连接
接收区设置
接收转向文件...
自动换行显示
十六进制显示
暂停接收显示
保存数据 清除显示
发送区设置
启用文件数据源...
自动发送附加位
发送完自动清空
按十六进制发送
数据流循环发送
发送间隔 1000 毫秒
文件载入 清除输入
网络数据接收
1，填写服务器的相关信息
2，点击链接
3. 输入数据
4.点击发送
发送
就绪!
发送 : 1329
接收 : 206
复位计数


网络调试助手 (CM精装版 V3.8)
网络设置
(1) 协议类型
TCP Client
(2) 服务器IP地址
192.168. 1 .100
(2) 服务器端口
7788
连接
接收区设置
接收转向文件...
自动换行显示
十六进制显示
暂停接收显示
保存数据 清除显示
发送区设置
启用文件数据源...
自动发送附加位
发送完自动清空
按十六进制发送
数据流循环发送
发送间隔 1000 毫秒
文件载入 清除输入
网络数据接收
thank you !
接收到的数据
hello world
发送
就绪!
发送 : 1340
接收 : 217
复位计数

tcp客户端

tcp客户端，并不是像之前一个段子：一个顾客去饭馆吃饭，这个顾客要点菜，就问服务员咱们饭店有客户端么，然后这个服务员非常客气的说道：先生 我们饭店不用客户端，我们直接送到您的餐桌上

如果，不学习网络的知识是不是 说不定也会发生那样的笑话 ， 哈哈

所谓的服务器端：就是提供服务的一方，而客户端，就是需要被服务的一方

tcp客户端构建流程

tcp的客户端要比服务器端简单很多，如果说服务器端是需要自己买手机、查手机卡、设置铃声、等待别人打电话流程的话，那么客户端就只需要找一个电话亭，拿起电话拨打即可，流程要少很多

示例代码：

```
#coding=utf-8
from socket import *

# 创建socket
tcpClientSocket = socket(AF_INET, SOCK_STREAM)

# 链接服务器
serAddr = ('192.168.1.102', 7788)
tcpClientSocket.connect(serAddr)

# 提示用户输入数据
sendData = raw_input("请输入要发送的数据：")

tcpClientSocket.send(sendData)

# 接收对方发送过来的数据，最大接收1024个字节
recvData = tcpClientSocket.recv(1024)
```

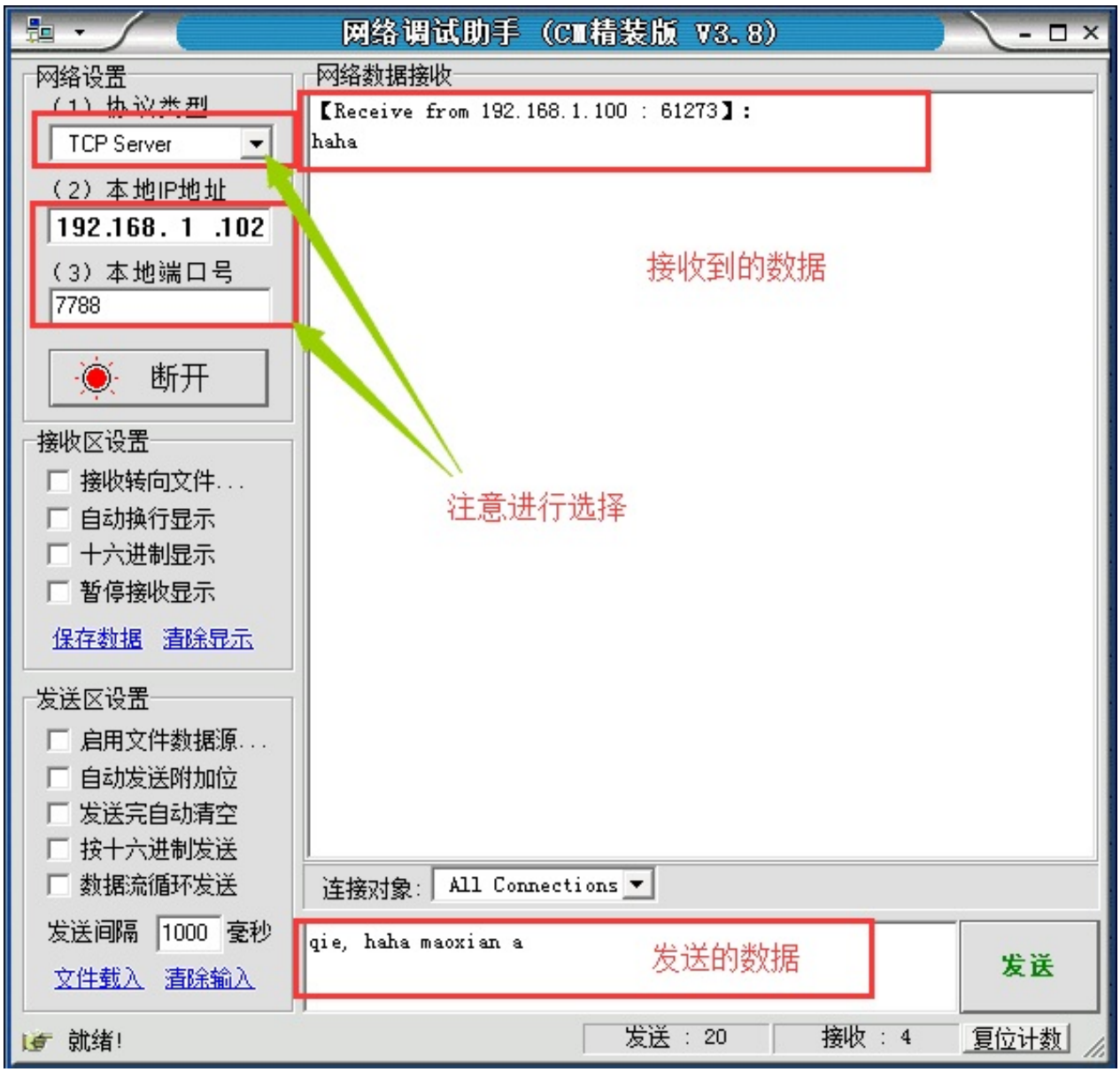
```
print '接收到的数据为:',recvData

# 关闭套接字
tcpClientSocket.close()
```

运行流程：

<1>tcp客户端

```
[dongGe@localhost 02-网络编程$ python network_test.py
请输入要发送的数据: haha
接收到的数据为: qie, haha maoxian a
```

<2>网络调试助手：


网络调试助手 (C■精装版 V3.8)
网络设置
(1) 协议类型
TCP Server
(2) 本地IP地址
192.168. 1 .102
(3) 本地端口号
7788
断开
接收区设置
接收转向文件...
自动换行显示
十六进制显示
暂停接收显示
保存数据 清除显示
发送区设置
启用文件数据源...
自动发送附加位
发送完自动清空
按十六进制发送
数据流循环发送
发送间隔 1000 毫秒
文件载入 清除输入
网络数据接收
【Receive from 192.168.1.100 : 61273】:
haha
接收到的数据
注意进行选择
连接对象: All Connections
qie, haha maoxian a
发送的数据
发送
就绪!
发送 : 20
接收 : 4
复位计数

应用：模拟QQ聊天

客户端参考代码

```
#coding=utf-8
from socket import *

# 创建socket
tcpClientSocket = socket(AF_INET, SOCK_STREAM)

# 链接服务器
serAddr = ('192.168.1.102', 7788)
tcpClientSocket.connect(serAddr)

while True:

    # 提示用户输入数据
    sendData = raw_input("send: ")

    if len(sendData)>0:
        tcpClientSocket.send(sendData)
    else:
        break

    # 接收对方发送过来的数据，最大接收1024个字节
    recvData = tcpClientSocket.recv(1024)
    print 'recv:',recvData

# 关闭套接字
tcpClientSocket.close()
```

服务器端参考代码

```
#coding=utf-8
from socket import *

# 创建socket
tcpSerSocket = socket(AF_INET, SOCK_STREAM)

# 绑定本地信息
address = ('', 7788)
tcpSerSocket.bind(address)

# 使用socket创建的套接字默认的属性是主动的，使用listen将其变为被动的，这样
tcpSerSocket.listen(5)

while True:

    # 如果有新的客户端来链接服务器，那么就产生一个信心的套接字专门为这个客户
    # newSocket用来为这个客户端服务
    # tcpSerSocket就可以省下来专门等待其他新客户端的链接
    newSocket, clientAddr = tcpSerSocket.accept()

    while True:

        # 接收对方发送过来的数据，最大接收1024个字节
        recvData = newSocket.recv(1024)

        # 如果接收的数据的长度为0，则意味着客户端关闭了链接
        if len(recvData)>0:
            print 'recv:',recvData
        else:
            break

        # 发送一些数据到客户端
        sendData = raw_input("send:")
        newSocket.send(sendData)

    # 关闭为这个客户端服务的套接字，只要关闭了，就意味着为不能再为这个客户端
    newSocket.close()
```

```
# 关闭监听套接字，只要这个套接字关闭了，就意味着整个程序不能再接收任何新的客
tcpSerSocket.close()
```

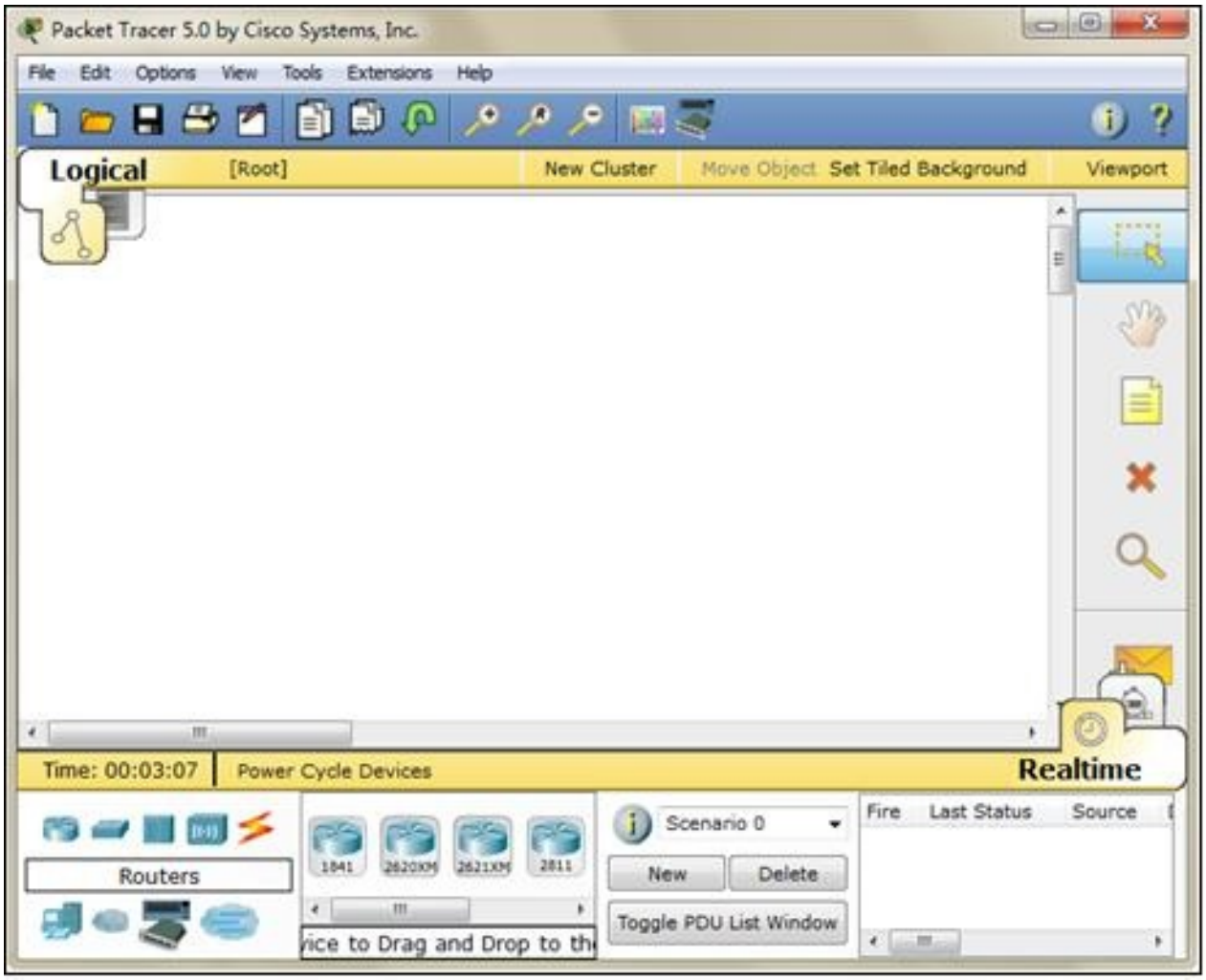
Packet Tracer 介绍&安装

1. Packet Tracer介绍

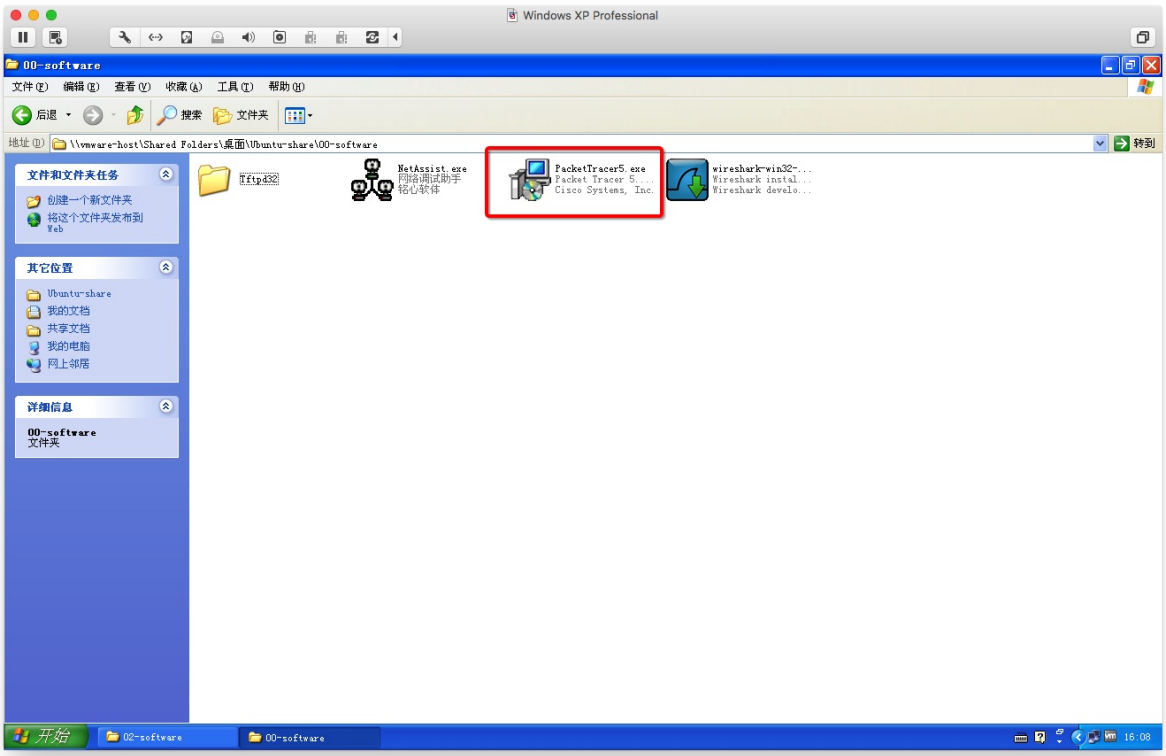
Packet Tracer 是由Cisco(著名网络公司，思科)公司发布的一个辅助学习工具，

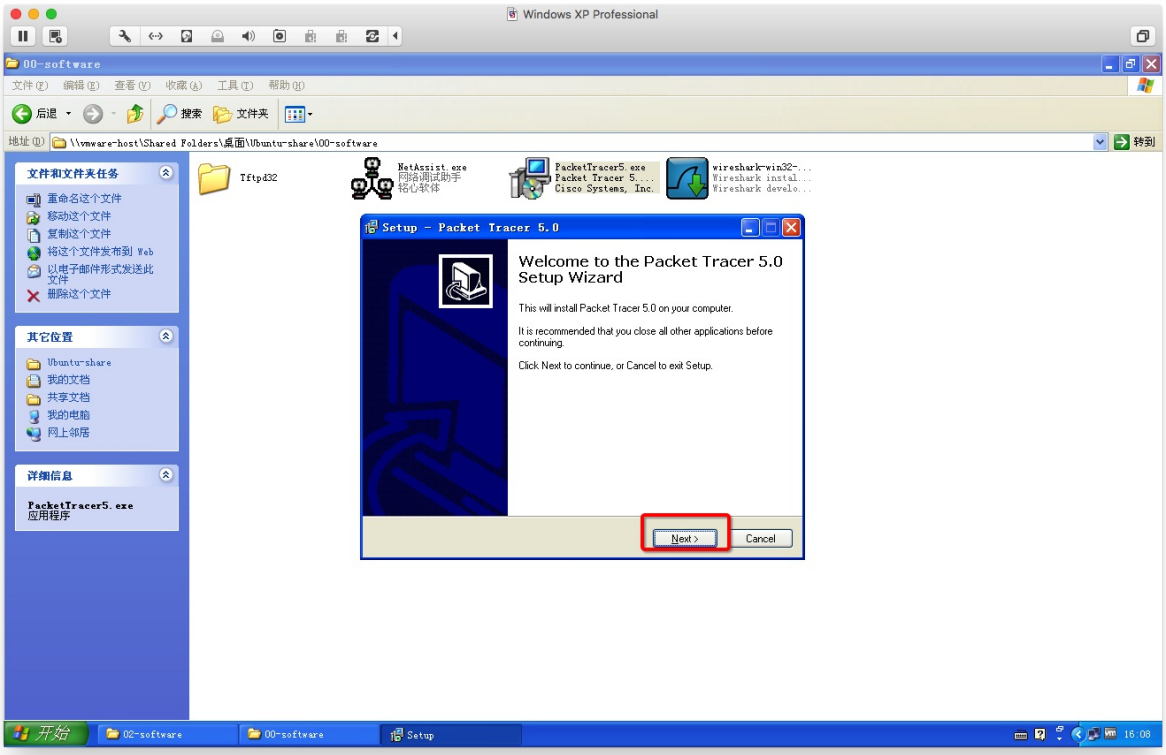
为学习思科网络课程的初学者去设计、配置、排除网络故障提供了网络模拟环境。

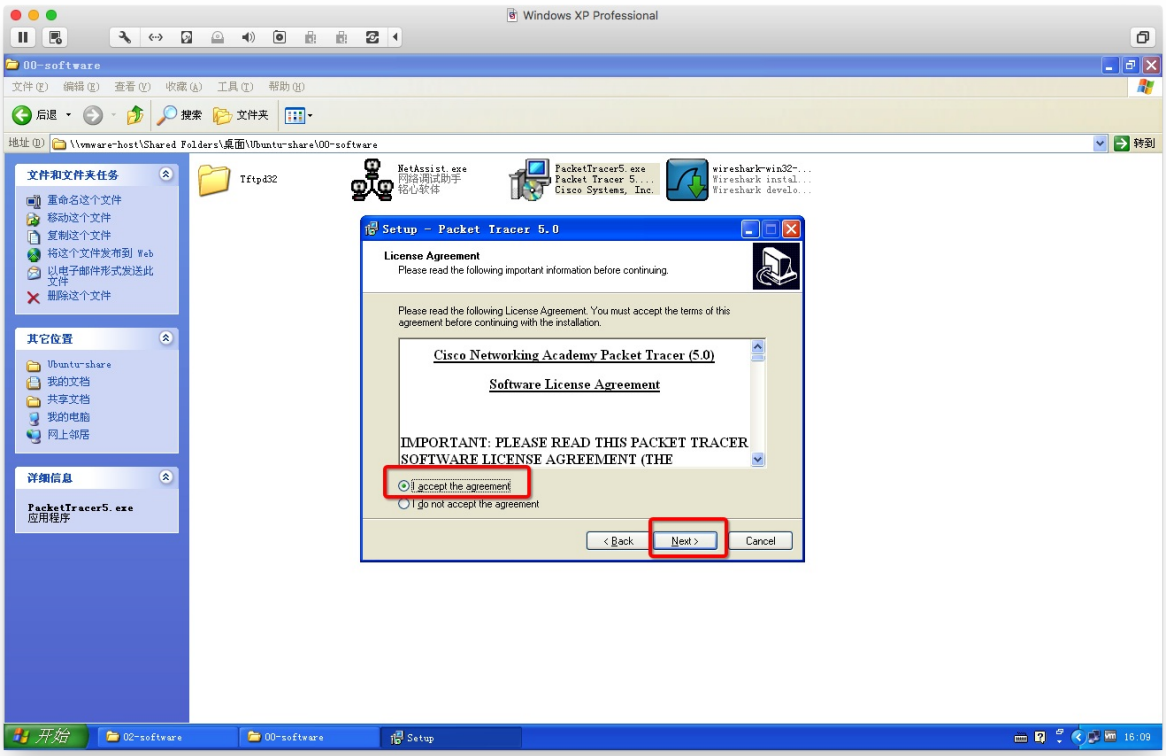
用户可以在软件的图形用户界面上直接使用拖曳方法建立网络拓扑，并可提供数据包在网络中行进的详细处理过程，观察网络实时运行情况。

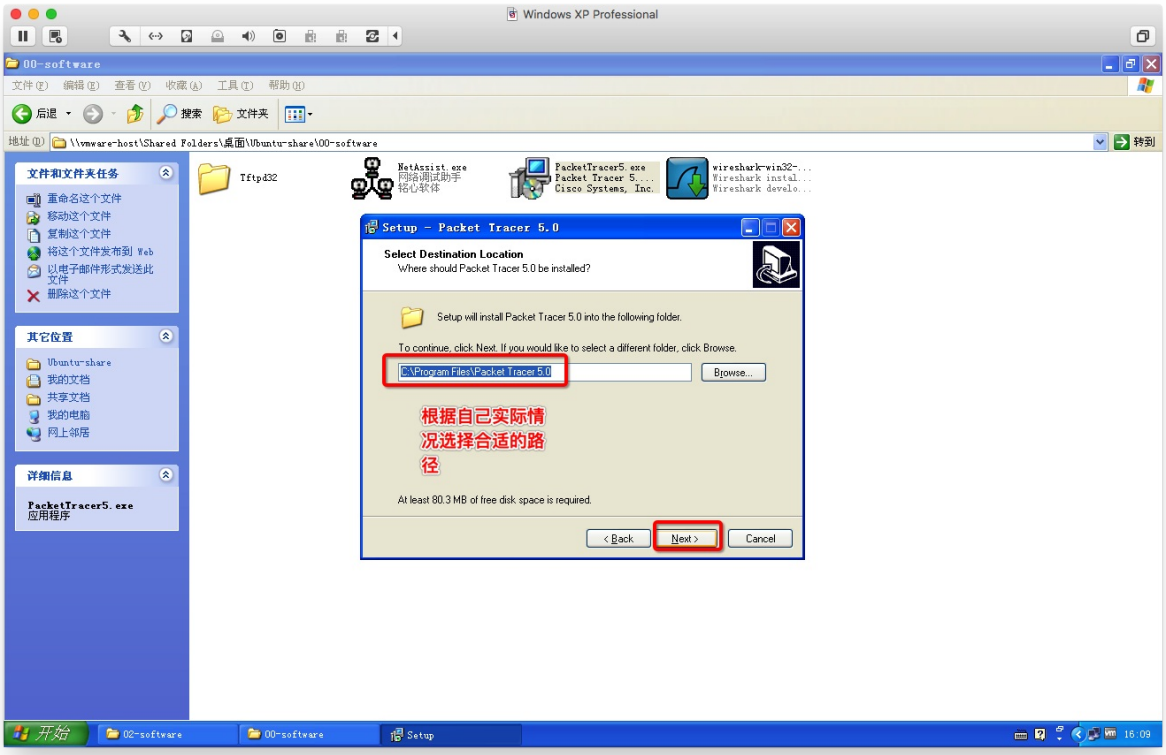


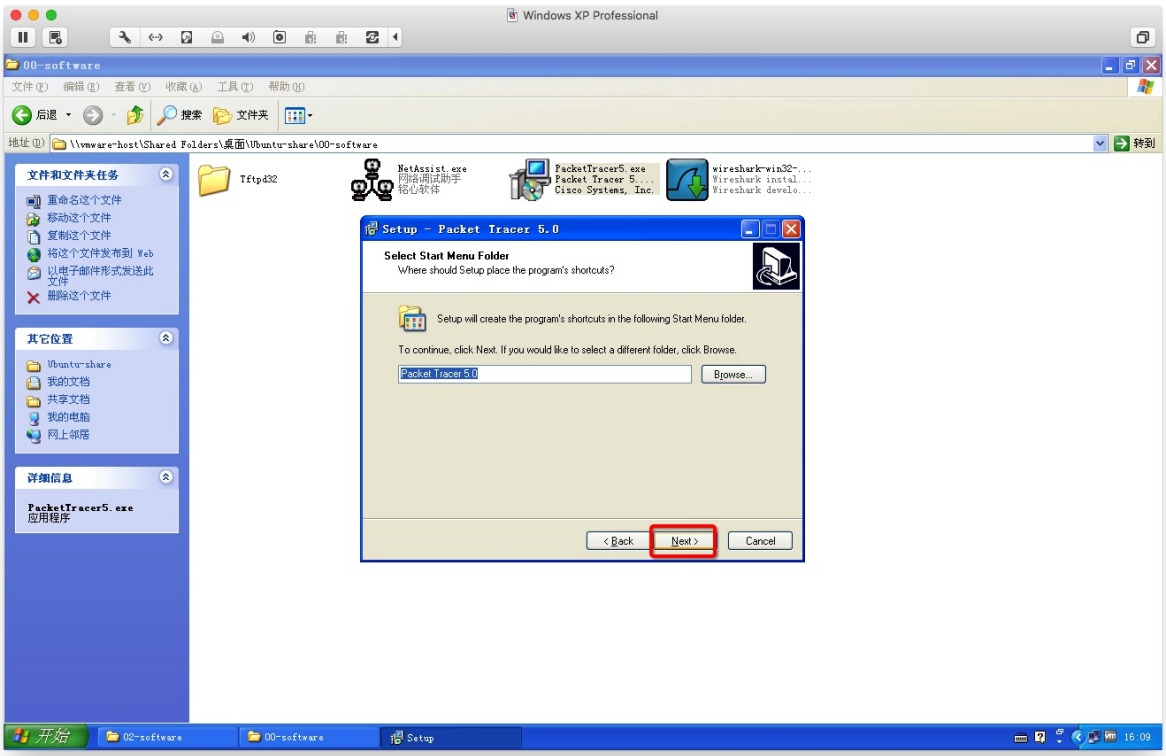
2. 安装

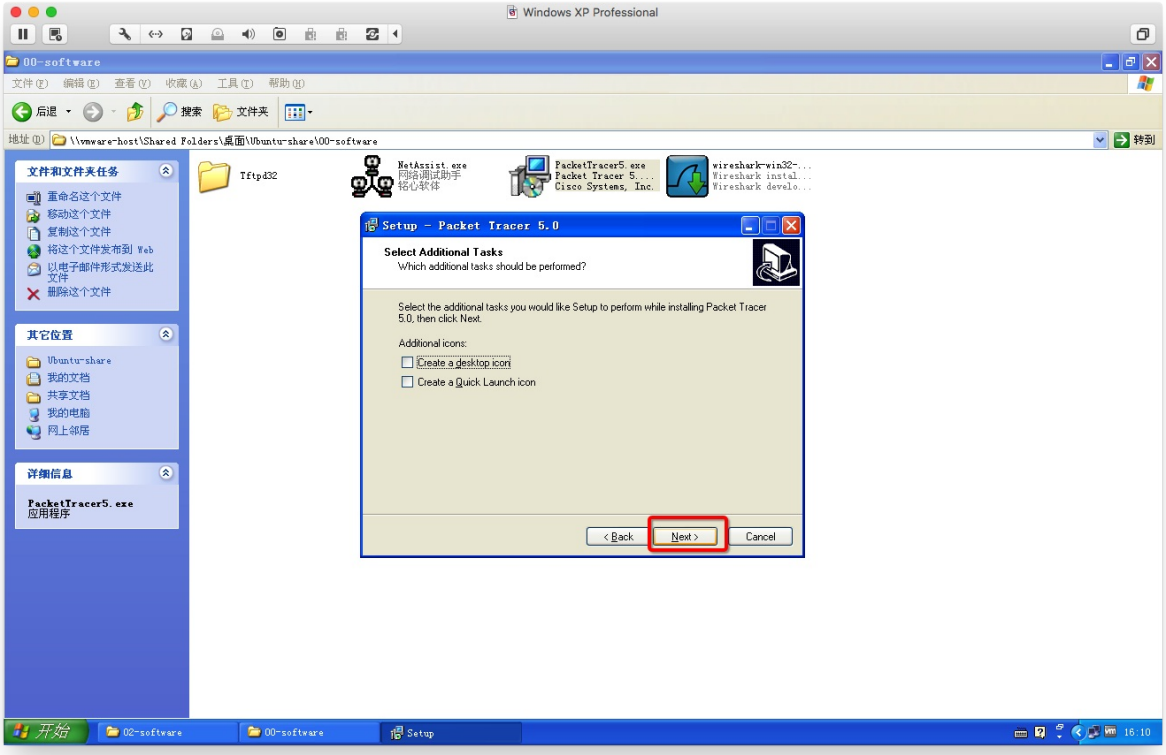

Windows XP Professional
00-software
文件(F) 编辑(E) 查看(V) 收藏(A) 工具(T) 帮助(H)
后退 搜索 文件夹
地址(D) \\vmware-host\Shared Folders\桌面\Ubuntu-share\00-software
转到
文件和文件夹任务
创建一个新文件夹
将这个文件夹发布到 Web
其它位置
Ubuntu-share
我的文档
共享文档
我的电脑
网上邻居
详细信息
00-software
文件夹
Tftpd32
NetAssist.exe
网络调试助手
铭心软体
PacketTracer5.exe
Packet Tracer 5...
Cisco Systems, Inc.
wireshark-win32-...
Wireshark instal...
Wireshark develo...
开始
02-software
00-software
16:08

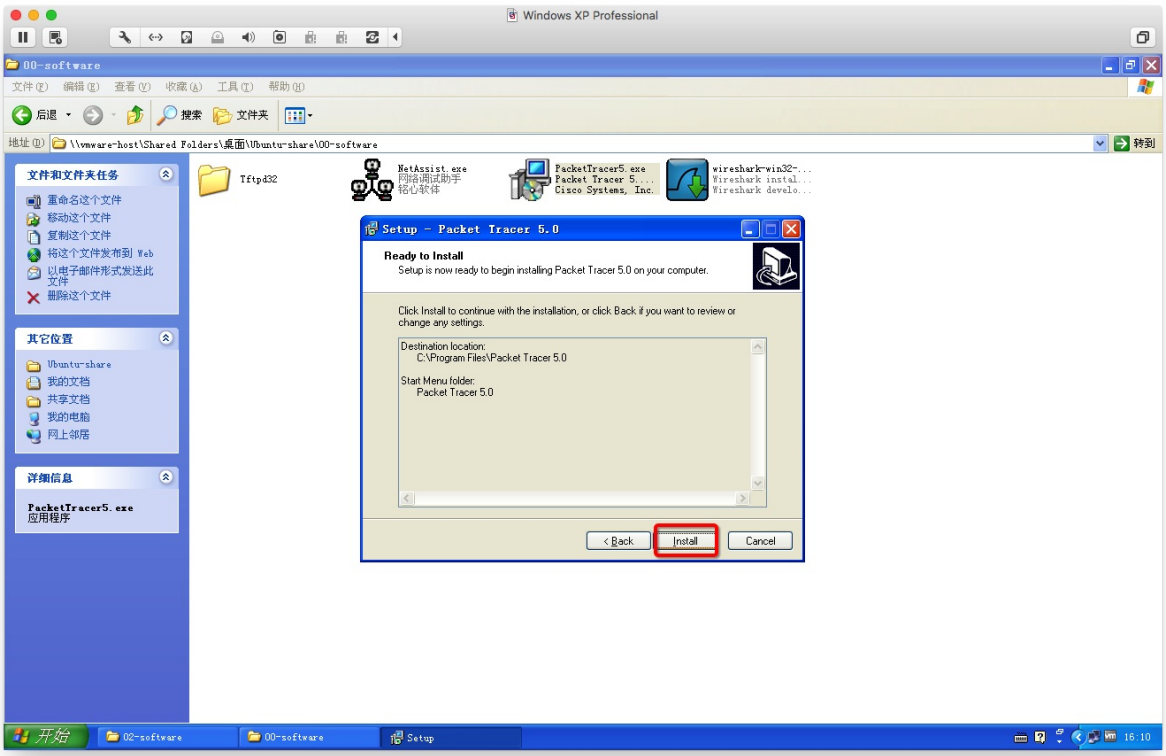

Windows XP Professional
00-software
文件(F) 编辑(E) 查看(V) 收藏(A) 工具(T) 帮助(H)
后退 搜索 文件夹
地址(D) \\vmware-host\Shared Folders\桌面\Ubuntu-share\00-software
转到
文件和文件夹任务
重命名这个文件
移动这个文件
复制这个文件
将这个文件发布到 Web
以电子邮件形式发送此文件
删除这个文件
其它位置
Ubuntu-share
我的文档
共享文档
我的电脑
网上邻居
详细信息
PacketTracer5.exe
应用程序
Tftpd32
NetAssist.exe
网络调试助手
铭心软体
PacketTracer5.exe
Packet Tracer 5...
Cisco Systems, Inc.
wireshark-win32-...
Wireshark instal...
Wireshark develo...
Setup - Packet Tracer 5.0
Welcome to the Packet Tracer 5.0 Setup Wizard
This will install Packet Tracer 5.0 on your computer.
It is recommended that you close all other applications before continuing.
Click Next to continue, or Cancel to exit Setup.
Next >
Cancel
开始
02-software
00-software
Setup
16:08

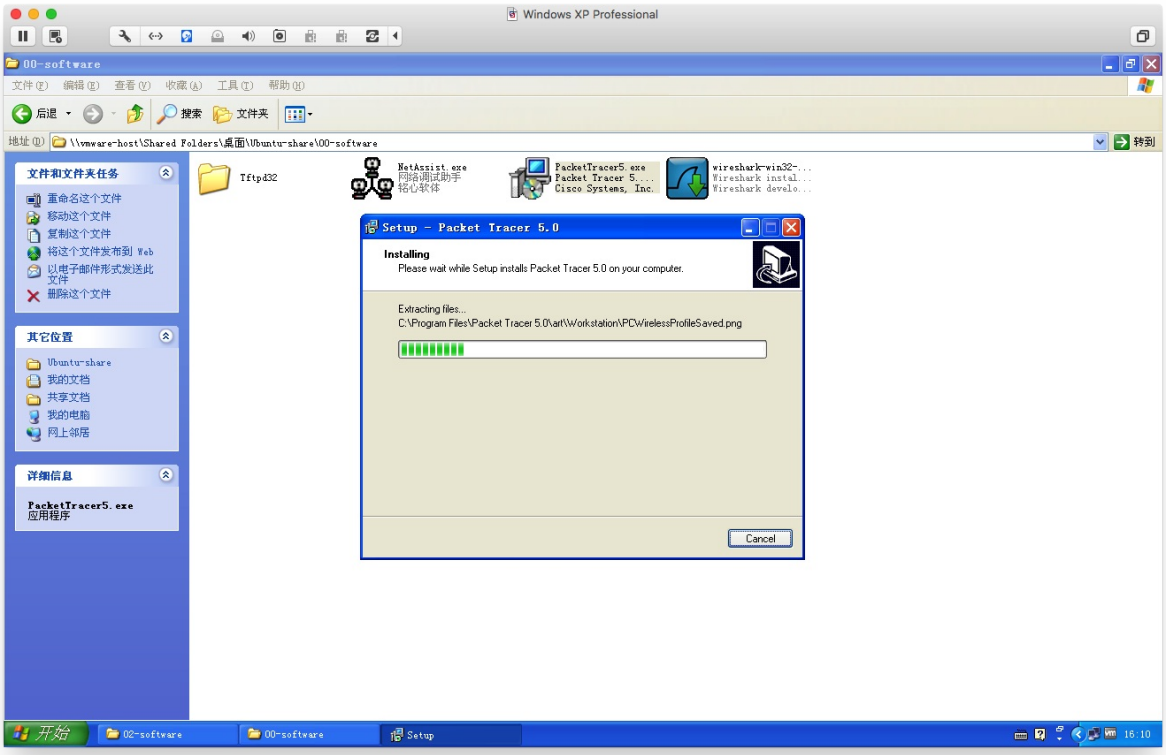

Windows XP Professional
00-software
Tftpd32
NetAssist.exe
PacketTracer5.exe
wireshark-win32-...
Setup - Packet Tracer 5.0
License Agreement
Please read the following important information before continuing.
Please read the following License Agreement. You must accept the terms of this agreement before continuing with the installation.
Cisco Networking Academy Packet Tracer (5.0)
Software License Agreement
IMPORTANT: PLEASE READ THIS PACKET TRACER SOFTWARE LICENSE AGREEMENT (THE
I accept the agreement
I do not accept the agreement
< Back
Next >
Cancel

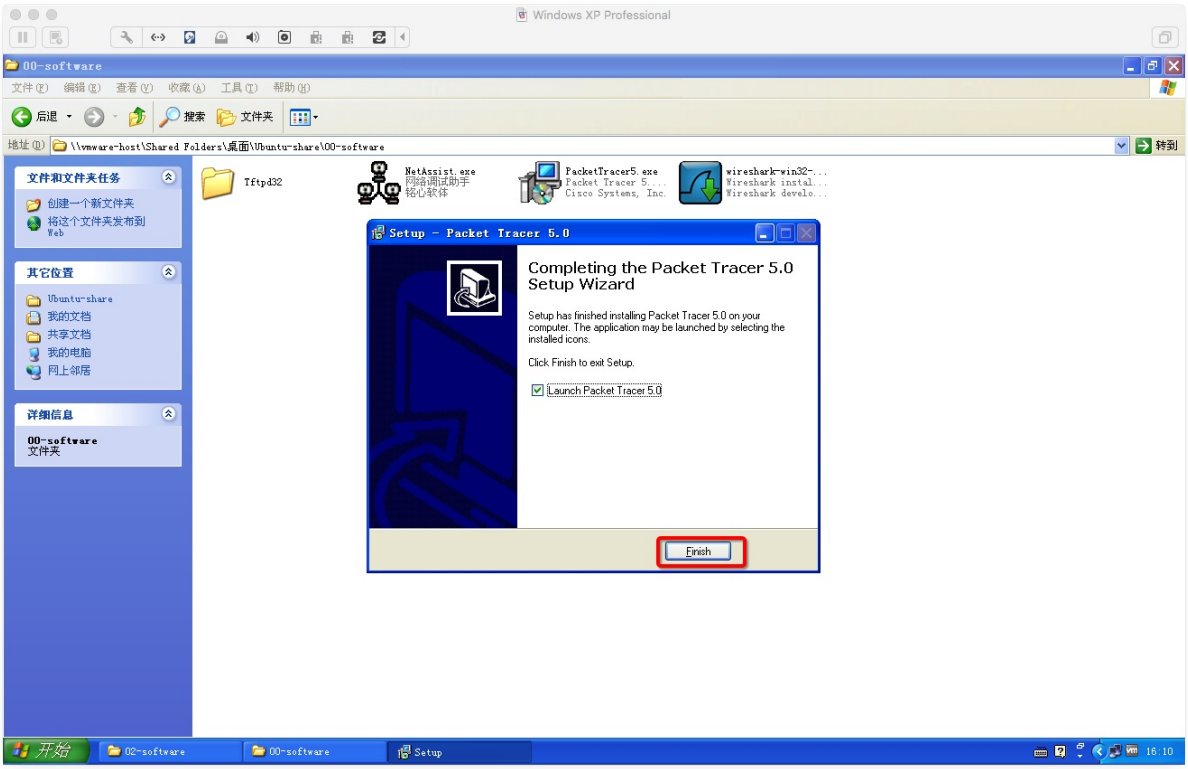

Windows XP Professional
00-software
Tftpd32
NetAssist.exe
PacketTracer5.exe
wireshark-win32-...
Setup - Packet Tracer 5.0
Select Destination Location
Where should Packet Tracer 5.0 be installed?
Setup will install Packet Tracer 5.0 into the following folder.
To continue, click Next. If you would like to select a different folder, click Browse.
C:\Program Files\Packet Tracer 5.0
Browse...
根据自己实际情况选择合适的路径
At least 80.3 MB of free disk space is required.
< Back
Next >
Cancel


Windows XP Professional
00-software
Tftpd32
Setup - Packet Tracer 5.0
Select Start Menu Folder
Where should Setup place the program's shortcuts?
Setup will create the program's shortcuts in the following Start Menu folder.
To continue, click Next. If you would like to select a different folder, click Browse.
Packet Tracer 5.0
Browse...
< Back
Next >
Cancel


Windows XP Professional
00-software
Tftpd32
Setup - Packet Tracer 5.0
Select Additional Tasks
Which additional tasks should be performed?
Select the additional tasks you would like Setup to perform while installing Packet Tracer 5.0, then click Next.
Additional icons:
Create a desktop icon
Create a Quick Launch icon
< Back
Next >
Cancel

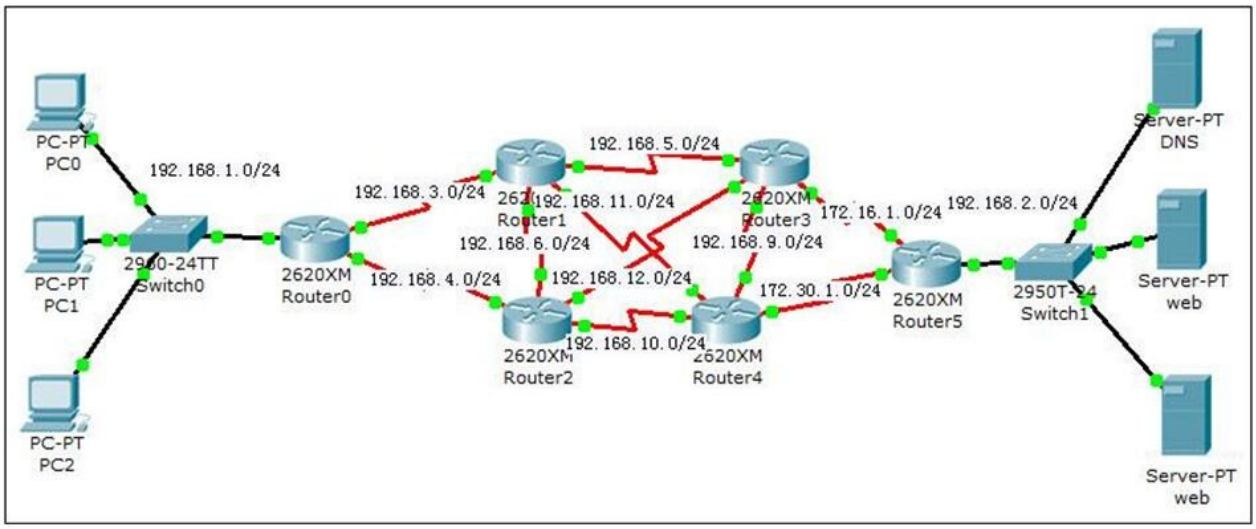

Windows XP Professional
00-software
Tftpd32
NetAssist.exe
PacketTracer5.exe
wireshark-win32-...
Setup - Packet Tracer 5.0
Ready to Install
Setup is now ready to begin installing Packet Tracer 5.0 on your computer.
Click Install to continue with the installation, or click Back if you want to review or change any settings.
Destination location:
C:\Program Files\Packet Tracer 5.0
Start Menu folder:
Packet Tracer 5.0
< Back
Install
Cancel
文件和文件夹任务
其它位置
详细信息
PacketTracer5.exe
应用程序

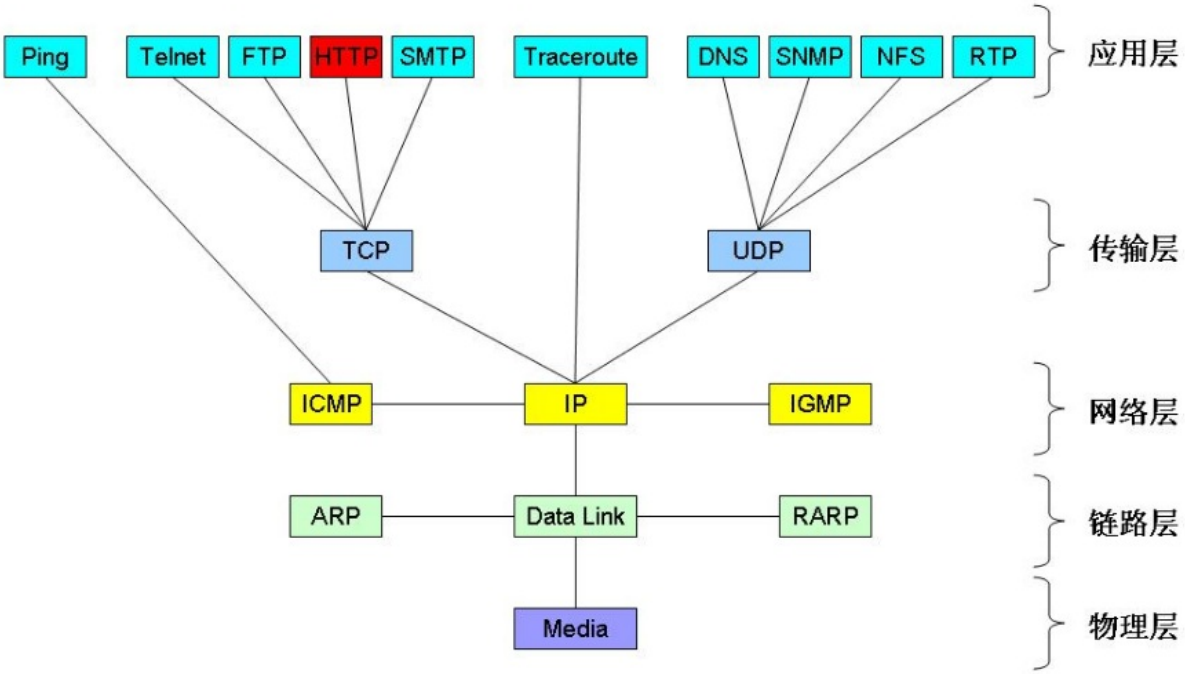

Windows XP Professional
00-software
Tftpd32
NetAssist.exe
PacketTracer5.exe
wireshark-win32-...
Setup - Packet Tracer 5.0
Installing
Please wait while Setup installs Packet Tracer 5.0 on your computer.
Extracting files...
C:\Program Files\Packet Tracer 5.0\art\Workstation\PCWirelessProfileSaved.png
Cancel
文件和文件夹任务
其它位置
详细信息
PacketTracer5.exe
应用程序



学习网通信过程的重要性

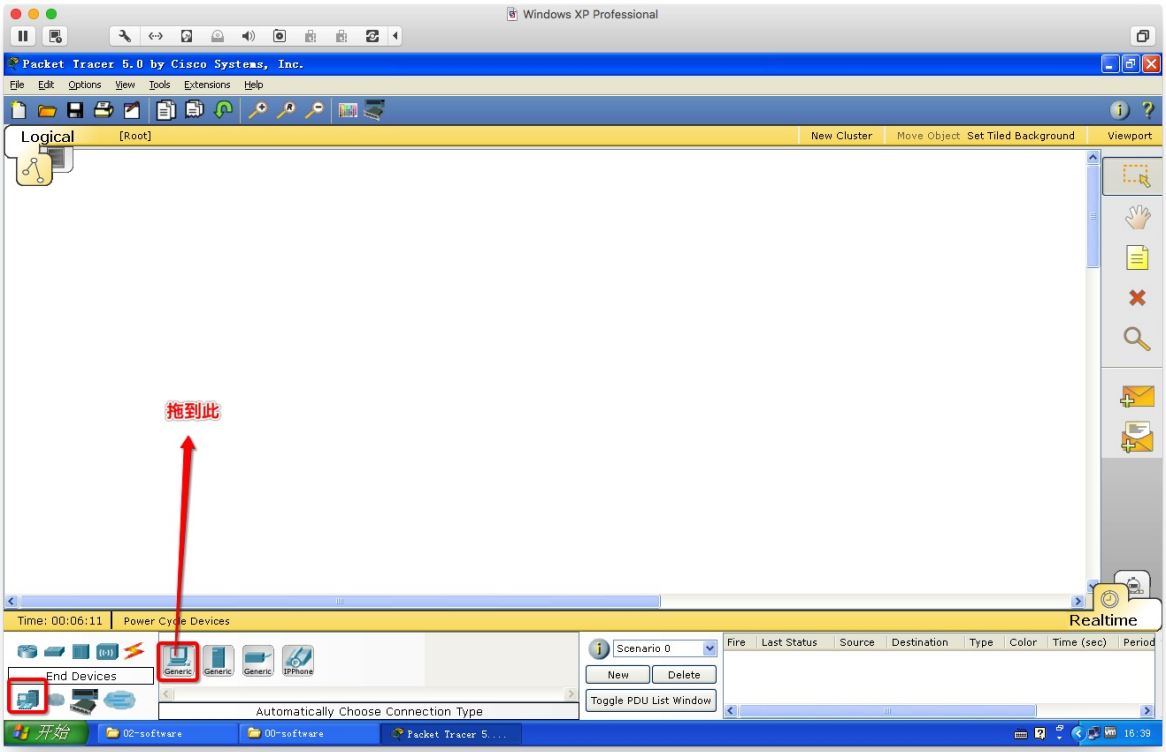
在浏览器中输入 www.itcast.cn后，访问的整个过程有哪些?

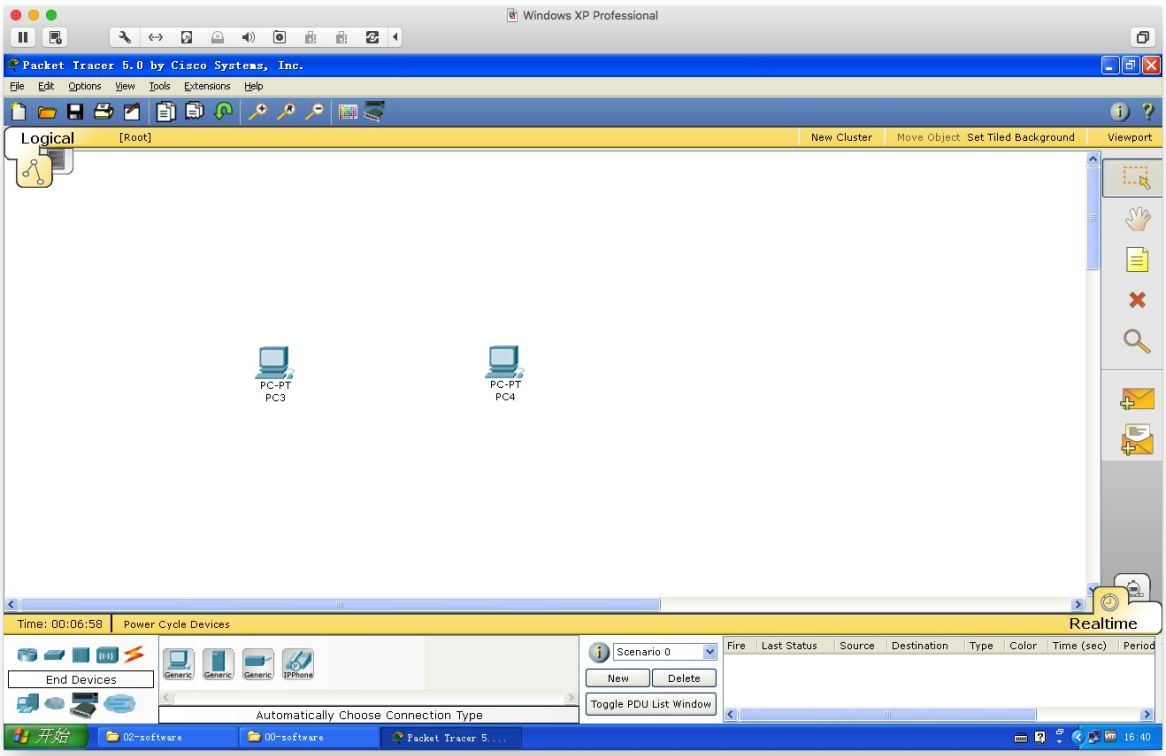


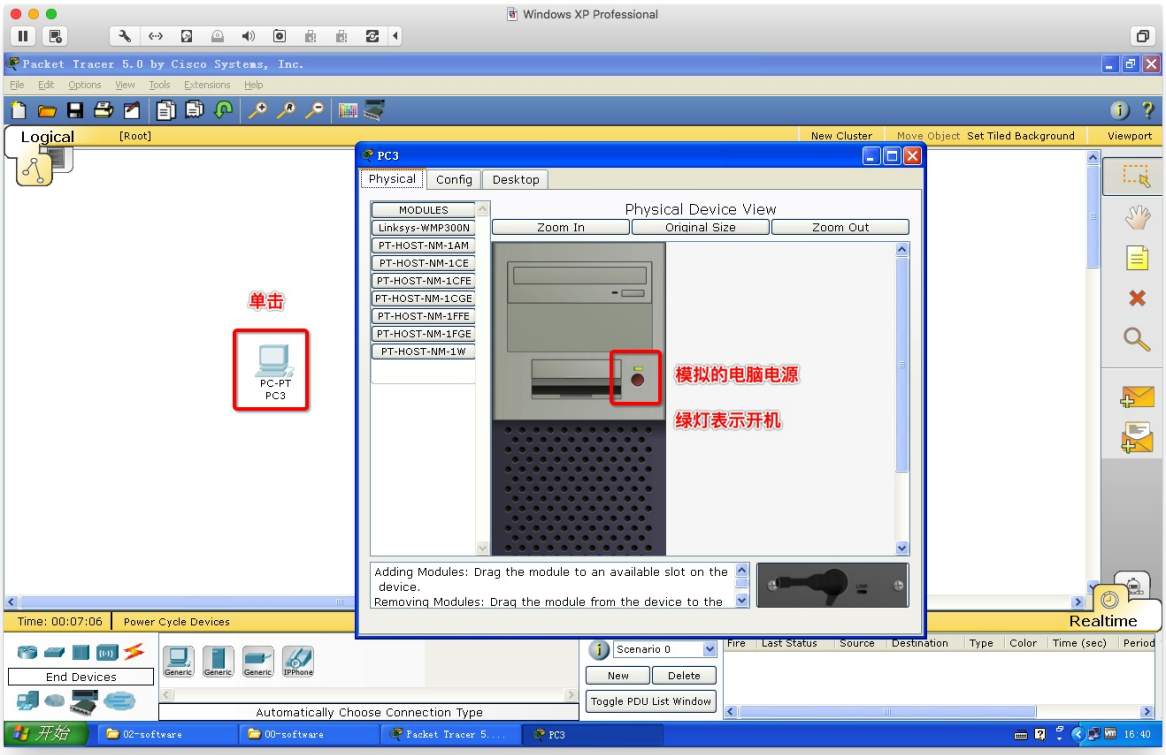

Ping
Telnet
FTP
HTTP
SMTP
Traceroute
DNS
SNMP
NFS
RTP
应用层
TCP
UDP
传输层
ICMP
IP
IGMP
网络层
ARP
Data Link
RARP
链路层
Media
物理层

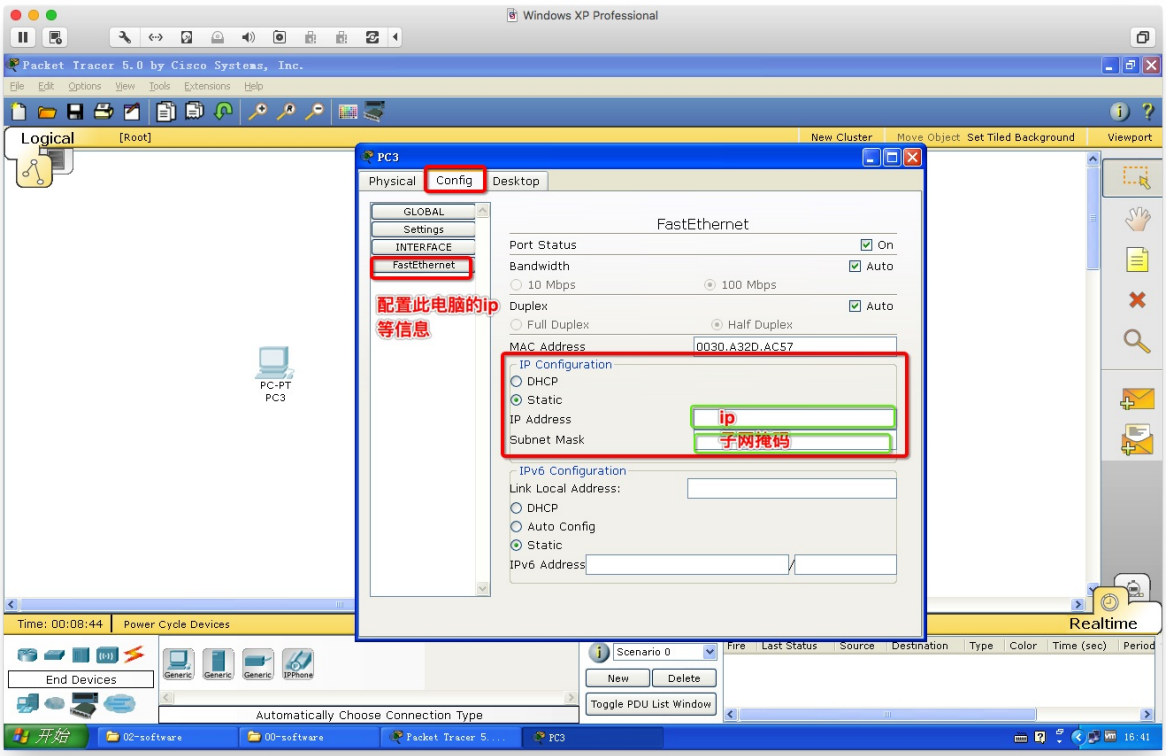
2台电脑组网

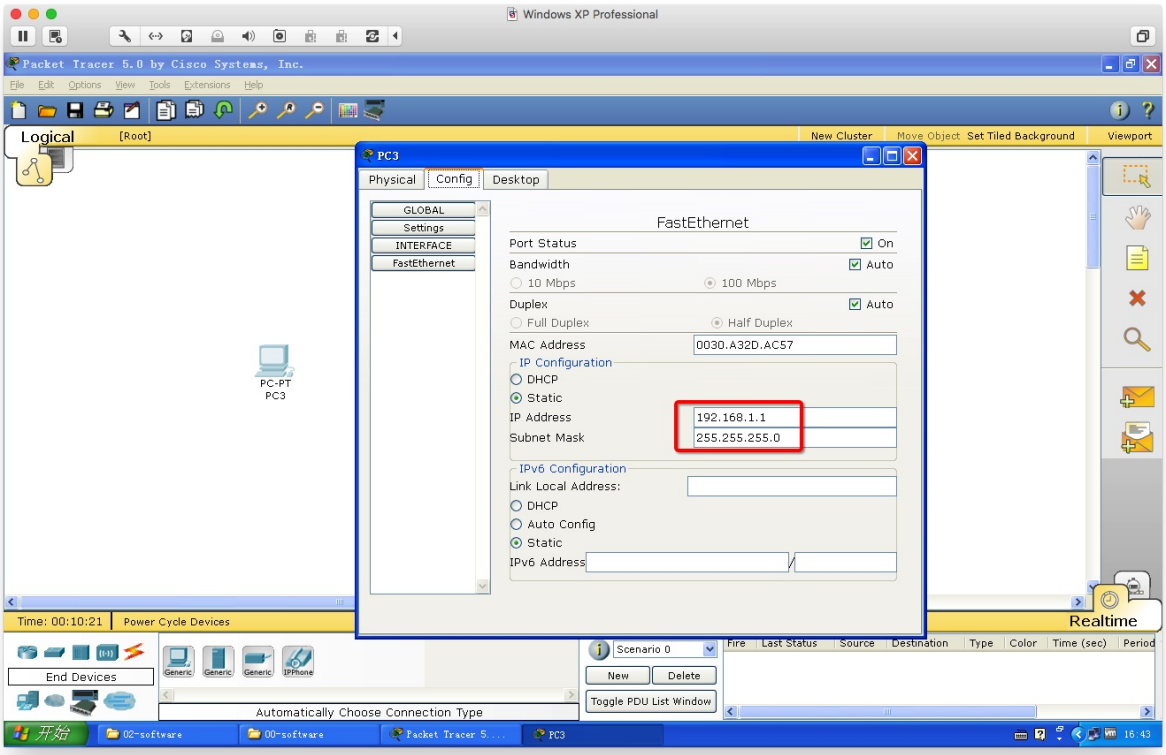
1.在packet tracer中完成如下示图的操作

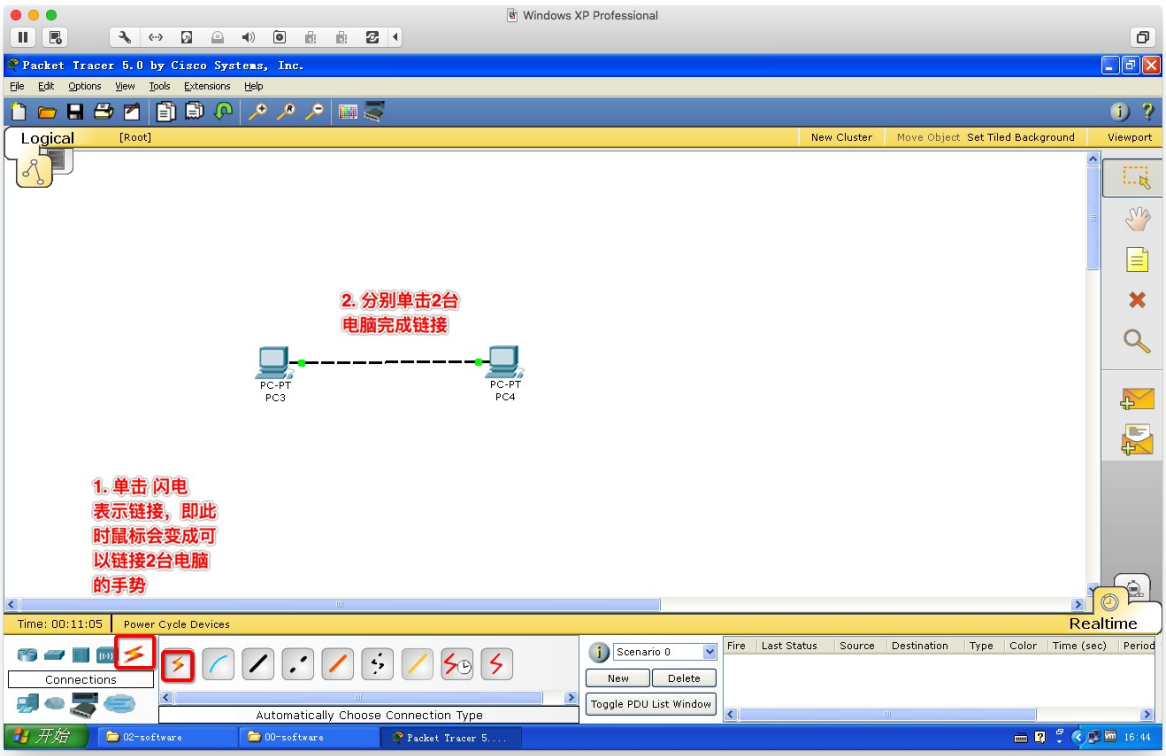


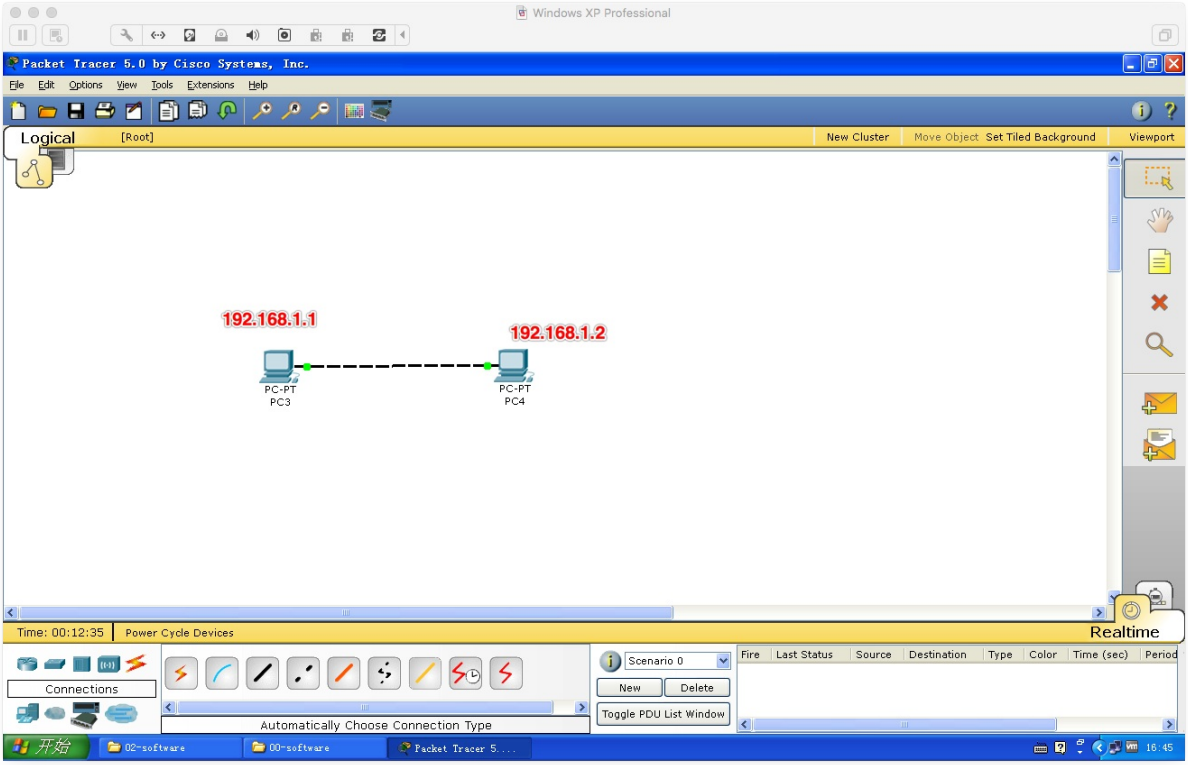

Windows XP Professional
Packet Tracer 5.0 by Cisco Systems, Inc.
File Edit Options View Tools Extensions Help
Logical
[Root]
New Cluster
Move Object
Set Tiled Background
Viewport
PC-PT
PC3
PC-PT
PC4
Time: 00:06:58
Power Cycle Devices
Realtime
End Devices
Automatically Choose Connection Type
Scenario 0
New
Delete
Toggle PDU List Window
Fire
Last Status
Source
Destination
Type
Color
Time (sec)
Period
开始
02-software
00-software
Packet Tracer 5....
16:40

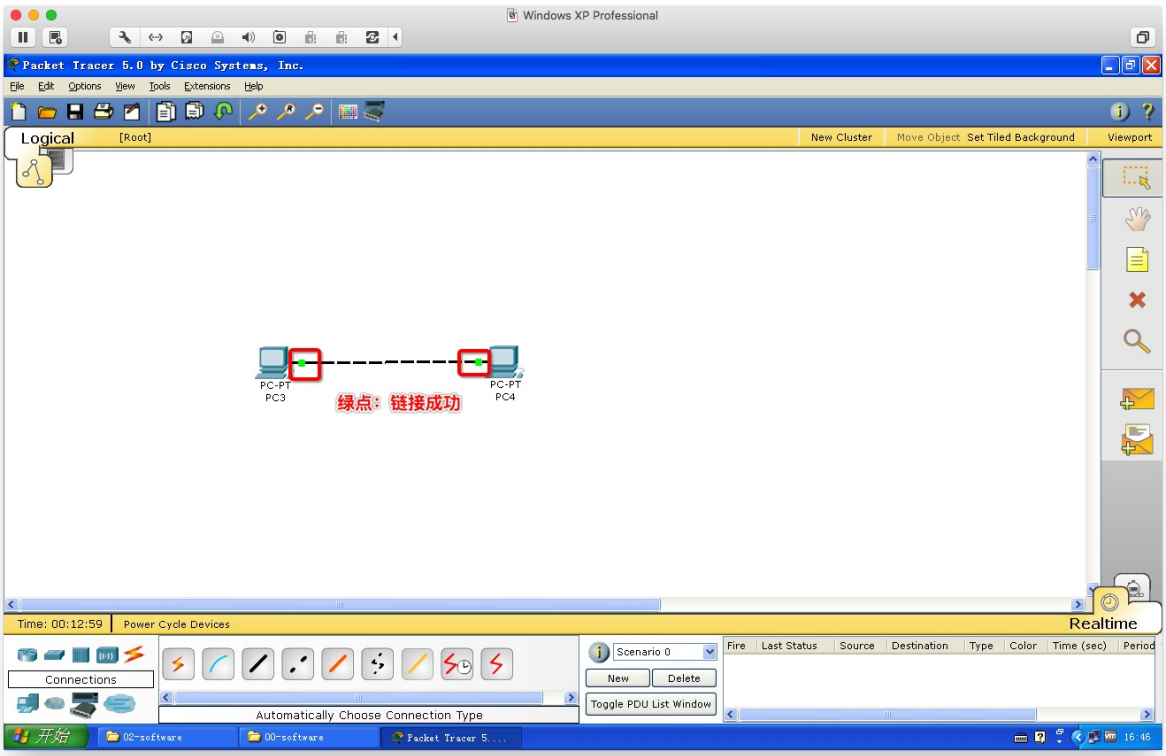

Windows XP Professional
Packet Tracer 5.0 by Cisco Systems, Inc.
Logical
[Root]
New Cluster
Move Object
Set Tiled Background
Viewport
单击
PC-PT
PC3
PC3
Physical
Config
Desktop
MODULES
Linksys-WMP300N
PT-HOST-NM-1AM
PT-HOST-NM-1CE
PT-HOST-NM-1CFE
PT-HOST-NM-1CGE
PT-HOST-NM-1FFE
PT-HOST-NM-1FGE
PT-HOST-NM-1W
Physical Device View
Zoom In
Original Size
Zoom Out
模拟的电脑电源
绿灯表示开机
Adding Modules: Drag the module to an available slot on the device.
Removing Modules: Drag the module from the device to the
Time: 00:07:06
Power Cycle Devices
Realtime
End Devices
Automatically Choose Connection Type
Scenario 0
New
Delete
Toggle PDU List Window
Fire
Last Status
Source
Destination
Type
Color
Time (sec)
Period
开始
02-software
00-software
Packet Tracer 5....
PC3
16:40

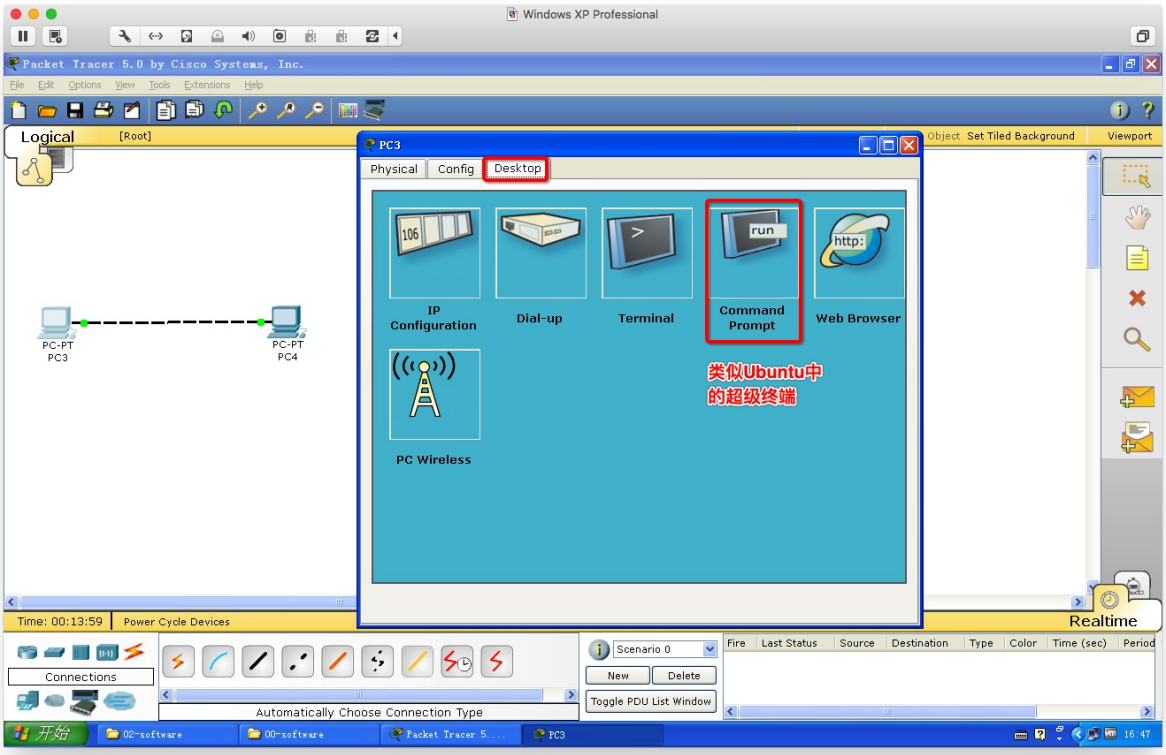

Windows XP Professional
Packet Tracer 5.0 by Cisco Systems, Inc.
Logical
[Root]
PC3
Physical
Config
Desktop
GLOBAL
Settings
INTERFACE
FastEthernet
配置此电脑的ip
等信息
FastEthernet
Port Status
On
Bandwidth
Auto
10 Mbps
100 Mbps
Duplex
Auto
Full Duplex
Half Duplex
MAC Address
0030.A32D.AC57
IP Configuration
DHCP
Static
IP Address
ip
Subnet Mask
子网掩码
IPv6 Configuration
Link Local Address:
DHCP
Auto Config
Static
IPv6 Address
PC-PT
PC3
Time: 00:08:44
Power Cycle Devices
Realtime
End Devices
Automatically Choose Connection Type
Scenario 0
New
Delete
Toggle PDU List Window
Fire
Last Status
Source
Destination
Type
Color
Time (sec)
Period

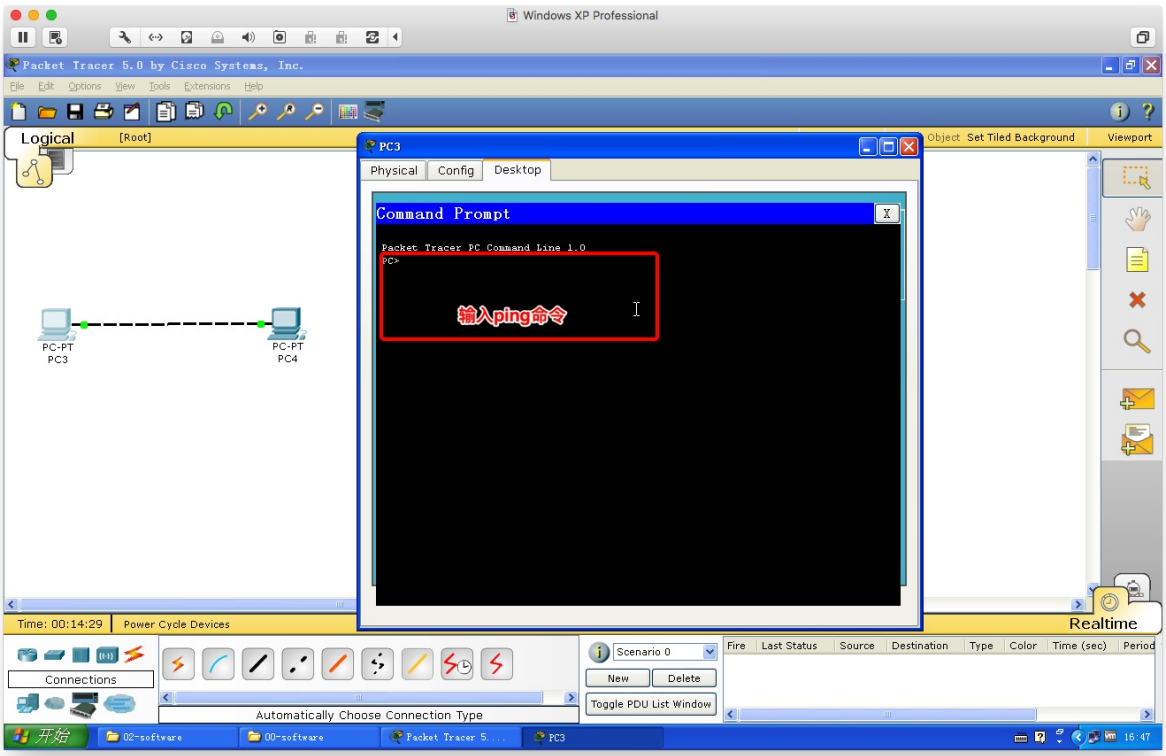

Windows XP Professional
Packet Tracer 5.0 by Cisco Systems, Inc.
Logical
[Root]
PC3
Physical
Config
Desktop
GLOBAL
Settings
INTERFACE
FastEthernet
FastEthernet
Port Status
On
Bandwidth
Auto
10 Mbps
100 Mbps
Duplex
Auto
Full Duplex
Half Duplex
MAC Address
0030.A32D.AC57
IP Configuration
DHCP
Static
IP Address
192.168.1.1
Subnet Mask
255.255.255.0
IPv6 Configuration
Link Local Address:
DHCP
Auto Config
Static
IPv6 Address
PC-PT
PC3
Time: 00:10:21
Power Cycle Devices
Realtime
End Devices
Automatically Choose Connection Type
Scenario 0
New
Delete
Toggle PDU List Window
Fire
Last Status
Source
Destination
Type
Color
Time (sec)
Period

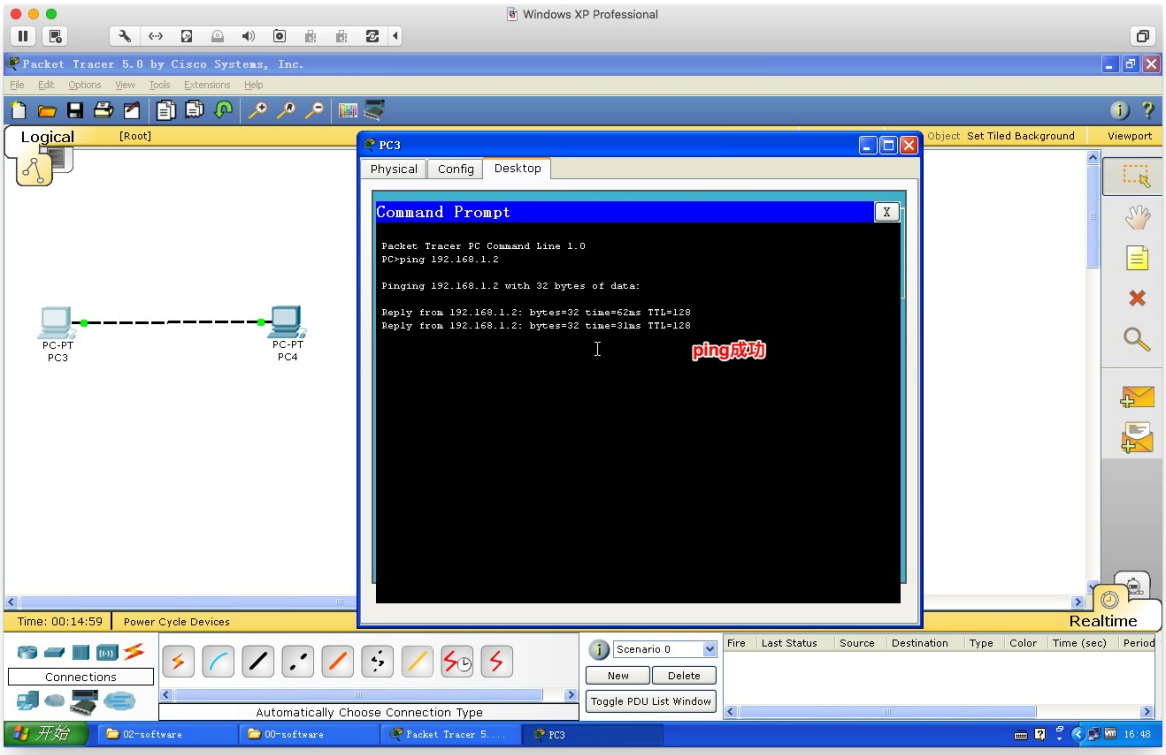

Windows XP Professional
Packet Tracer 5.0 by Cisco Systems, Inc.
File Edit Options View Tools Extensions Help
Logical
[Root]
New Cluster
Move Object
Set Tiled Background
Viewport
2. 分别单击2台
电脑完成链接
PC-PT
PC3
PC-PT
PC4
1. 单击 闪电
表示链接，即此
时鼠标会变成可
以链接2台电脑
的手势
Time: 00:11:05
Power Cycle Devices
Realtime
Connections
Automatically Choose Connection Type
Scenario 0
New
Delete
Toggle PDU List Window
Fire
Last Status
Source
Destination
Type
Color
Time (sec)
Period
开始
02-software
00-software
Packet Tracer 5....
16:44

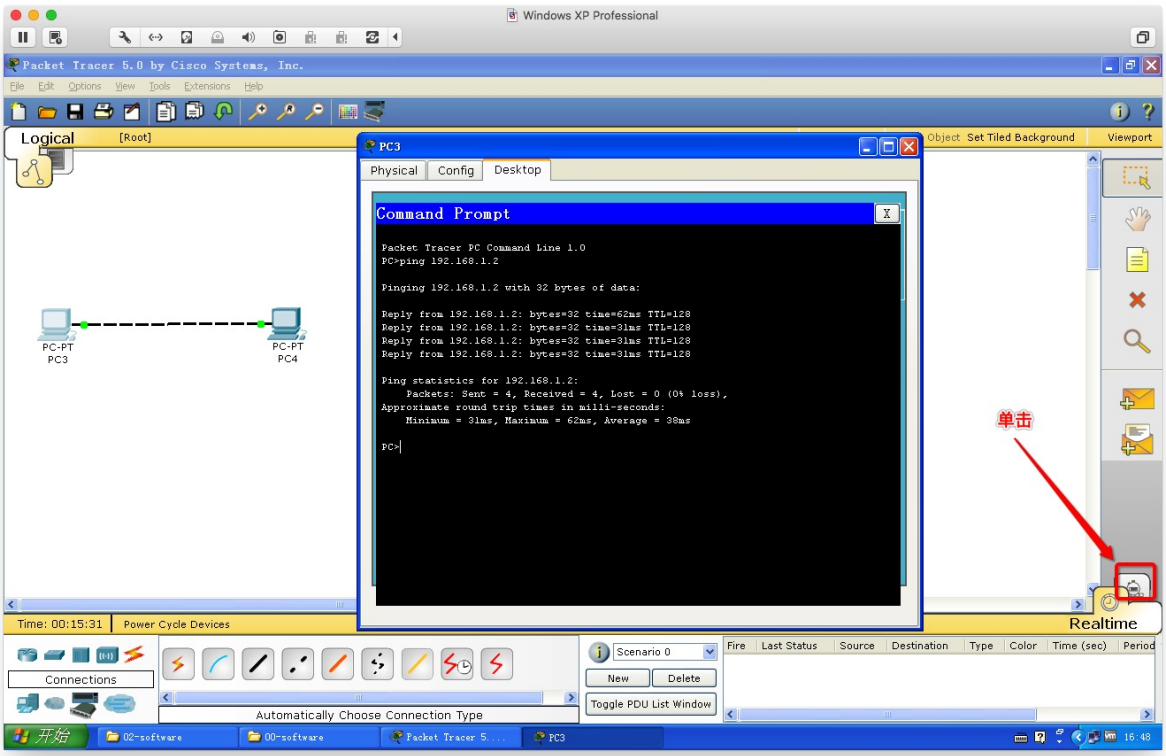

Windows XP Professional
Packet Tracer 5.0 by Cisco Systems, Inc.
File Edit Options View Tools Extensions Help
Logical
[Root]
New Cluster
Move Object
Set Tiled Background
Viewport
192.168.1.1
192.168.1.2
PC-PT
PC3
PC-PT
PC4
Time: 00:12:35
Power Cycle Devices
Realtime
Connections
Automatically Choose Connection Type
Scenario 0
New
Delete
Toggle PDU List Window
Fire
Last Status
Source
Destination
Type
Color
Time (sec)
Period
开始
02-software
00-software
Packet Tracer 5....
16:45

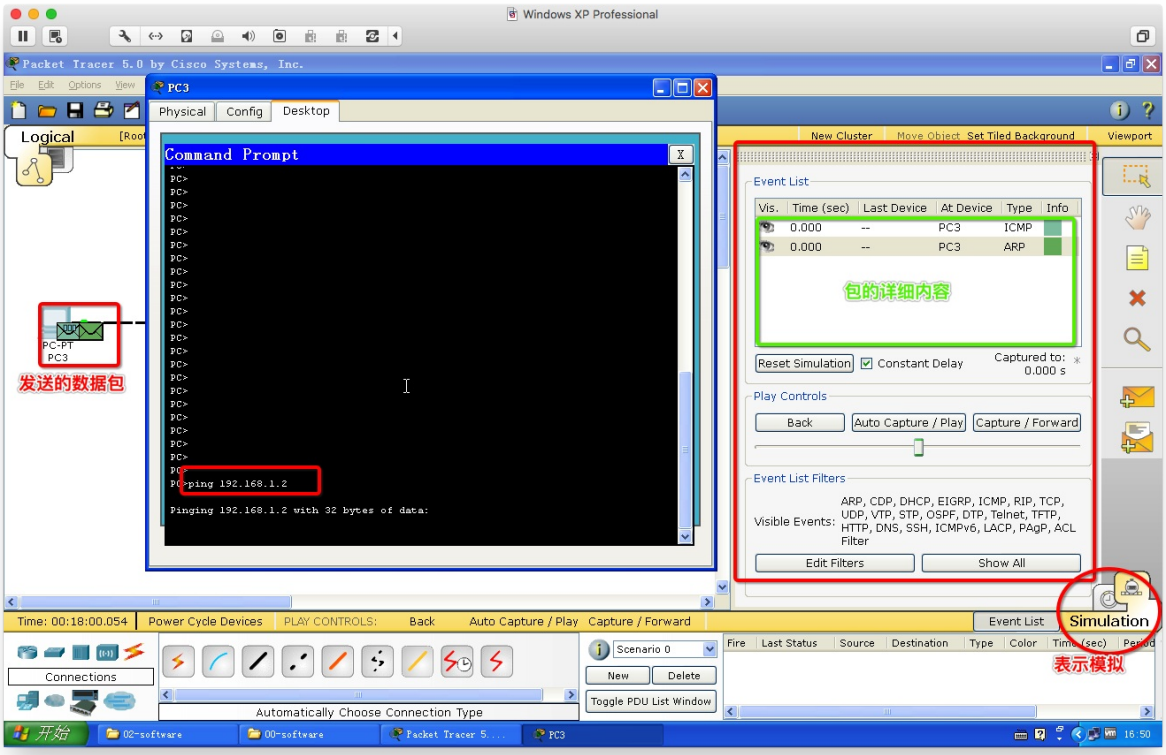

Windows XP Professional
Packet Tracer 5.0 by Cisco Systems, Inc.
File Edit Options View Tools Extensions Help
Logical
[Root]
New Cluster
Move Object
Set Tiled Background
Viewport
PC-PT
PC3
PC-PT
PC4
绿点：链接成功
Time: 00:12:59
Power Cycle Devices
Realtime
Connections
Automatically Choose Connection Type
Scenario 0
New
Delete
Toggle PDU List Window
Fire
Last Status
Source
Destination
Type
Color
Time (sec)
Period

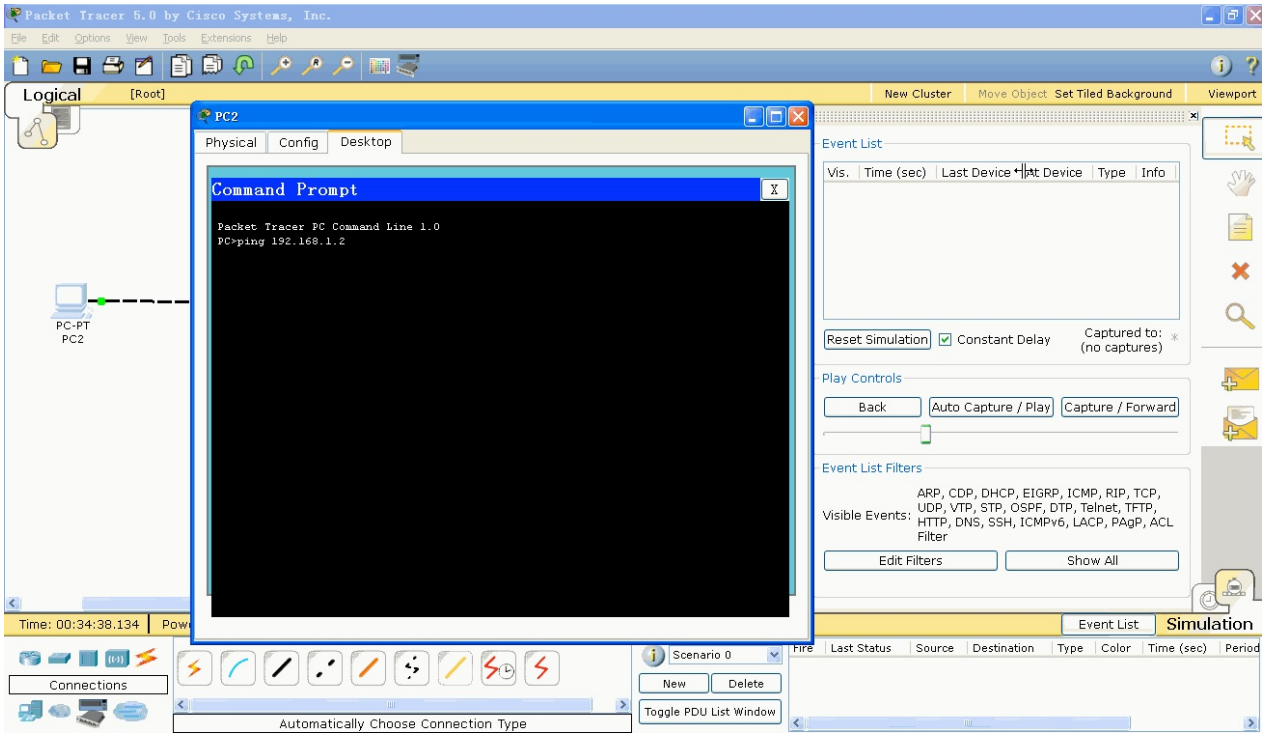

Windows XP Professional
Packet Tracer 5.0 by Cisco Systems, Inc.
Logical
[Root]
PC3
Physical
Config
Desktop
IP Configuration
Dial-up
Terminal
Command Prompt
Web Browser
PC Wireless
类似Ubuntu中的超级终端
PC-PT
PC3
PC-PT
PC4
Time: 00:13:59
Power Cycle Devices
Realtime
Connections
Automatically Choose Connection Type
Scenario 0
New
Delete
Toggle PDU List Window


Windows XP Professional
Packet Tracer 5.0 by Cisco Systems, Inc.
Logical
[Root]
PC3
Physical
Config
Desktop
Command Prompt
Packet Tracer PC Command Line 1.0
PC>
输入ping命令
PC-PT PC3
PC-PT PC4
Time: 00:14:29
Power Cycle Devices
Realtime
Connections
Automatically Choose Connection Type
Scenario 0
New
Delete
Toggle PDU List Window
Fire
Last Status
Source
Destination
Type
Color
Time (sec)
Period


Windows XP Professional
Packet Tracer 5.0 by Cisco Systems, Inc.
Logical
[Root]
PC3
Physical
Config
Desktop
Command Prompt
Packet Tracer PC Command Line 1.0
PC>ping 192.168.1.2
Pinging 192.168.1.2 with 32 bytes of data:
Reply from 192.168.1.2: bytes=32 time=62ms TTL=128
Reply from 192.168.1.2: bytes=32 time=31ms TTL=128
ping成功
PC-PT PC3
PC-PT PC4
Time: 00:14:59
Power Cycle Devices
Realtime
Connections
Automatically Choose Connection Type
Scenario 0
New
Delete
Toggle PDU List Window
Fire
Last Status
Source
Destination
Type
Color
Time (sec)
Period


Windows XP Professional
Packet Tracer 5.0 by Cisco Systems, Inc.
Logical
PC3
Physical
Config
Desktop
Command Prompt
Packet Tracer PC Command Line 1.0
PC>ping 192.168.1.2
Pinging 192.168.1.2 with 32 bytes of data:
Reply from 192.168.1.2: bytes=32 time=62ms TTL=128
Reply from 192.168.1.2: bytes=32 time=31ms TTL=128
Reply from 192.168.1.2: bytes=32 time=31ms TTL=128
Reply from 192.168.1.2: bytes=32 time=31ms TTL=128
Ping statistics for 192.168.1.2:
Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
Minimum = 31ms, Maximum = 62ms, Average = 38ms
PC>
PC-PT PC3
PC-PT PC4
单击
Realtime
Time: 00:15:31
Power Cycle Devices
Connections
Automatically Choose Connection Type
Scenario 0
New
Delete
Toggle PDU List Window

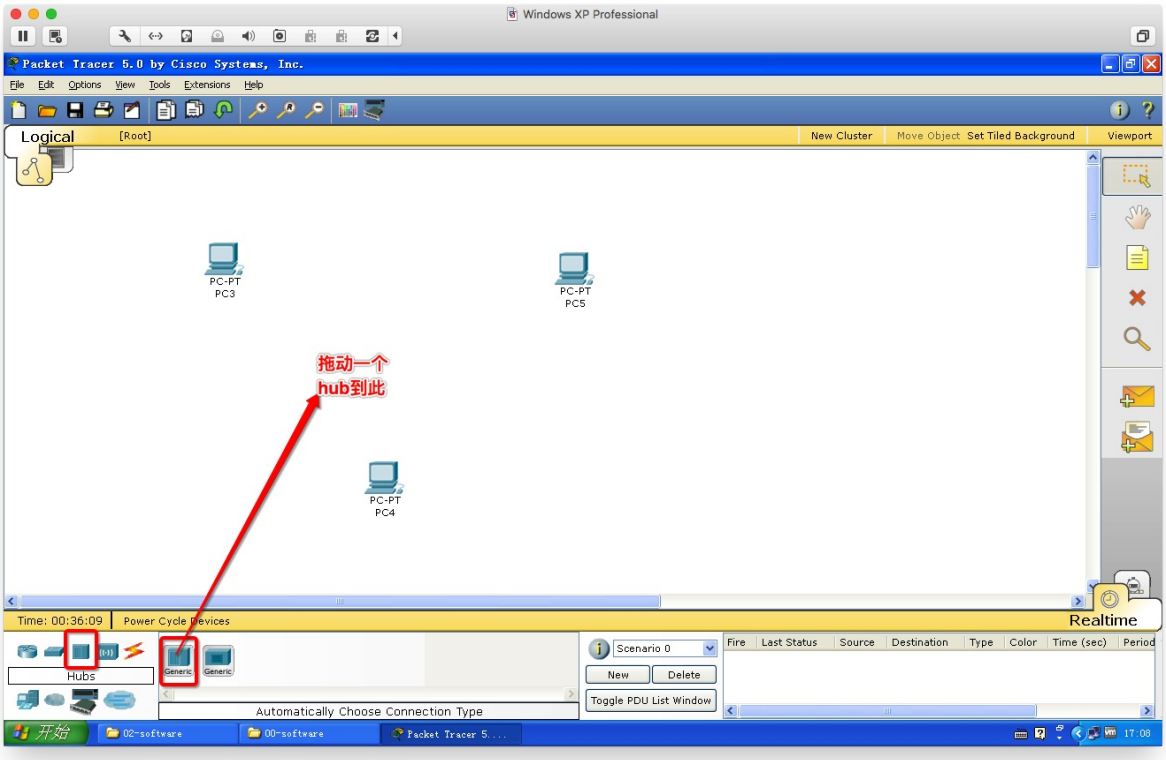

Windows XP Professional
Packet Tracer 5.0 by Cisco Systems, Inc.
PC3
Physical
Config
Desktop
Command Prompt
PC>ping 192.168.1.2
Pinging 192.168.1.2 with 32 bytes of data:
Logical
PC-PT PC3
发送的数据包
Event List
Vis. Time (sec) Last Device At Device Type Info
0.000 -- PC3 ICMP
0.000 -- PC3 ARP
包的详细内容
Reset Simulation
Constant Delay
Captured to: 0.000 s
Play Controls
Back
Auto Capture / Play
Capture / Forward
Event List Filters
Visible Events: ARP, CDP, DHCP, EIGRP, ICMP, RIP, TCP, UDP, VTP, STP, OSPF, DTP, Telnet, TFTP, HTTP, DNS, SSH, ICMPv6, LACP, PAgP, ACL Filter
Edit Filters
Show All
Simulation
表示模拟
Time: 00:18:00.054
Power Cycle Devices
PLAY CONTROLS:
Back
Auto Capture / Play
Capture / Forward
Event List
Connections
Automatically Choose Connection Type
Scenario 0
New
Delete
Toggle PDU List Window

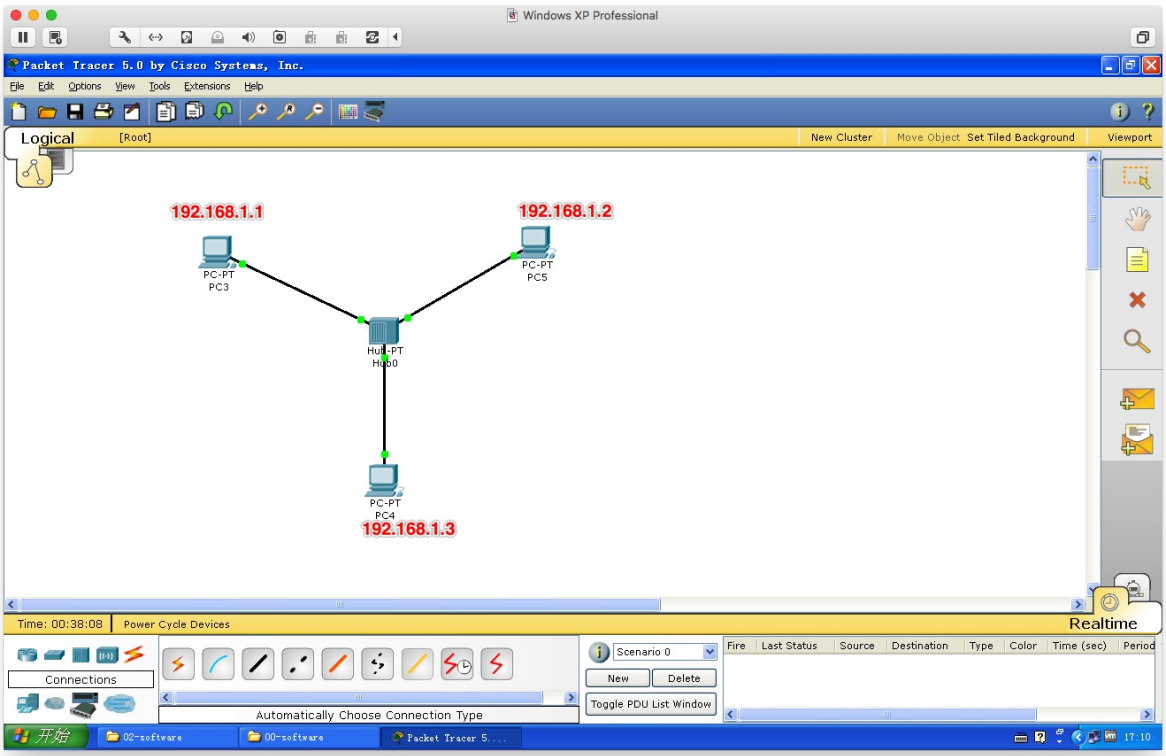

Packet Tracer 5.0 by Cisco Systems, Inc.
File Edit Options View Tools Extensions Help
Logical
[Root]
New Cluster
Move Object
Set Tiled Background
Viewport
PC2
Physical
Config
Desktop
Command Prompt
Packet Tracer PC Command Line 1.0
PC>ping 192.168.1.2
PC-PT
PC2
Event List
Vis.
Time (sec)
Last Device
At Device
Type
Info
Reset Simulation
Constant Delay
Captured to:
(no captures)
Play Controls
Back
Auto Capture / Play
Capture / Forward
Event List Filters
Visible Events:
ARP, CDP, DHCP, EIGRP, ICMP, RIP, TCP, UDP, VTP, STP, OSPF, DTP, Telnet, TFTP, HTTP, DNS, SSH, ICMPv6, LACP, PAgP, ACL Filter
Edit Filters
Show All
Time: 00:34:38.134
Event List
Simulation
Fire
Last Status
Source
Destination
Type
Color
Time (sec)
Period
Connections
Scenario 0
New
Delete
Toggle PDU List Window
Automatically Choose Connection Type

通过集线器组网

集线器又名hub

1. 通过hub链接网络





2. 总结

- hub（集线器）能够完成多个电脑的链接
- 每个数据包的发送都是以广播的形式进行的，容易堵塞网络

通过交换机组网

1. 网络交换机介绍

网络交换机（又称“网络交换器”），是一个扩大网络的器材，能为子网络中提供更多的连接端口，以便连接更多的计算机 具有性能价格比高、高度灵活、相对简单、易于实现等特点 以太网技术已成为当今最重要的一种局域网组网技术，网络交换机也就成为了最普及的交换机

家用级：

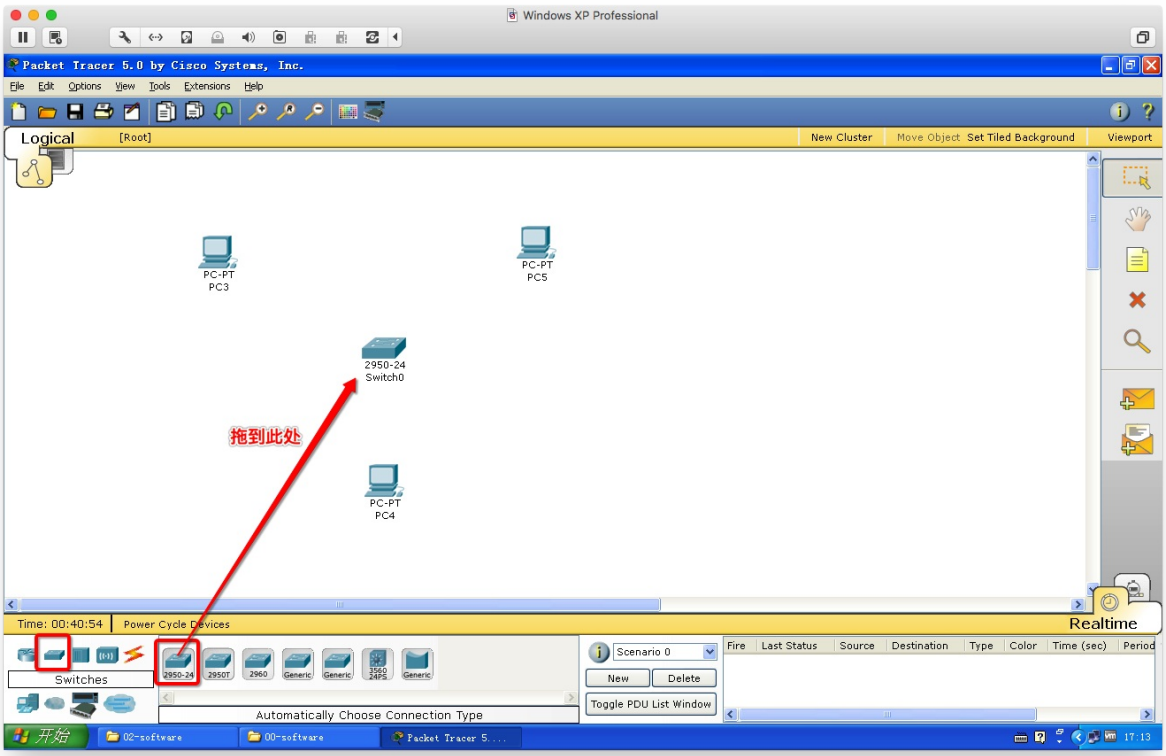
企业级：

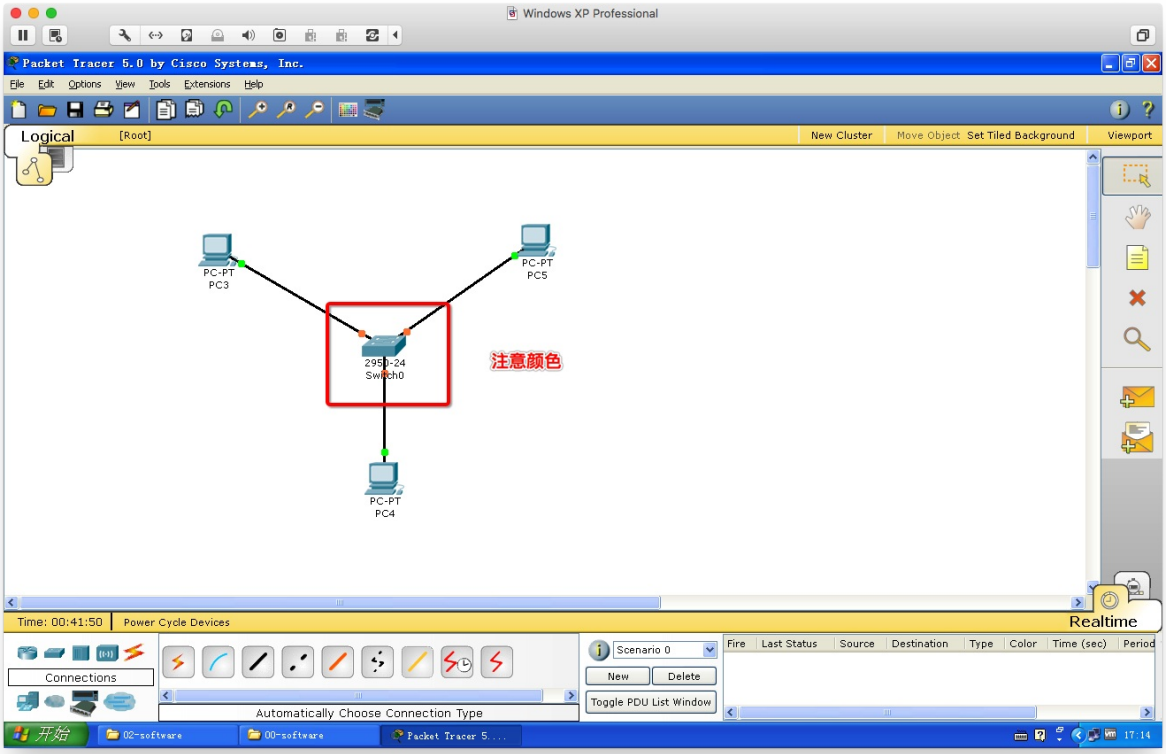
交换机的作用：

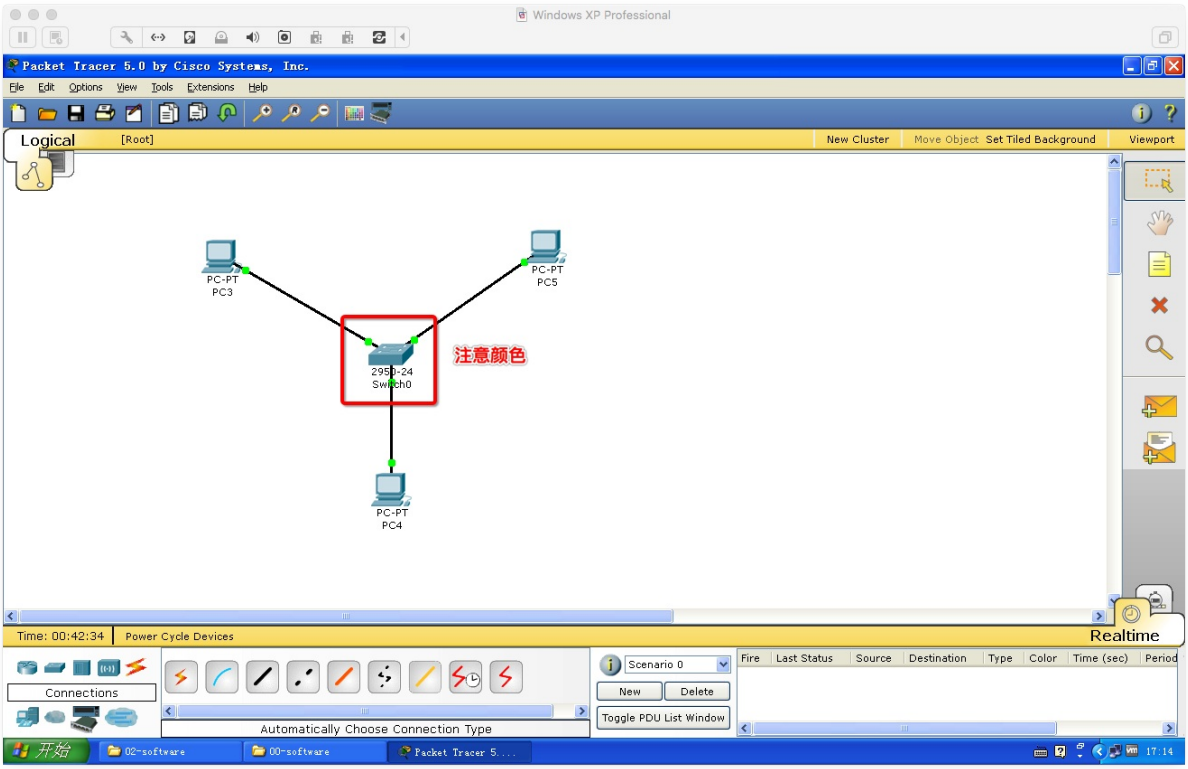
- 转发过滤：当一个数据帧的目的地址在MAC地址表中有映射时，它被转发到连接目的节点的端口而不是所有端口（如该数据帧为广播帧则转发至所有端口）
- 学习功能：以太网交换机了解每一端口相连设备的MAC地址，并将地址同相应的端口映射起来存放在交换机缓存中的MAC地址表中

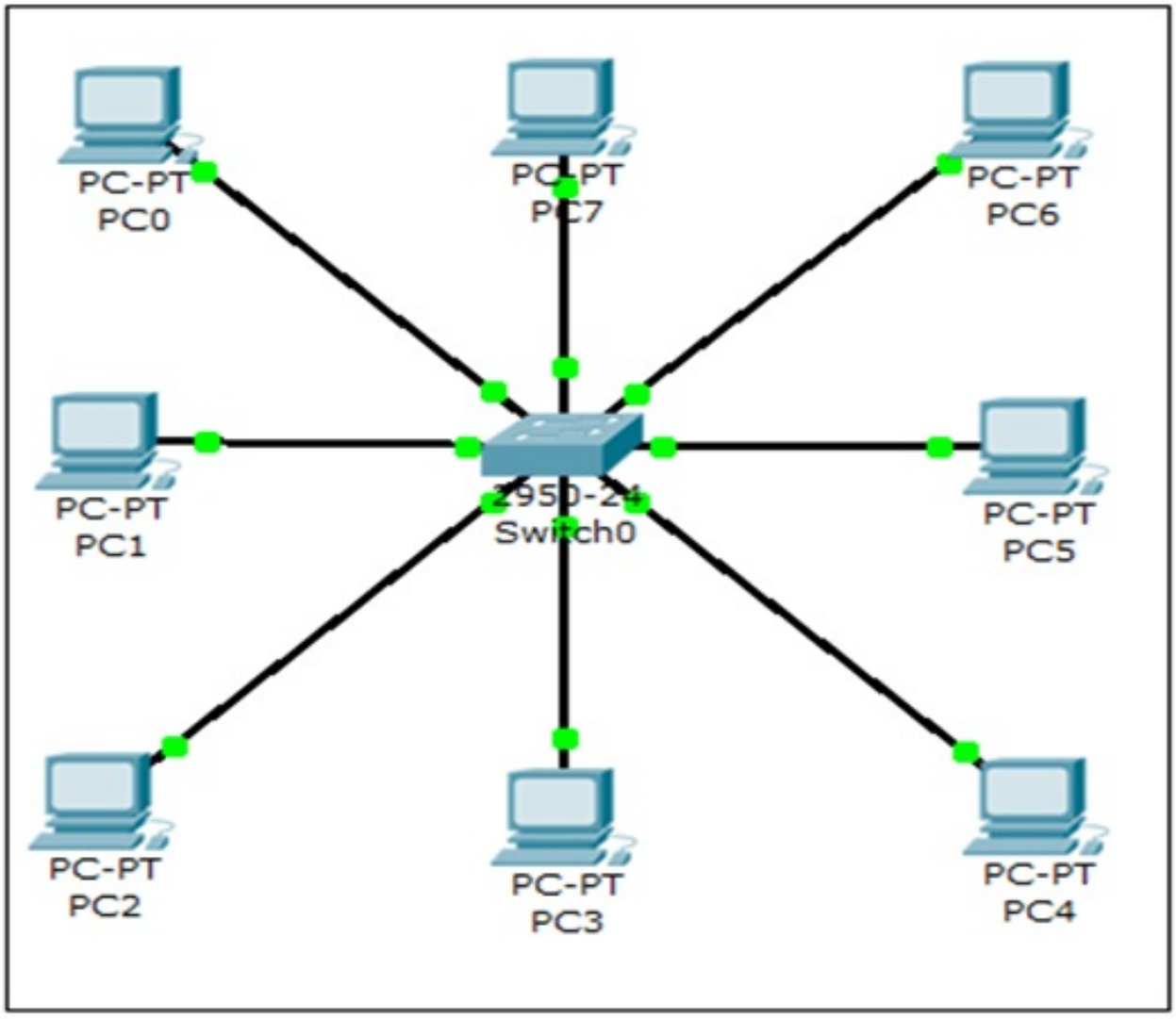
2. 通信过程（pc+switch）

在Packet Tracer 中，完成如下图示的链接：






Windows XP Professional
Packet Tracer 5.0 by Cisco Systems, Inc.
File Edit Options View Tools Extensions Help
Logical
[Root]
New Cluster
Move Object
Set Tiled Background
Viewport
PC-PT PC3
PC-PT PC5
注意颜色
2950-24 Switch0
PC-PT PC4
Time: 00:42:34
Power Cycle Devices
Realtime
Connections
Automatically Choose Connection Type
Scenario 0
New
Delete
Toggle PDU List Window
Fire
Last Status
Source
Destination
Type
Color
Time (sec)
Period
开始
02-software
00-software
Packet Tracer 5....
17:14


PC-PT PC0
PC-PT PC7
PC-PT PC6
PC-PT PC1
2950-24 Switch0
PC-PT PC5
PC-PT PC2
PC-PT PC3
PC-PT PC4

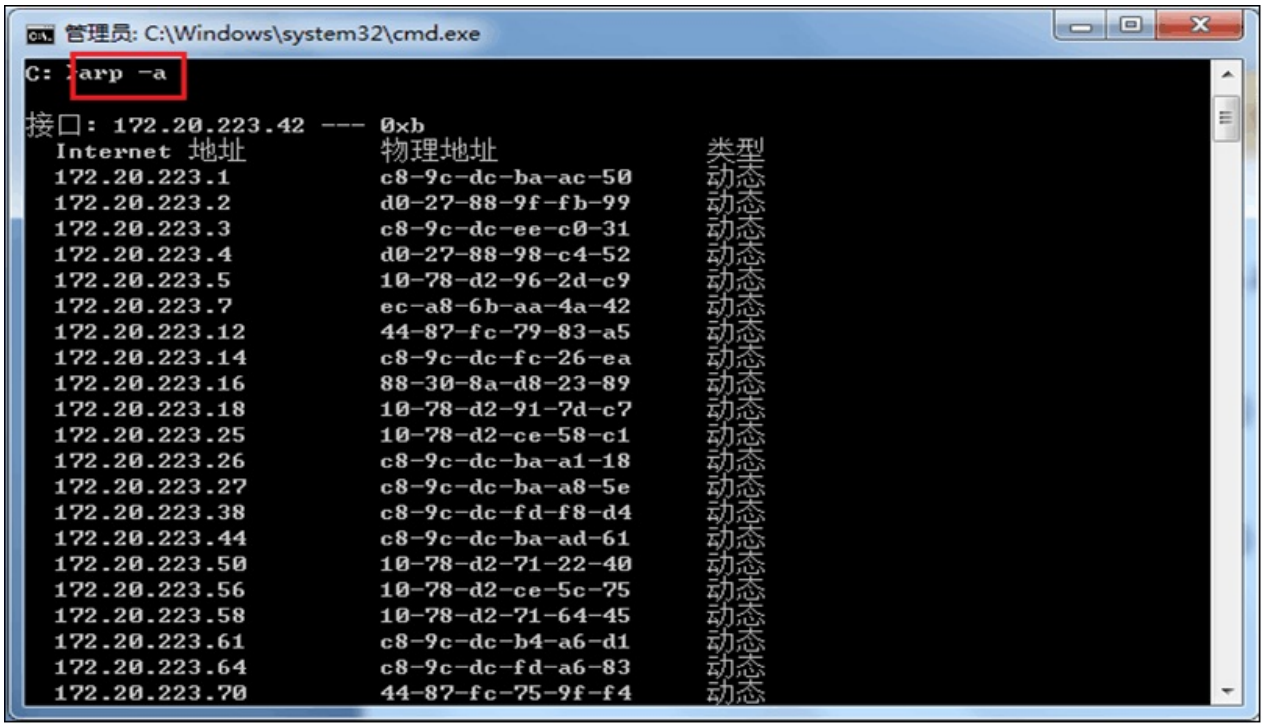
注意：

- 每台pc必须手动设置IP、netmask

  192.168.1.1/255.255.255.0 …… 192.168.1.8/255.255.255.0

arp缓存表

- 每台pc都会有一个arp缓存表，用来记录IP所对应的的MAC



2. 总结

- 交换机能够完成多个电脑的链接
- 每个数据包的发送都是以广播的形式进行的，容易堵塞网络
- 如果PC不知目标IP所对应的的MAC，那么可以看出，pc会先发送arp广播，得到对方的MAC然后，在进行数据的传送
- 当switch第一次收到arp广播数据，会把arp广播数据包转发给所有端口（除来源端口）；如果以后还有pc询问此IP的MAC，那么只是向目标的端口进行转发数据

通过路由器组网

1. 路由器简介

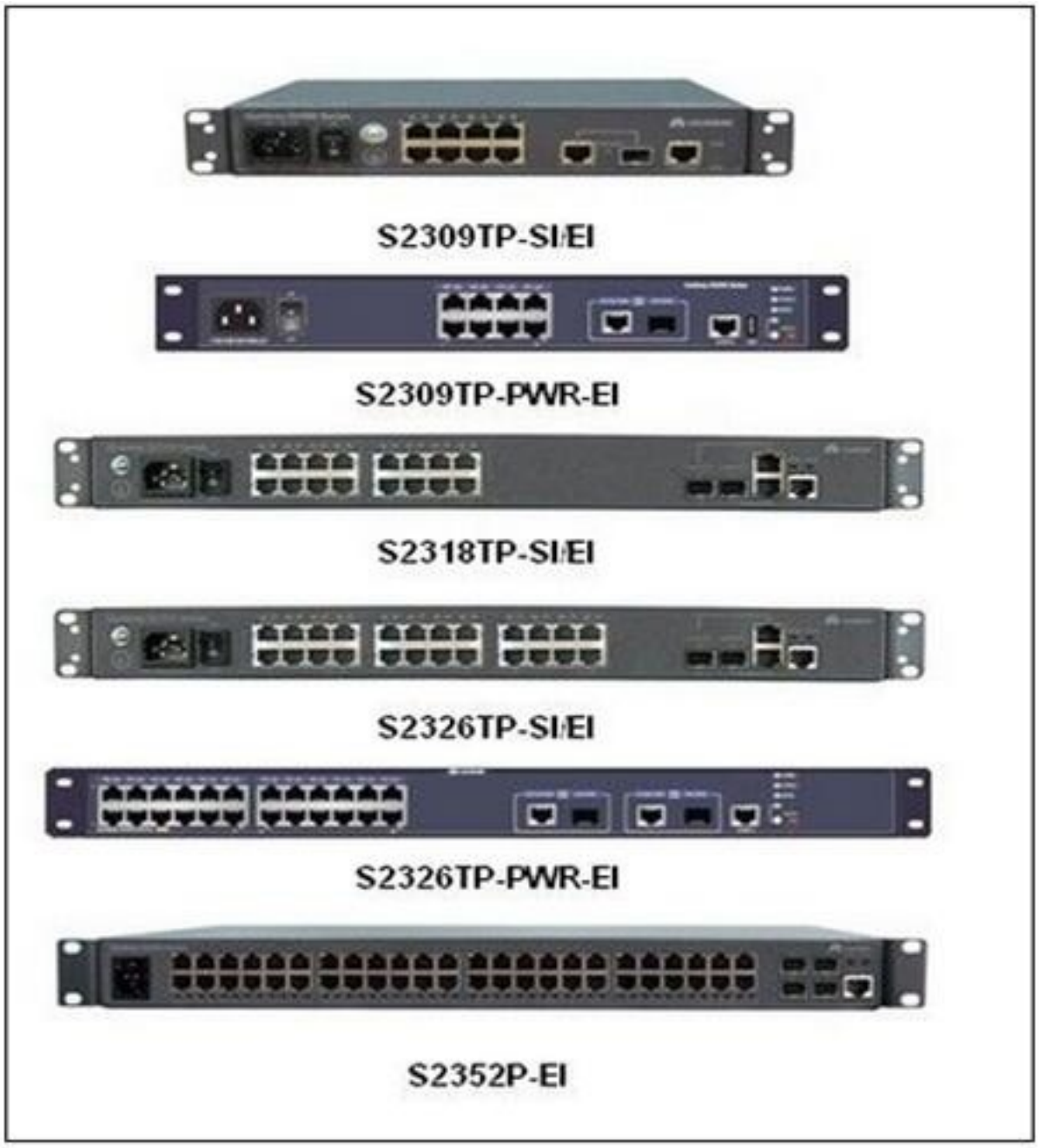
路由器（Router）又称网关设备（Gateway）是用于连接多个逻辑上分开的网络

所谓逻辑网络是代表一个单独的网络或者一个子网。当数据从一个子网传输到另一个子网时，可通过路由器的路由功能来完成

具有判断网络地址和选择IP路径的功能

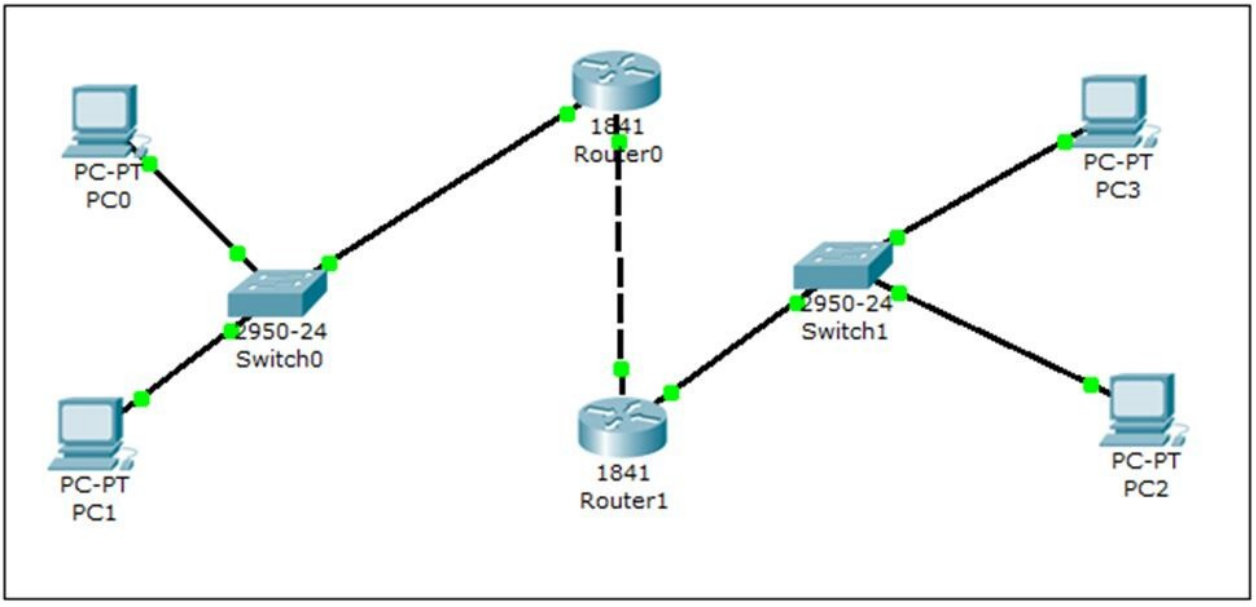
家用级：

企业级：



2. 通信过程（pc+switch+router）

通过2个router，2个switch，4台pc组成的网络，如下：



说明：

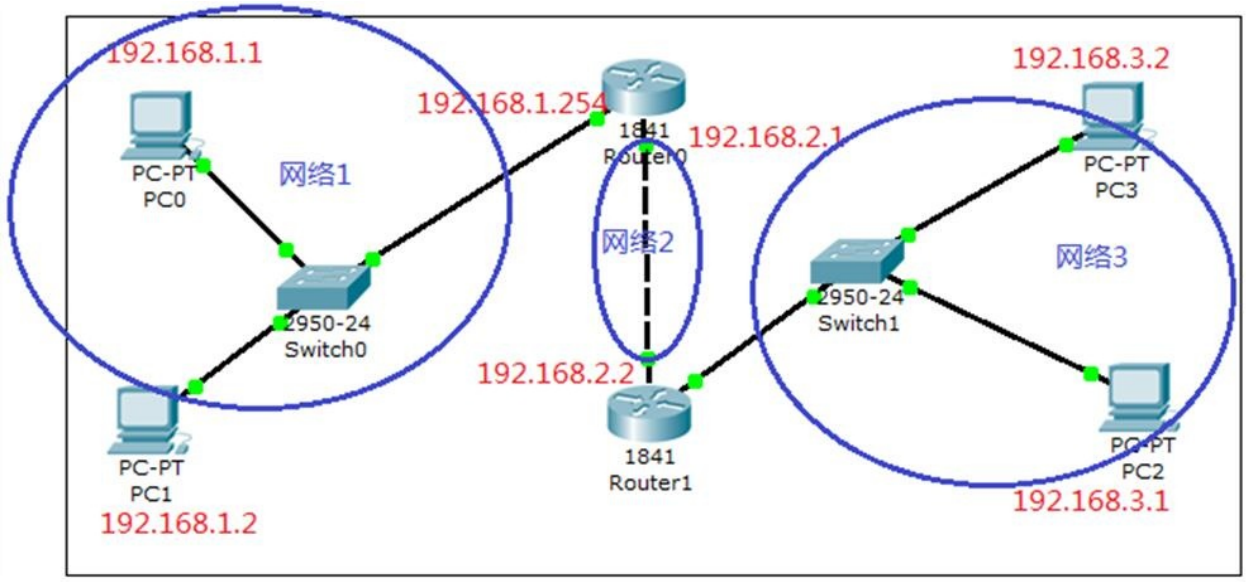
- 配置所有pc的IP、netmask

  pc0（192.168.1.1/24） pc1（192.168.1.2/24） pc2（192.168.3.1/24）
  pc3（192.168.3.2/24）

- 配置router（每个router有2个IP）

  router0(192.168.1.254/24,192.168.2.1/24)
  router1(192.168.3.254/24,192.168.2.2/24)

配置后的网络信息如下：



3. 总结

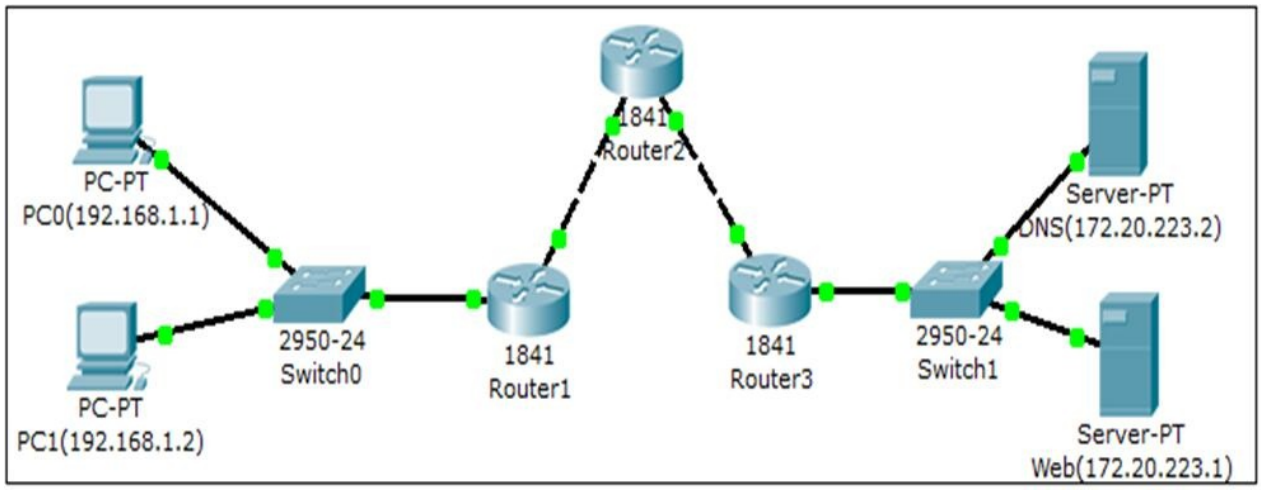
- 不在同一网段的pc，需要设置默认网关才能把数据传送过去 通常情况下，都会把路由器默认网关
- 当路由器收到一个其它网段的数据包时，会根据“路由表”来决定，把此数据包发送到哪个端口；路由表的设定有静态和动态方法
- 每经过一次路由器，那么TTL值就会减一

在dos控制端下，可通过输入命令查看路由表：

交换机、路由器、服务器组网

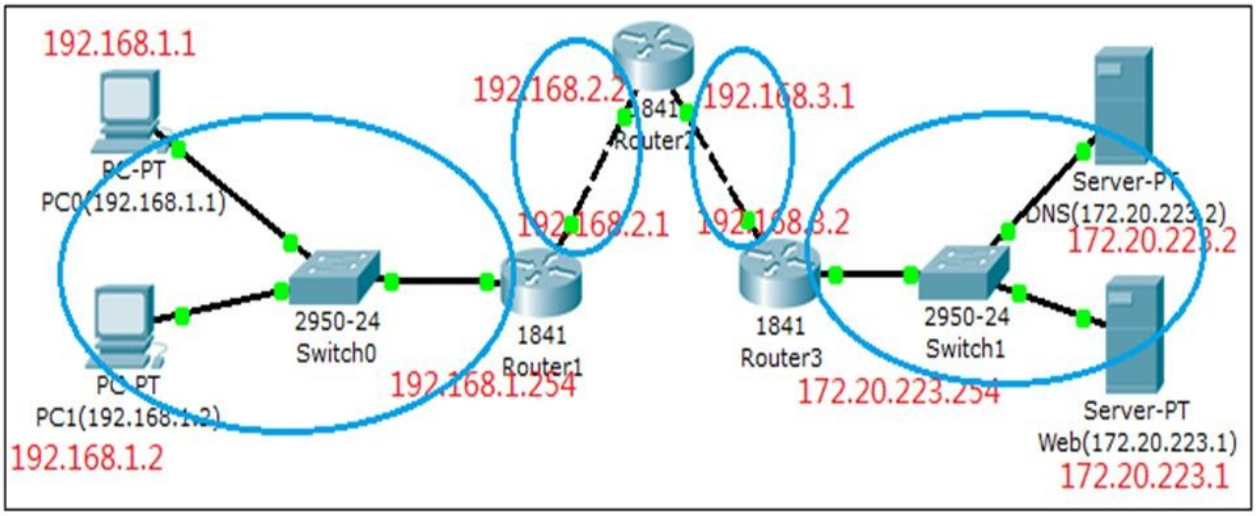
1. 通信过程（pc+switch+router+server）

较为复杂的通信过程如：访问 www.itheima.com



注意：一定要配置

- PC：IP、NETMASK、DFGATEWAY、DNS
- ROUTER：IP、NETMASK、路由表



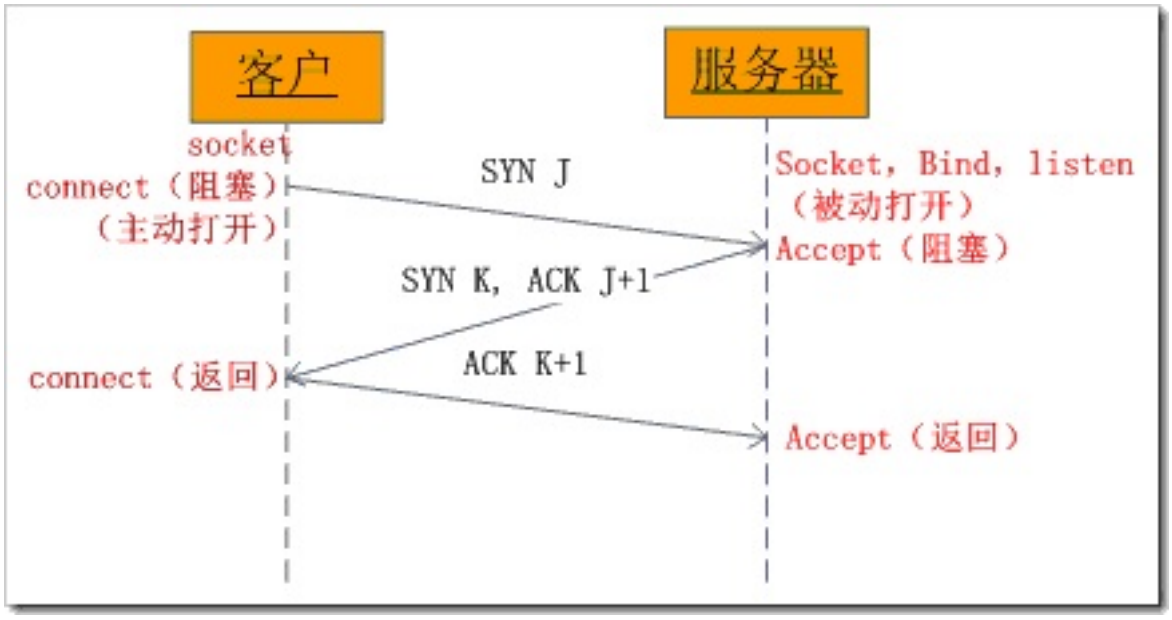
总结

- DNS服务器用来解析出IP（类似电话簿）
- DFGATEWAY（默认网关）用来对顶，当发送的数据包的目的ip不是当

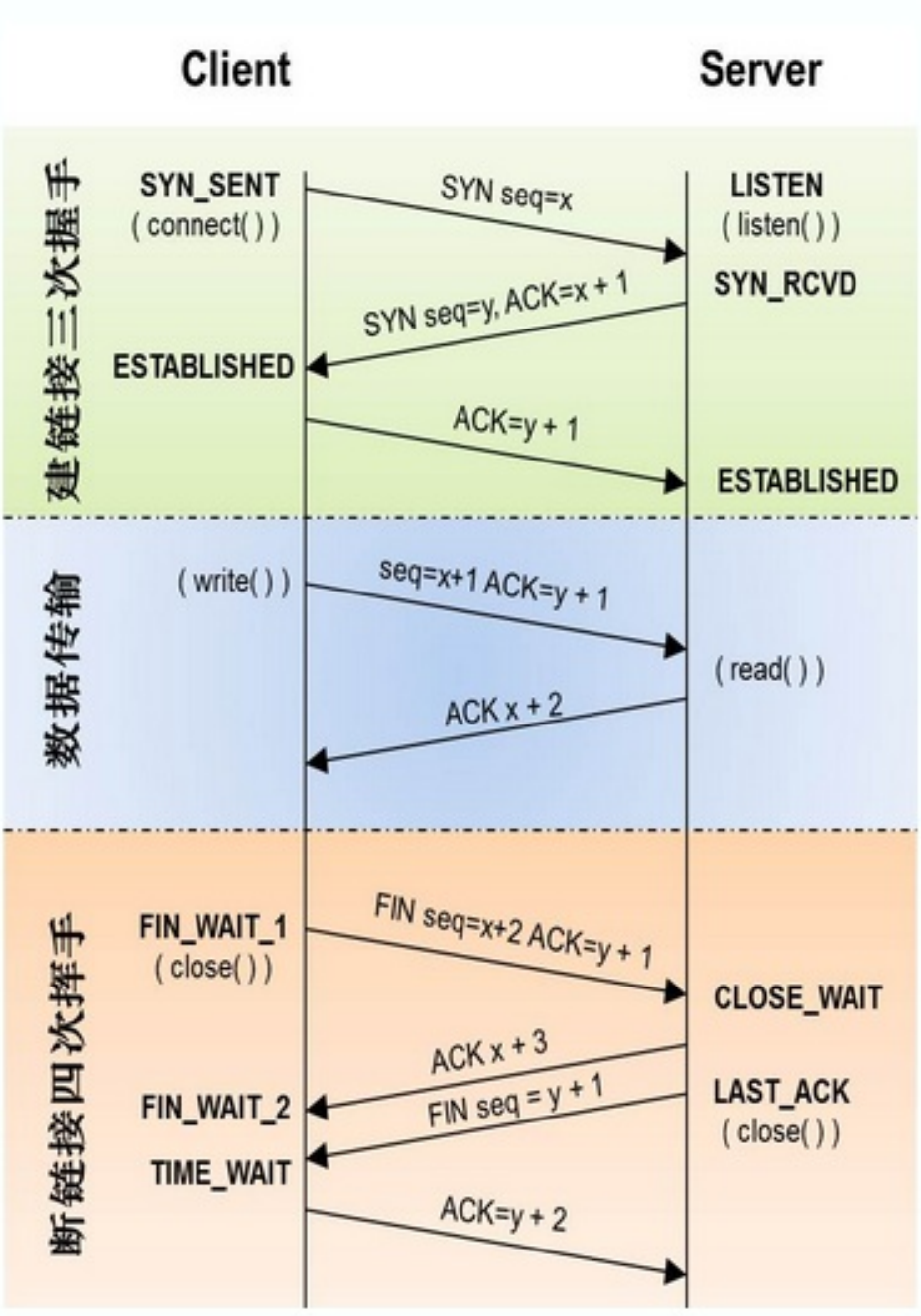
前网络时，此数据包包转发的目的ip

- 在路由器中路由表指定数据包的”下一跳”的地址

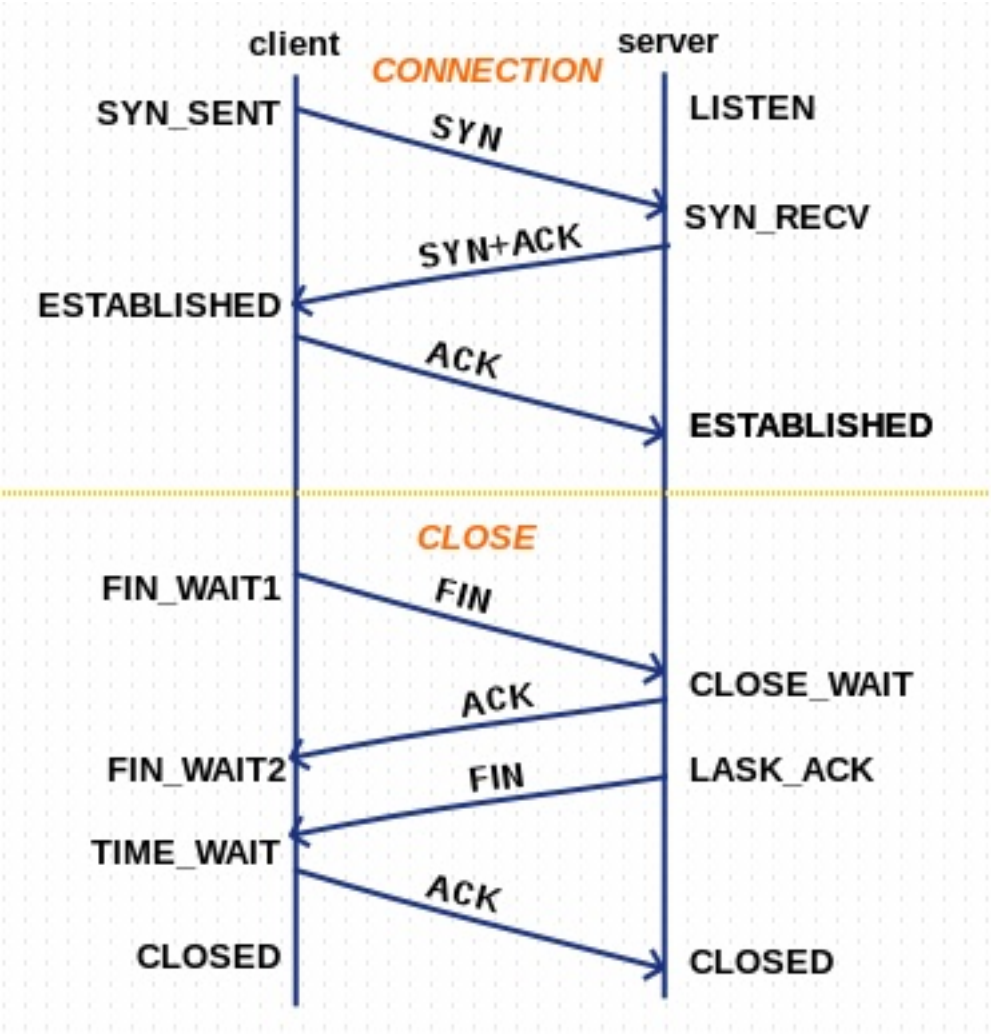
tcp三次握手


客户
服务器
socket
connect（阻塞）
（主动打开）
SYN J
Socket, Bind, listen
（被动打开）
Accept（阻塞）
SYN K, ACK J+1
connect（返回）
ACK K+1
Accept（返回）

tcp四次挥手

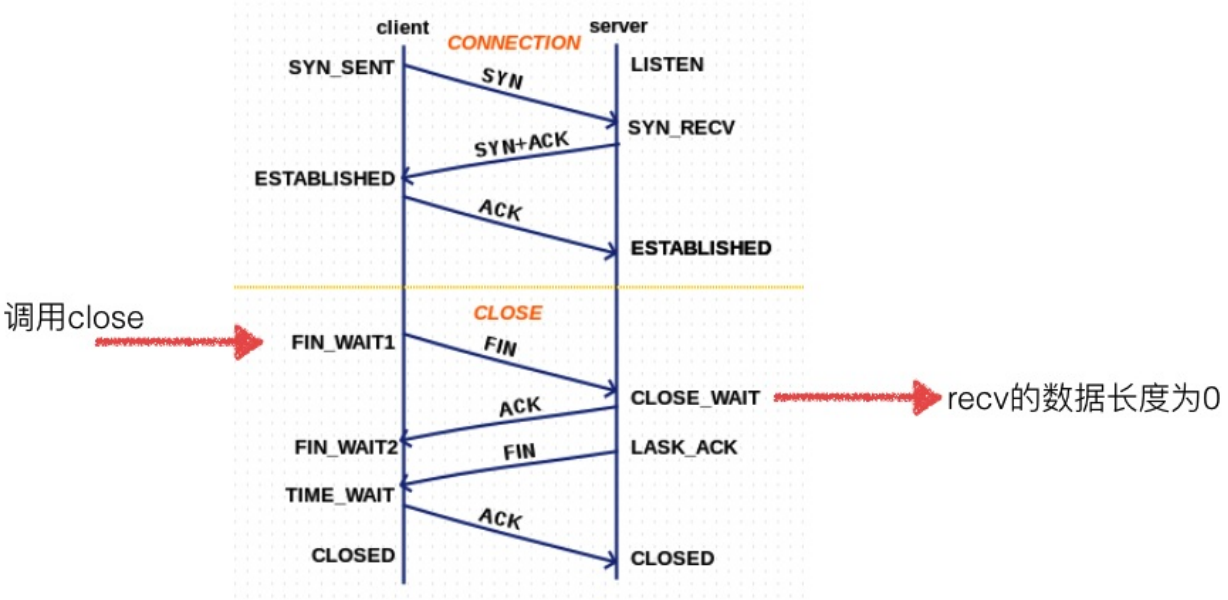

Client
Server
建链接三次握手
SYN_SENT
(connect( ))
SYN seq=x
LISTEN
(listen( ))
SYN seq=y, ACK=x + 1
SYN_RCVD
ESTABLISHED
ACK=y + 1
ESTABLISHED
数据传输
(write( ))
seq=x+1 ACK=y + 1
(read( ))
ACK x + 2
断链接四次挥手
FIN_WAIT_1
(close( ))
FIN seq=x+2 ACK=y + 1
CLOSE_WAIT
ACK x + 3
FIN_WAIT_2
FIN seq = y + 1
LAST_ACK
(close( ))
TIME_WAIT
ACK=y + 2

tcp十种状态

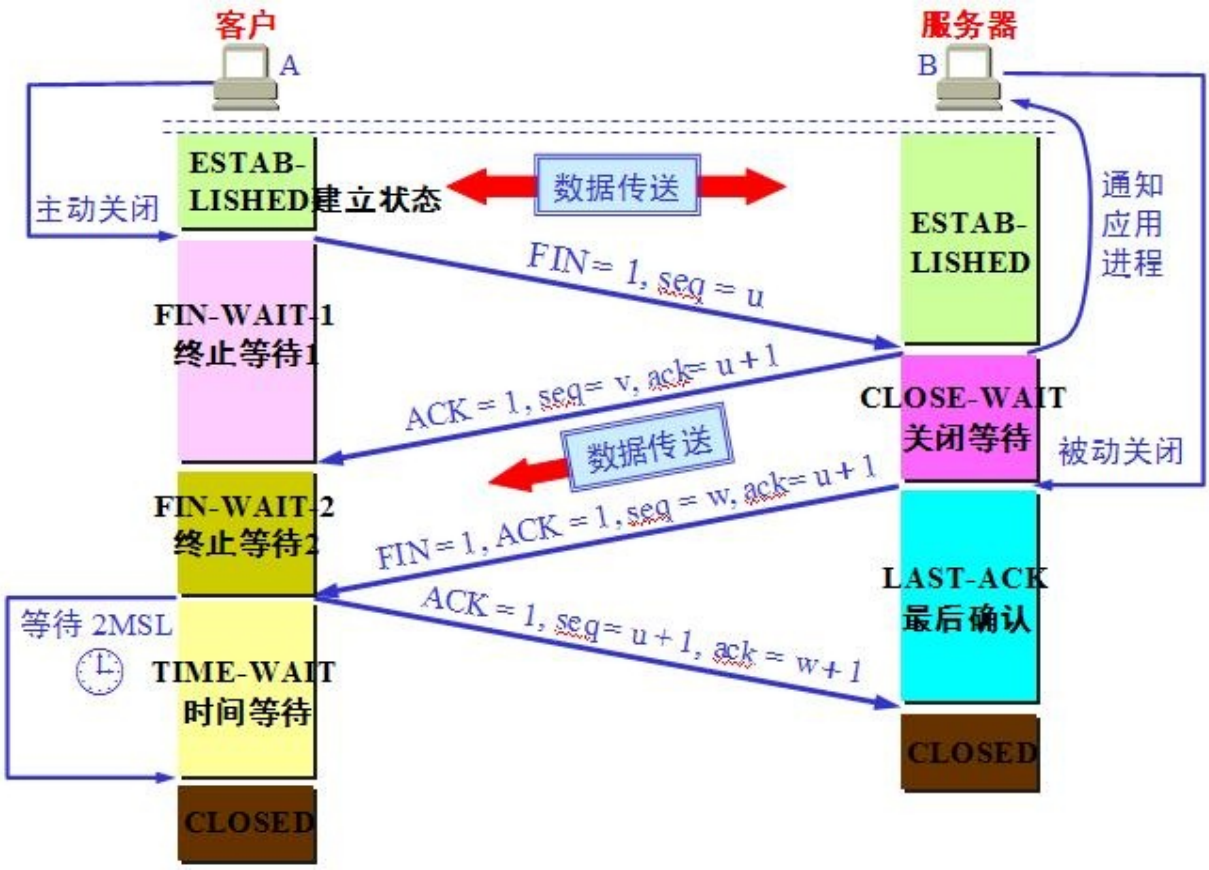


注意:

- 当一端收到一个FIN，内核让read返回0来通知应用层另一端已经终止了向本端的数据传送
- 发送FIN通常是应用层对socket进行关闭的结果


client
server
CONNECTION
SYN_SENT
SYN
LISTEN
SYN_RECV
SYN+ACK
ESTABLISHED
ACK
ESTABLISHED
调用close
CLOSE
FIN_WAIT1
FIN
CLOSE_WAIT
recv的数据长度为0
ACK
FIN_WAIT2
FIN
LASK_ACK
TIME_WAIT
ACK
CLOSED
CLOSED

tcp的2MSL问题



说明

2MSL即两倍的MSL，TCP的TIME_WAIT状态也称为2MSL等待状态，

当TCP的一端发起主动关闭，在发出最后一个ACK包后，

即第3次握 手完成后发送了第四次握手的ACK包后就进入了TIME_WAIT状态，

必须在此状态上停留两倍的MSL时间，

等待2MSL时间主要目的是怕最后一个 ACK包对方没收到，

那么对方在超时后将重发第三次握手的FIN包，

主动关闭端接到重发的FIN包后可以再发一个ACK应答包。

在TIME_WAIT状态 时两端的端口不能使用，要等到2MSL时间结束才可继续使用。

当连接处于2MSL等待阶段时任何迟到的报文段都将被丢弃。

不过在实际应用中可以通过设置 SO_REUSEADDR选项达到不必等待2MSL时间结束再使用此端口。

tcp长连接和短连接

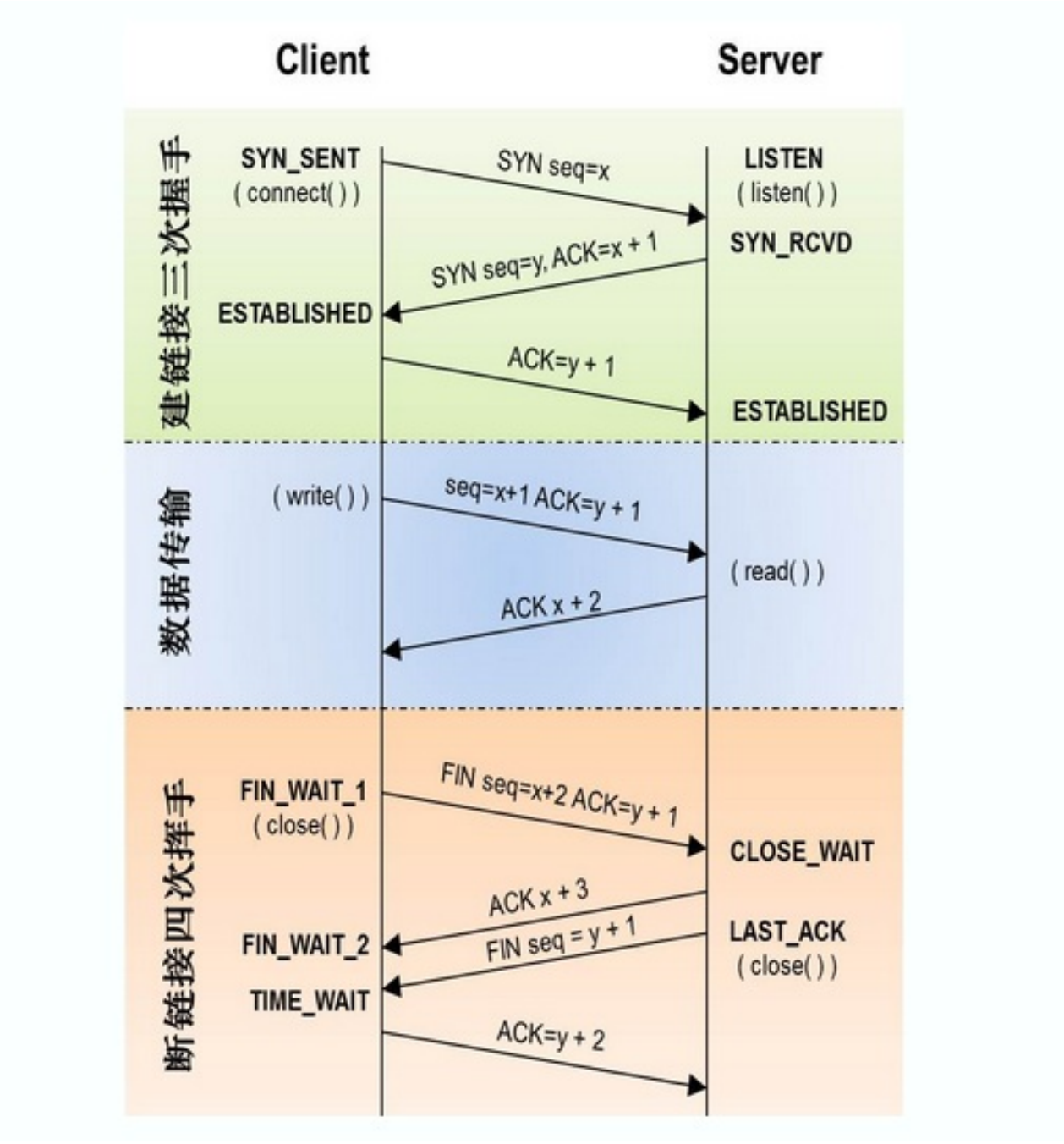
TCP在真正的读写操作之前，server与client之间必须建立一个连接，

当读写操作完成后，双方不再需要这个连接时它们可以释放这个连接，

连接的建立通过三次握手，释放则需要四次握手，

所以说每个连接的建立都是需要资源消耗和时间消耗的。

TCP通信的整个过程，如下图:



1. TCP短连接

模拟一种TCP短连接的情况:

1. client 向 server 发起连接请求
2. server 接到请求，双方建立连接
3. client 向 server 发送消息
4. server 回应 client
5. 一次读写完成，此时双方任何一个都可以发起 close 操作

在第 步骤5中，一般都是 client 先发起 close 操作。当然也不排除有特殊的情况。

从上面的描述看，短连接一般只会在 client/server 间传递一次读写操作！

2. TCP长连接

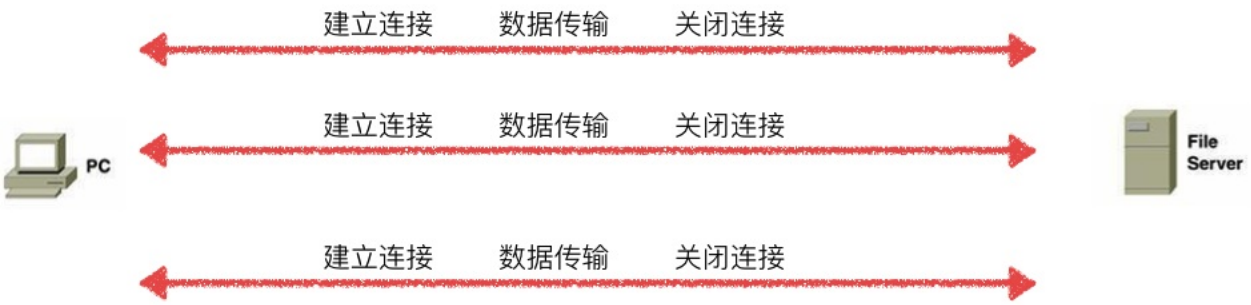
再模拟一种长连接的情况:

1. client 向 server 发起连接
2. server 接到请求，双方建立连接
3. client 向 server 发送消息
4. server 回应 client
5. 一次读写完成，连接不关闭
6. 后续读写操作...
7. 长时间操作之后client发起关闭请求

3. TCP长/短连接操作过程

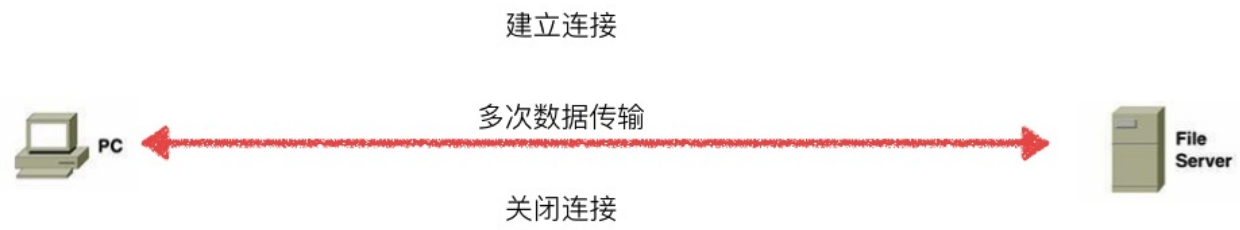
3.1 短连接的操作步骤是：

建立连接——数据传输——关闭连接...建立连接——数据传输——关闭连接



3.2 长连接的操作步骤是：

建立连接——数据传输...（保持连接）...数据传输——关闭连接



4. TCP长/短连接的优点和缺点

- 长连接可以省去较多的TCP建立和关闭的操作，减少浪费，节约时间。

  对于频繁请求资源的客户来说，较适用长连接。

- client与server之间的连接如果一直不关闭的话，会存在一个问题，

  随着客户端连接越来越多，server早晚有扛不住的时候，这时候server端需要采取一些策略，

  如关闭一些长时间没有读写事件发生的连接，这样可以避免一些恶意连接导致server端服务受损；

  如果条件再允许就可以以客户端机器为颗粒度，限制每个客户端的最大长连接数，

  这样可以完全避免某个蛋疼的客户端连累后端服务。

- 短连接对于服务器来说管理较为简单，存在的连接都是有用的连接，不需要额外的控制手段。

- 但如果客户请求频繁，将在TCP的建立和关闭操作上浪费时间和带宽。

5. TCP长/短连接的应用场景

- 长连接多用于操作频繁，点对点的通讯，而且连接数不能太多情况。

  每个TCP连接都需要三次握手，这需要时间，如果每个操作都是先连接，

再操作的话那么处理速度会降低很多，所以每个操作完后都不断开，

再次处理时直接发送数据包就OK了，不用建立TCP连接。

例如：数据库的连接用长连接，如果用短连接频繁的通信会造成socket错误，

而且频繁的socket 创建也是对资源的浪费。

- 而像WEB网站的http服务一般都用短链接，因为长连接对于服务端来说会耗费一定的资源，

  而像WEB网站这么频繁的成千上万甚至上亿客户端的连接用短连接会更省一些资源，

  如果用长连接，而且同时有成千上万的用户，如果每个用户都占用一个连接的话，

  那可想而知吧。所以并发量大，但每个用户无需频繁操作情况下需用短连好。

listen的队列长度

服务器端运行

```
#coding=utf-8
from socket import *
from time import sleep

# 创建socket
tcpSerSocket = socket(AF_INET, SOCK_STREAM)

# 绑定本地信息
address = ('', 7788)
tcpSerSocket.bind(address)

connNum = int(raw_input("请输入要最大的链接数:"))

# 使用socket创建的套接字默认的属性是主动的，使用listen将其变为被动的，这样
tcpSerSocket.listen(connNum)

while True:

    # 如果有新的客户端来链接服务器，那么就产生一个新的套接字专门为这个客户端
    newSocket, clientAddr = tcpSerSocket.accept()
    print clientAddr
    sleep(1)
```

客户端运行

```
#coding=utf-8
```

```
from socket import *

connNum = raw_input("请输入要链接服务器的次数:")
for i in range(int(connNum)):
    s = socket(AF_INET, SOCK_STREAM)
    s.connect(("192.168.1.102", 7788))
    print(i)
```

总结

- listen中的black表示已经建立链接和半链接的总数
- 如果当前已建立链接数和半链接数以达到设定值，那么新客户端就不会connect成功，而是等待服务器

手动配置ip

1. 设置IP和掩码

```
ifconfig eth0 192.168.5.40 netmask 255.255.255.0
```

2. 设置网关

```
route add default gw 192.168.5.1
```

常见网络攻击案例

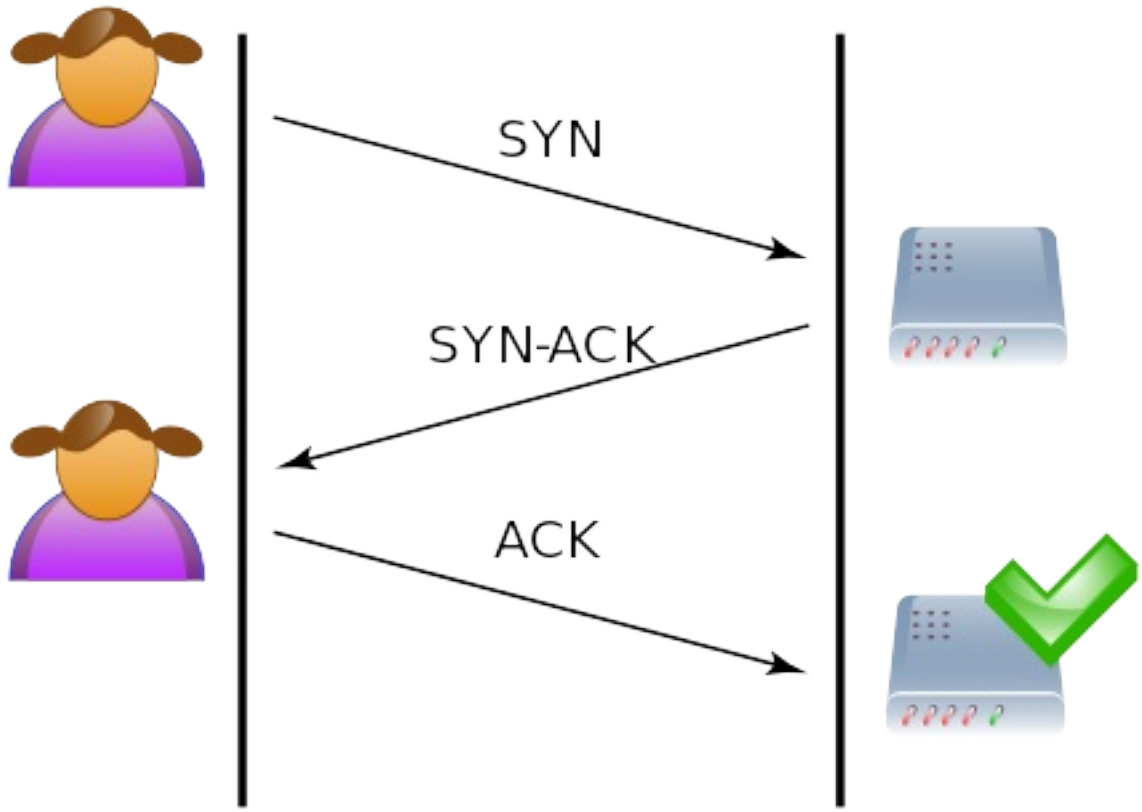
1. tcp半链接攻击

tcp半链接攻击也称为：SYN Flood (SYN洪水)

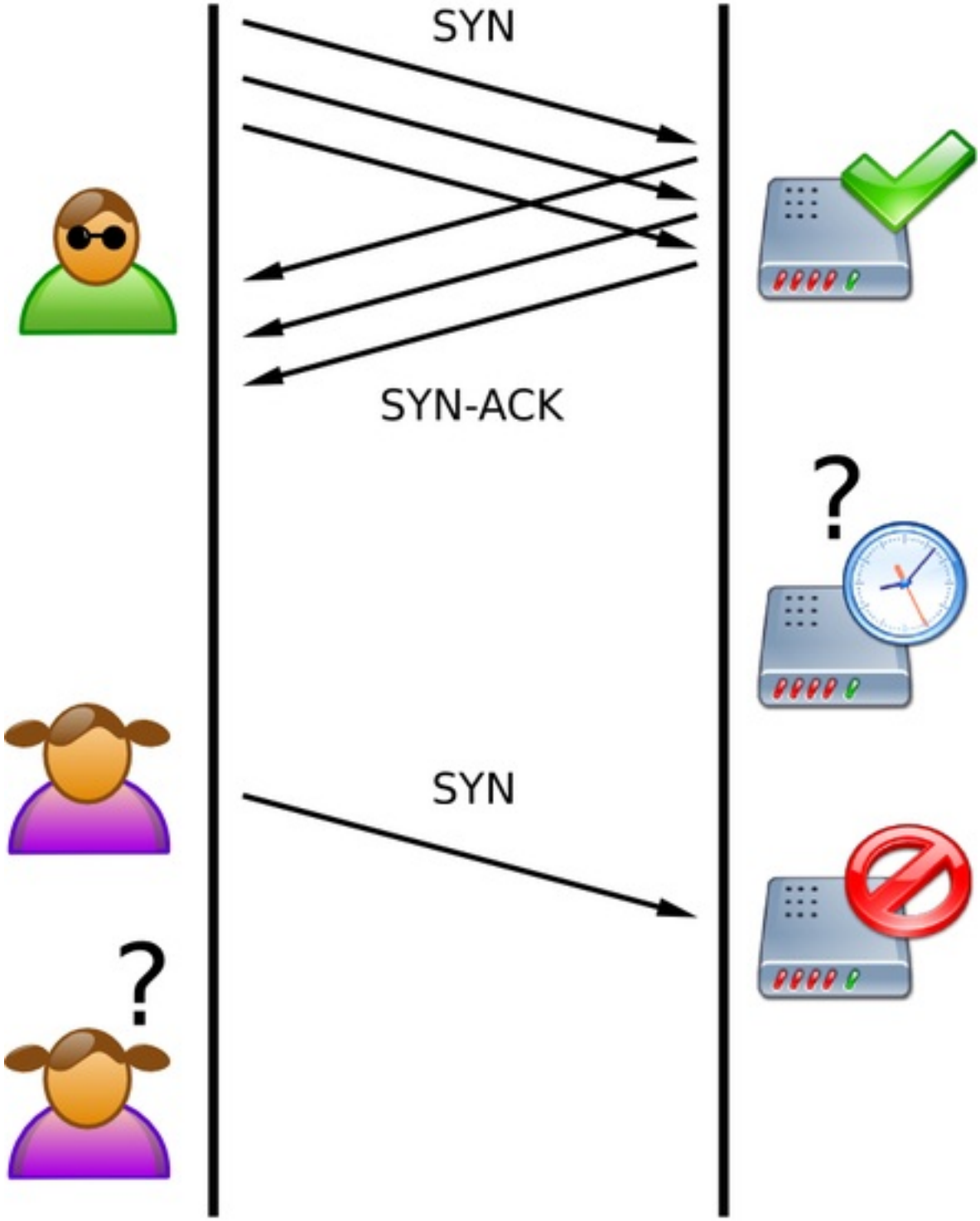
是种典型的DoS (Denial of Service，拒绝服务) 攻击

效果就是服务器TCP连接资源耗尽，停止响应正常的TCP连接请求

1.1 正常链接时的情况



1.2 半链接攻击时的情况



2. dns攻击

2.1 dns服务器被劫持

我们知道一个域名服务器对其区域内的用户解析请求负责，

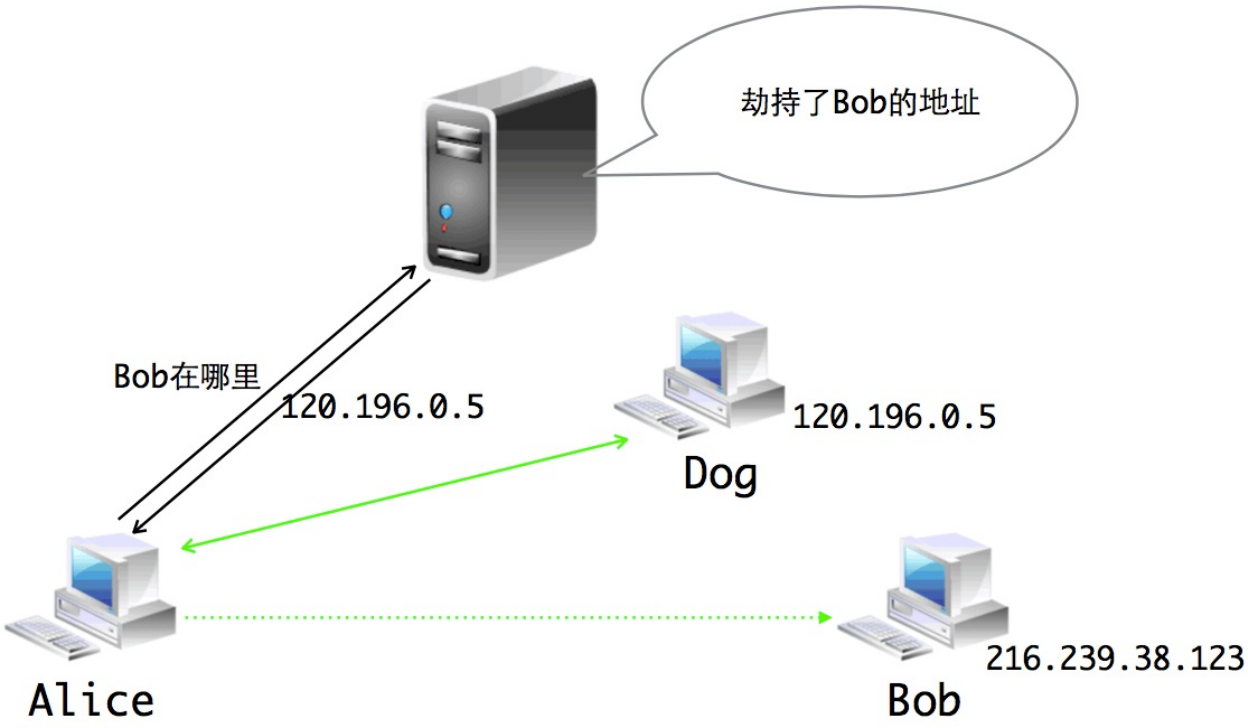
但是并没有一个机制去监督它有没有真地负责。

也就是说域名服务器的权力并没有被关在笼子里，

所以它既可以认真地“为人民服务”，也可以“指鹿为马”。

于是有些流氓的域名服务器故意更改一些域名的解析结果，

将用户引向一个错误的目标地址。这就叫作 DNS 劫持，主要用来阻止用户访问某些特定的网站，或者是将用户引导到广告页面。



2.2 dns欺骗

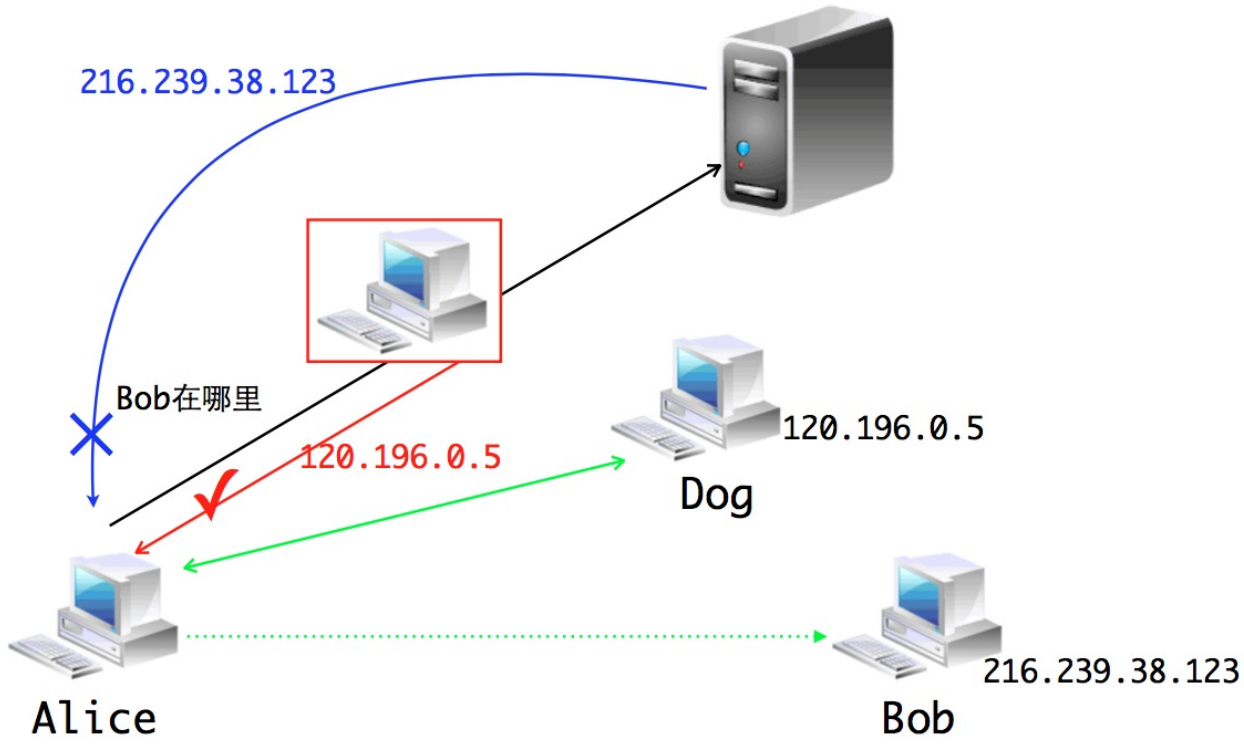
DNS 欺骗简单来说就是用一个假的 DNS 应答来欺骗用户计算机，

让其相信这个假的地址，并且抛弃真正的 DNS 应答。

在一台主机发出 DNS 请求后，它就开始等待应答，

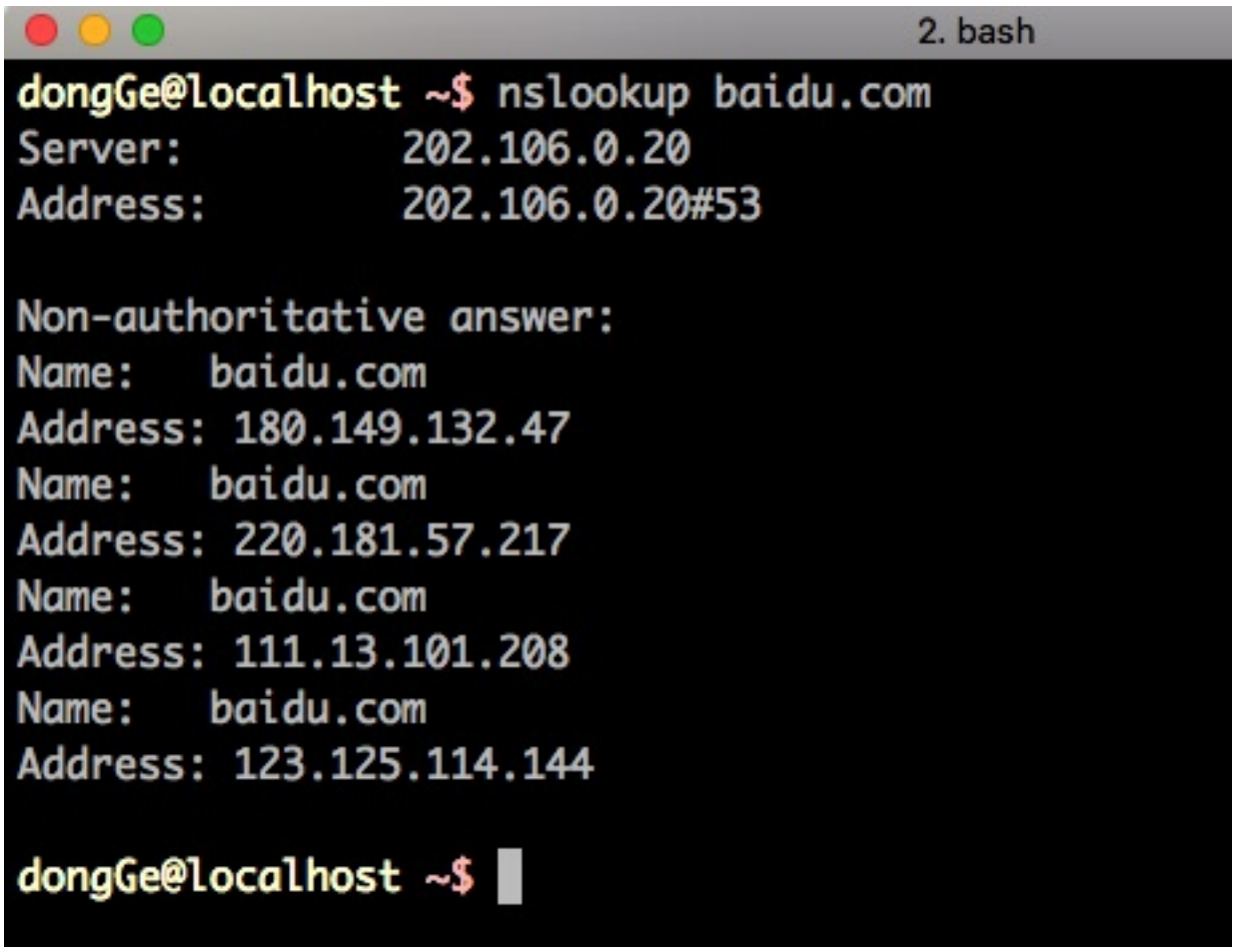
如果此时有一个看起来正确（拥有和DNS请求一样的序列号）的应答包，

它就会信以为真，并且丢弃稍晚一点到达的应答。

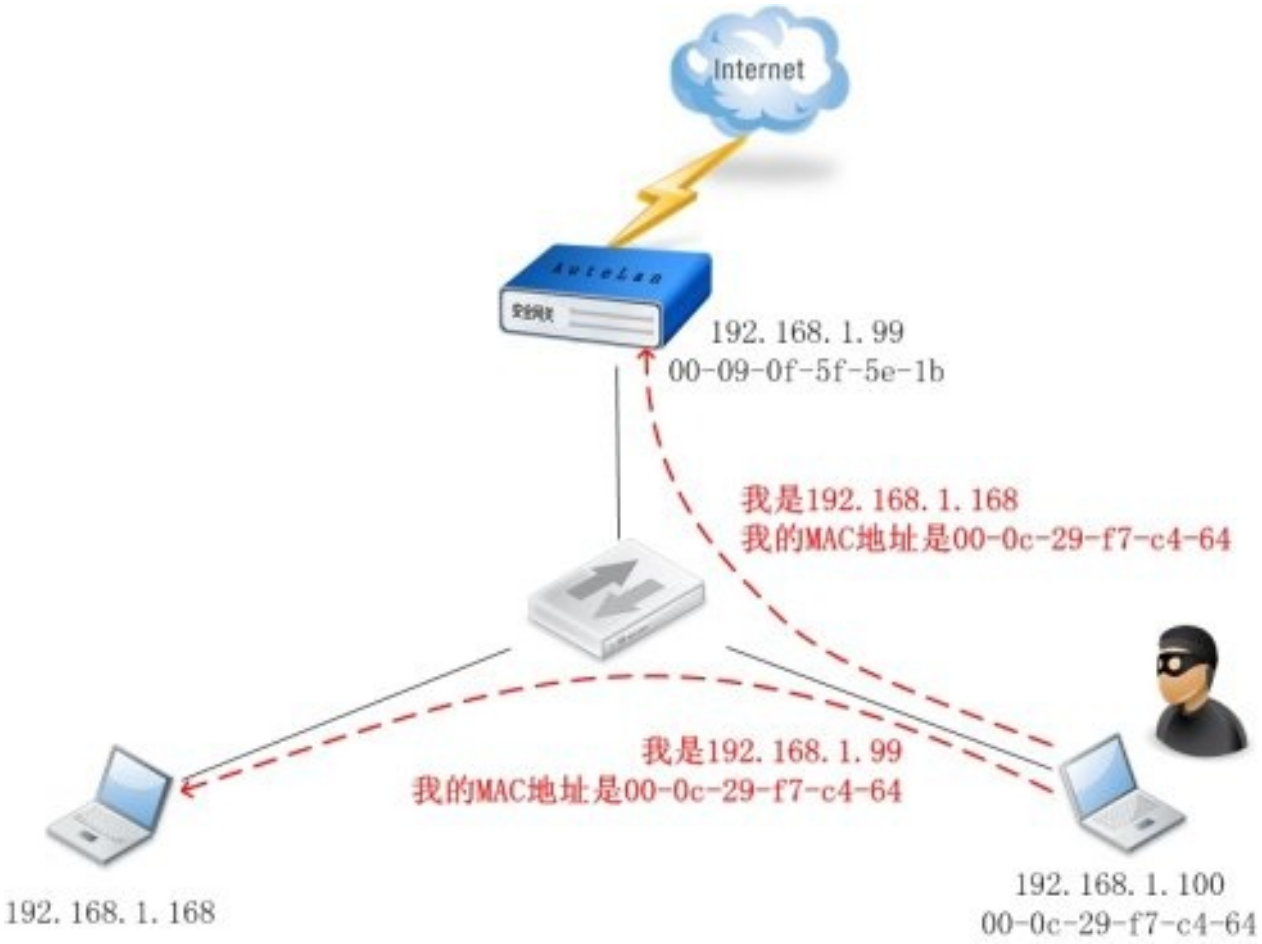


2.3 查看域名解析的ip地址方法

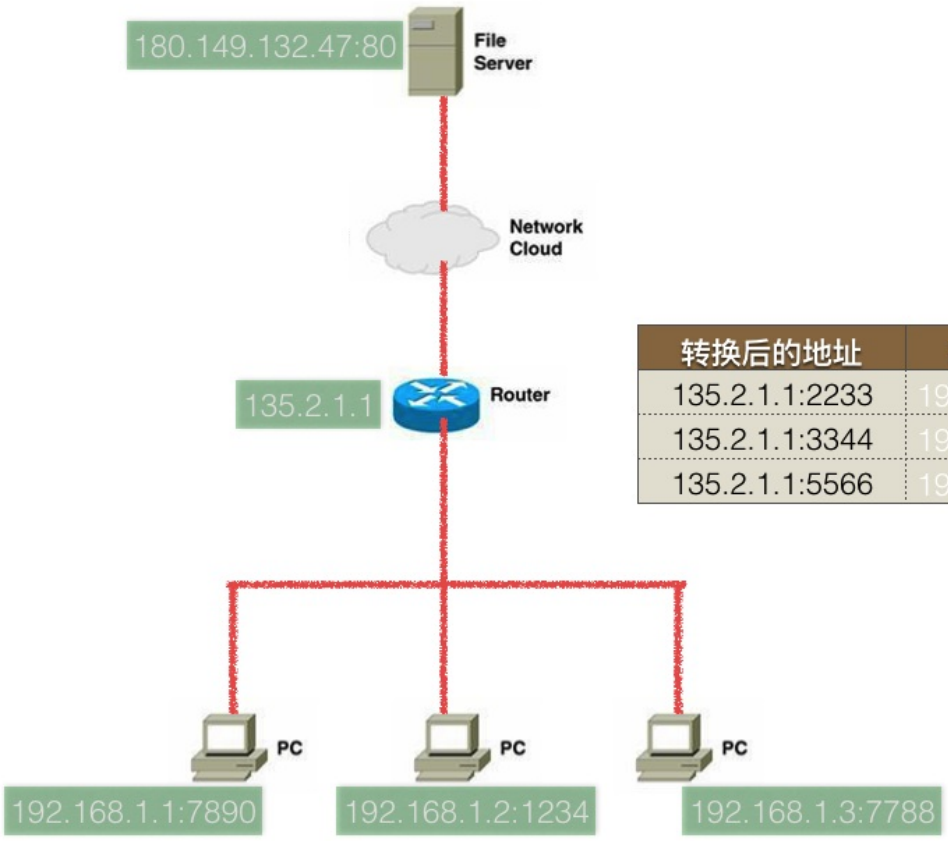
```
nslookup 域名

例如:
nslookup baidu.com
```


2. bash
dongGe@localhost ~$ nslookup baidu.com
Server: 202.106.0.20
Address: 202.106.0.20#53
Non-authoritative answer:
Name: baidu.com
Address: 180.149.132.47
Name: baidu.com
Address: 220.181.57.217
Name: baidu.com
Address: 111.13.101.208
Name: baidu.com
Address: 123.125.114.144
dongGe@localhost ~$

3. arp攻击


Internet
192.168.1.99
00-09-0f-5f-5e-1b
我是192.168.1.168
我的MAC地址是00-0c-29-f7-c4-64
我是192.168.1.99
我的MAC地址是00-0c-29-f7-c4-64
192.168.1.168
192.168.1.100
00-0c-29-f7-c4-64

家庭上网解析



转换后的地址	转换前的地址
135.2.1.1:2233	192.168.1.1:7890
135.2.1.1:3344	192.168.1.2:1234
135.2.1.1:5566	192.168.1.3:7788

单进程服务器

1. 完成一个简单的TCP服务器

```
from socket import *

serSocket = socket(AF_INET, SOCK_STREAM)

# 重复使用绑定的信息
serSocket.setsockopt(SOL_SOCKET, SO_REUSEADDR  , 1)

localAddr = ('', 7788)

serSocket.bind(localAddr)

serSocket.listen(5)

while True:

    print('-----主进程，，等待新客户端的到来------')

    newSocket,destAddr = serSocket.accept()

    print('-----主进程，，接下来负责数据处理[%s]-----'%str(destAddr))

    try:
        while True:
            recvData = newSocket.recv(1024)
            if len(recvData)>0:
                print('recv[%s]:%s'%(str(destAddr), recvData))
            else:
                print('[%s]客户端已经关闭'%str(destAddr))
                break
    finally:
        newSocket.close()
```

```
serSocket.close()
```

2. 总结

- 同一时刻只能为一个客户进行服务，不能同时为多个客户服务
- 类似于找一个“明星”签字一样，客户需要耐心等待才可以获取到服务
- 当服务器为一个客户端服务时，而另外的客户端发起了connect，只要服务器listen的队列有空闲的位置，就会为这个新客户端进行连接，并且客户端可以发送数据，但当服务器为这个新客户端服务时，可能一次性把所有数据接收完毕
- 当recv接收数据时，返回值为空，即没有返回数据，那么意味着客户端已经调用了close关闭了；因此服务器通过判断recv接收数据是否为空 来判断客户端是否已经下线

多进程服务器

1. 多进程服务器

```
from socket import *
from multiprocessing import *
from time import sleep

# 处理客户端的请求并为其服务
def dealWithClient(newSocket,destAddr):
    while True:
        recvData = newSocket.recv(1024)
        if len(recvData)>0:
            print('recv[%s]:%s'%(str(destAddr), recvData))
        else:
            print('[%s]客户端已经关闭'%str(destAddr))
            break

    newSocket.close()


def main():

    serSocket = socket(AF_INET, SOCK_STREAM)
    serSocket.setsockopt(SOL_SOCKET, SO_REUSEADDR  , 1)
    localAddr = ('', 7788)
    serSocket.bind(localAddr)
    serSocket.listen(5)

    try:
        while True:
            print('-----主进程，，等待新客户端的到来------')
            newSocket,destAddr = serSocket.accept()
```

```
            print('-----主进程，，接下来创建一个新的进程负责数据处理[%s
            client = Process(target=dealWithClient, args=(newSoc
            client.start()

            #因为已经向子进程中copy了一份（引用），并且父进程中这个套接字
            #所以关闭
            newSocket.close()
    finally:
        #当为所有的客户端服务完之后再进行关闭，表示不再接收新的客户端的链接
        serSocket.close()

if __name__ == '__main__':
    main()
```

2. 总结

- 通过为每个客户端创建一个进程的方式，能够同时为多个客户端进行服务
- 当客户端不是特别多的时候，这种方式还行，如果有几百上千个，就不可取了，因为每次创建进程等过程需要好较大的资源

多线程服务器

```
#coding=utf-8
from socket import *
from threading import Thread
from time import sleep

# 处理客户端的请求并执行事情
def dealWithClient(newSocket,destAddr):
    while True:
        recvData = newSocket.recv(1024)
        if len(recvData)>0:
            print('recv[%s]:%s'%(str(destAddr), recvData))
        else:
            print('[%s]客户端已经关闭'%str(destAddr))
            break

    newSocket.close()


def main():

    serSocket = socket(AF_INET, SOCK_STREAM)
    serSocket.setsockopt(SOL_SOCKET, SO_REUSEADDR  , 1)
    localAddr = ('', 7788)
    serSocket.bind(localAddr)
    serSocket.listen(5)

    try:
        while True:
            print('-----主进程，，等待新客户端的到来------')
            newSocket,destAddr = serSocket.accept()

            print('-----主进程，，接下来创建一个新的进程负责数据处理[%s
```

```
            client = Thread(target=dealWithClient, args=(newSock
            client.start()

            #因为线程中共享这个套接字，如果关闭了会导致这个套接字不可用，
            #但是此时在线程中这个套接字可能还在收数据，因此不能关闭
            #newSocket.close()
    finally:
        serSocket.close()

if __name__ == '__main__':
    main()
```

单进程服务器-非堵塞模式

服务器

```
#coding=utf-8
from socket import *
import time

# 用来存储所有的新链接的socket
g_socketList = []

def main():
    serSocket = socket(AF_INET, SOCK_STREAM)
    serSocket.setsockopt(SOL_SOCKET, SO_REUSEADDR  , 1)
    localAddr = ('', 7788)
    serSocket.bind(localAddr)
    #可以适当修改listen中的值来看看不同的现象
    serSocket.listen(1000)
    #将套接字设置为非堵塞
    #设置为非堵塞后，如果accept时，恰巧没有客户端connect，那么accept会
    #产生一个异常，所以需要try来进行处理
    serSocket.setblocking(False)

    while True:

        #用来测试
        #time.sleep(0.5)

        try:
            newClientInfo = serSocket.accept()
        except Exception as result:
            pass
        else:
            print("一个新的客户端到来:%s"%str(newClientInfo))
            newClientInfo[0].setblocking(False)
```

```
            g_socketList.append(newClientInfo)

        # 用来存储需要删除的客户端信息
        needDelClientInfoList = []

        for clientSocket,clientAddr in g_socketList:
            try:
                recvData = clientSocket.recv(1024)
                if len(recvData)>0:
                    print('recv[%s]:%s'%(str(clientAddr), recvDa
                else:
                    print('[%s]客户端已经关闭'%str(clientAddr))
                    clientSocket.close()
                    g_needDelClientInfoList.append((clientSocket
            except Exception as result:
                pass

        for needDelClientInfo in needDelClientInfoList:
            g_socketList.remove(needDelClientInfo)

if __name__ == '__main__':
    main()
```

客户端

```
#coding=utf-8
from socket import *
import random
import time

serverIp = raw_input("请输入服务器的ip:")
connNum = raw_input("请输入要链接服务器的次数(例如1000):")
g_socketList = []
for i in range(int(connNum)):
    s = socket(AF_INET, SOCK_STREAM)
```

```
        s.connect((serverIp, 7788))
        g_socketList.append(s)
        print(i)

    while True:
        for s in g_socketList:
            s.send(str(random.randint(0,100)))

        # 用来测试用
        #time.sleep(1)
```

select版-TCP服务器

1. select 原理

在多路复用的模型中，比较常用的有select模型和epoll模型。这两个都是系统接口，由操作系统提供。当然，Python的select模块进行了更高级的封装。

网络通信被Unix系统抽象为文件的读写，通常是一个设备，由设备驱动程序提供，驱动可以知道自身的数据是否可用。支持阻塞操作的设备驱动通常会实现一组自身的等待队列，如读/写等待队列用于支持上层(用户层)所需的block或non-block操作。设备的文件的资源如果可用（可读或者可写）则会通知进程，反之则会让进程睡眠，等到数据到来可用的时候，再唤醒进程。

这些设备的文件描述符被放在一个数组中，然后select调用的时候遍历这个数组，如果对于的文件描述符可读则会返回改文件描述符。当遍历结束之后，如果仍然没有一个可用设备文件描述符，select让用户进程则会睡眠，直到等待资源可用的时候在唤醒，遍历之前那个监视的数组。每次遍历都是依次进行判断的。

2. select 回显服务器

使用python的select模块很容易写出下面一个echo(回显)服务器：

```
import select
import socket
import sys


server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server.bind(('', 7788))
server.listen(5)
```

```
inputs = [server, sys.stdin]

running = True

while True:

    # 调用 select 函数，阻塞等待
    readable, writeable, exceptional = select.select(inputs, [],

    # 数据抵达，循环
    for sock in readable:

        # 监听到有新的连接
        if sock == server:
            conn, addr = server.accept()
            # select 监听的socket
            inputs.append(conn)

        # 监听到键盘有输入
        elif sock == sys.stdin:
            cmd = sys.stdin.readline()
            running = False
            break

        # 有数据到达
        else:
            # 读取客户端连接发送的数据
            data = sock.recv(1024)
            if data:
                sock.send(data)
            else:
                # 移除select监听的socket
                inputs.remove(sock)
                sock.close()

    # 如果检测到用户输入敲击键盘，那么就退出
    if not running:
        break
```

```
server.close()
```

在windows中，使用‘网络调试助手’，进行连接服务器即可测试

另外一个服务器（包含writeList）：

```
#coding=utf-8
import socket
import Queue
from select import select
```

```
SERVER_IP = ('', 9999)

# 保存客户端发送过来的消息,将消息放入队列中
message_queue = {}
input_list = []
output_list = []

if __name__ == "__main__":
    server = socket.socket()
    server.bind(SERVER_IP)
    server.listen(10)
    # 设置为非阻塞
    server.setblocking(False)

    # 初始化将服务端加入监听列表
    input_list.append(server)

    while True:
        # 开始 select 监听,对input_list中的服务端server进行监听
        stdinput, stdoutput, stderr = select(input_list, output_

        # 循环判断是否有客户端连接进来,当有客户端连接进来时select将触发
        for obj in stdinput:
            # 判断当前触发的是不是服务端对象, 当触发的对象是服务端对象时,
            if obj == server:
                # 接收客户端的连接, 获取客户端对象和客户端地址信息
                conn, addr = server.accept()
                print("Client %s connected! "%str(addr))
                # 将客户端对象也加入到监听的列表中, 当客户端发送消息时 s
                input_list.append(conn)
                # 为连接的客户端单独创建一个消息队列, 用来保存客户端发送的
                message_queue[conn] = Queue.Queue()

            else:
                # 由于客户端连接进来时服务端接收客户端连接请求, 将客户端加
                # 所以判断是否是客户端对象触发
                try:
```

```
                    recv_data = obj.recv(1024)
                    # 客户端未断开
                    if recv_data:
                        print("received %s from client %s"%(recv
                        # 将收到的消息放入到各客户端的消息队列中
                        message_queue[obj].put(recv_data)

                        # 将回复操作放到output列表中，让select监听
                        if obj not in output_list:
                            output_list.append(obj)

                except ConnectionResetError:
                    # 客户端断开连接了，将客户端的监听从input列表中移除
                    input_list.remove(obj)
                    # 移除客户端对象的消息队列
                    del message_queue[obj]
                    print("\n[input] Client %s disconnected"%str

        # 如果现在没有客户端请求,也没有客户端发送消息时，开始对发送消息列表
        for sendobj in output_list:
            try:
                # 如果消息队列中有消息,从消息队列中获取要发送的消息
                if not message_queue[sendobj].empty():
                    # 从该客户端对象的消息队列中获取要发送的消息
                    send_data = message_queue[sendobj].get()
                    sendobj.send(send_data)
                else:
                    # 将监听移除等待下一次客户端发送消息
                    output_list.remove(sendobj)

            except ConnectionResetError:
                # 客户端连接断开了
                del message_queue[sendobj]
                output_list.remove(sendobj)
                print("\n[output] Client  %s disconnected"%str(a
```

3. 总结

优点

select目前几乎在所有的平台上支持，其良好跨平台支持也是它的一个优点。

缺点

select的一个缺点在于单个进程能够监视的文件描述符的数量存在最大限制，在Linux上一般为1024，可以通过修改宏定义甚至重新编译内核的方式提升这一限制，但是这样也会造成效率的降低。

一般来说这个数目和系统内存关系很大，具体数目可以cat /proc/sys/fs/file-max察看。32位机默认是1024个。64位机默认是2048.

对socket进行扫描时是依次扫描的，即采用轮询的方法，效率较低。

当套接字比较多的时候，每次select()都要通过遍历FD_SETSIZE个Socket来完成调度，不管哪个Socket是活跃的，都遍历一遍。这会浪费很多CPU时间。

epoll版-TCP服务器

1. epoll的优点：

1. 没有最大并发连接的限制，能打开的FD(指的是文件描述符，通俗的理解就是套接字对应的数字编号)的上限远大于1024
2. 效率提升，不是轮询的方式，不会随着FD数目的增加效率下降。只有活跃可用的FD才会调用callback函数；即epoll最大的优点就在于它只管你“活跃”的连接，而跟连接总数无关，因此在实际的网络环境中，epoll的效率就会远远高于select和poll。

2. epoll使用参考代码

```
import socket
import select

# 创建套接字
s = socket.socket(socket.AF_INET,socket.SOCK_STREAM)

# 设置可以重复使用绑定的信息
s.setsockopt(socket.SOL_SOCKET,socket.SO_REUSEADDR,1)

# 绑定本机信息
s.bind(("",7788))

# 变为被动
s.listen(10)

# 创建一个epoll对象
epoll=select.epoll()

# 测试，用来打印套接字对应的文件描述符
# print s.fileno()
```

```
# print select.EPOLLIN|select.EPOLLET

# 注册事件到epoll中
# epoll.register(fd[, eventmask])
# 注意，如果fd已经注册过，则会发生异常
# 将创建的套接字添加到epoll的事件监听中
epoll.register(s.fileno(),select.EPOLLIN|select.EPOLLET)


connections = {}
addresses = {}

# 循环等待客户端的到来或者对方发送数据
while True:

    # epoll 进行 fd 扫描的地方 -- 未指定超时时间则为阻塞等待
    epoll_list=epoll.poll()

    # 对事件进行判断
    for fd,events in epoll_list:

        # print fd
        # print events

        # 如果是socket创建的套接字被激活
        if fd == s.fileno():
            conn,addr=s.accept()

            print('有新的客户端到来%s'%str(addr))

            # 将 conn 和 addr 信息分别保存起来
            connections[conn.fileno()] = conn
            addresses[conn.fileno()] = addr

            # 向 epoll 中注册 连接 socket 的 可读 事件
            epoll.register(conn.fileno(), select.EPOLLIN | selec
```

```
        elif events == select.EPOLLIN:
            # 从激活 fd 上接收
            recvData = connections[fd].recv(1024)

            if len(recvData)>0:
                print('recv:%s'%recvData)
            else:
                # 从 epoll 中移除该 连接 fd
                epoll.unregister(fd)

                # server 侧主动关闭该 连接 fd
                connections[fd].close()

                print("%s---offline---"%str(addresses[fd]))
```

2. 说明

- EPOLLIN （可读）
- EPOLLOUT （可写）
- EPOLLET （ET模式）

epoll对文件描述符的操作有两种模式：LT（level trigger）和ET（edge trigger）。LT模式是默认模式，LT模式与ET模式的区别如下：

```
LT模式：当epoll检测到描述符事件发生并将此事件通知应用程序，应用程序可以不立

ET模式：当epoll检测到描述符事件发生并将此事件通知应用程序，应用程序必须立即
```

协程

协程，又称微线程，纤程。英文名Coroutine。

协程是啥

首先我们得知道协程是啥？协程其实可以认为是比线程更小的执行单元。 为啥说他是一个执行单元，因为他自带CPU上下文。这样只要在合适的时机，我们可以把一个协程 切换到另一个协程。 只要这个过程中保存或恢复 CPU上下文那么程序还是可以运行的。

通俗的理解：在一个线程中的某个函数，可以在任何地方保存当前函数的一些临时变量等信息，然后切换到另外一个函数中执行，注意不是通过调用函数的方式做到的，并且切换的次数以及什么时候再切换到原来的函数都由开发者自己确定

协程和线程差异

那么这个过程看起来比线程差不多。其实不然, 线程切换从系统层面远不止保存和恢复 CPU上下文这么简单。 操作系统为了程序运行的高效性每个线程都有自己缓存Cache等等数据，操作系统还会帮你做这些数据的恢复操作。所以线程的切换非常耗性能。但是协程的切换只是单纯的操作CPU的上下文，所以一秒钟切换个上百万次系统都抗的住。

协程的问题

但是协程有一个问题，就是系统并不感知，所以操作系统不会帮你做切换。那么谁来帮你做切换？让需要执行的协程更多的获得CPU时间才是问题的关键。

例子

目前的协程框架一般都是设计成 1:N 模式。所谓 1:N 就是一个线程作为一个容器里面放置多个协程。 那么谁来适时的切换这些协程？答案是有协程自己主动让出CPU，也就是每个协程池里面有一个调度器， 这个调度器是被动调度的。意思就是他不会主动调度。而且当一个协程发现自己执行不下去了(比如异步等待网络的数据回来，但是当前还没有数据到)， 这个时候就可以由这个协程通知调度器，这个时候执行到调度器的代码，调度器根据事先设计好的调度算法找到当前最需要CPU的协程。 切换这个协程的CPU上下文把CPU的运行权交个这个协程，直到这个协程出现执行不下去需要等等的情况，或者它调用主动让出CPU的API之类，触发下一次调度。

那么这个实现有没有问题?

其实是有问题的，假设这个线程中有一个协程是CPU密集型的他没有IO操作， 也就是自己不会主动触发调度器调度的过程，那么就会出现其他协程得不到执行的情况， 所以这种情况下需要程序员自己避免。这是一个问题，假设业务开发的人员并不懂这个原理的话就可能会出现问题。

协程的好处

在IO密集型的程序中由于IO操作远远慢于CPU的操作，所以往往需要CPU去等IO操作。 同步IO下系统需要切换线程，让操作系统可以在IO过程中执行其他的东西。 这样虽然代码是符合人类的思维习惯但是由于大量的线程切换带来了大量的性能的浪费，尤其是IO密集型的程序。

所以人们发明了异步IO。就是当数据到达的时候触发我的回调。来减少线程切换带来性能损失。 但是这样的坏处也是很大的，主要的坏处就是操作被“分片”了，代码写的不是“一气呵成”这种。 而是每次来段数据就要判断 数据够不够处理哇，够处理就处理吧，不够处理就在等等吧。这样代码的可读性很低，其实也不符合人类的习惯。

但是协程可以很好解决这个问题。比如 把一个IO操作 写成一个协程。当触发IO操作的时候就自动让出CPU给其他协程。要知道协程的切换很轻的。 协程通过这种对异步IO的封装 既保留了性能也保证了代码的容易编写和可读性。在高IO密集型的程序下很好。但是高CPU密集型的程序下没啥好处。

协程一个简单实现

```
import time

def A():
    while True:
        print("----A---")
        yield
        time.sleep(0.5)

def B(c):
    while True:
        print("----B---")
        c.next()
        time.sleep(0.5)

if __name__=='__main__':
    a = A()
    B(a)
```

运行结果：

```
--B--
--A--
--B--
--A--
--B--
--A--
--B--
--A--
--B--
--A--
--B--
--A--
...省略...
```

协程-greenlet版

为了更好使用协程来完成多任务，python中的greenlet模块对其封装，从而使得切换任务变的更加简单

安装方式

使用如下命令安装greenlet模块:

```
    sudo pip install greenlet
```

```
#coding=utf-8

from greenlet import greenlet
import time

def test1():
    while True:
        print "---A--"
        gr2.switch()
        time.sleep(0.5)

def test2():
    while True:
        print "---B--"
        gr1.switch()
        time.sleep(0.5)

gr1 = greenlet(test1)
gr2 = greenlet(test2)

#切换到gr1中运行
gr1.switch()
```

运行效果

```
---A--
---B--
---A--
---B--
---A--
---B--
---A--
---B--
...省略...
```

gevent

greenlet已经实现了协程，但是这个还的人工切换，是不是觉得太麻烦了，不要捉急，python还有一个比greenlet更强大的并且能够自动切换任务的模块 `gevent`

其原理是当一个greenlet遇到IO(指的是input output 输入输出，比如网络、文件操作等)操作时，比如访问网络，就自动切换到其他的greenlet，等到IO操作完成，再在适当的时候切换回来继续执行。

由于IO操作非常耗时，经常使程序处于等待状态，有了gevent为我们自动切换协程，就保证总有greenlet在运行，而不是等待IO

1. gevent的使用

```
#coding=utf-8

#请使用python 2 来执行此程序

import gevent

def f(n):
    for i in range(n):
        print gevent.getcurrent(), i

g1 = gevent.spawn(f, 5)
g2 = gevent.spawn(f, 5)
g3 = gevent.spawn(f, 5)
g1.join()
g2.join()
g3.join()
```

运行结果

```
<Greenlet at 0x10e49f550: f(5)> 0
<Greenlet at 0x10e49f550: f(5)> 1
<Greenlet at 0x10e49f550: f(5)> 2
<Greenlet at 0x10e49f550: f(5)> 3
<Greenlet at 0x10e49f550: f(5)> 4
<Greenlet at 0x10e49f910: f(5)> 0
<Greenlet at 0x10e49f910: f(5)> 1
<Greenlet at 0x10e49f910: f(5)> 2
<Greenlet at 0x10e49f910: f(5)> 3
<Greenlet at 0x10e49f910: f(5)> 4
<Greenlet at 0x10e49f4b0: f(5)> 0
<Greenlet at 0x10e49f4b0: f(5)> 1
<Greenlet at 0x10e49f4b0: f(5)> 2
<Greenlet at 0x10e49f4b0: f(5)> 3
<Greenlet at 0x10e49f4b0: f(5)> 4
```

可以看到，3个greenlet是依次运行而不是交替运行

2. gevent切换执行

```
import gevent

def f(n):
    for i in range(n):
        print gevent.getcurrent(), i
        #用来模拟一个耗时操作，注意不是time模块中的sleep
        gevent.sleep(1)

g1 = gevent.spawn(f, 5)
g2 = gevent.spawn(f, 5)
g3 = gevent.spawn(f, 5)
g1.join()
g2.join()
g3.join()
```

运行结果

```
<Greenlet at 0x7fa70ffa1c30: f(5)> 0
<Greenlet at 0x7fa70ffa1870: f(5)> 0
<Greenlet at 0x7fa70ffa1eb0: f(5)> 0
<Greenlet at 0x7fa70ffa1c30: f(5)> 1
<Greenlet at 0x7fa70ffa1870: f(5)> 1
<Greenlet at 0x7fa70ffa1eb0: f(5)> 1
<Greenlet at 0x7fa70ffa1c30: f(5)> 2
<Greenlet at 0x7fa70ffa1870: f(5)> 2
<Greenlet at 0x7fa70ffa1eb0: f(5)> 2
<Greenlet at 0x7fa70ffa1c30: f(5)> 3
<Greenlet at 0x7fa70ffa1870: f(5)> 3
<Greenlet at 0x7fa70ffa1eb0: f(5)> 3
<Greenlet at 0x7fa70ffa1c30: f(5)> 4
<Greenlet at 0x7fa70ffa1870: f(5)> 4
<Greenlet at 0x7fa70ffa1eb0: f(5)> 4
```

3个greenlet交替运行

3. gevent并发下载器

当然，实际代码里，我们不会用gevent.sleep()去切换协程，而是在执行到IO操作时，gevent自动切换，代码如下

```
#coding=utf-8

from gevent import monkey;
import gevent
import urllib2

#有IO才做时需要这一句
monkey.patch_all()

def myDownLoad(url):
    print('GET: %s' % url)
```

```
    resp = urllib2.urlopen(url)
    data = resp.read()
    print('%d bytes received from %s.' % (len(data), url))

gevent.joinall([
        gevent.spawn(myDownLoad, 'http://www.baidu.com/'),
        gevent.spawn(myDownLoad, 'http://www.itcast.cn/'),
        gevent.spawn(myDownLoad, 'http://www.itheima.com/'),
])
```

运行结果

```
GET: http://www.baidu.com/
GET: http://www.itcast.cn/
GET: http://www.itheima.com/
102247 bytes received from http://www.baidu.com/.
166903 bytes received from http://www.itheima.com/.
162294 bytes received from http://www.itcast.cn/.
```

从上能够看到是先发送的获取baidu的相关信息，然后依次是itcast、itheima，但是收到数据的先后顺序不一定与发送顺序相同，这也就体现出了异步，即不确定什么时候会收到数据，顺序不一定

gevent版-TCP服务器

```
import sys
import time
import gevent

from gevent import socket,monkey
monkey.patch_all()

def handle_request(conn):
    while True:
        data = conn.recv(1024)
        if not data:
            conn.close()
            break
        print("recv:", data)
        conn.send(data)


def server(port):
    s = socket.socket()
    s.bind(('', port))
    s.listen(5)
    while True:
        cli, addr = s.accept()
        gevent.spawn(handle_request, cli)

if __name__ == '__main__':
    server(7788)
```