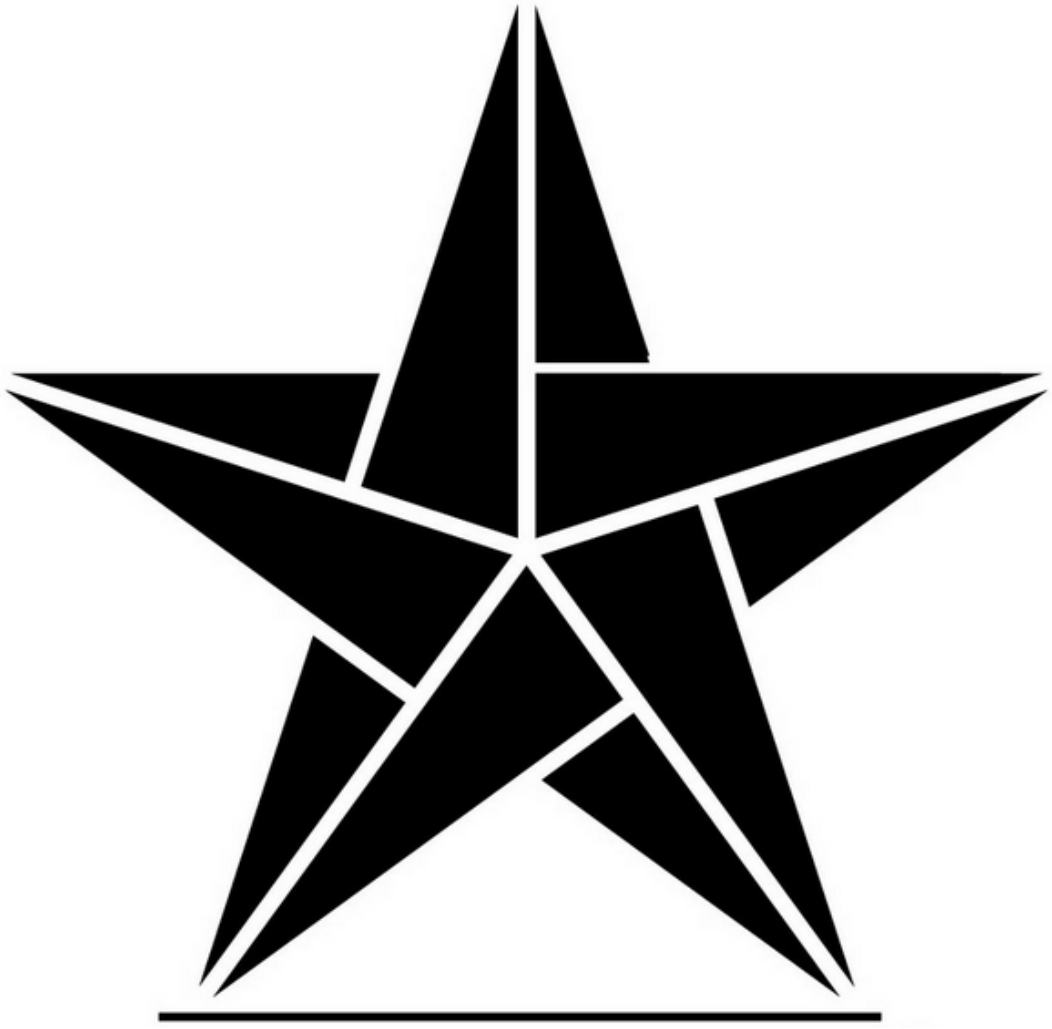
Wireshark 使用说明



版本号：20120905-A

目　　录

◆ 简介

1.1.、什么是 Wireshark

Wireshark 是网络包分析工具。网络包分析工具的主要作用是尝试捕获网络包， 并尝试显示包的尽可能详细的情况。过去的此类工具要么是过于昂贵，要么是属于某人私有，或者是二者兼顾。Wireshark 出现以后，这种现状得以改变。

2.1、主要应用

网络管理员用来解决网络通讯问题；

网络安全工程师用来检测已知的和未知的安全隐患；

开发人员用来测试协议执行情况；

用来学习网络协议；

除了上面提到的，Wireshark 还可以用在其它许多场合。

1.1.2.特性

支持 UNIX 和 Windows 平台；

在接口实时捕捉包；

能详细显示包的详细协议信息；

可以打开/保存捕捉的包；

可以导入导出其他捕捉程序支持的包数据格式；

可以通过多种方式过滤包；

多种方式查找包；

通过过滤以多种色彩显示包；

创建多种统计分析；

捕捉多种网络接口；

支持多种其它程序捕捉的文件；

支持多格式输出；

对多种协议解码提供支持；

开源软件；

不管怎么说，要想真正了解它的强大，您还得使用它才行！

◆ 安装

2.1、windows 平台上的安装

在互联网搜索到我们想要安装的软件版本，执行安装包后弹出如下欢迎界面：

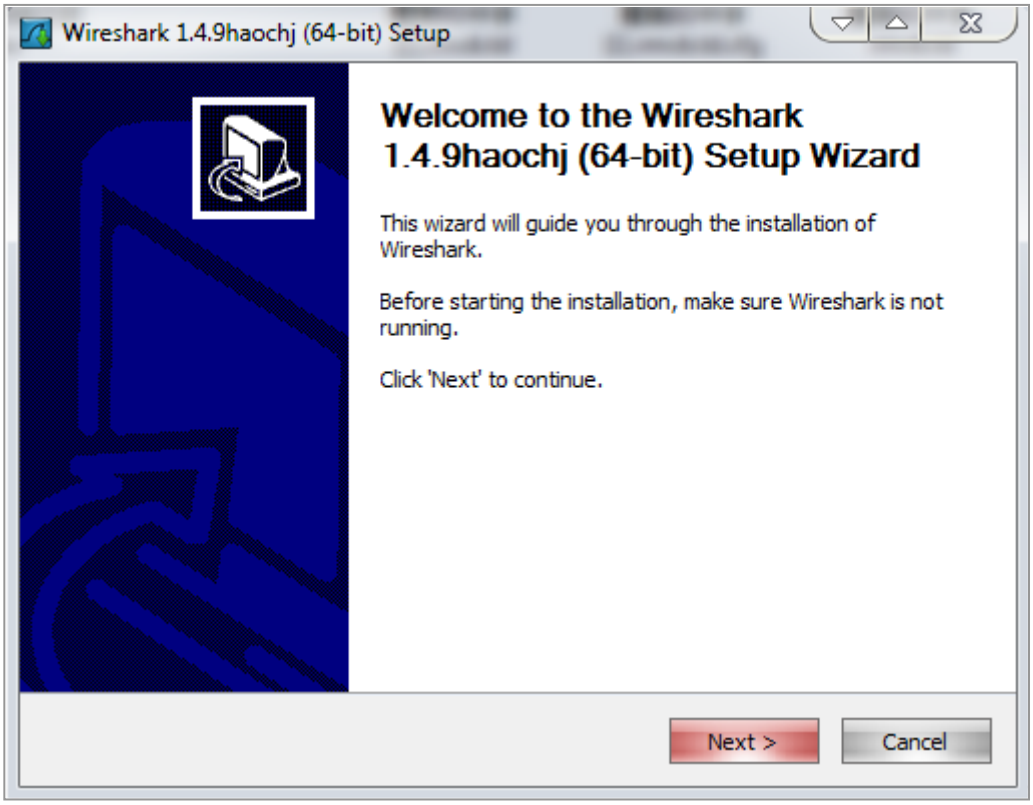


图 2-1-1

选择“Next”我们进入到下一步安装步骤；

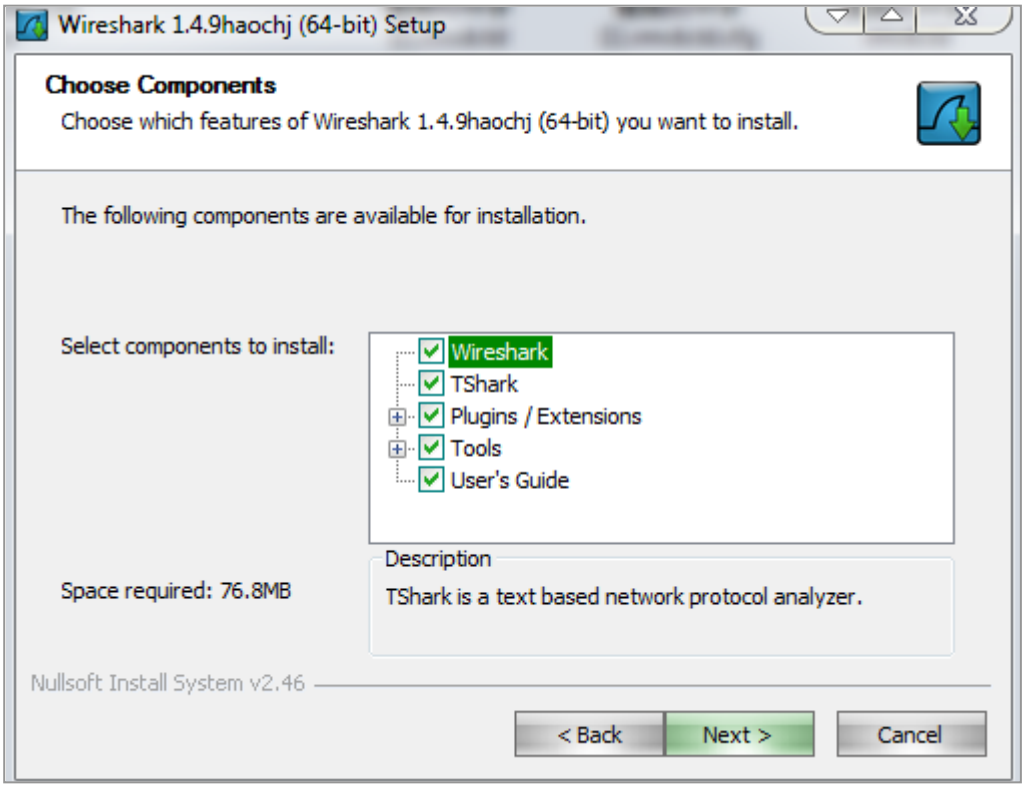


图 2-1-2

如图 2-1-2 这一步 wireshark 会询问我们需要安装哪些程序组件。我们可以选择性的安装我们必须的一些 wireshark 附带的工具；

通常使用到的有 :wireshark 主程序(图形界面)、TSshark(基于命令行的)、plugins/extensions（ wireshark 以及 TSshark 的一些扩展插件 ）等...在此我们选择安装全部组件程序后点击“Next”进入到下一步安装；

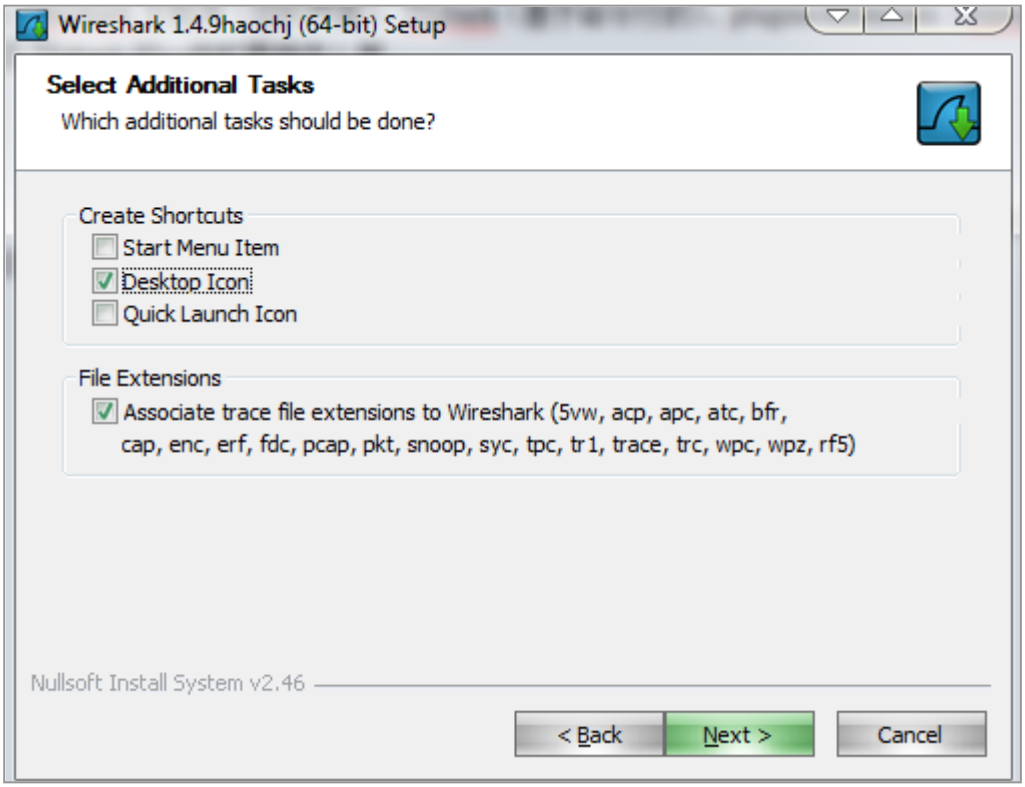


图 2-1-3

如图 2-1-3 所示在此界面 wireshark 安装程序会询问我们是否要在开始菜单（Start Menu Item）、桌面快捷方式（Desktop Icon）、快速启动栏（Quick Launch Icon）以及是否关联所列出的后缀名文件。在此我们选择建立桌面快捷方式并且关联到列出后缀名文件后点击“Next”进入到下一步安装；

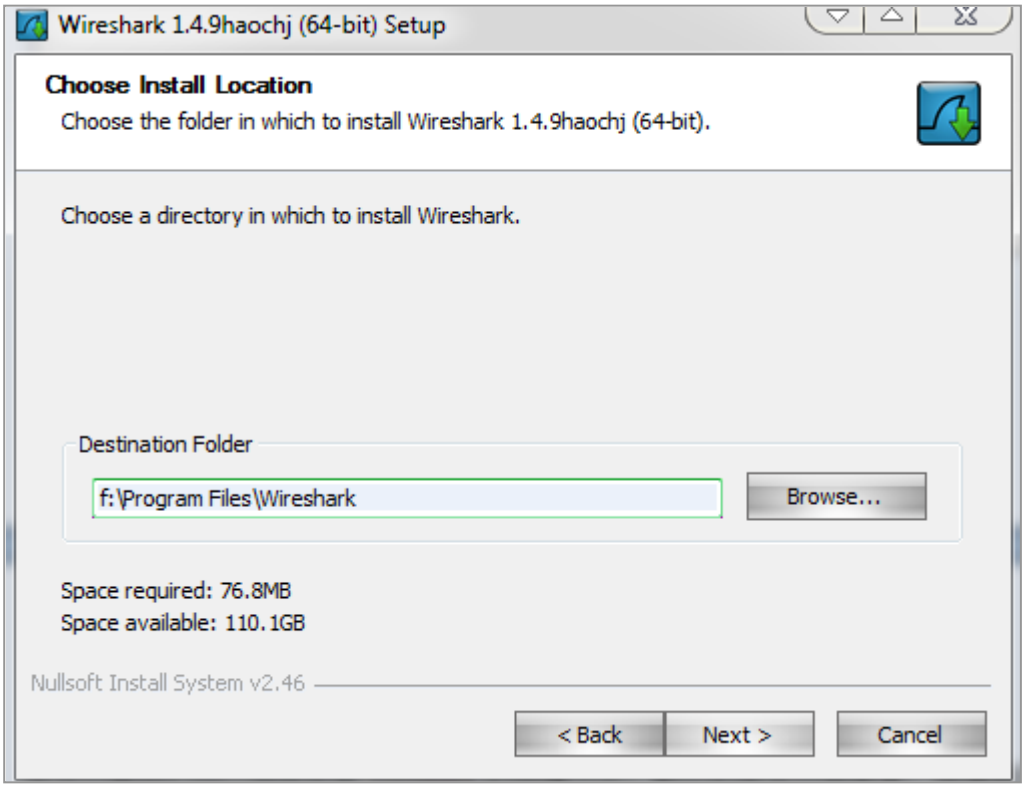


图 2-1-4

如图 2-1-4 所示 wireshark 询问我们的安装位置。我们选择 F 盘点击“Next”；

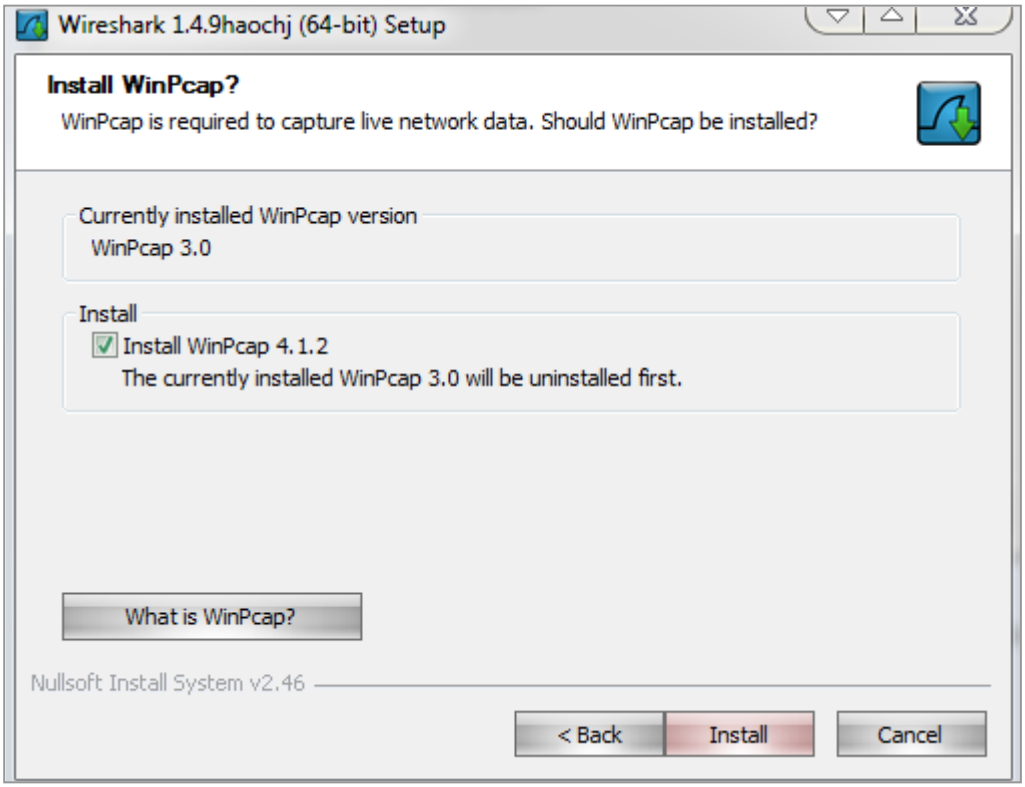


图 2-1-5

图 2-1-5 所示，wireshark 安装程序询问我们是否安装依赖软件 WinPcap 3.0（该软件主要用于捕获网络底层数据包），假如您的电脑上已经安装了该软件，那么我们可以取消掉该软甲的安装。

在此处由于我们没有安装过该软件我们勾选该复选框后点击“Install”执行安装；

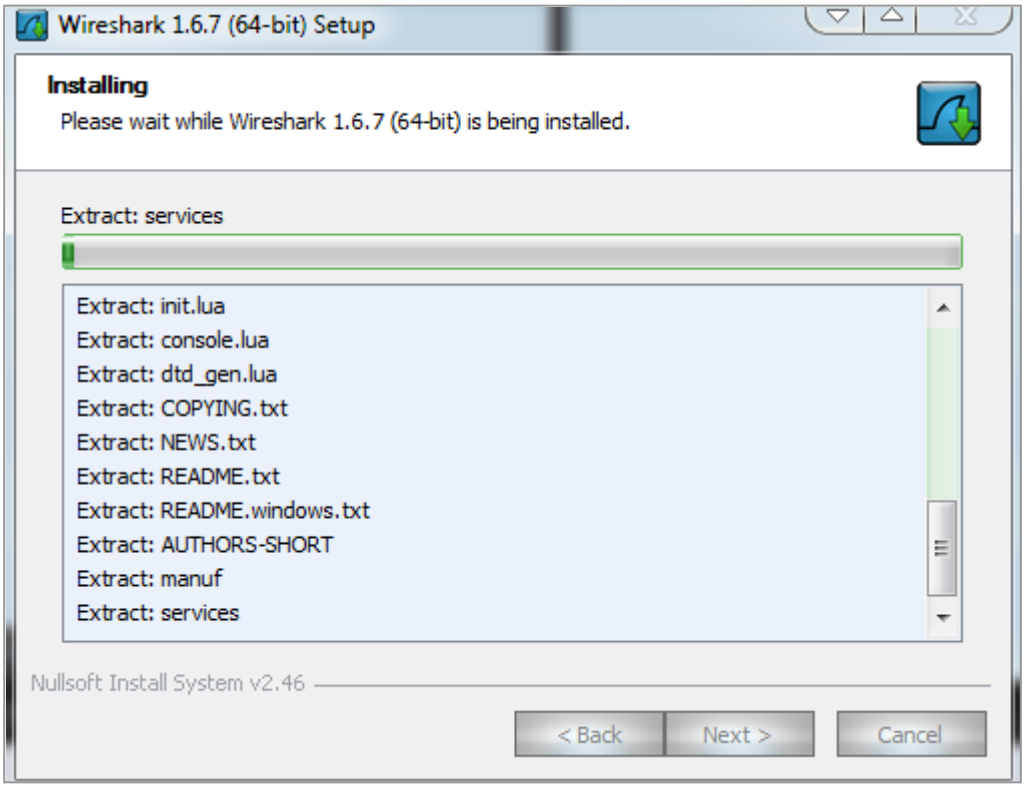

Wireshark 1.6.7 (64-bit) Setup
Installing
Please wait while Wireshark 1.6.7 (64-bit) is being installed.
Extract: services
Extract: init.lua
Extract: console.lua
Extract: dtd_gen.lua
Extract: COPYING.txt
Extract: NEWS.txt
Extract: README.txt
Extract: README.windows.txt
Extract: AUTHORS-SHORT
Extract: manuf
Extract: services
Nullsoft Install System v2.46
< Back
Next >
Cancel

图 2-1-6

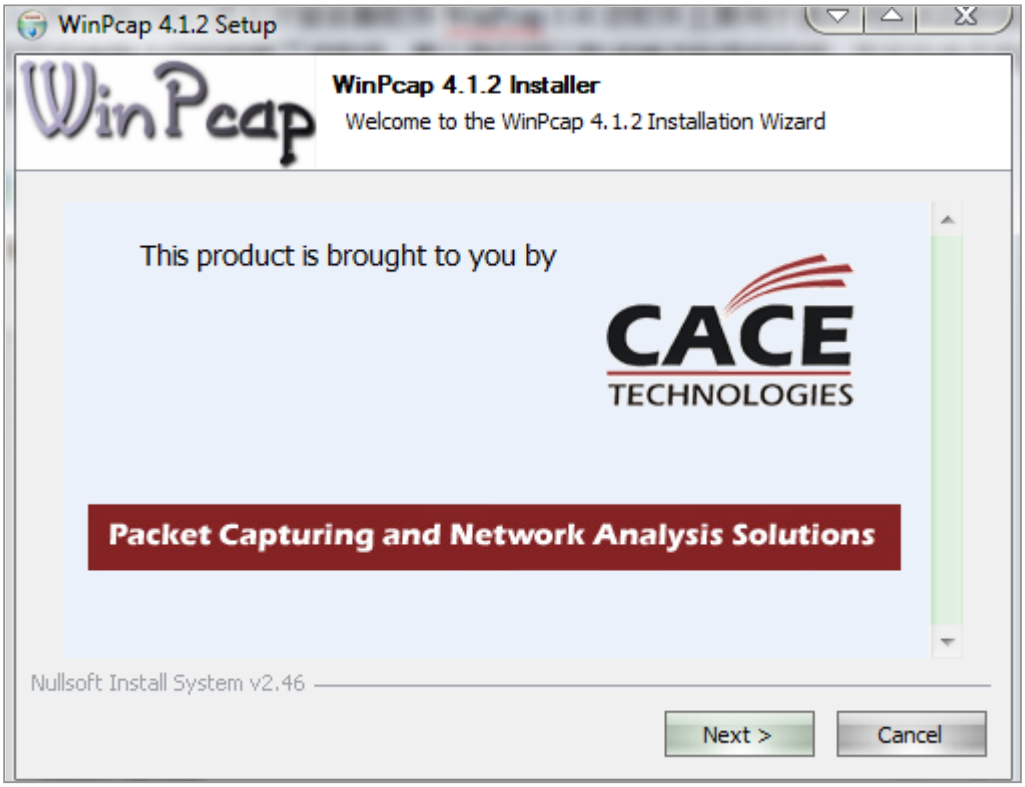

WinPcap 4.1.2 Setup
WinPcap
WinPcap 4.1.2 Installer
Welcome to the WinPcap 4.1.2 Installation Wizard
This product is brought to you by
CACE
TECHNOLOGIES
Packet Capturing and Network Analysis Solutions
Nullsoft Install System v2.46
Next >
Cancel

图 2-1-7

如图 2-1-7 所示我们进入到 WinPcap 的安装界面点击“Next”进入到下一步安装；

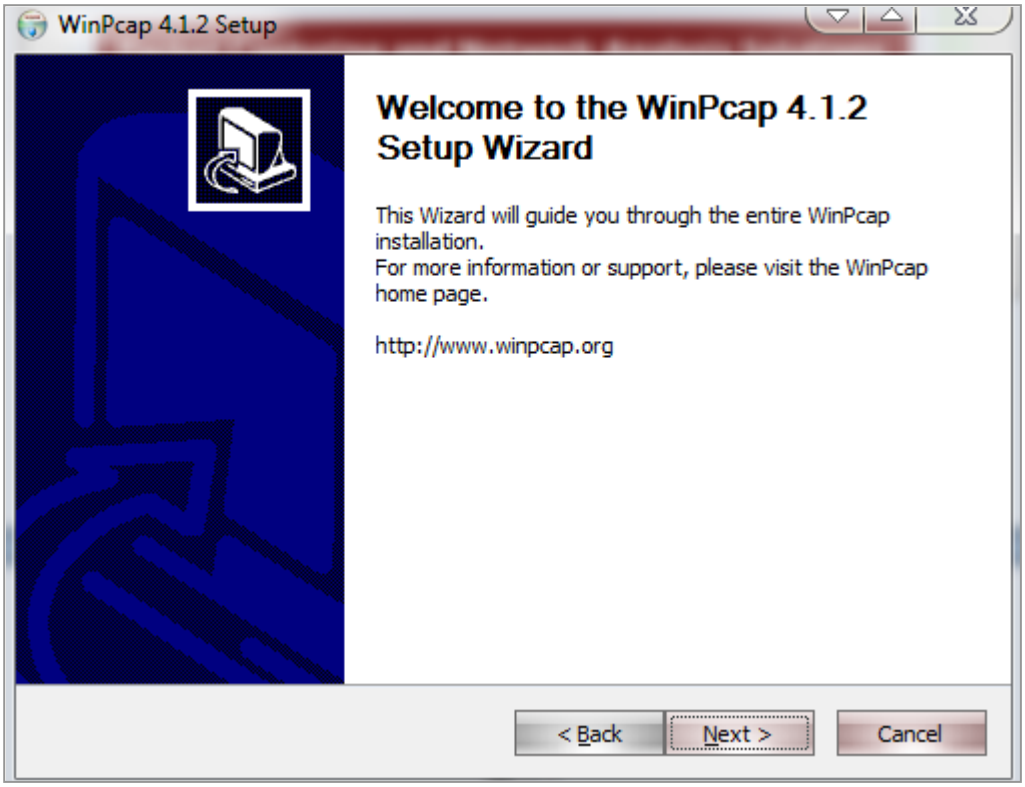


图 2-1-8

如图 2-1-8 点击“Next”进行到 WinPcap 的下一步安装操作；

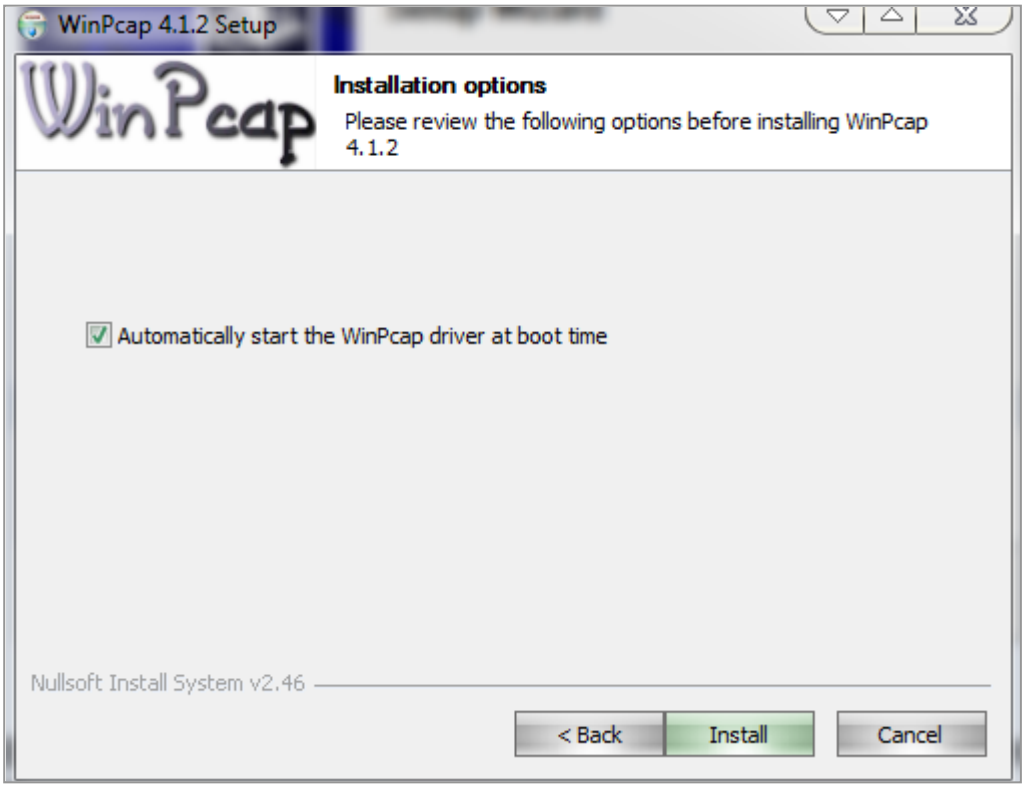


图 2-1-9

图 2-1-9 所示，WinPcap 询问我们是否开机自动启动 WinPcap 驱动程序。我们采取默认选项后点击“Install”；

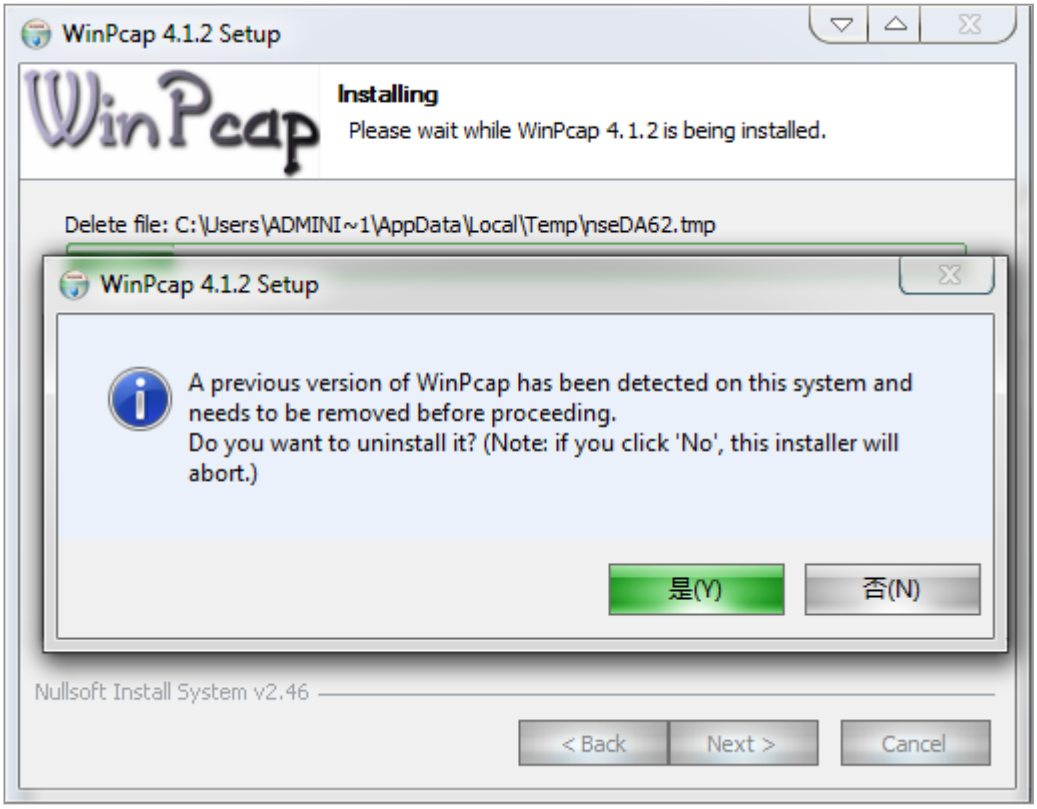


图 2-1-10

假如出现图 2-1-10 的提示，那么就是您的电脑已经安装了该类软件，WinPcap 询问我们是否卸载以前所安装的 WinPcap。在此我们选择“是”后 WinPcap 开始自动安装；

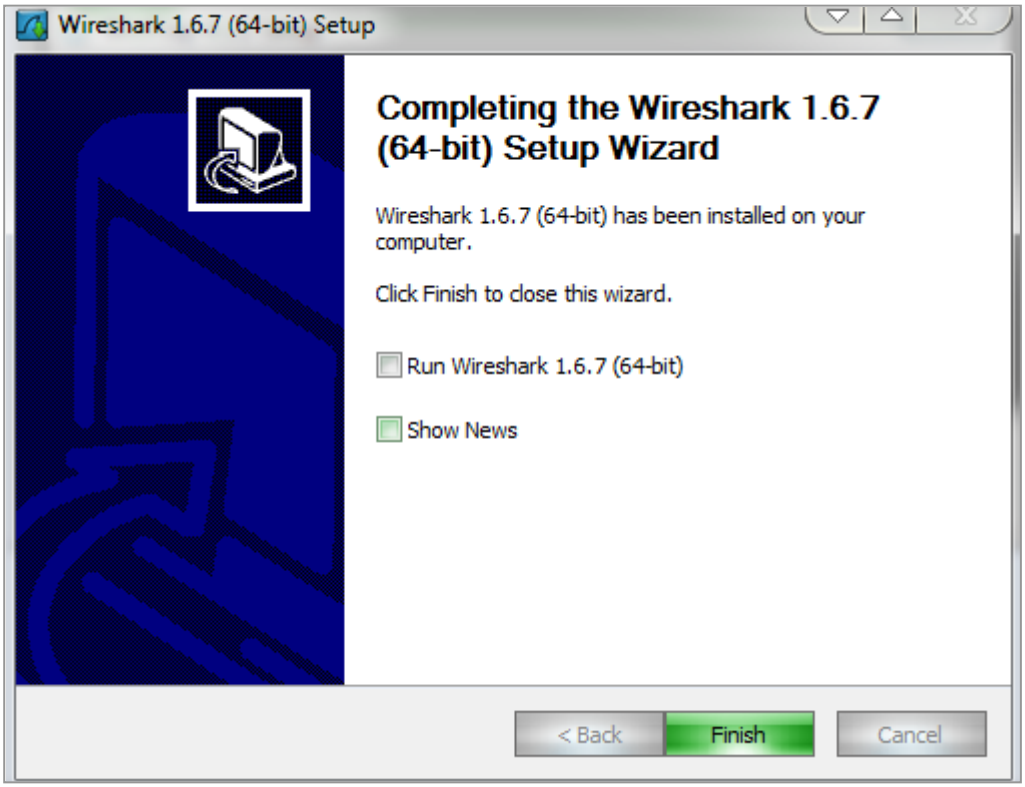


图 2-1-11

到了如图 2-1-11 所示的界面后说名 wireshark 已经安装成功了，点击“Finish”后在桌面我们可以看到 wireshark 的快捷方式；

2.2、linux 平台上的安装

2.2.1、RedHat 版本

2.2.1.1、tcpdump 源码安装方式

首先安装 tcpdump 需要先安装 libpcap，把我们下载到的 tcpdump 以及 libpcap 安装包放在 linux 一个目录下面，然后我们使用 cd 进入该目录后执行 libpcap 的解压操作。如下图 2-2-1，2-2-2 所示：

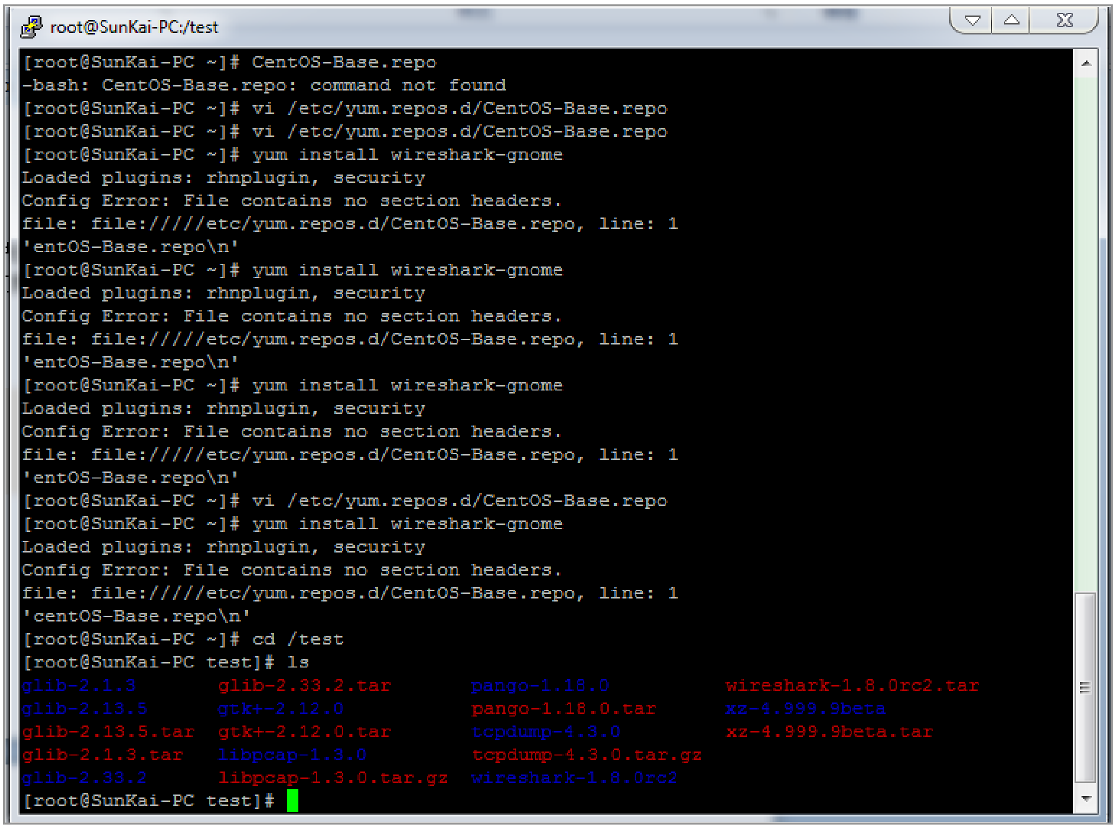


图 2-2-1

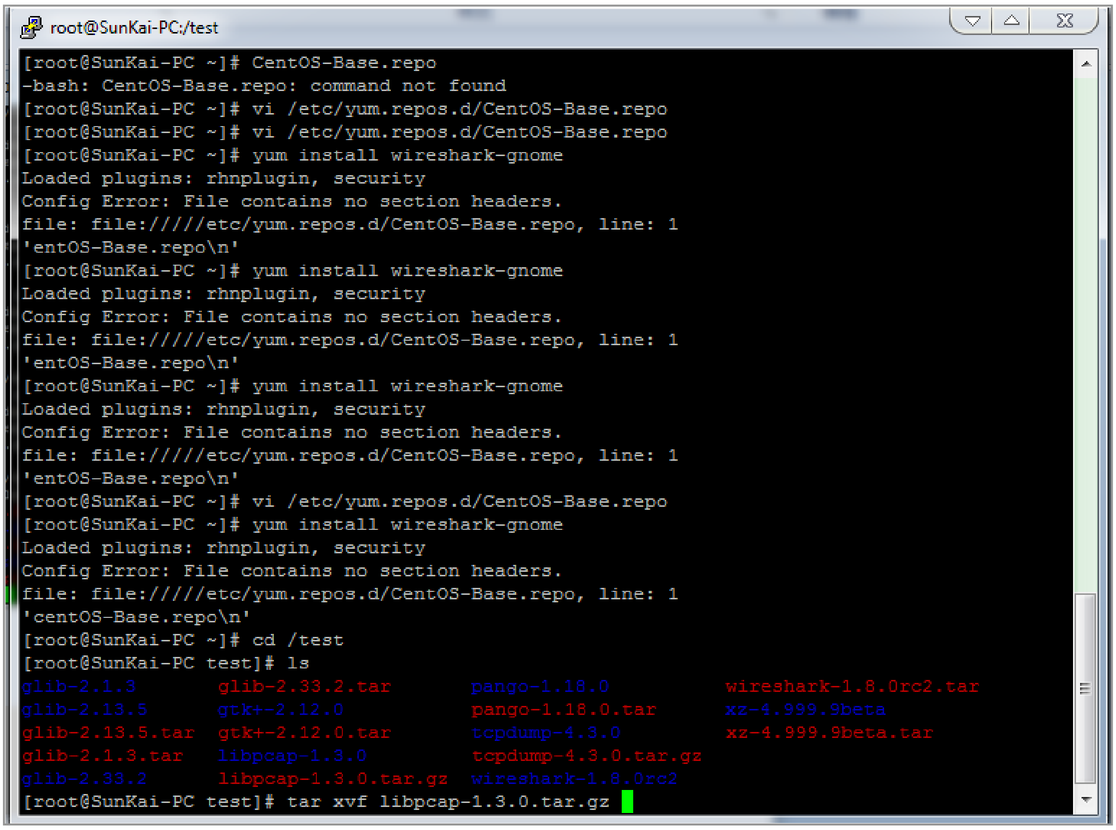


图 2-2-2

解压完毕后的；libpca 目录如下图 2-2-3 所示：

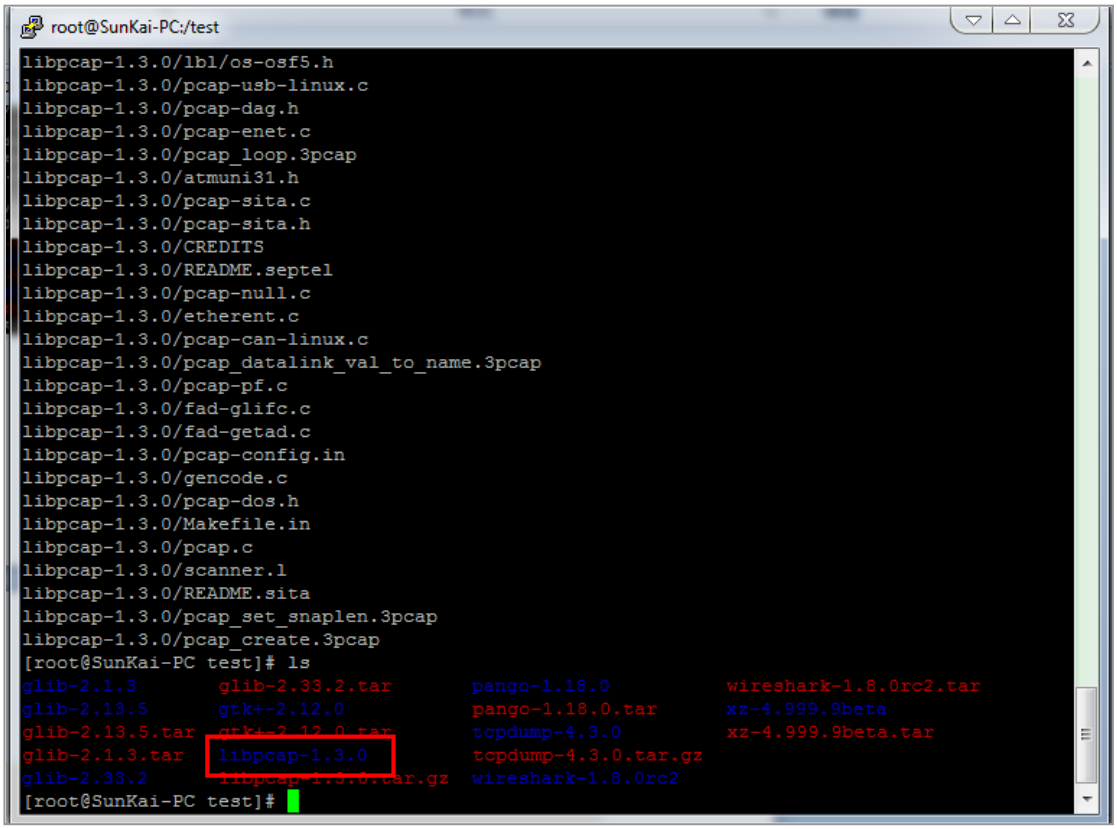


图 2-2-3

我们进入到该目录，然后执行./configure 然后执行 make&make install 就开始安装了；安装完毕后的正常状态如下图 2-2-4 所示：

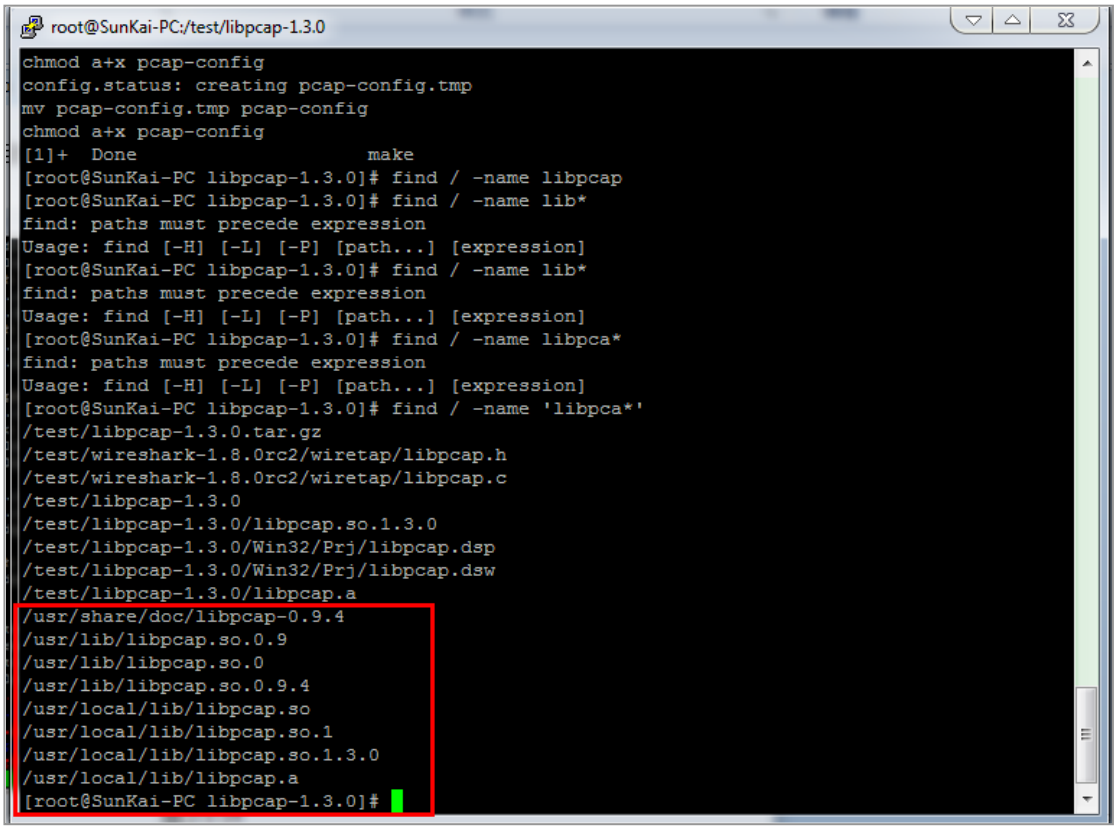


图 2-2-4

然后我们进入到 tcpdump 文件的存放目录，再次执行解压操作；如下图 2-2-5，2-2-6 所示：

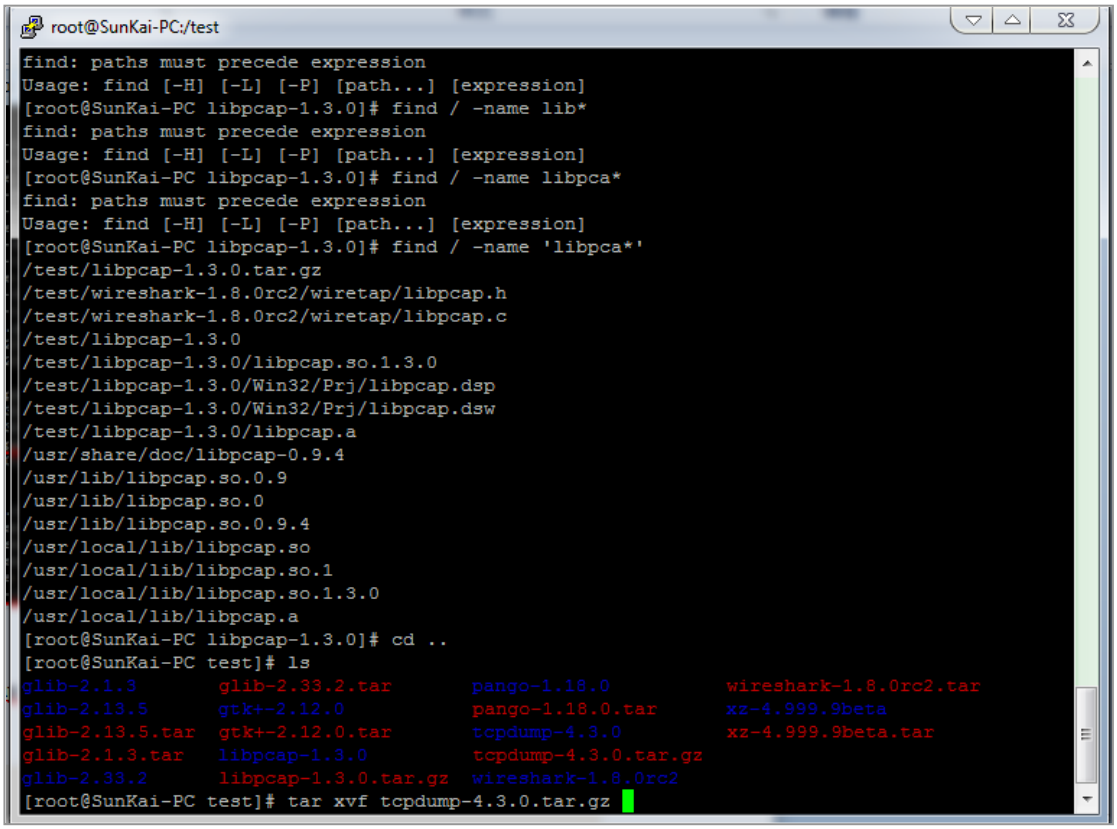

root@SunKai-PC:/test
find: paths must precede expression
Usage: find [-H] [-L] [-P] [path...] [expression]
[root@SunKai-PC libpcap-1.3.0]# find / -name lib*
find: paths must precede expression
Usage: find [-H] [-L] [-P] [path...] [expression]
[root@SunKai-PC libpcap-1.3.0]# find / -name libpca*
find: paths must precede expression
Usage: find [-H] [-L] [-P] [path...] [expression]
[root@SunKai-PC libpcap-1.3.0]# find / -name 'libpca*'
/test/libpcap-1.3.0.tar.gz
/test/wireshark-1.8.0rc2/wiretap/libpcap.h
/test/wireshark-1.8.0rc2/wiretap/libpcap.c
/test/libpcap-1.3.0
/test/libpcap-1.3.0/libpcap.so.1.3.0
/test/libpcap-1.3.0/Win32/Prj/libpcap.dsp
/test/libpcap-1.3.0/Win32/Prj/libpcap.dsw
/test/libpcap-1.3.0/libpcap.a
/usr/share/doc/libpcap-0.9.4
/usr/lib/libpcap.so.0.9
/usr/lib/libpcap.so.0
/usr/lib/libpcap.so.0.9.4
/usr/local/lib/libpcap.so
/usr/local/lib/libpcap.so.1
/usr/local/lib/libpcap.so.1.3.0
/usr/local/lib/libpcap.a
[root@SunKai-PC libpcap-1.3.0]# cd ..
[root@SunKai-PC test]# ls
glib-2.1.3 glib-2.33.2.tar pango-1.18.0 wireshark-1.8.0rc2.tar
glib-2.13.5 gtk+-2.12.0 pango-1.18.0.tar xz-4.999.9beta
glib-2.13.5.tar gtk+-2.12.0.tar tcpdump-4.3.0 xz-4.999.9beta.tar
glib-2.1.3.tar libpcap-1.3.0 tcpdump-4.3.0.tar.gz
glib-2.33.2 libpcap-1.3.0.tar.gz wireshark-1.8.0rc2
[root@SunKai-PC test]# tar xvf tcpdump-4.3.0.tar.gz

图 2-2-5

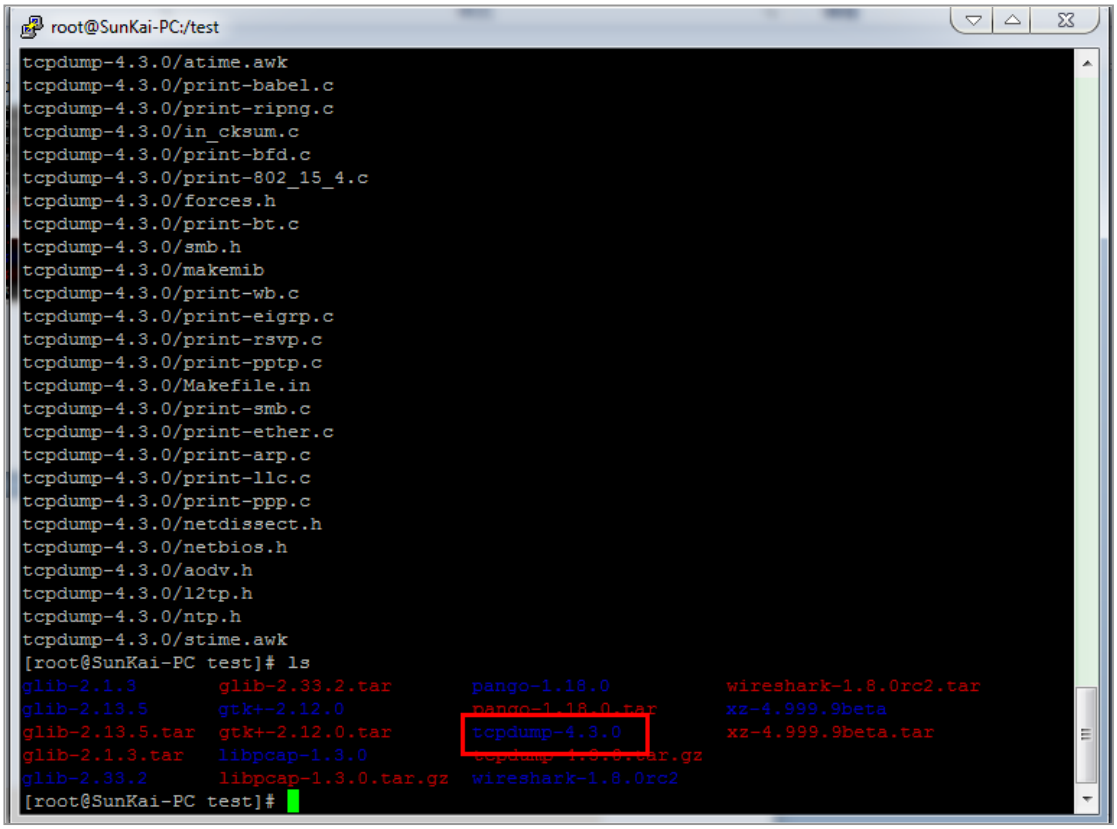

root@SunKai-PC:/test
tcpdump-4.3.0/atime.awk
tcpdump-4.3.0/print-babel.c
tcpdump-4.3.0/print-ripng.c
tcpdump-4.3.0/in_cksum.c
tcpdump-4.3.0/print-bfd.c
tcpdump-4.3.0/print-802_15_4.c
tcpdump-4.3.0/forces.h
tcpdump-4.3.0/print-bt.c
tcpdump-4.3.0/smb.h
tcpdump-4.3.0/makemib
tcpdump-4.3.0/print-wb.c
tcpdump-4.3.0/print-eigrp.c
tcpdump-4.3.0/print-rsvp.c
tcpdump-4.3.0/print-pptp.c
tcpdump-4.3.0/Makefile.in
tcpdump-4.3.0/print-smb.c
tcpdump-4.3.0/print-ether.c
tcpdump-4.3.0/print-arp.c
tcpdump-4.3.0/print-llc.c
tcpdump-4.3.0/print-ppp.c
tcpdump-4.3.0/netdissect.h
tcpdump-4.3.0/netbios.h
tcpdump-4.3.0/aodv.h
tcpdump-4.3.0/l2tp.h
tcpdump-4.3.0/ntp.h
tcpdump-4.3.0/stime.awk
[root@SunKai-PC test]# ls
glib-2.1.3 glib-2.33.2.tar pango-1.18.0 wireshark-1.8.0rc2.tar
glib-2.13.5 gtk+-2.12.0 pango-1.18.0.tar xz-4.999.9beta
glib-2.13.5.tar gtk+-2.12.0.tar tcpdump-4.3.0 xz-4.999.9beta.tar
glib-2.1.3.tar libpcap-1.3.0
glib-2.33.2 libpcap-1.3.0.tar.gz wireshark-1.8.0rc2
[root@SunKai-PC test]#

图 2-2-6

进入到 tcpdump 目录下后我们一样执行./configure，完毕后我们再次执行：make&make install；安装完毕后我们就可以使用该软件了；如下图 2-2-7，2-2-8 所示：

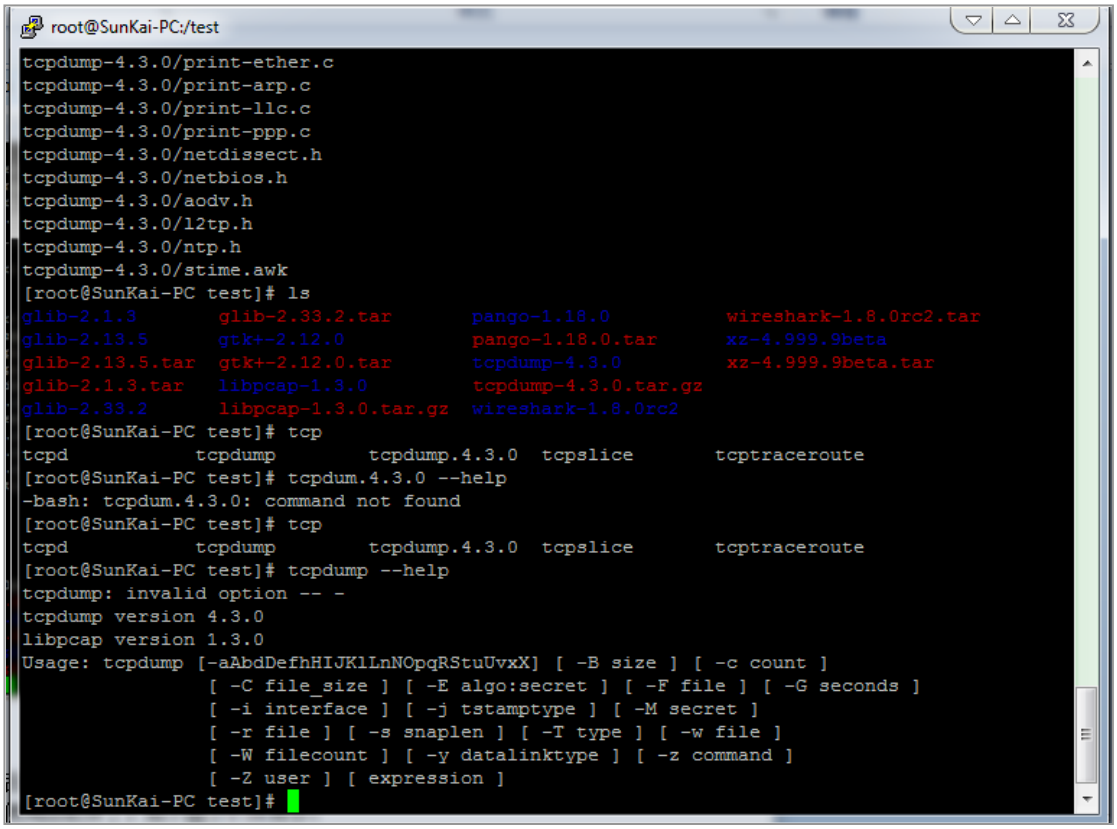


图 2-2-7

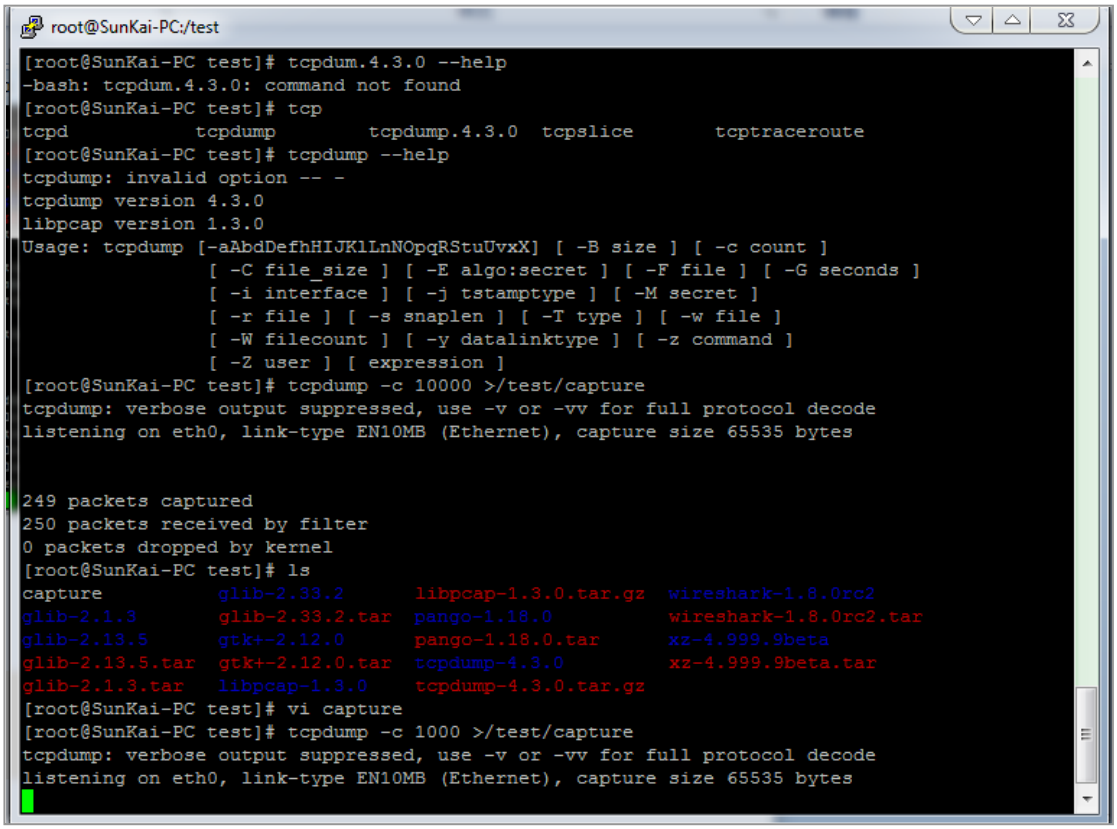


图 2-2-8

Tcpdum 软件很早之前就有，在此只为了让大家比较一下 wireshark 以及 tcpdump 的一些不同之处。如果大家对于 tcpdump 有兴趣也可以自己去百度一下；我们在这里就不多做赘述了。

2.2.2.2、Linux yum 安装方式

首先输入 yum install wireshark-gnome,启动 wireshark 以及依赖软件包的下载；如图 2-2-9，2-2-10 所示：

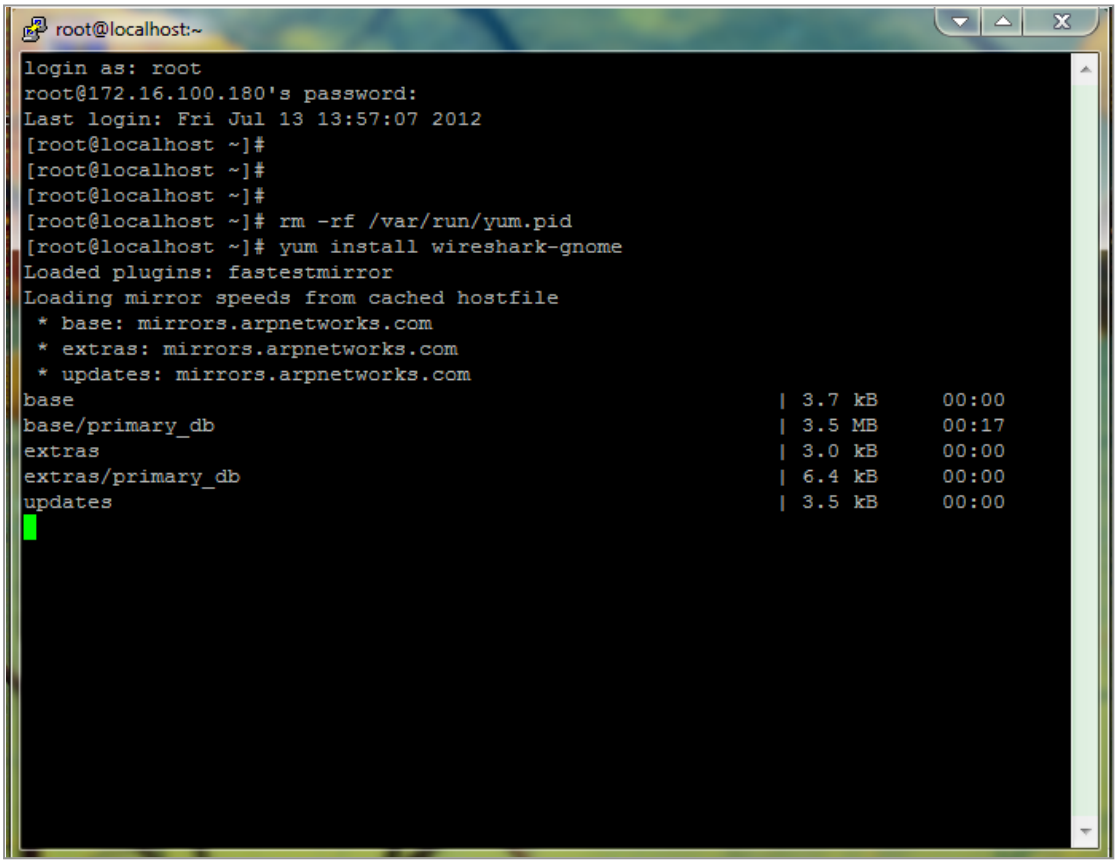


图 2-2-9

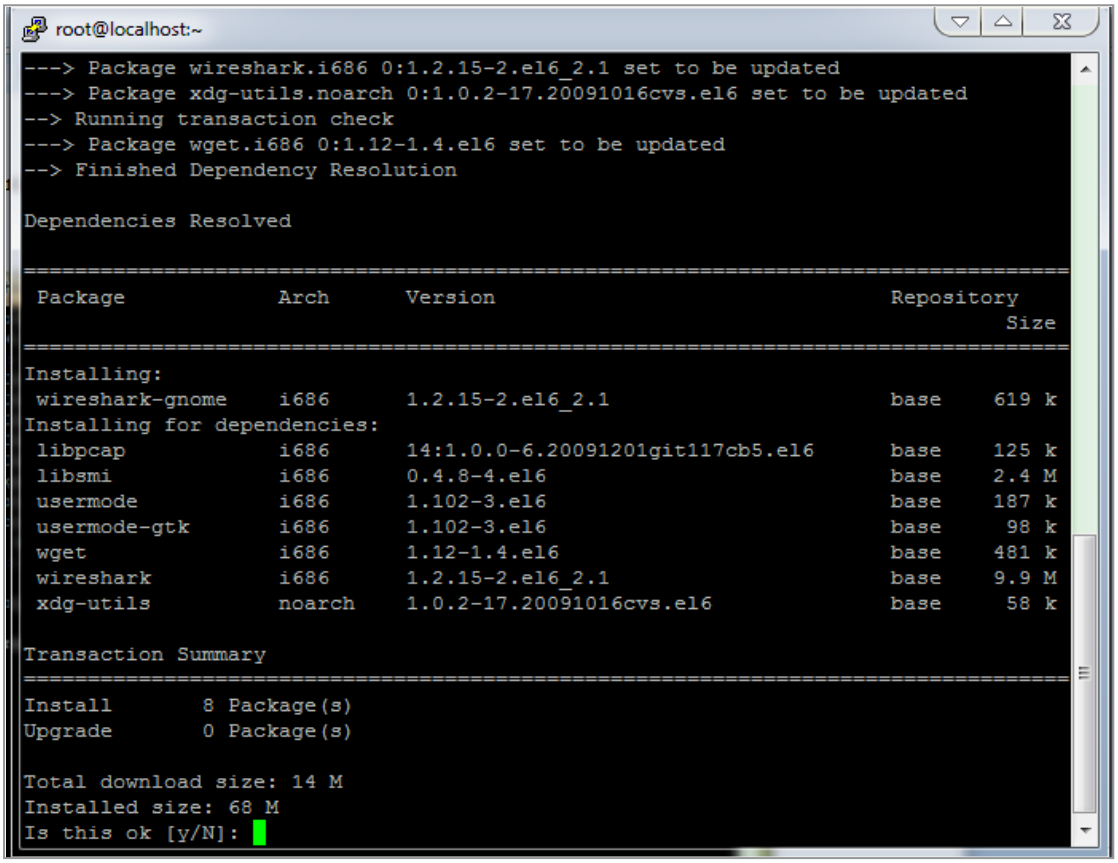


图 2-2-10

如上图 2-2-10，我们输入 y 回车后 wireshark 开始自动下载 wireshark 以及依赖软件；

下载完后系统会提示是否执行 wireshark 以及一些相关软件的安装 如下图 2-2-11,2-2-12 所示：

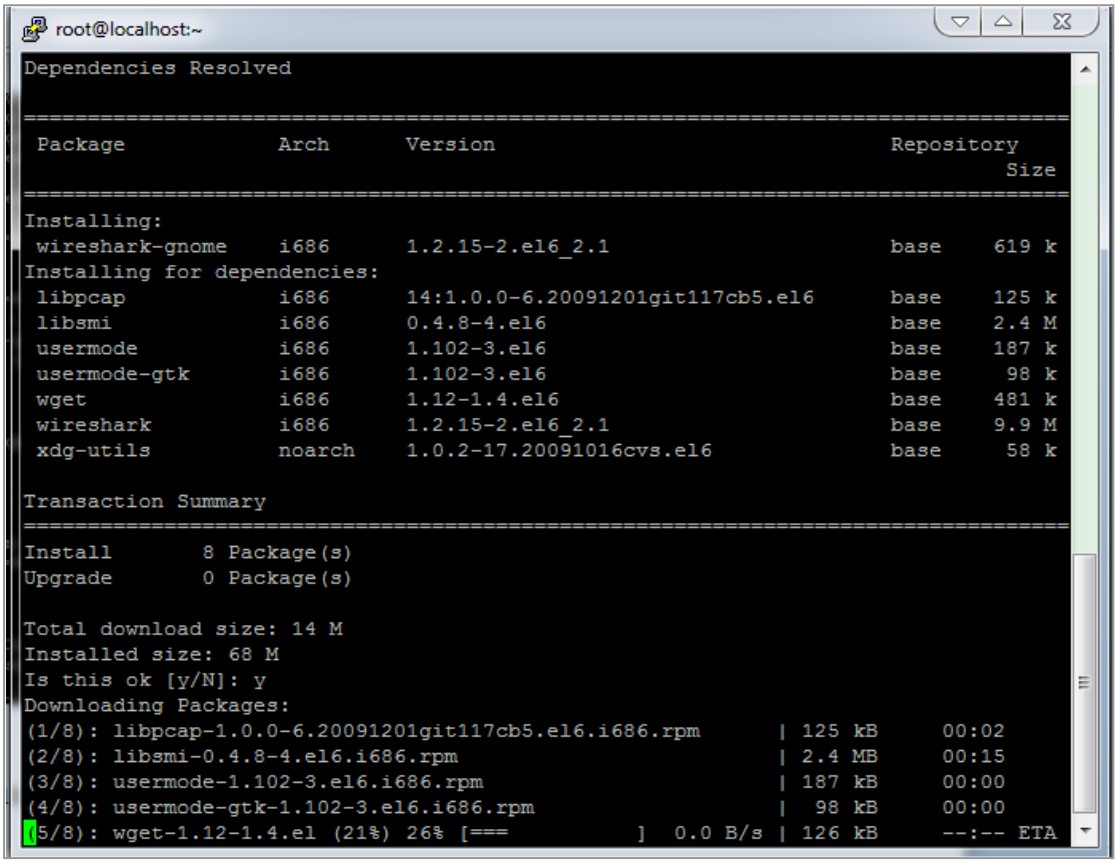

root@localhost:~
Dependencies Resolved
Package Arch Version Repository Size
Installing:
wireshark-gnome i686 1.2.15-2.el6_2.1 base 619 k
Installing for dependencies:
libpcap i686 14:1.0.0-6.20091201git117cb5.el6 base 125 k
libsmi i686 0.4.8-4.el6 base 2.4 M
usermode i686 1.102-3.el6 base 187 k
usermode-gtk i686 1.102-3.el6 base 98 k
wget i686 1.12-1.4.el6 base 481 k
wireshark i686 1.2.15-2.el6_2.1 base 9.9 M
xdg-utils noarch 1.0.2-17.20091016cvs.el6 base 58 k
Transaction Summary
Install 8 Package(s)
Upgrade 0 Package(s)
Total download size: 14 M
Installed size: 68 M
Is this ok [y/N]: y
Downloading Packages:
(1/8): libpcap-1.0.0-6.20091201git117cb5.el6.i686.rpm | 125 kB 00:02
(2/8): libsmi-0.4.8-4.el6.i686.rpm | 2.4 MB 00:15
(3/8): usermode-1.102-3.el6.i686.rpm | 187 kB 00:00
(4/8): usermode-gtk-1.102-3.el6.i686.rpm | 98 kB 00:00
(5/8): wget-1.12-1.4.el (21%) 26% [===] 0.0 B/s | 126 kB --:-- ETA

图 2-2-11

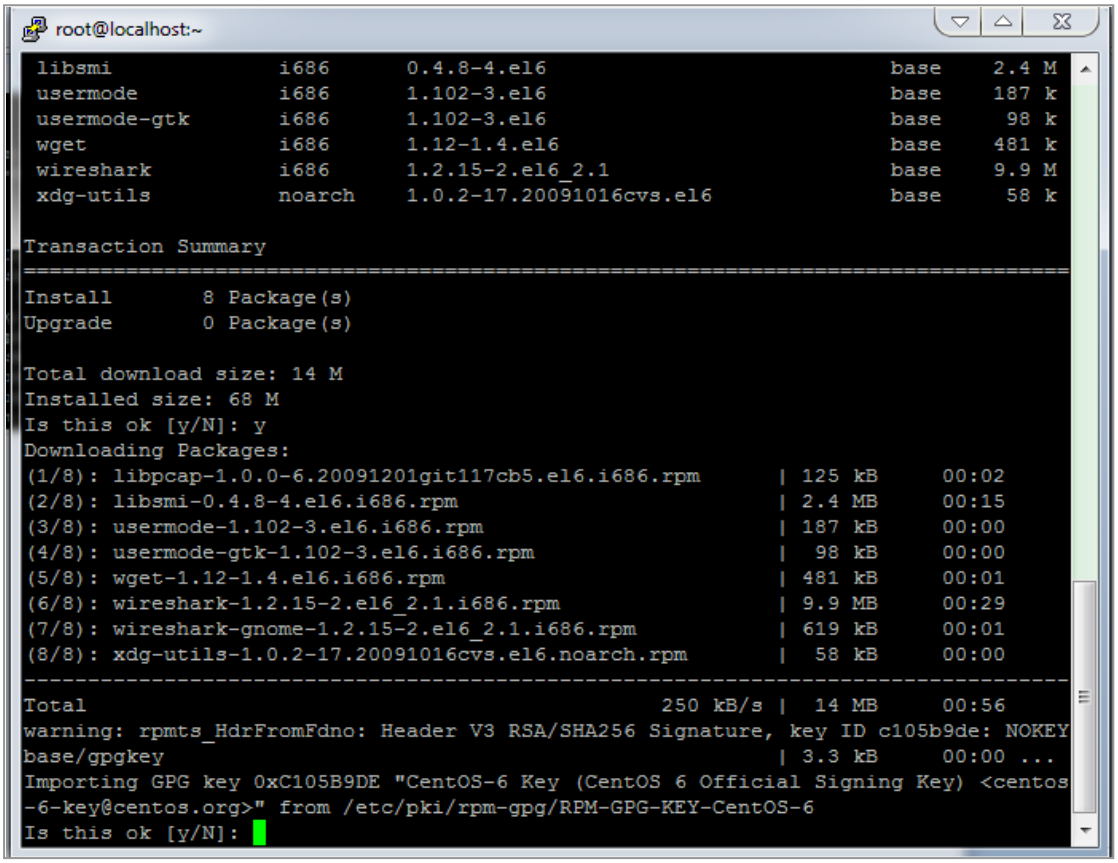


图 2-2-12

我们输入 y 回车后系统将自动执行一系列的软件安装；如下图 2-2-13，2-2-14 所示：

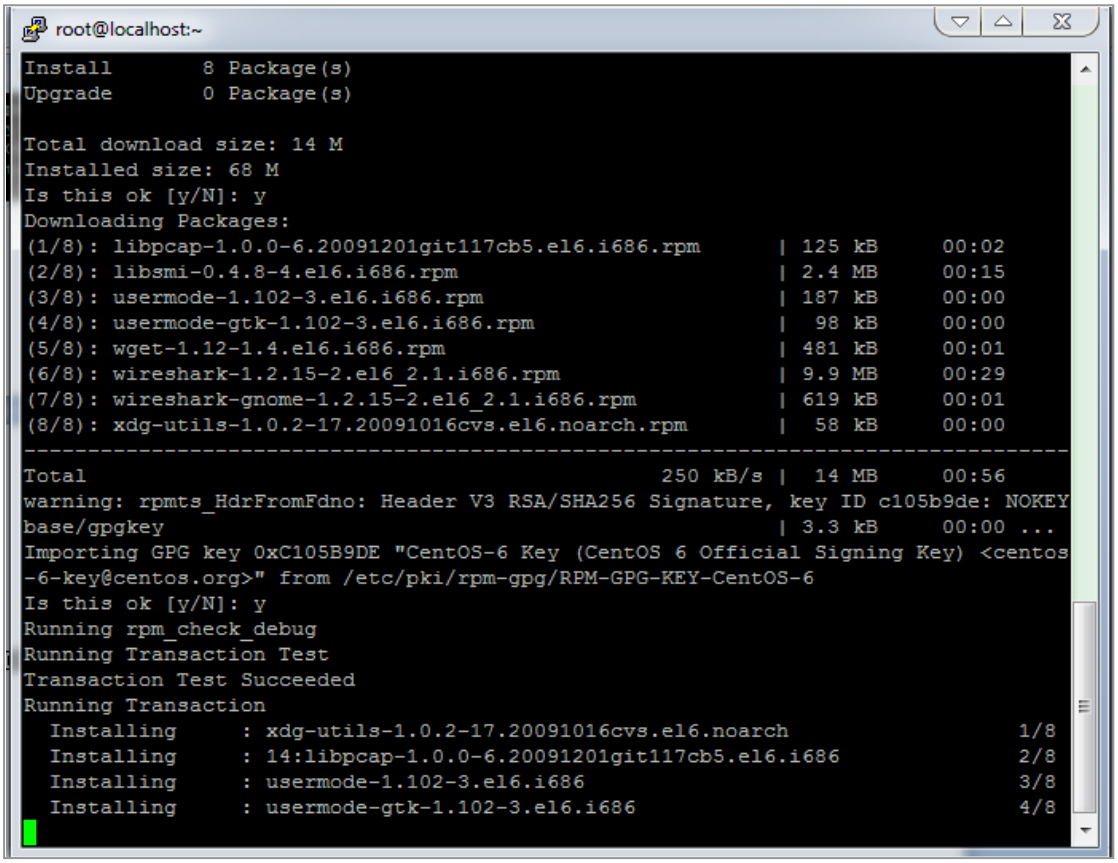

root@localhost:~
Install 8 Package(s)
Upgrade 0 Package(s)
Total download size: 14 M
Installed size: 68 M
Is this ok [y/N]: y
Downloading Packages:
(1/8): libpcap-1.0.0-6.20091201git117cb5.el6.i686.rpm | 125 kB 00:02
(2/8): libsmi-0.4.8-4.el6.i686.rpm | 2.4 MB 00:15
(3/8): usermode-1.102-3.el6.i686.rpm | 187 kB 00:00
(4/8): usermode-gtk-1.102-3.el6.i686.rpm | 98 kB 00:00
(5/8): wget-1.12-1.4.el6.i686.rpm | 481 kB 00:01
(6/8): wireshark-1.2.15-2.el6_2.1.i686.rpm | 9.9 MB 00:29
(7/8): wireshark-gnome-1.2.15-2.el6_2.1.i686.rpm | 619 kB 00:01
(8/8): xdg-utils-1.0.2-17.20091016cvs.el6.noarch.rpm | 58 kB 00:00
Total 250 kB/s | 14 MB 00:56
warning: rpmts_HdrFromFdno: Header V3 RSA/SHA256 Signature, key ID c105b9de: NOKEY
base/gpgkey | 3.3 kB 00:00 ...
Importing GPG key 0xC105B9DE "CentOS-6 Key (CentOS 6 Official Signing Key) <centos-6-key@centos.org>" from /etc/pki/rpm-gpg/RPM-GPG-KEY-CentOS-6
Is this ok [y/N]: y
Running rpm_check_debug
Running Transaction Test
Transaction Test Succeeded
Running Transaction
Installing : xdg-utils-1.0.2-17.20091016cvs.el6.noarch 1/8
Installing : 14:libpcap-1.0.0-6.20091201git117cb5.el6.i686 2/8
Installing : usermode-1.102-3.el6.i686 3/8
Installing : usermode-gtk-1.102-3.el6.i686 4/8

图 2-2-13

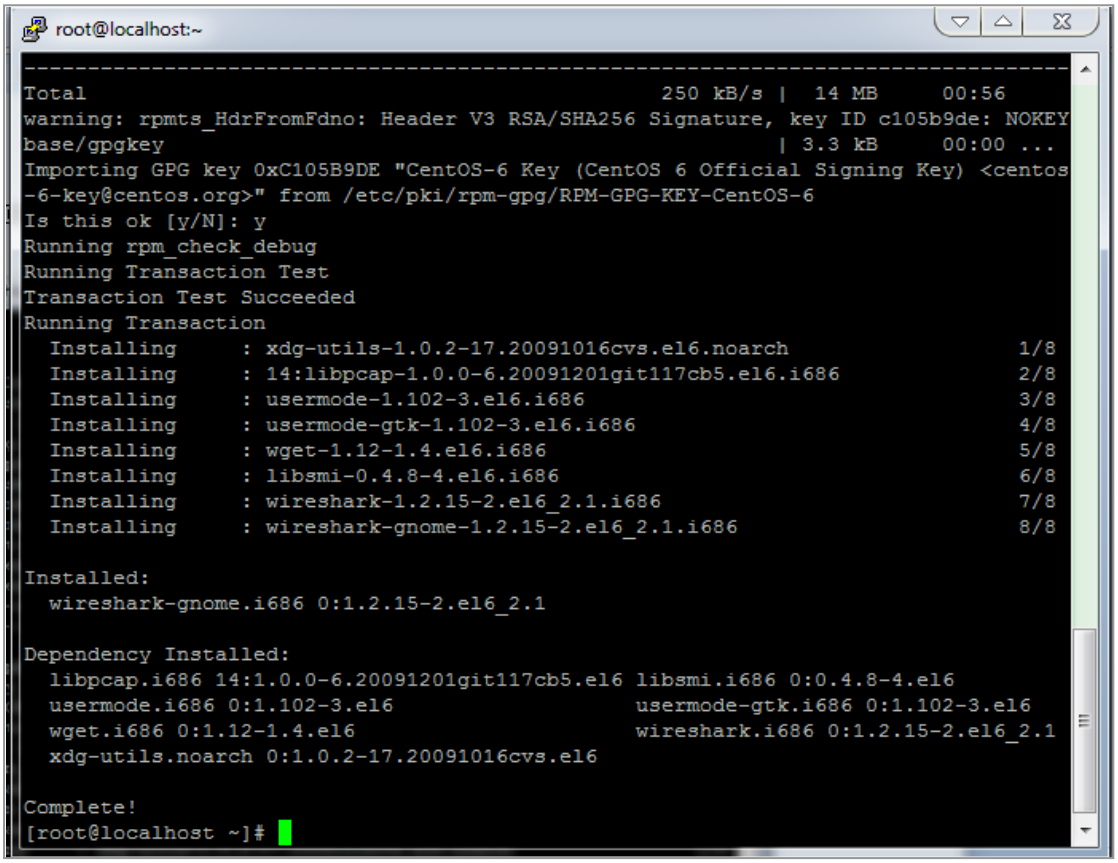


图 2-2-14

如上图 2-2-14，系统自动完成了 wireshark 以及相关依赖软件的安装；这时候我们输入 wireshark 这条命令回车后发下报错！如下图 2-2-15：

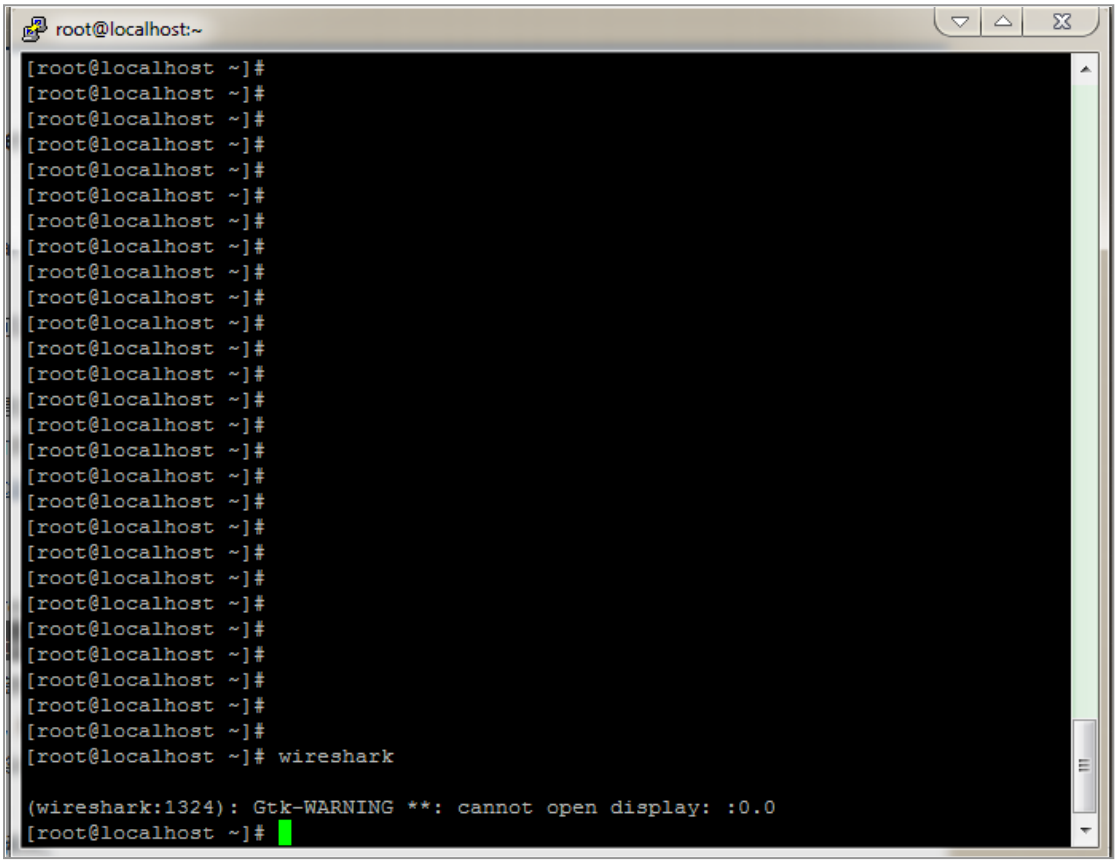


图 2-2-15

ethereal 和 wireshark 都是基于图形的程序，所以我们使用 ssh 的远程终端是无法运行的。

比如说 ,使用 ethereal 命令就可以启动它了 ,但是远程终端会提示 Gtk-WARNING **: cannot open display:；遇到该问题怎么办呢？我们可以使用对应的命令行程序 tethereal；这条命令来启动 wireshark 进行数据包捕捉。如下图：2-2-16 所示：

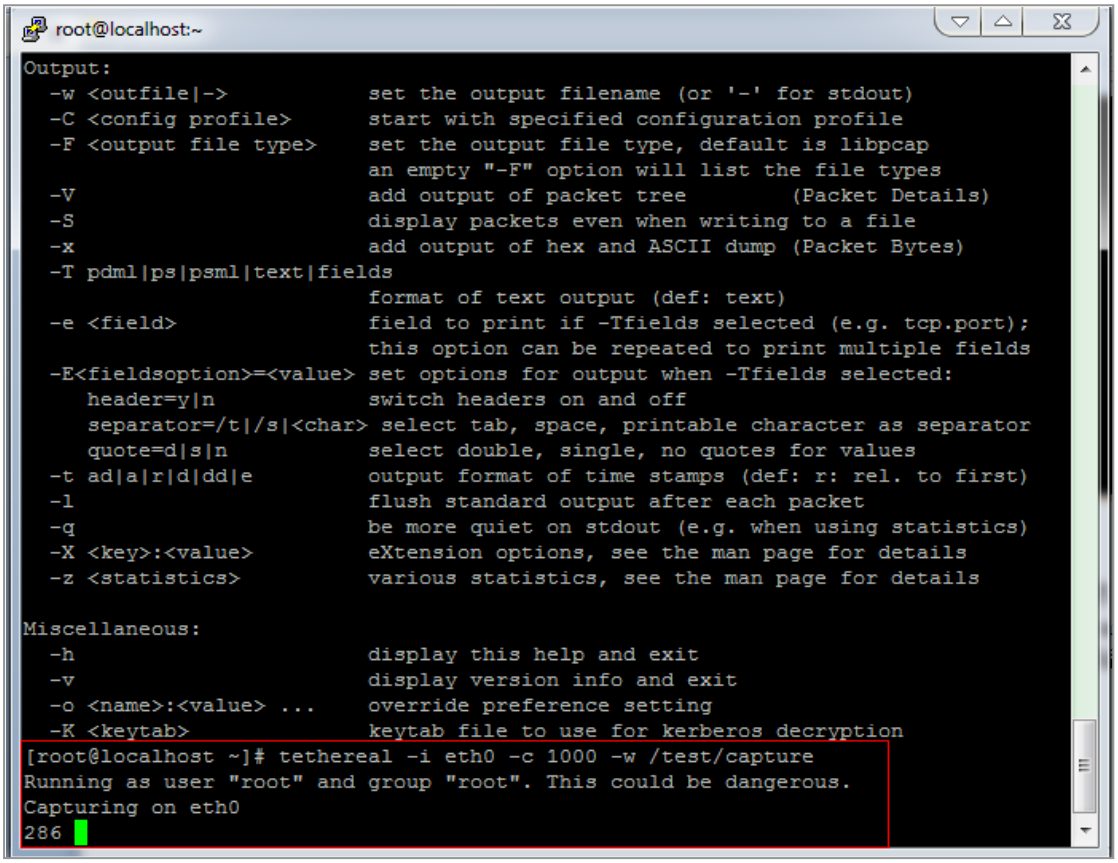


图 2-2-16

关于 linux 下的 wireshark 的安装我们就讲这么多，但是 wireshark 并不止这一种安装方式，还有其他的一些更为可选的安装方式。这些需要大家自己去慢慢的摸索发现了 ,我们在此对于 wireshark 在命令行界面的使用以及其他安装方式就不做过多的解释了。

2.3、Ubuntu apt-get 安装方式

首先打开 Ubuntu 终端，然后输入 apt-get install wireshark。系统会自动从软件中心下载以及安装 wireshark，如下图 2-2-17 所示：

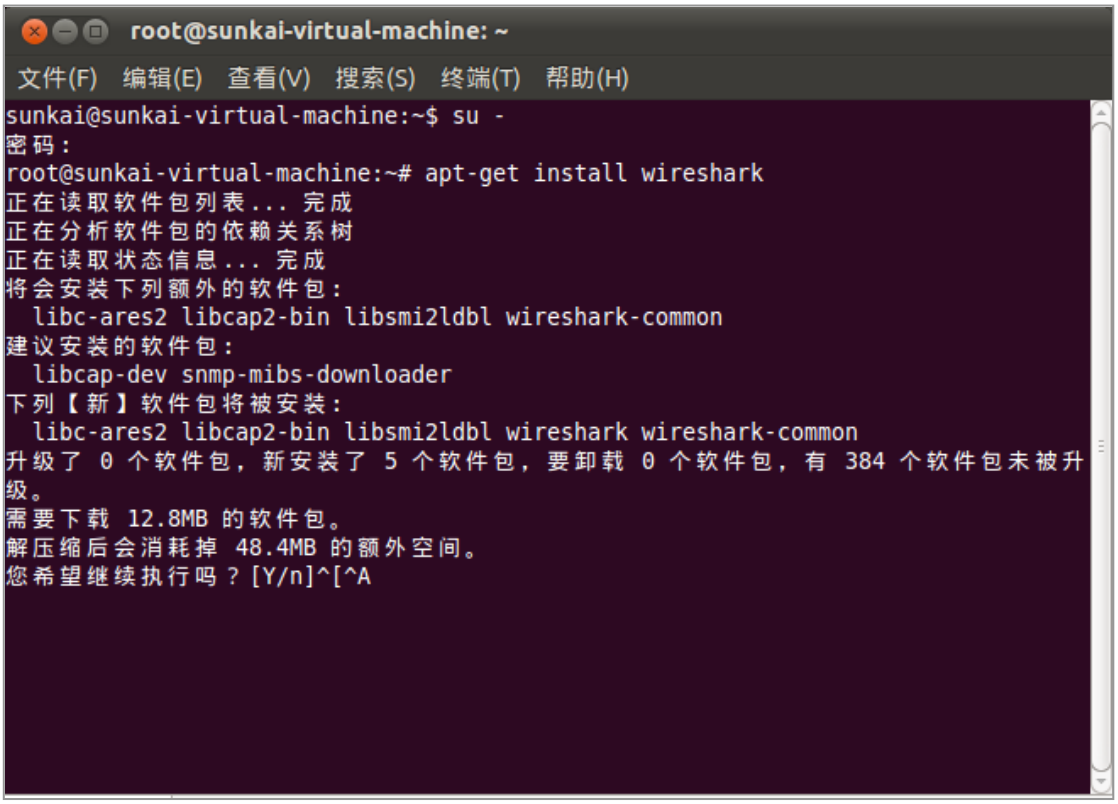


图 2-2-17

输入 Y 回车后我们继续安装过程；如下图 2-2-18，2-2-19 所示：

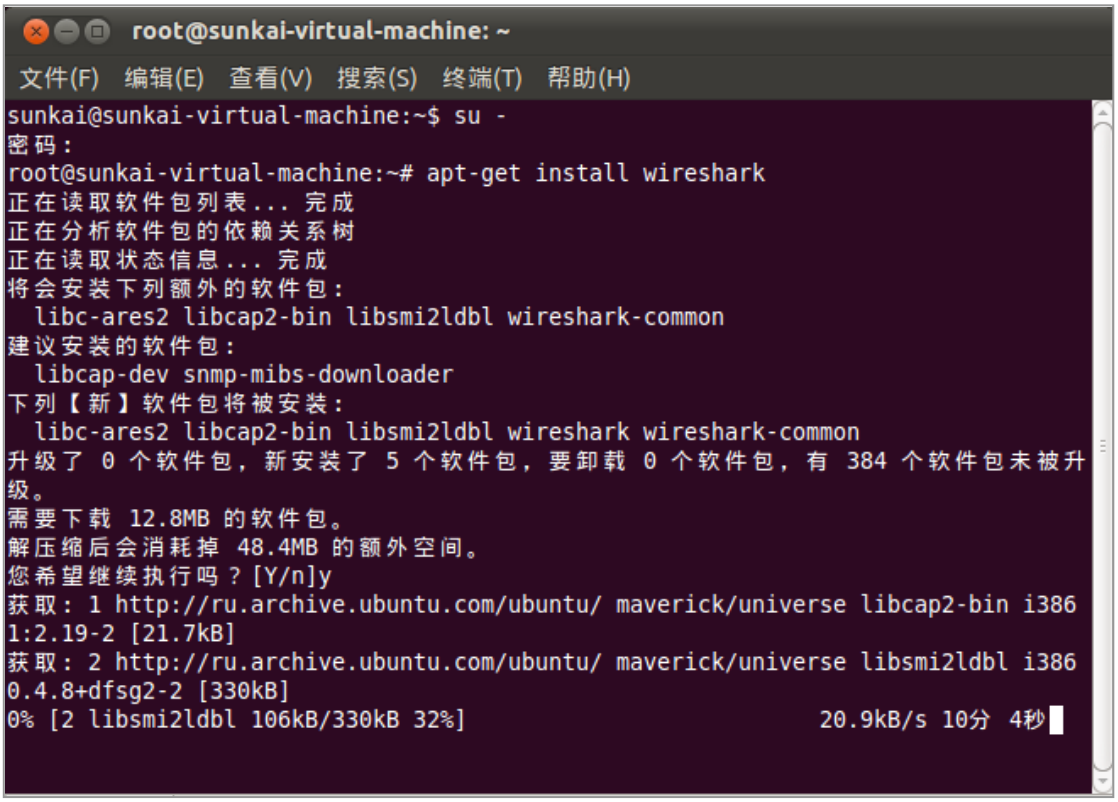

root@sunkai-virtual-machine: ~
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
sunkai@sunkai-virtual-machine:~$ su -
密码：
root@sunkai-virtual-machine:~# apt-get install wireshark
正在读取软件包列表... 完成
正在分析软件包的依赖关系树
正在读取状态信息... 完成
将会安装下列额外的软件包：
libc-ares2 libcap2-bin libsmi2ldbl wireshark-common
建议安装的软件包：
libcap-dev snmp-mibs-downloader
下列【新】软件包将被安装：
libc-ares2 libcap2-bin libsmi2ldbl wireshark wireshark-common
升级了 0 个软件包，新安装了 5 个软件包，要卸载 0 个软件包，有 384 个软件包未被升级。
需要下载 12.8MB 的软件包。
解压缩后会消耗掉 48.4MB 的额外空间。
您希望继续执行吗？[Y/n]y
获取：1 http://ru.archive.ubuntu.com/ubuntu/ maverick/universe libcap2-bin i386 1:2.19-2 [21.7kB]
获取：2 http://ru.archive.ubuntu.com/ubuntu/ maverick/universe libsmi2ldbl i386 0.4.8+dfsg2-2 [330kB]
0% [2 libsmi2ldbl 106kB/330kB 32%] 20.9kB/s 10分 4秒

图 2-2-18

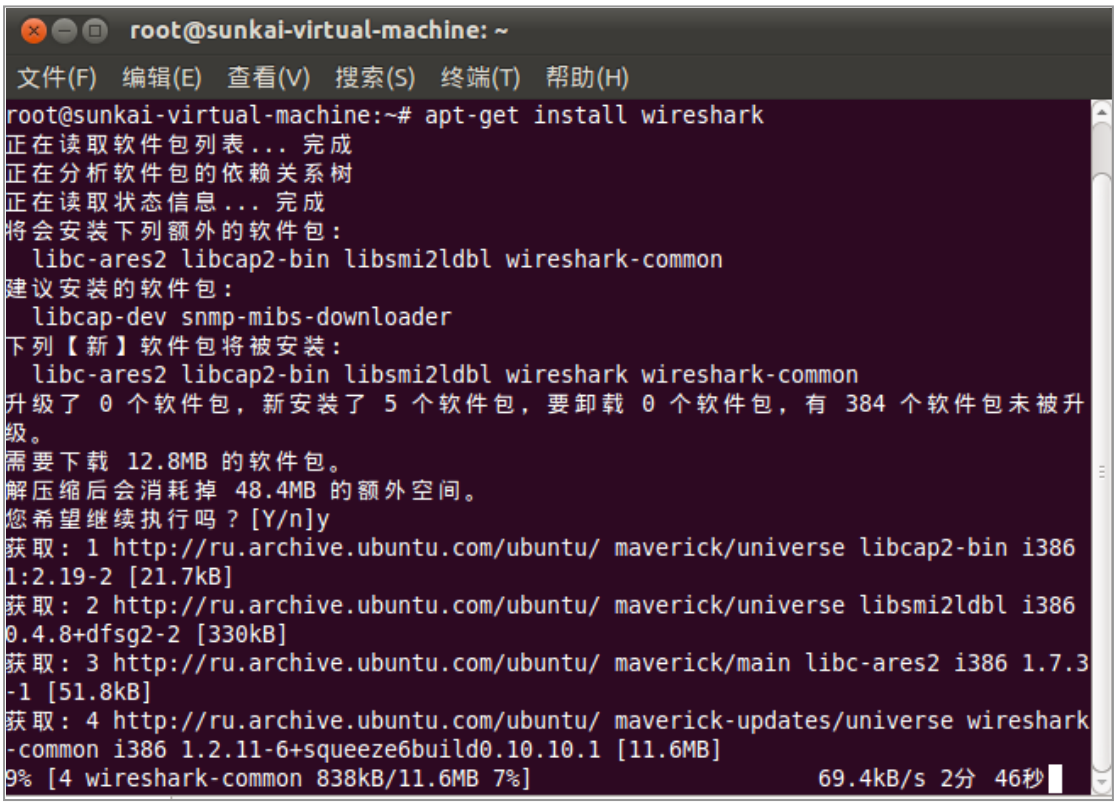

root@sunkai-virtual-machine: ~
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
root@sunkai-virtual-machine:~# apt-get install wireshark
正在读取软件包列表... 完成
正在分析软件包的依赖关系树
正在读取状态信息... 完成
将会安装下列额外的软件包：
libc-ares2 libcap2-bin libsmi2ldbl wireshark-common
建议安装的软件包：
libcap-dev snmp-mibs-downloader
下列【新】软件包将被安装：
libc-ares2 libcap2-bin libsmi2ldbl wireshark wireshark-common
升级了 0 个软件包，新安装了 5 个软件包，要卸载 0 个软件包，有 384 个软件包未被升级。
需要下载 12.8MB 的软件包。
解压缩后会消耗掉 48.4MB 的额外空间。
您希望继续执行吗？[Y/n]y
获取：1 http://ru.archive.ubuntu.com/ubuntu/ maverick/universe libcap2-bin i386 1:2.19-2 [21.7kB]
获取：2 http://ru.archive.ubuntu.com/ubuntu/ maverick/universe libsmi2ldbl i386 0.4.8+dfsg2-2 [330kB]
获取：3 http://ru.archive.ubuntu.com/ubuntu/ maverick/main libc-ares2 i386 1.7.3-1 [51.8kB]
获取：4 http://ru.archive.ubuntu.com/ubuntu/ maverick-updates/universe wireshark-common i386 1.2.11-6+squeeze6build0.10.10.1 [11.6MB]
9% [4 wireshark-common 838kB/11.6MB 7%] 69.4kB/s 2分 46秒

图 2-2-19

自动处理依赖关系；如下图 2-2-20 示：

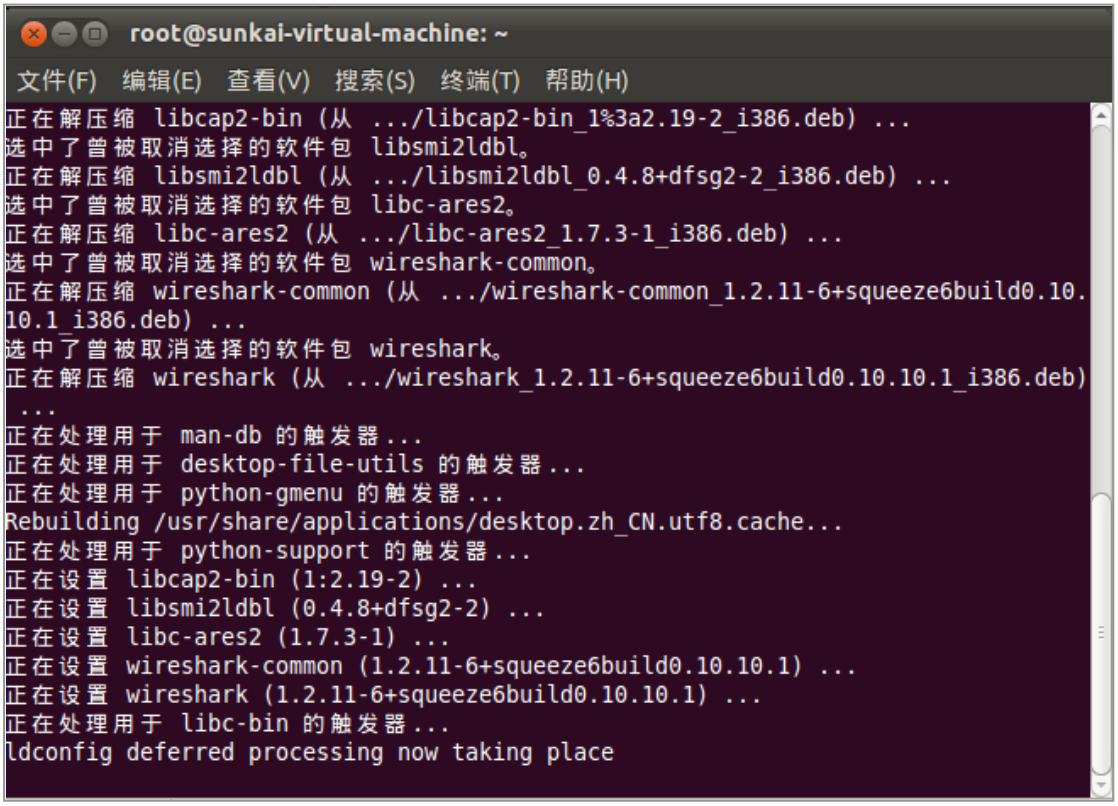


图 2-2-20

安装完成后我们输入：wireshark 来启动图形化的界面程序；如下图 2-2-21，2-2-22 所示：

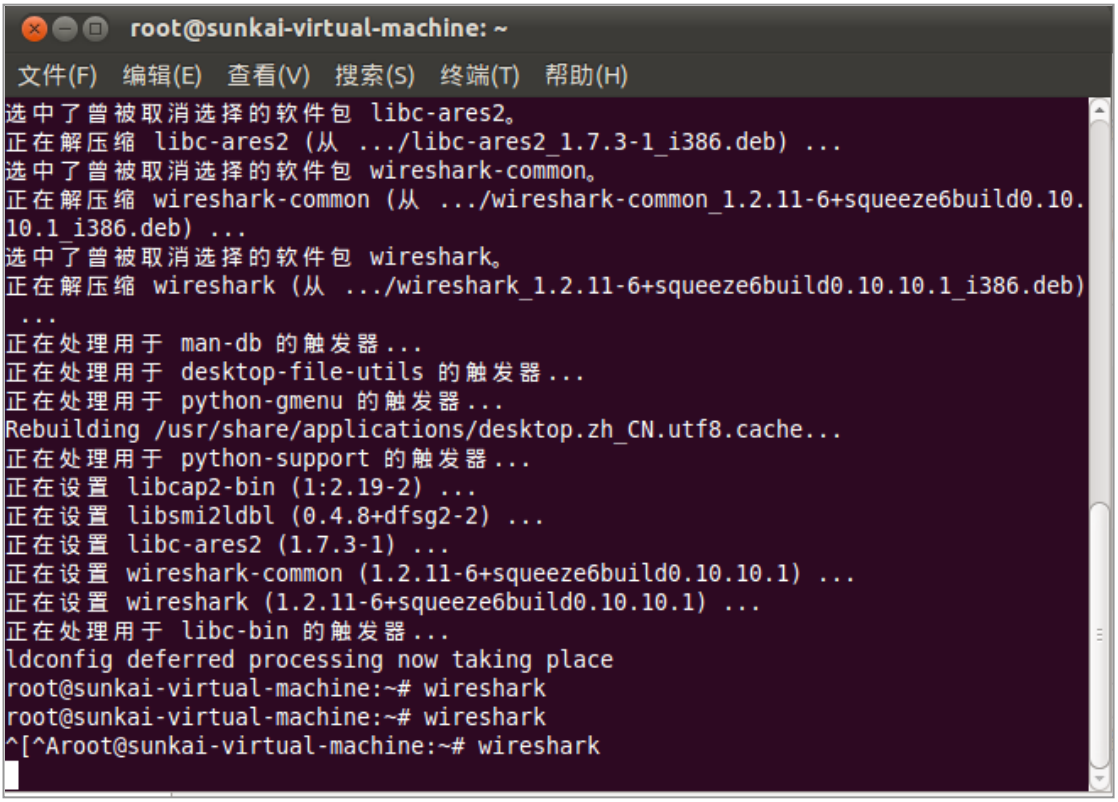

root@sunkai-virtual-machine: ~
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
选中了曾被取消选择的软件包 libc-ares2。
正在解压缩 libc-ares2 (从 .../libc-ares2_1.7.3-1_i386.deb) ...
选中了曾被取消选择的软件包 wireshark-common。
正在解压缩 wireshark-common (从 .../wireshark-common_1.2.11-6+squeeze6build0.10.10.1_i386.deb) ...
选中了曾被取消选择的软件包 wireshark。
正在解压缩 wireshark (从 .../wireshark_1.2.11-6+squeeze6build0.10.10.1_i386.deb) ...
正在处理用于 man-db 的触发器...
正在处理用于 desktop-file-utils 的触发器...
正在处理用于 python-gmenu 的触发器...
Rebuilding /usr/share/applications/desktop.zh_CN.utf8.cache...
正在处理用于 python-support 的触发器...
正在设置 libcap2-bin (1:2.19-2) ...
正在设置 libsmi2ldbl (0.4.8+dfsg2-2) ...
正在设置 libc-ares2 (1.7.3-1) ...
正在设置 wireshark-common (1.2.11-6+squeeze6build0.10.10.1) ...
正在设置 wireshark (1.2.11-6+squeeze6build0.10.10.1) ...
正在处理用于 libc-bin 的触发器...
ldconfig deferred processing now taking place
root@sunkai-virtual-machine:~# wireshark
root@sunkai-virtual-machine:~# wireshark
^[^Aroot@sunkai-virtual-machine:~# wireshark

图 2-2-21

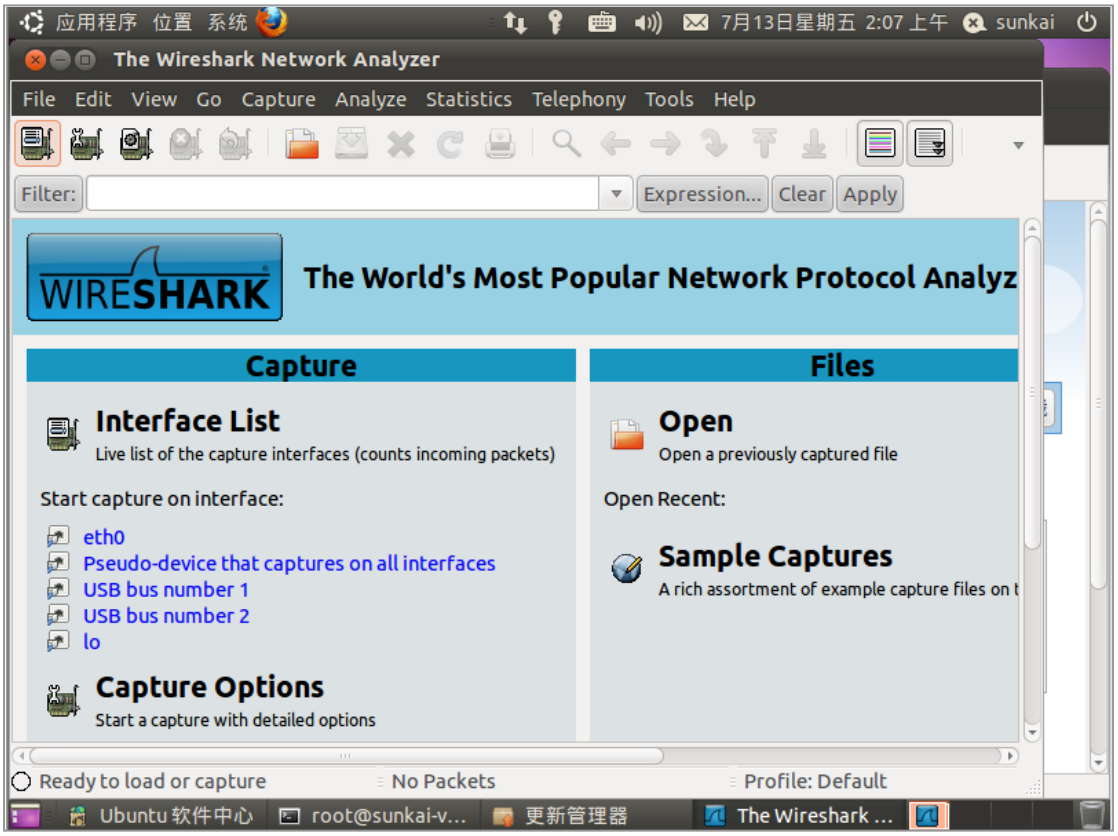

应用程序 位置 系统
7月13日星期五 2:07 上午
sunkai
The Wireshark Network Analyzer
File Edit View Go Capture Analyze Statistics Telephony Tools Help
Filter:
Expression... Clear Apply
WIRESHARK
The World's Most Popular Network Protocol Analyz
Capture
Interface List
Live list of the capture interfaces (counts incoming packets)
Start capture on interface:
eth0
Pseudo-device that captures on all interfaces
USB bus number 1
USB bus number 2
lo
Capture Options
Start a capture with detailed options
Files
Open
Open a previously captured file
Open Recent:
Sample Captures
A rich assortment of example capture files on t
Ready to load or capture
No Packets
Profile: Default
Ubuntu 软件中心
root@sunkai-v...
更新管理器
The Wireshark ...

图 2-2-22

至此 wireshark 在 Ubuntu 系统上面的 apt-get 安装方式就完成了，我们可以和在 windows 上一样来使用 wireshark 来进行数据包捕捉操作了！

◆ 界面概括

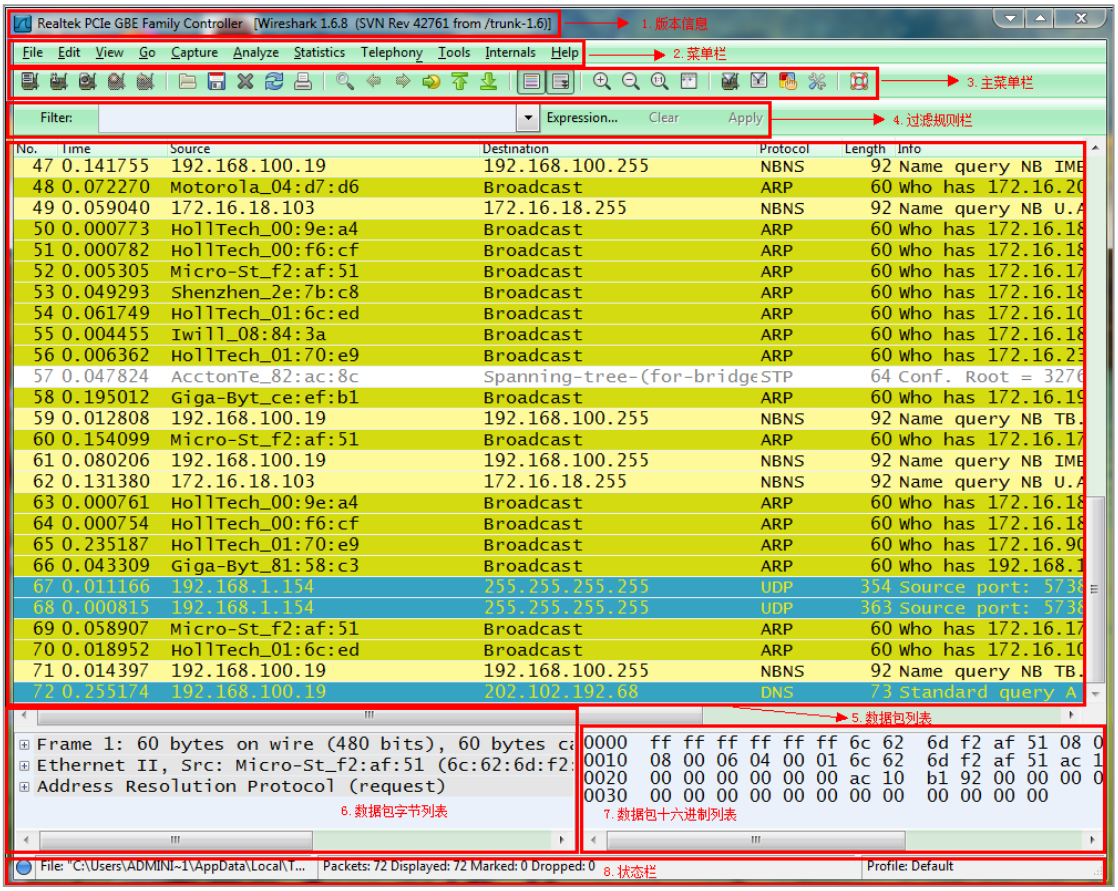


图 3-1

从图 3-1 所示为 wireshark 的一个主界面概括信息，从图中的红色矩形框以及标注可看出 wireshark 一共分为 8 块。以下我们一一介绍各个板块的功能选项；

3.1.0、主菜单栏

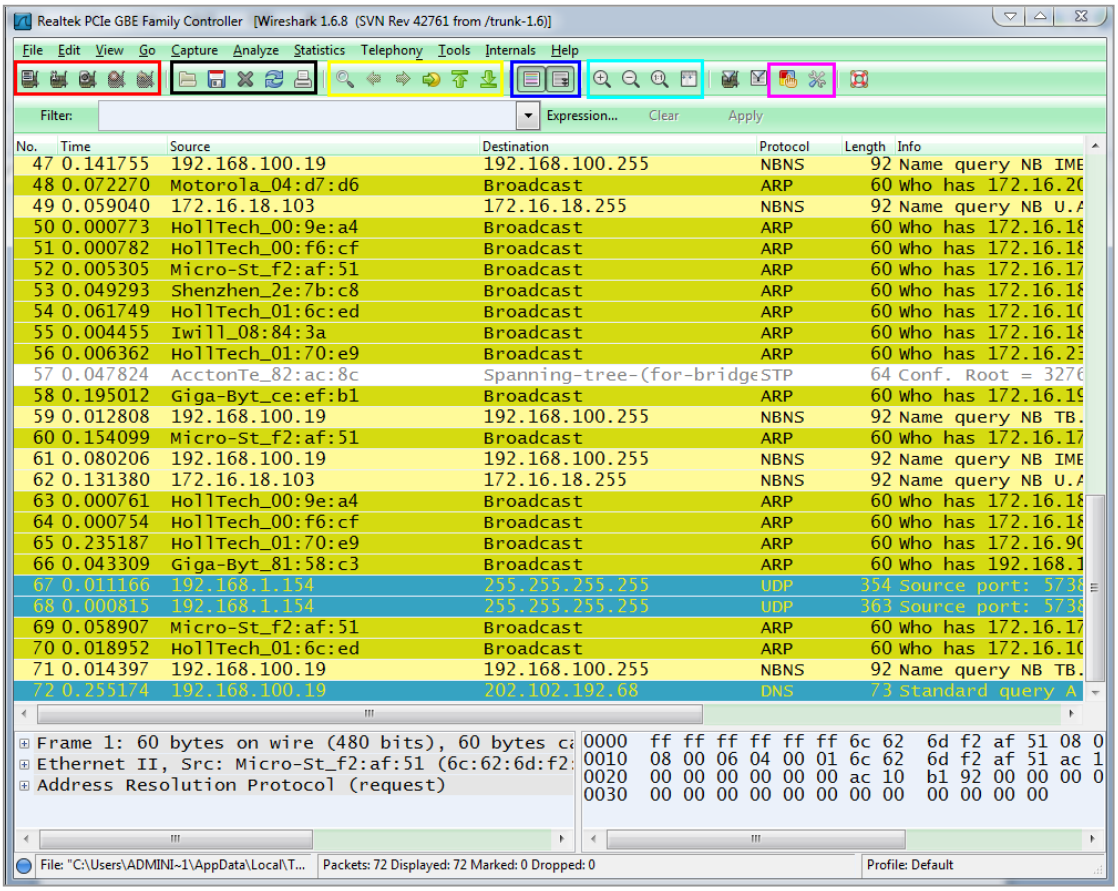


图 3-1-1

如图 3-1-1 所示带颜色矩形框内，主菜单栏又分为 6 个子功能选项栏。从左至右依次为：抓包工具栏、文件工具栏、包定位工具栏、颜色以及滚动界面工具栏、数据包列表字体定义工具栏以及首选项工具栏；下面我们从左至右分别介绍各个工具栏的功能。

3.1.1、抓包工具栏

如图 3-1-1 红色矩形框内所示，抓包工具栏显示共有 5 个功能图标。从左至右依次为：

 列出计算机可以抓包的接口（与菜单--Capture--Interfaces 功能相同）；

 定义抓包参数选项（与菜单--Capture--Options 功能相同）；

 开始一个新的抓包操作（与菜单--Capture--Start 功能相同）；

 停止抓包操作（与菜单--Capture--Stop 功能相同；

 重新开始一个新的抓包操作（与菜单--Capture--Restart 功能相同）；

点击  图标后，wireshark 会自动弹出一个对话框，该对话框列出了我们计算机所安装的网卡信息。从上面我们可以看出网卡的 IP 地址以及网卡的型号，在有数据包收发时候“Packets”字段会有数字滚动。如果计算机安装了多块网卡并且每个网卡配置了多个 IP 地址的情况下，想针对某一块网卡开始一个抓包操作可以根据 IP 地址或者 MAC 地址来区分不同的网卡。然后可直接点击与网卡相对应的“Start”按钮；如图3-1-2,；

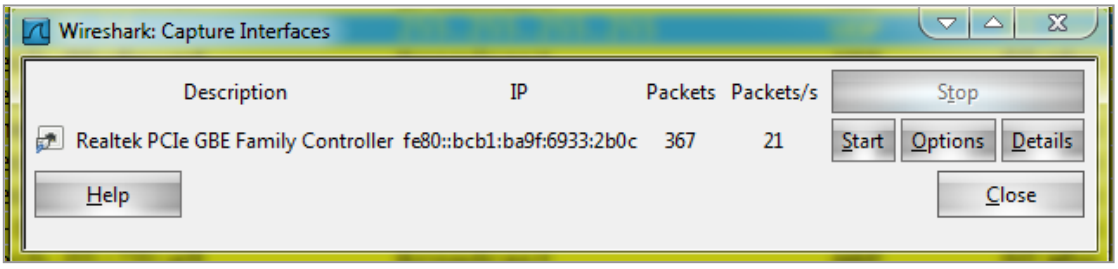


图3-1-2

如果我们想对抓包动作做相应的设置的话，我们可以点击该  图标或者使用图3-1-2所示的“Options”按钮。在中文版界面该图标被翻译为“抓包参数选项”。点击图标后弹出一个对话框，该对话框主要如图3-1--3所示分为以下5大块：Capture（捕获）、Capture Files（抓包文件）、Stop Capture（停止抓包）、Display Options（查看设置）以及 Name Resolution（名字映射）。首先 Capture 列出了可以抓包的网卡，从图3-1-4可以看出列出了我本机的一块网卡，假如你主机装了多块网卡的话 wireshark 会一一列出来。你可以选择特定的网卡捕捉进出该网卡的数据包；

Capture on all interfaces（捕捉所有接口）选择该复选框后本机安装的所有显示到网卡列表的网卡都会执行数据包捕捉操作；

Capture all in promiscuous mode （设置网卡为杂合模式）如果不选择该复选框那么 wireshark 将只会捕捉本机网卡所发收的数据包而不会捕捉局域网其他主机所发出的数据包；在这里我们选择默认操作。

Capture Files（捕捉数据包为文件）这一栏如图3-1-3所示有以下几个选项：Use multiple files（使用多个文件）、Use pcap-ng format （该功能 Wireshark 暂未正式支持）、Next file every n megabyte(s)（到达指定文件大小）、Next file every n minutes(s) （到达指定时间）、Ring buffer with n files（到达缓冲区指定文件数目）以及 Stop caputure after n file(s) （到达指定文件数目）；以上几个选项有效前提是选择了第一个即：Use multiple files（使用多个文件）复选框时生效。

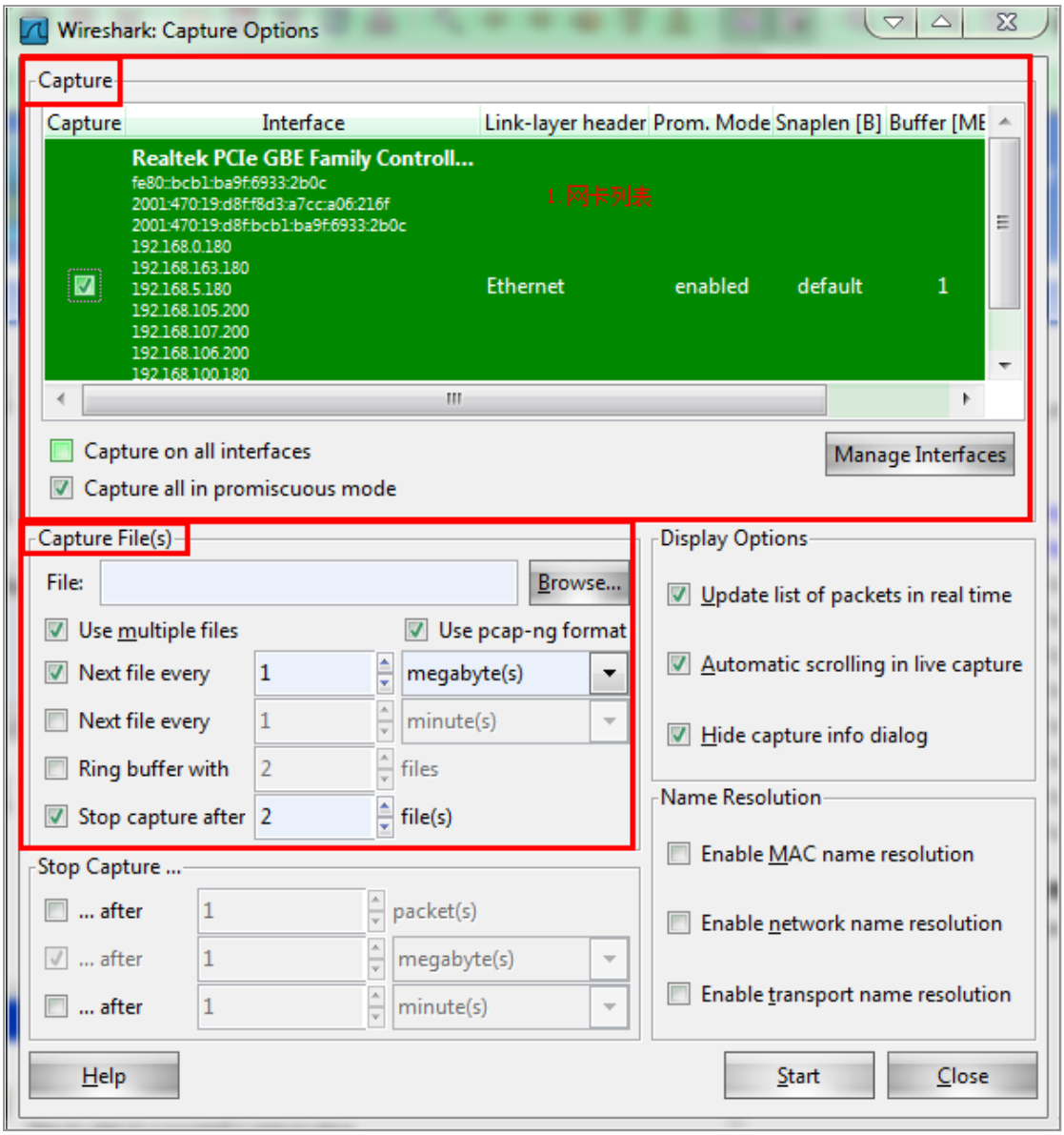


图3-1-3

如上图3-1-3所示我们选择了 Use multiple files（使用多个文件）、Next file every n megabyte(s)（到达指定文件大小）以及 Stop caputure after n file(s) （到达指定文件数目）复选框后指定到达1024KB 时自动换一个新的文件进行存储，存储文件到达2个时自动停止抓包进程。然后我们选择“Browse”浏览到我们新建的.cap 后缀名的文件（使用以上功能时候要新建一个后缀名为.cap 的空文件，假如没有指定文件 wireshark 将找不到所要根据的文件名来自动创建文件，将会弹出如图3-1-4所示错误信息）。然后点击“Start”按钮执行抓包操作。Wireshark 会根据我们要求在抓取大小到1024KB（下拉菜单有多个存储单位选项）数据包后自动创建2个1M 大小的.cap 文件来对数据包进行存储并且存储文件数目达到2个时自动停止抓包。如图3-1-5所示；

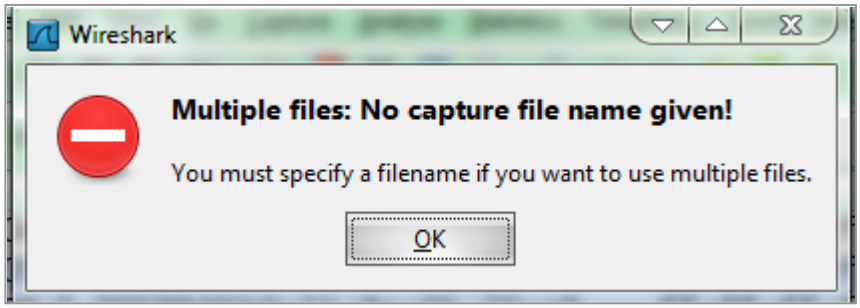


图3-1-4

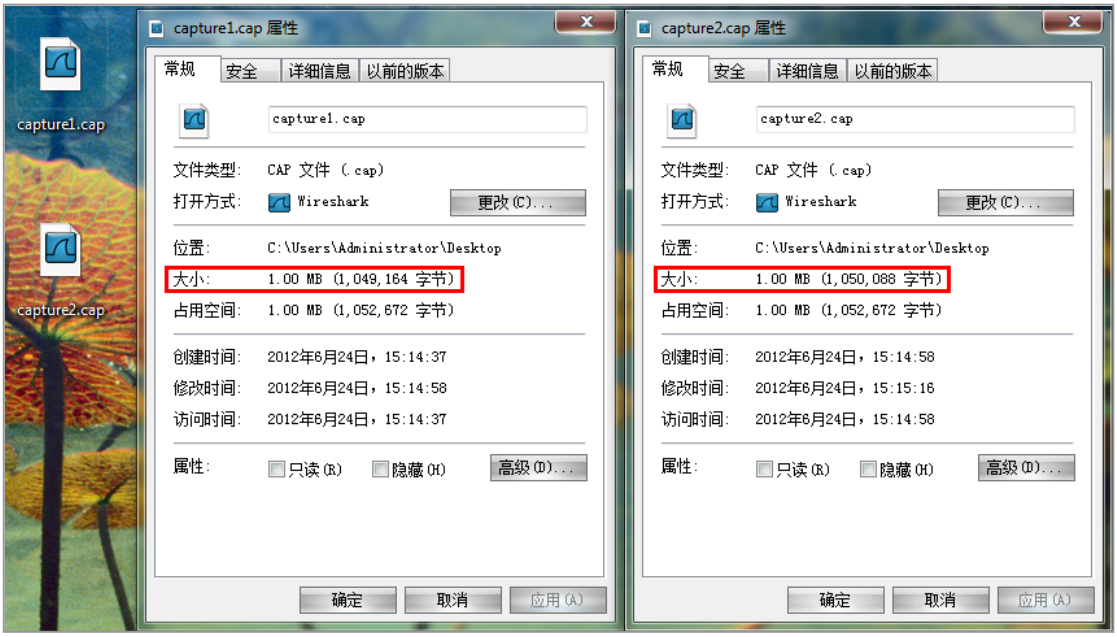


图3-1-5

除了上述根据抓取数据包大小变换文件外，我们还可以使用 Next file every n minutes(s) （到达指定时间）指定一个抓包时间来进行数据包抓取操作。如下图3-1-6所示，我们选择抓包时间为30秒（下拉菜单具有多个时间选项）。点击“Start”后 wireshark 执行抓包操作；

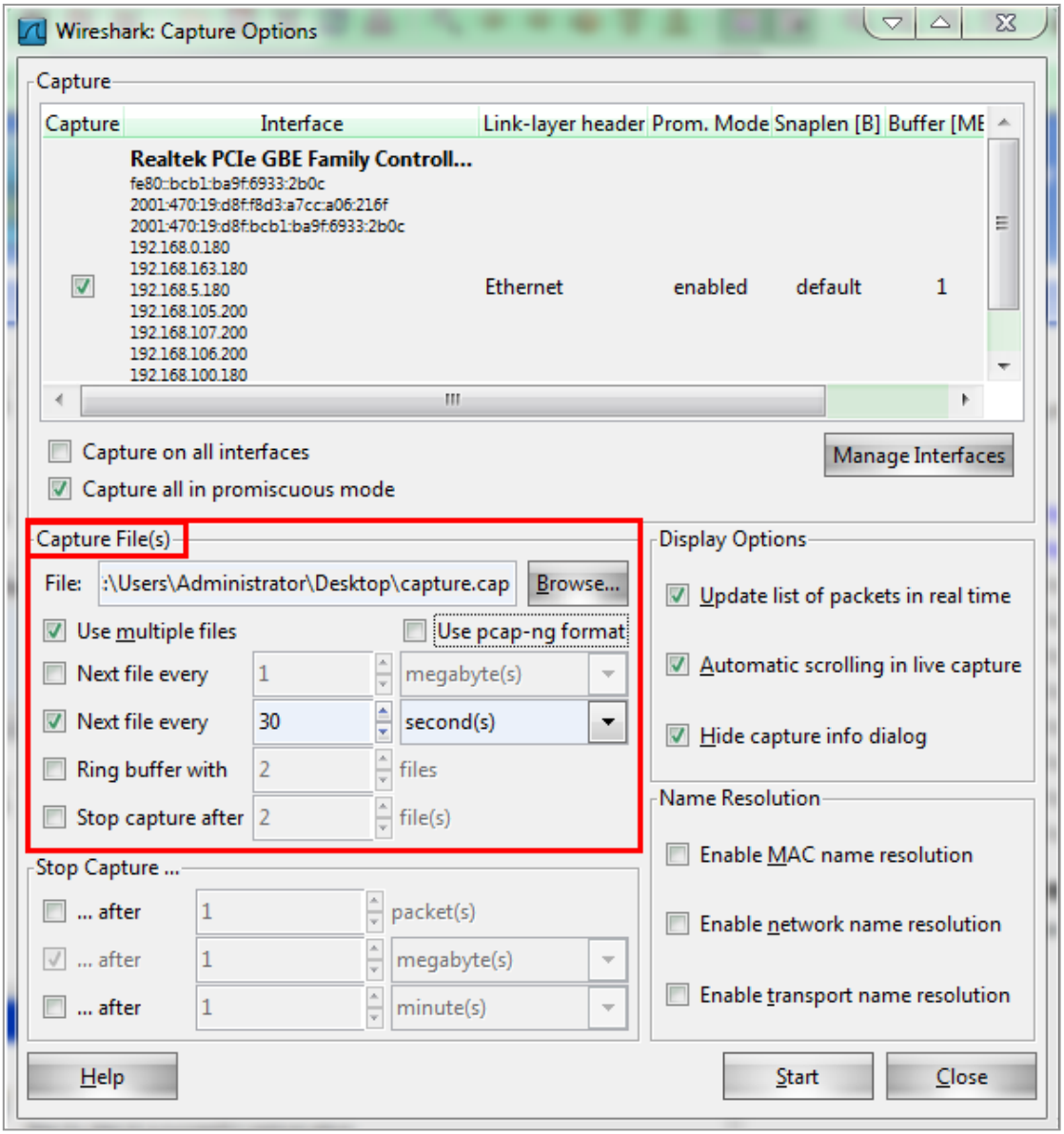


图3-1-6

如上图3-1-6所示，我们假如没有选择 Stop caputure after n file(s) （到达指定文件数目）复选框，那么 wireshark 将每30秒创建一个新的.cap 文件，并且会一直持续抓包除非人工手动停止。所以建议和 Stop caputure after n file(s) （到达指定文件数目）该复选框联合使用；

Ring buffer with n files（到达缓冲区指定文件数目）该复选框是规定所抓取的数据包到达指定文件大小后覆盖文件内容循环进行抓包。相应的操作如图3-1-7所示：

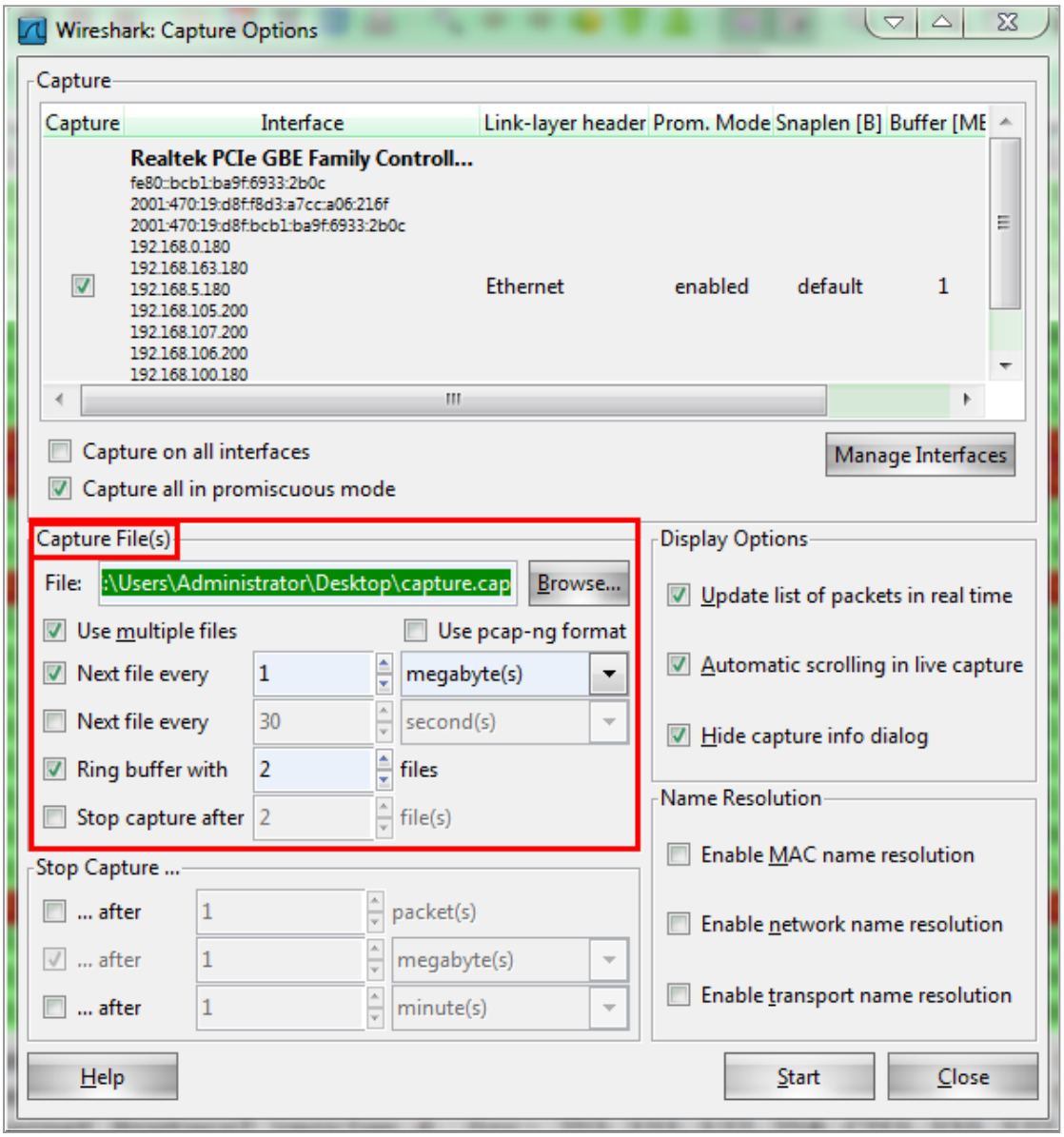


图3-1-7

按上图3-1-7配置抓包操作，wireshark 将会在抓包数量到达1M 时自动创建第2个文件，当第2个文件所存储数据包数量也到达指定的1M 时 wireshark 将会覆盖之前第一个文件的内容。继而继续存储新的数据包；

Stop Capture（停止抓包）：该栏具有与上面 Capture fils 一栏类似的功能，也是限制抓取数据包数量，唯一不同的是该栏是先抓取后存储即不用预先指定参照文件名。在此栏我们可以定义在抓取多少数据包之后停止抓包，抓取数据包到达多大存储大小以及抓包多长时间后停止抓包。假如我们选择了如图3-1-8所示配置，那么 wireshark 将会在抓取了10个数据包之后停止抓包动作；

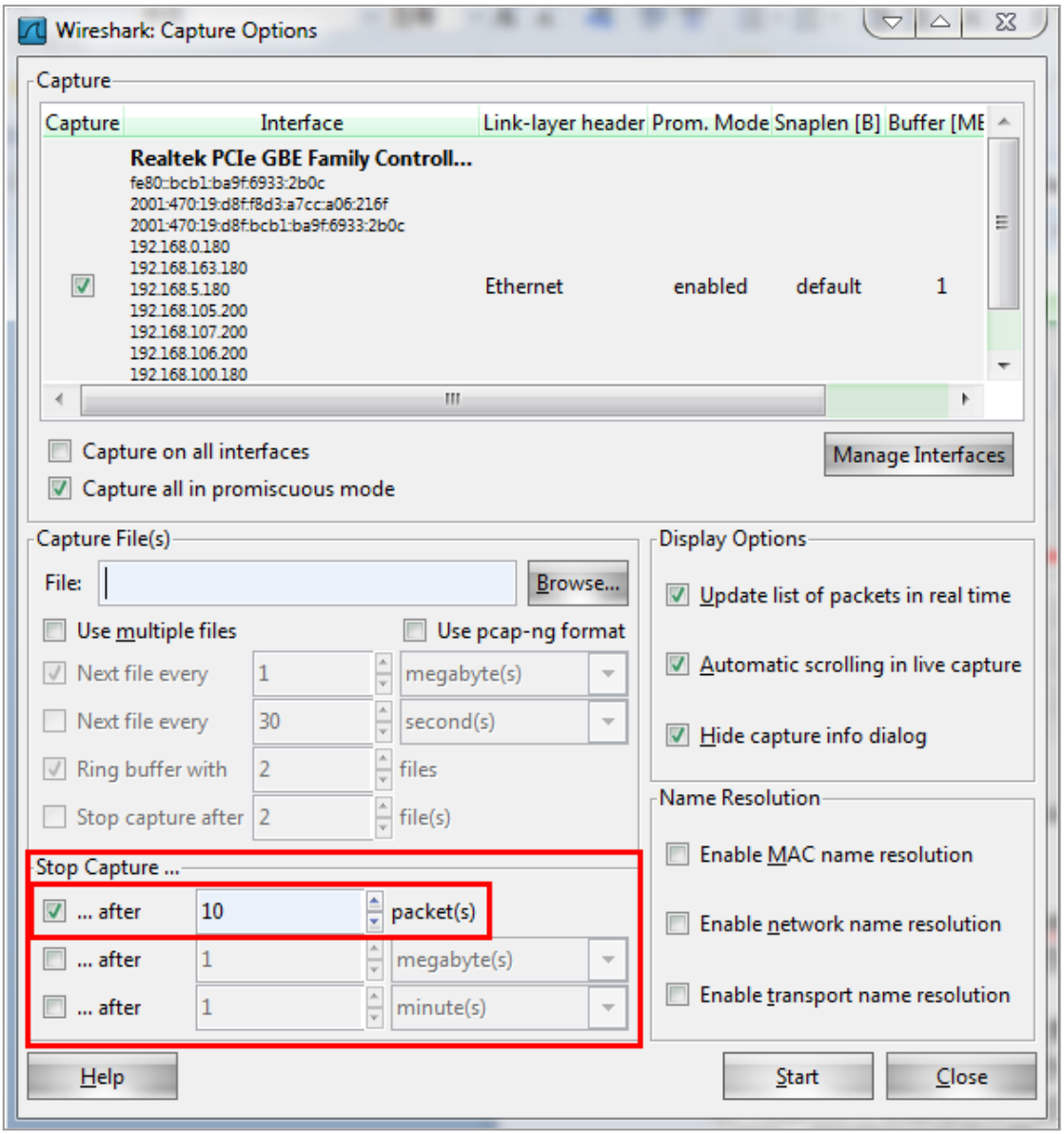


图3-1-8

如图3-1-9所示，wireshark 在抓取了10个数据包之后停止了抓包进程；

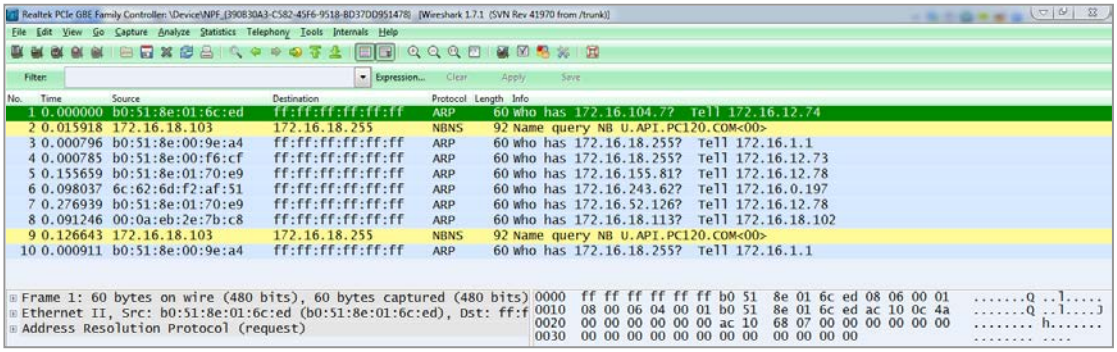


图3-1-9

除了按包数量捕捉外，我们还可以指定捕捉数据包到达指定大小以及捕捉动作执行多长时间后停止抓包动作，方法是勾选相应的复选框后制定一个数值。在这里我们不在做过多赘述；

Display Options（显示设置）：顾名思义该选项与显示有关，如下图3-1-10所示，从上至下第一个复选框 Update list of packets in real time （实时更新数据包列表）此选项相当于主菜单栏--颜色设置栏--  图标。勾选此选项 wireshark 在捕捉数据包时我们可以实时看到数据包被捕获到，反之捕获进行时的界面将会是如图3-1-11所示 ,直到我们手动或者自动停止捕捉进程 wireshark 才会列出所捕捉到的数据包；

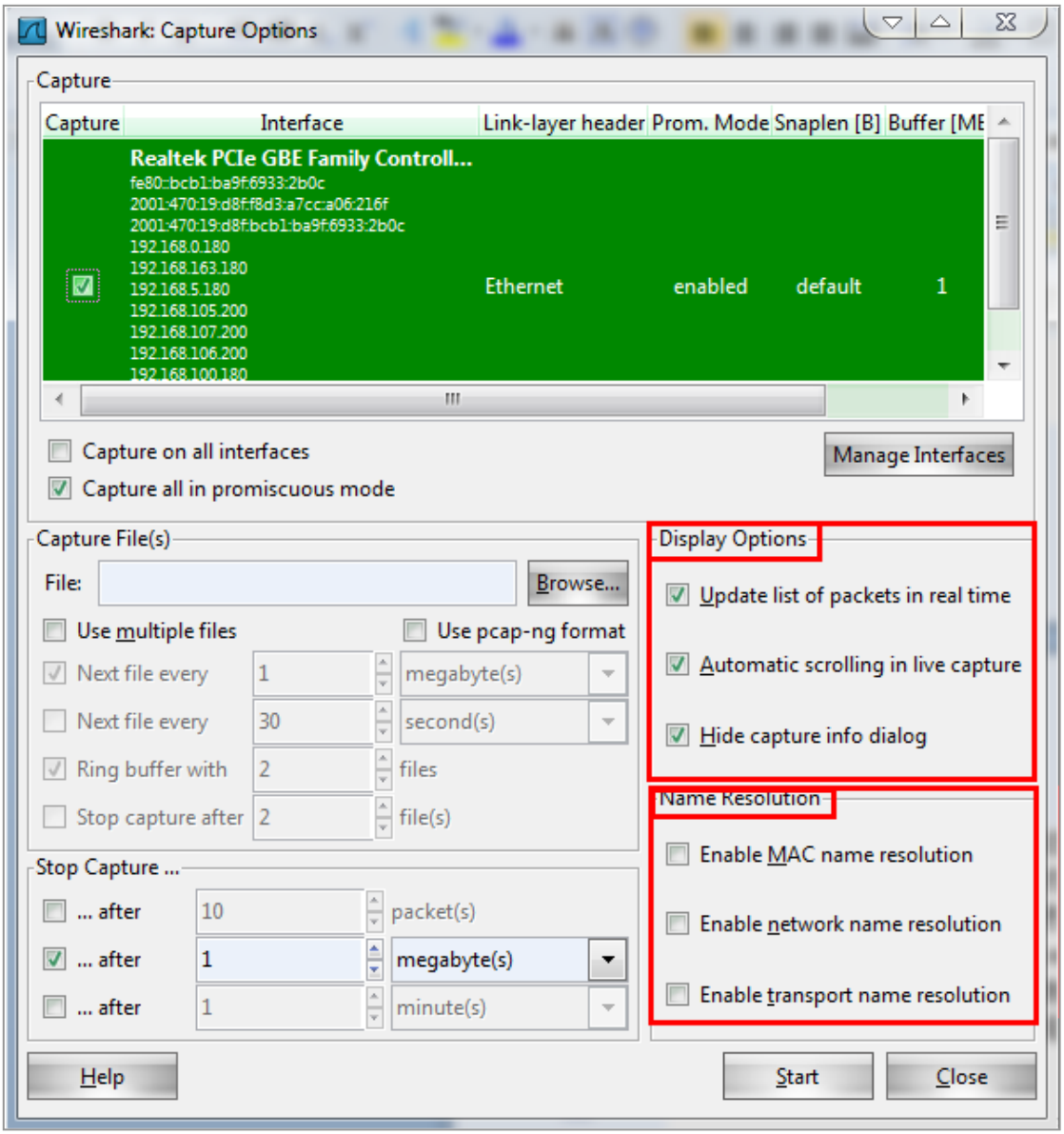

Wireshark: Capture Options
Capture
Capture
Interface
Link-layer header
Prom. Mode
Snaplen [B]
Buffer [ME
Realtek PCIe GBE Family Controll...
fe80::bcb1:ba9f:6933:2b0c
2001:470:19:d8ff:8d3:a7cc:a06:216f
2001:470:19:d8f:bcb1:ba9f:6933:2b0c
192.168.0.180
192.168.163.180
192.168.5.180
192.168.105.200
192.168.107.200
192.168.106.200
192.168.100.180
Ethernet
enabled
default
1
Capture on all interfaces
Manage Interfaces
Capture all in promiscuous mode
Capture File(s)
File:
Browse...
Use multiple files
Use pcap-ng format
Next file every
1
megabyte(s)
Next file every
30
second(s)
Ring buffer with
2
files
Stop capture after
2
file(s)
Display Options
Update list of packets in real time
Automatic scrolling in live capture
Hide capture info dialog
Name Resolution
Enable MAC name resolution
Enable network name resolution
Enable transport name resolution
Stop Capture ...
... after
10
packet(s)
... after
1
megabyte(s)
... after
1
minute(s)
Help
Start
Close

图3-1-10

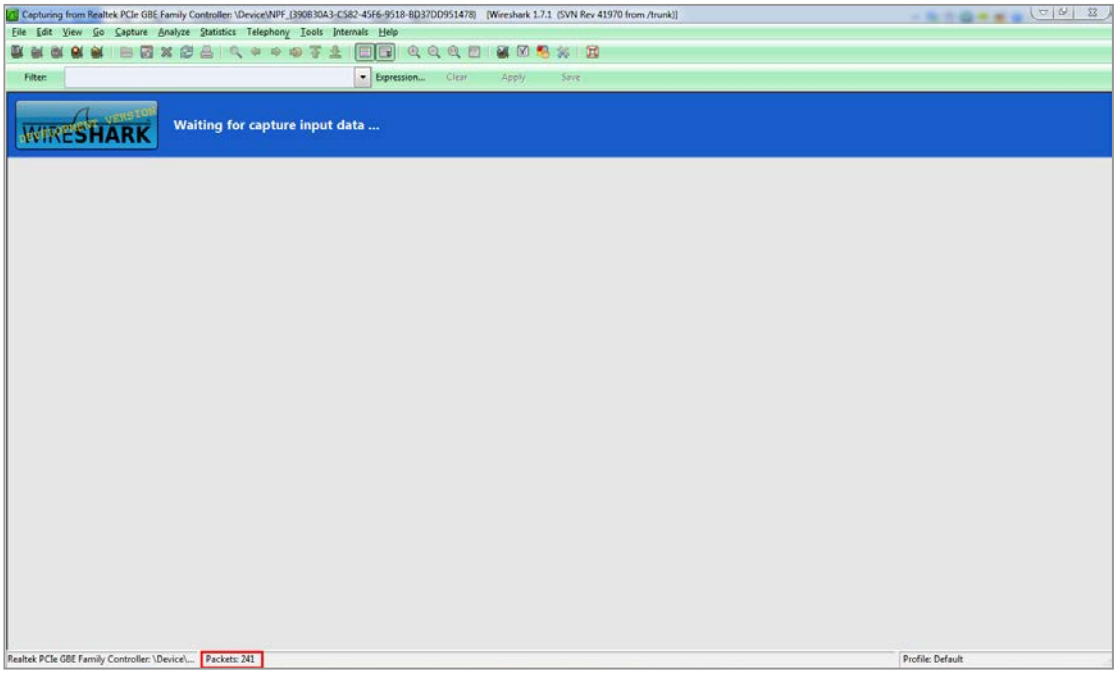


图3-1-11

从上图3-1-11所示底部状态栏红色矩形框内，我们可以看到数据包一直在增加，说明 wireshark 在工作；依赖于 Update list of packets in real time （实时更新数据包列表）该选项工作的是 Automatic scrolling in live capture （捕捉进行时自动滚动）。Wireshark 在有数据进入时实时滚动包列表面板，这样您将一直能看到最近的包。反之，则最新数据包会被放置在行末，但不会自动滚动面板。假如我们不选择 Update list of packets in real time （实时更新数据包列表）该选项的话，那么 Automatic scrolling in live capture （捕捉进行时自动滚动）选项将是灰色不可用状态。

Hide capture info dialog （隐藏捕捉信息对话框）如果我们取消选择该复选框，那么在 wireshark 执行捕捉进程时我们将看到一个数据包统计对话框。如图3-1-12所示：

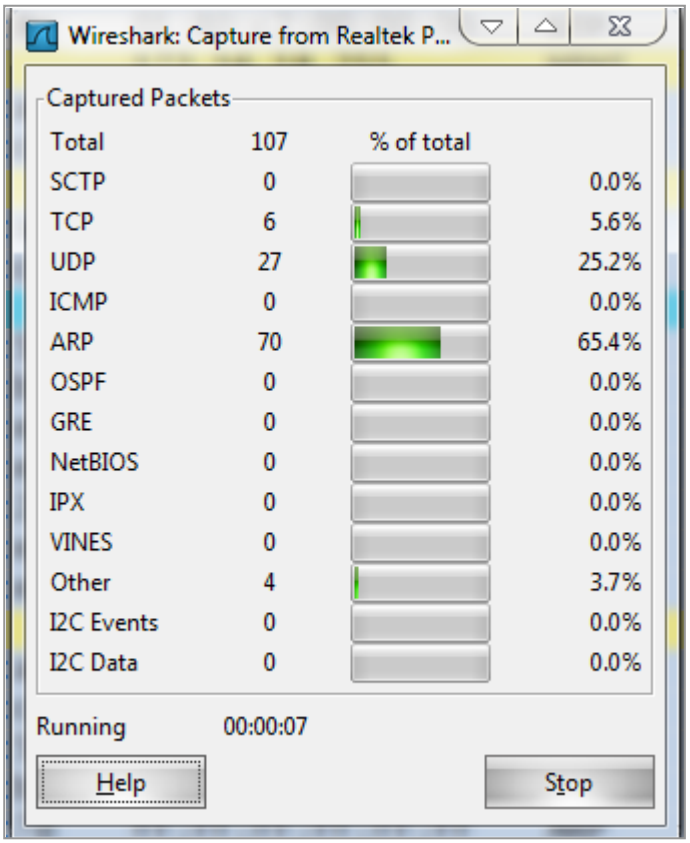


图3-1-12

依照数据包统计对话框我们可以看到在所有捕获到的数据包当中，哪种协议的数据包数量最多；可以根据实际应用情况选择以上三项功能；

最后要介绍的一项功能如图3-1-10所示，Name Resolution （名称解析）：该栏有三个选项可选，依次为 Enable MAC name resolution （开启 MAC 地址名字解析）、Enable network name resolution （开启网络地址名称解析）以及 Enable transport name resolution （开启传输协议解析）。

Enable MAC name resolutionARP（开启 MAC 地址名字解析）：名字解析 Wireshark 会向操作系统发出请求，将以太网地址转换为对应的 IP 地址(例如：00:09:5b:01:02:03->192.168.0.1)Ethernet codes(ethers file) 如果 ARP 解析错误，Wireshark 会尝试将以太网地址解析为已知设备名。这种解析需要用户指定一个 ethers 文件为 mac 地址分配名称。(e.g. 00:09:5b:01:02:03 -> homerouter).

Ethernet manufacturer codes (manuf file) 如果 ARP 解析和 ethers 文件都无法成功解析，Wireshark 会尝试转换 mac 地址的前三个字节为厂商名的缩写。mac 地址的前三个字节是 IEEE 为各厂商分配的独立地址(通过前三个字节可以得出每个网络设备的供应商，当然这些也是可以被篡改的。，)(e.g. 00:09:5b:01:02:03 -> Netgear_01:02:03).

Enable network name resolution （开启网络地址名称解析）：该功能开启和不开启基本无区别，建议不要开启；

Enable transport name resolution （开启传输协议解析）：开启该功能后，wireshark 会对所抓取的 TCP、UDP 协议的数据包发送接收的端口自动解析为相对应的服务。例如我们该功能后 wireshark 会自动将发送至 TCP 端口80的数据包解析为相对应的 HTTP 服务。如图3-1-13所示：

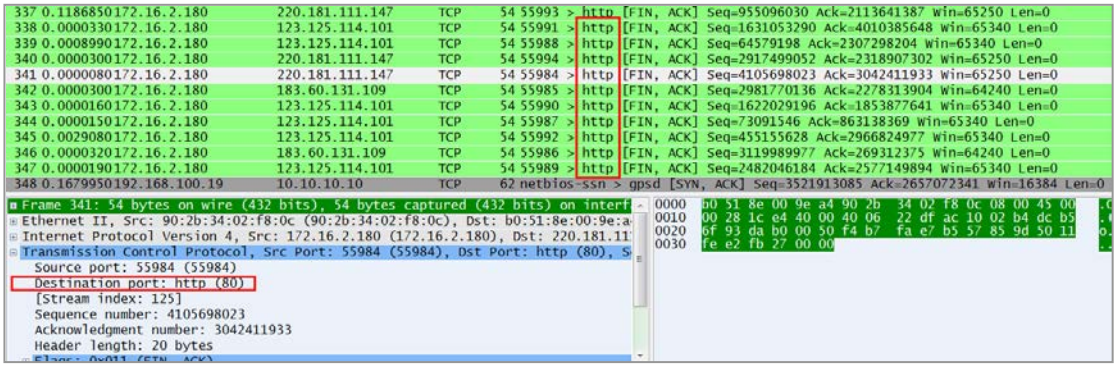
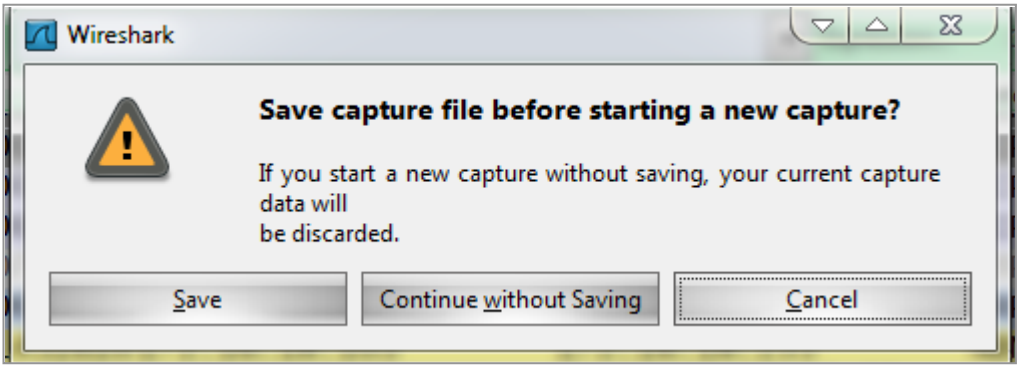


图3-1-13

如果我们不开启此功能的话那么 wireshark 将只会显示出所发送或接收数据包的端口号，而不会显示该端口用于什么服务；在这里我们不在做过多的演示操作；

 该图标用于开始一个新的抓包操作，点击后 wireshark 将会开始一个新的抓包进程，如果您已经抓取了数据包（数据包列表存在数据包）那么 wireshark 将会问您是不是对当前的数据包保存；如图3-1-14所示：



3-1-14

选择“Continue without Saving”（继续操作不做保存）后 wireshark 将会开始一个新的抓包进程；

 该图标用于停止一个抓包进程。当我们捕捉到合适数量的数据包后，我们点击该按钮 wireshark 将会停止抓包操作。这时我们即可对所抓取的数据包进行观察；

 重新开始一个新的抓包进程，如果我们没有抓取到我们需要的数据包，那么可以点击该按钮重新开始抓包；类似于  。在这里不再过多描述；

3.1.2、文件工具栏

如图3-1-1黄色矩形框内所示，文件工具栏有六个图标。他们分别代表：

 打开一个新的数据包文件；（与菜单栏--file--open 相同）

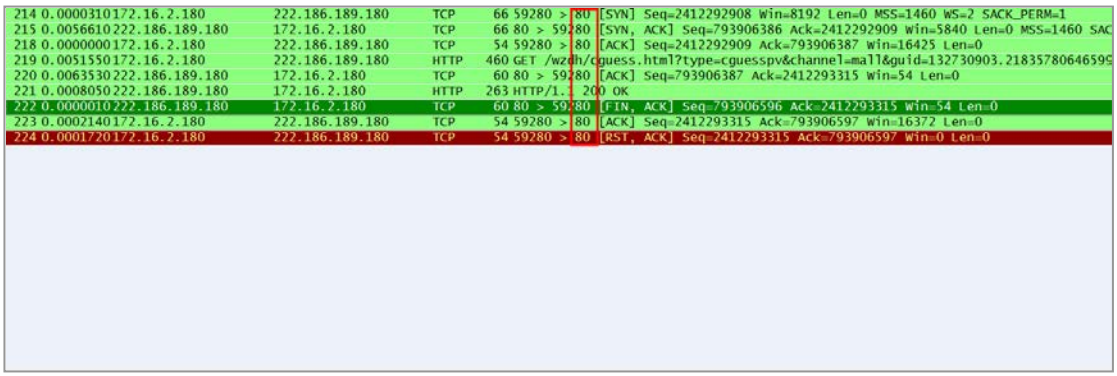
 保存当前所抓取的数据包为文件；（与菜单栏--file--Save 相同）

 关闭当前抓取的数据包列表显示界面；（与菜单栏--file--Close 相同）

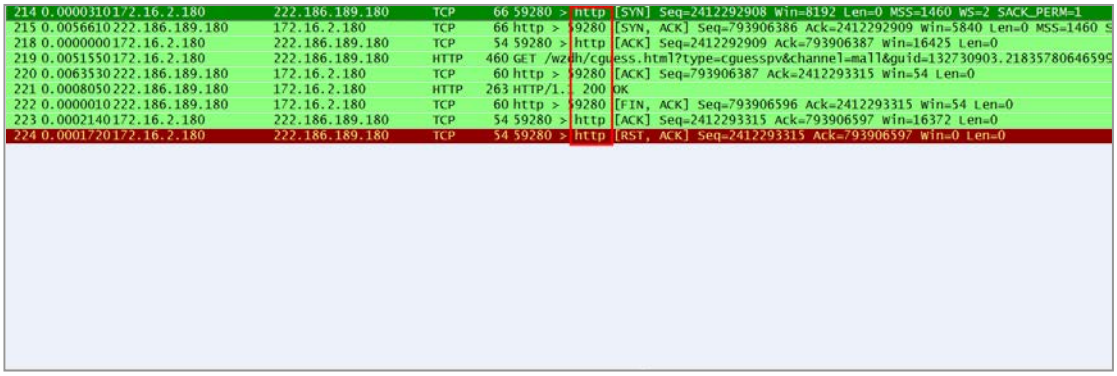
 重新载入当前的抓取的数据包;(与菜单栏--View--Reload 相同)

 打印当前捕捉的数据包；（与菜单栏--file--Print 相同）

从以上介绍大家也可以看出，该工具栏主要用于数据包问价的操作。由于有些操作简单在这里就不一一的介绍了。大家可以自己下载 wireshark 测试一下各个图标的功能。其中重新载入  ，当我们修改了某些显示或者关于数据包列表的操作后，在数据包列表我们可能看不到有什么变化。但是我们点击该按钮重新载入一下，即可我们看到操作所执行的结果。该功能就相当于我们的刷新功能。例如我们修改了主菜单栏--View--name resolution--enable for transport layer （该功能用于开启传输层名称解析，详见3.1.1小节 Enable transport name resolution （开启传输协议解析）选项）后我们观察数据包列表没有任何反应，这时候我们点击  后可以看到 HTTP 传输的目的端口由80变成了 HTTP。如图3-1-16：



（没有做修改刷新前）图3-1-15



（修改刷新后）图3-1-16

从以上两幅图红色矩形框内可以看到 wireshark 把80端口解析为常用的 HTTP 协议。当然在此我们只是举一个小小的示例 ,刷新功能还有其他的用法。这就需要大家实际操作中慢慢的来进行发现了；

3.1.3、包查找工具栏

如图3-1-1黄色矩形框内所示，该工具栏主用于数据包的定位操作。各图标的功能如下：

 该功能我们会在后续与抓包过滤器一起介绍；

 在数据包列表向后显示鼠标所点击过的数据包；

 在数据包列表向前显示鼠标所点击过的数据包；

 定位到特定数据包序号；(点击该按钮会弹出对话框，指定特定的数据包序列号点击 “jump to”)

 定位到第一个数据包；

 定位到最后一个数据包；

3.1.4、颜色定义工具栏

如图3-1-1深蓝色矩形框内所示，该栏具有两个图标。他们分别为：

 颜色控制按钮；

 抓取数据包滚动/不滚动；

 该按钮我们点击可取消/开启数据包列表颜色显示；

 如果我们不选择该按钮未开启状态那么我们抓取数据包时数据包列表显示将不会跟随最后一个抓取数据包来进行自动滚动（详见3.1.1小节 Update list of packets in real time （实时更新数据包列表）选项）；

3.1.5、字体大小工具栏

图3-1-1浅蓝色矩形框内所示，该栏主要设置数据包列表字体显示大小。每个图标如下所示：

 放大一级显示数据包列表字体；

 缩小一级显示数据包列表字体；

 调整数据包列表字体到原始大小；

 调整所有列宽到原始大小；

3.1.6、首选项工具栏

如图3-1-1紫色矩形框内所示，该栏我们主要有颜色以及首选项的设置两个按钮。它们分别是：


颜色规则定义；


wireshark 首选项设置；


点击该按钮弹出颜色显示规则定义对话框，我们可以随个人的习惯定义各种协议的数据包的颜色显示规则。如图3-1-17所示：

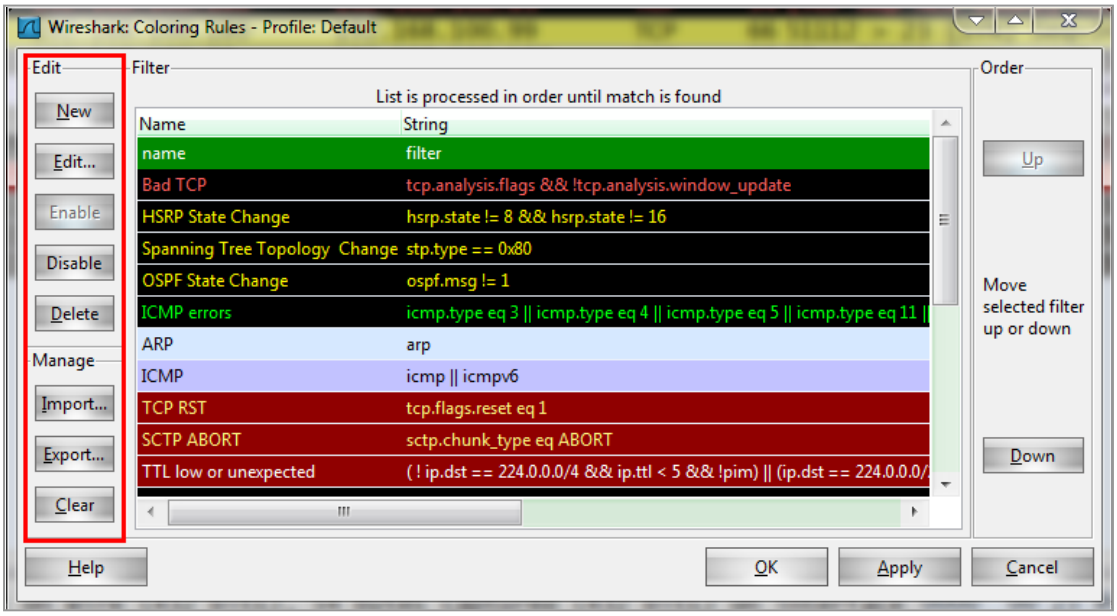


图3-1-17

在图3-1-17对话框红色矩形框内所示即“Edit”一栏，我们可以 New（新建规则）、Edit（编辑现有规则）、Enable（启用规则）、Disable（停用规则）以及删除颜色规则；例如我们定义 tcp 协议 flag 标记为 syn 的数据包显示为黑色字体黄色背景色，则我们可以如图3-1-19所示填写；

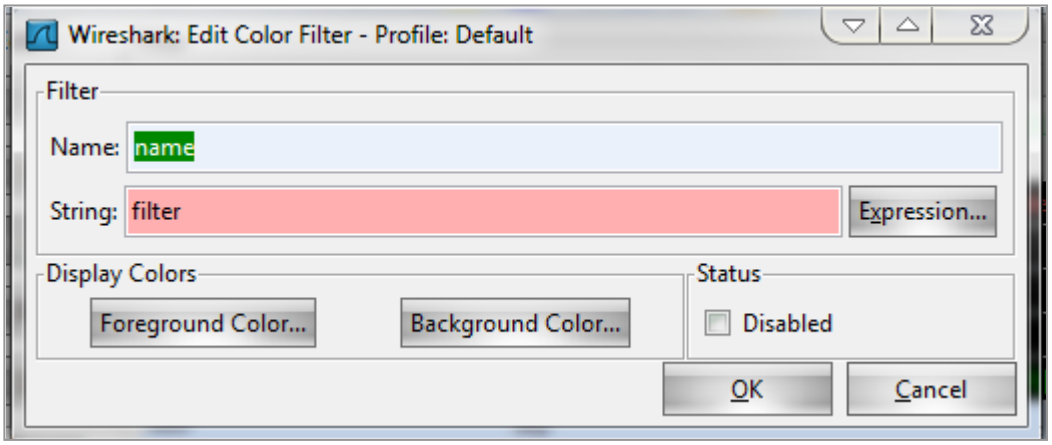


图3-1-18

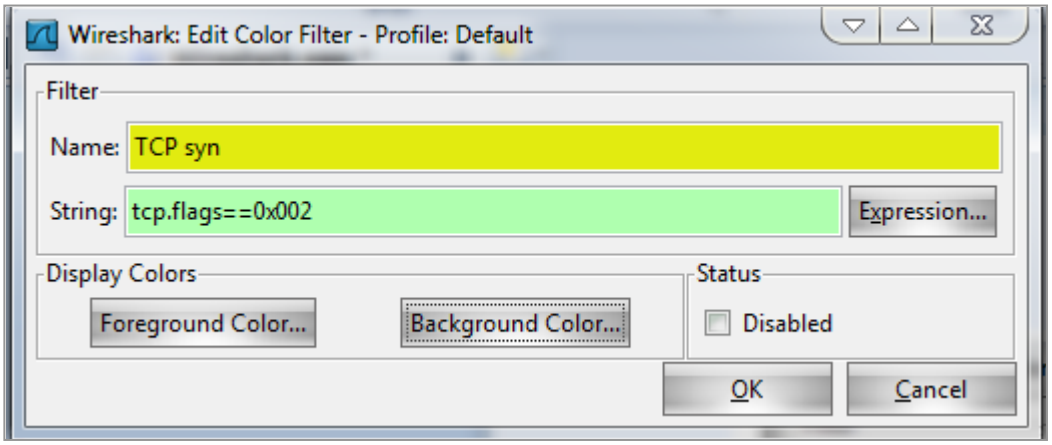


图3-1-19

而“Manage”一栏则可以执行导入导出规则。点击“Import”我们可以执行导入规则配置文件；“Export”可以对现有的规则执行导出操作，以便日后导入。再下面的“Clear”则是对现有的规则进行清除操作；

在所有的设置都做完的情况下我们可以点击“OK”或者“Aplly”直接应用；

点击该按钮我们可以对 wireshark 进行一些配置操作，例如我么你可以在此处更改 wireshark 的显示、抓包、名字解析等一些配置；首先我们第一个要说的就是 wireshark 的三个显示列表的布局，即数据包列表、数据包详细信息列表以及数据包字节数列表的布局；如图3-1-20所示：

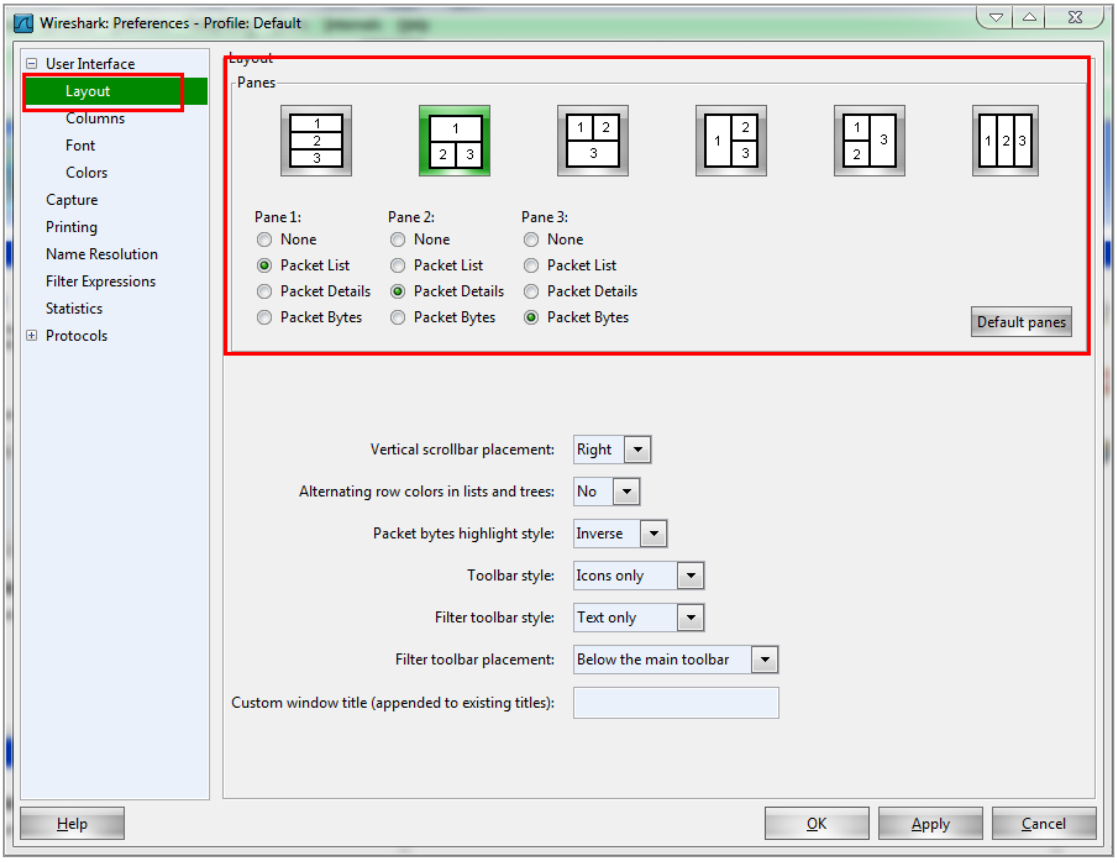


图3-1-20

打开首选项设置按钮后我们点击左边栏的“Layout”（布局），可以从右边栏看到有六种布局方式，我们可以根据自己的喜好选择他们的显示位置；如上图3-1-20所示那样我选择的是第二种方式。

点击左边栏的第三个“Columnt”我们可以定义数据包列表所显出的列。如图3-1-21所示那样，我选择的是显示所有列；

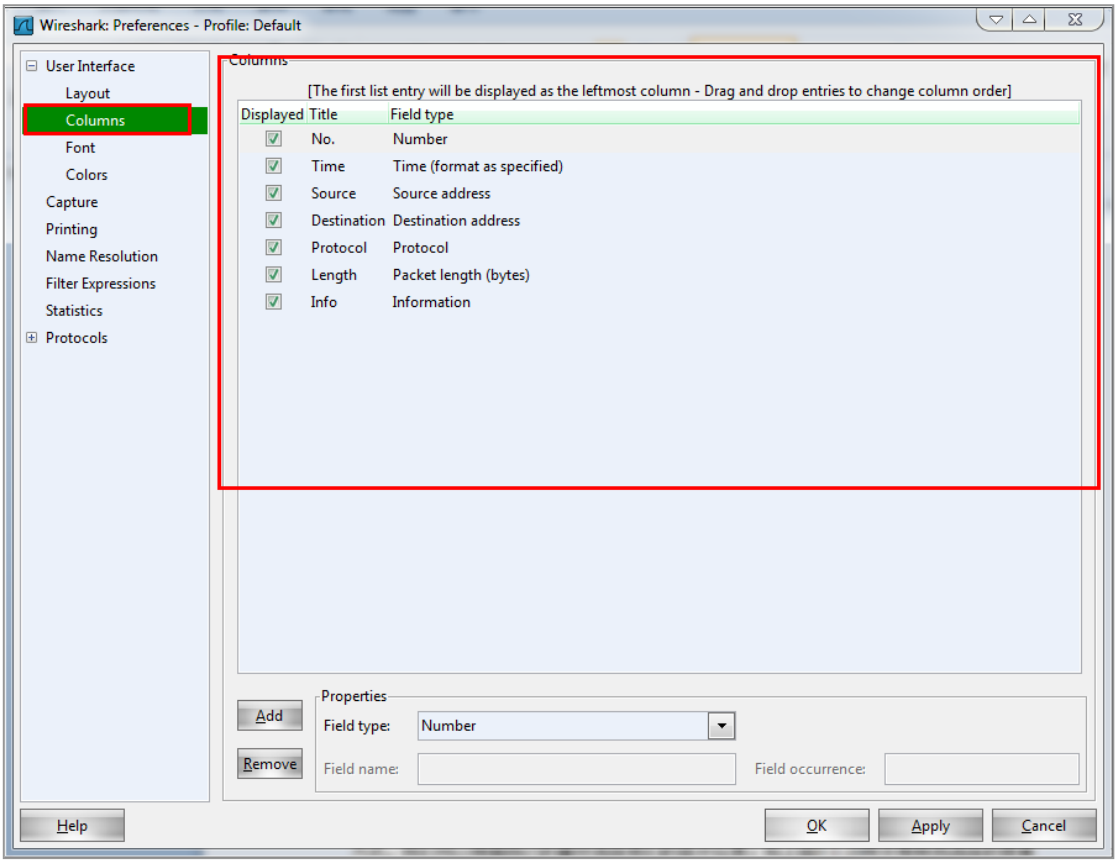


图3-1-21

“Font”（字体）在此处我么可定义 wireshark 的默认字体，如图3-1-22所示那样有多重字体供选择；可根据个人设置相应字体；

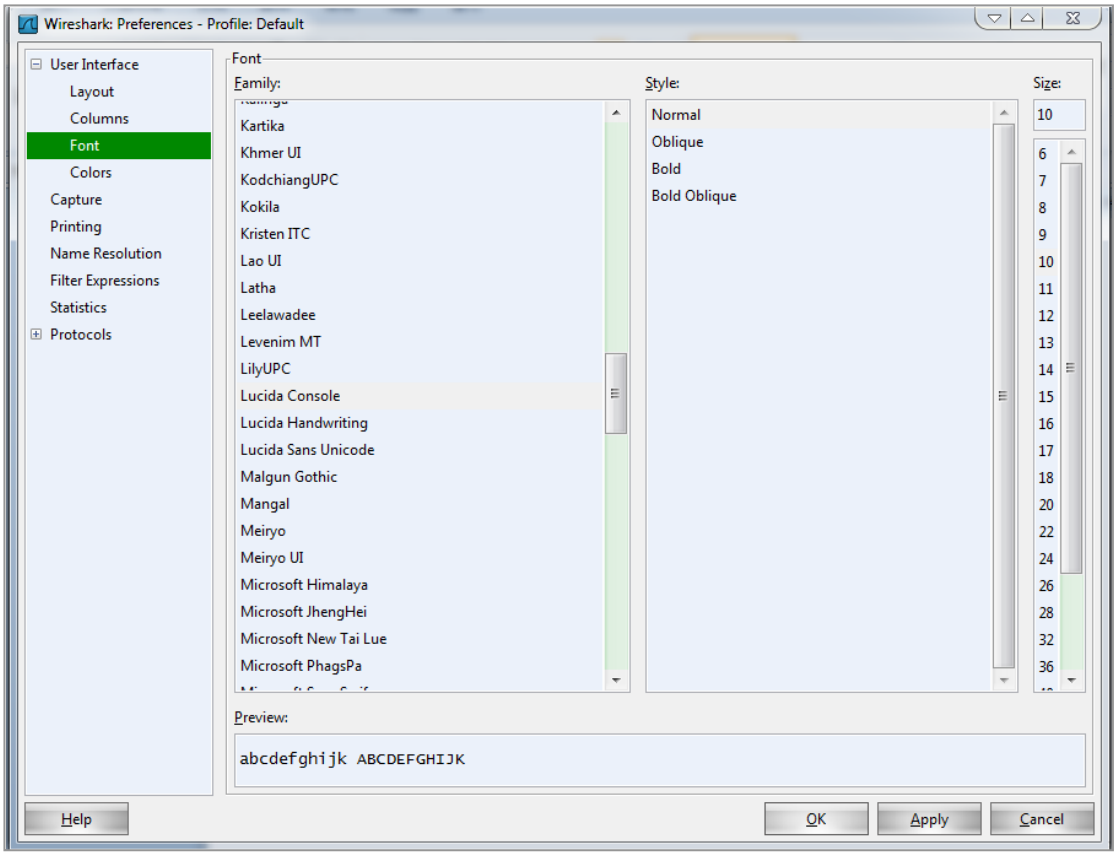


图3-1-22

再下面的就是“colors”（颜色），定义数据包列表各种协议数据包的显示颜色。图3-1-23所示，我们可以定义 TCP 流、UDP 流会话的颜色规则；

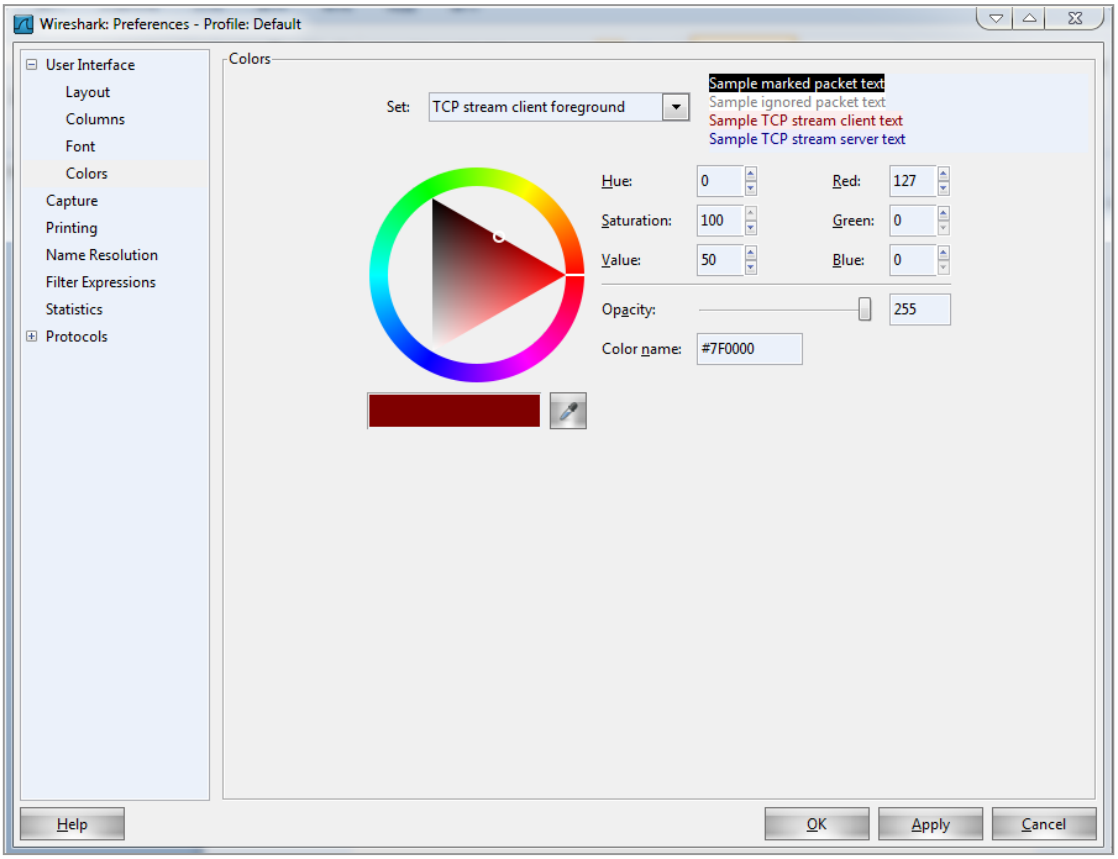


图3-1-23

“Capture”栏定义了抓取数据包时候的一些特性。详细解释如图3-1-24带色矩形框内所标识：

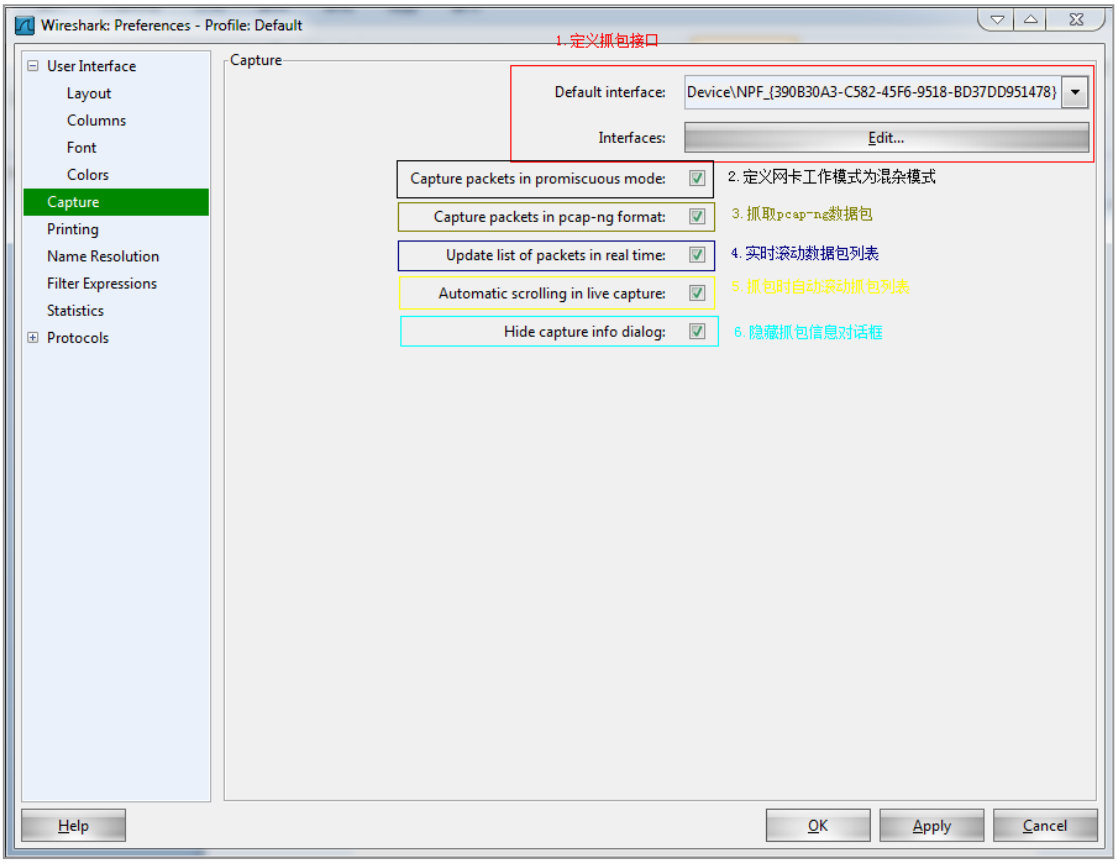


图3-1-24

综上使我们在首选项设置里面经常使用到的一些设置，而下面的“Printing”（打印）由于不常使用我们在这里就不多做介绍了。“Name Resolutins”（名称解析）这个选项相同于我们之前所介绍的抓包工具栏里面的名称解析一样用法，在此我么也不再赘述。“Protocol”（协议）一栏我们放在后面在做介绍。在此我们就把 wireshark 的主工具栏一栏基本介绍完了。以上所说的选项已经足够我们日常使用的了，而为了更加的详细说明 wireshark 软件，我们在后面还会相继探讨菜单栏，以及 wireshark 的一些日常抓包分析和常见使用问题。

◆ 菜单简介

4.2.0、菜单栏

如图4-2-1所示红色矩形框内的一排选项 ,我们称之为菜单栏 ;我们着重介绍常用的 :file(文件)、Edit (编辑)、View (查看)、Go (定位)、Captur (抓包)、Analyze (分析) 以及 Statistics (统计) 几个选项。由于之前所介绍的主菜单栏和菜单栏有诸多重复选项，在此为了节省时间我们对于重复的选项不再进行介绍。

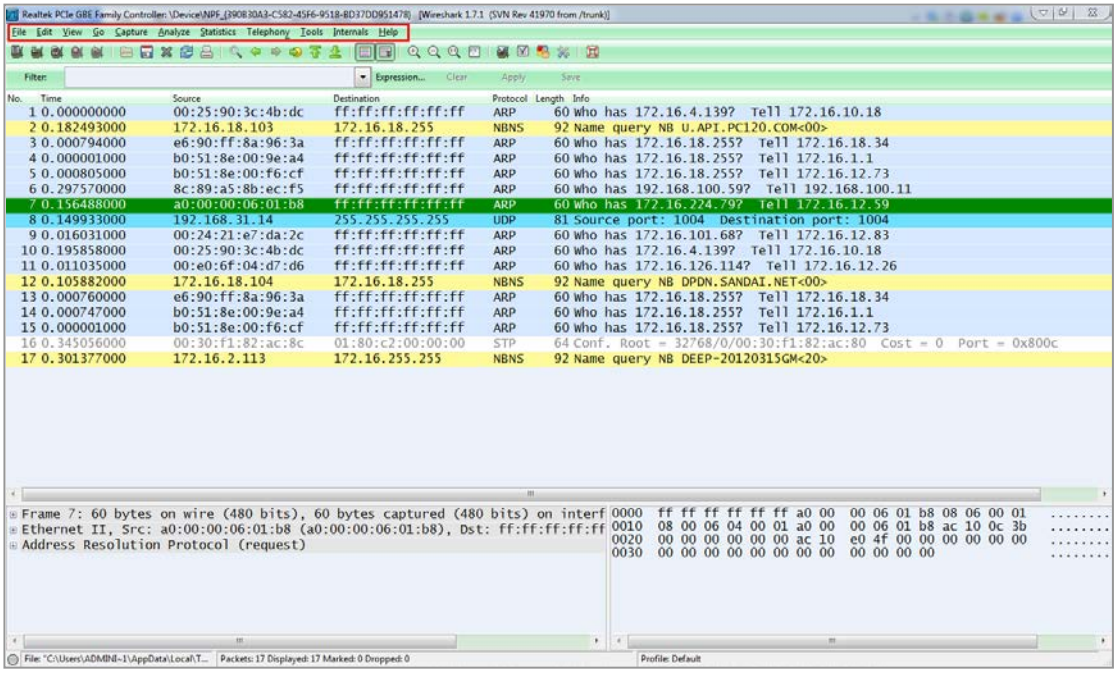


图4-2-1

4.2.1、file 菜单

File 菜单定义了一些数据包文件的操作，我们可以使用它对我们当前的抓取的数据包进行保存、打印、合并、现有数据包问价的打开等操作...如图4-2-2所示：

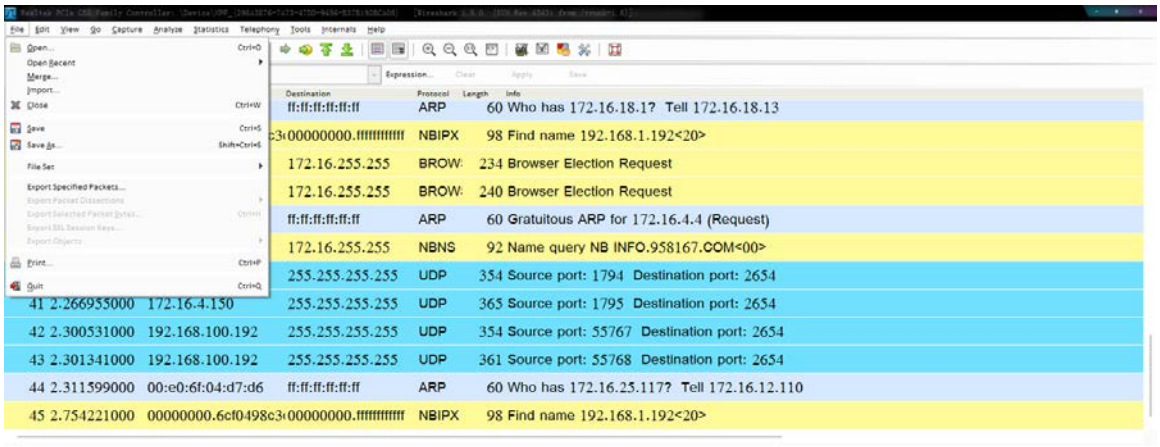


图4-2-2

从上到下主要应用到选项依次为：

Merge...（文件合并）：使用该功能我们可以同时打开两个.cap 文件并且显示在一起。首先我们先打开一个现有的.cap 文件，然后点击文件合并选项再次浏览到我们想打开的第二个.cap 文件，并执行打开操作；

Import（导入）：该选项目前测试有问题。

Close（关闭）：关闭当前抓取的数据包显示列表；

Save（储存）：保存为数据包文件；

Save as（另存为）：另存到数据包文件；

File Set(文件集合）:在该选项下面可以看到目前缓存的文件名，以及文件存储位置。如图4-2-3所示：

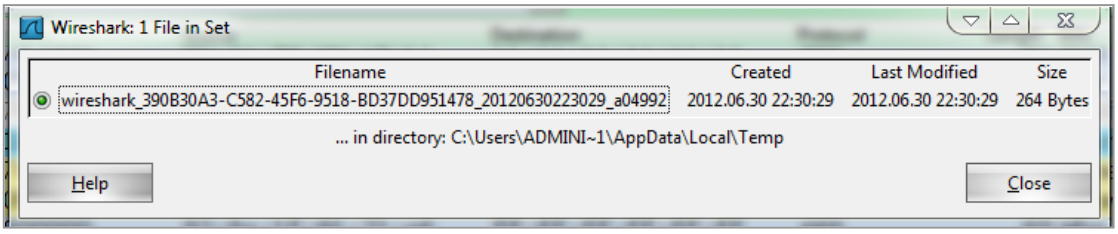


图4-2-3

Export（导出）：输出当前抓取数据包文件为.txt 文本模式。如图4-2-4所示：

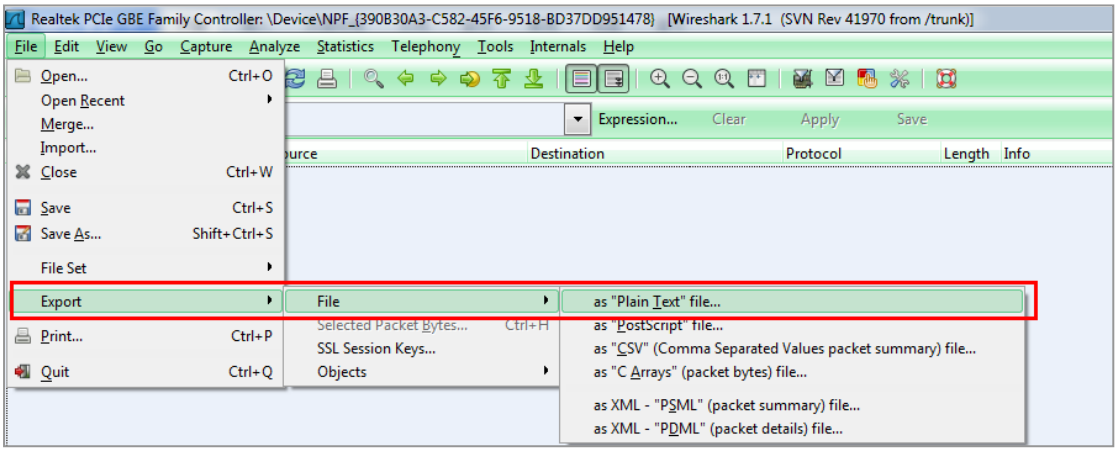


图4-2-4

4.2.2、Edit 菜单

Edit（编辑）菜单主要定义了一些数据包列表的编辑操作，其中有数据包的查找、标记、忽略以及设置参考时间等操作。如下图4-2-4示：

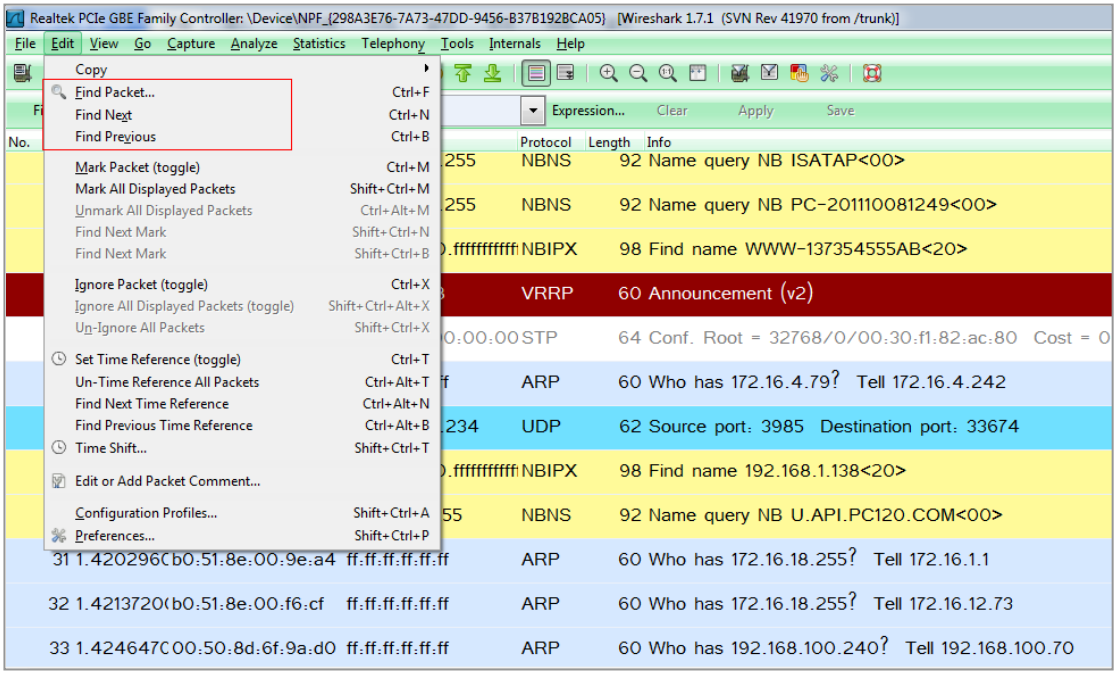


图4-2-4

Copy（拷贝）：该选项用于拷贝数据包详细列表各个字段的值或者描述；

Find Packet（查找数据包）：如图4-2-4红色矩形框内所示选项用于数据包的查找，我们将在后面的显示过滤器里面介绍；

Mark Packet（toggle）：（标记数据包）：被标记数据包显示颜色为黑底白字，被标记的数据包可用于定位查找；

Ignore Packet（toggle）：（忽略数据包）：选中想要忽略不显示的数据包列表条目，使用此选项既可以忽而略该数据包的显示；

Set Time Referencer（设置参考时间）：选中想要作为参考的数据包，使用该选项可以定义参考时间；参考时间可以用于定位查找操作；

Time Shift（时间修改）：修改数据包，该选项基本使用不到所以不做介绍；

Edit or Add Packet Comment（修改添加包描述）：选中数据包列表想要备注描述的数据包，然后点击该选项，弹出对话框后我们在弹出的对话框中可以对数据包添加注释；如图4-2-5所示：

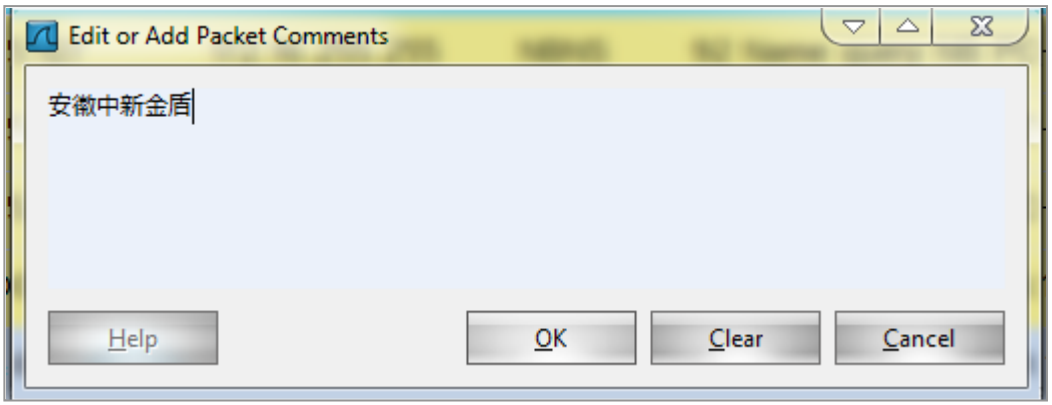


图4-2-5

图上所示为示例，实际上 wireshark 不支持中文描述。如果添加中文注释我们在下面数据包详细列表将看到的是乱码。大家以后也可以自己尝试下试试。

Configuration Profiles（配置文件）：这个里面存储着我们所对 wireshark 的配置，假如我们有多套配置的话，那么打开该选项弹出的对话框内是有多个条目的，我们可以选中想要使用的那套配置文件，点击"Apply"即可；如图示4-2-6:

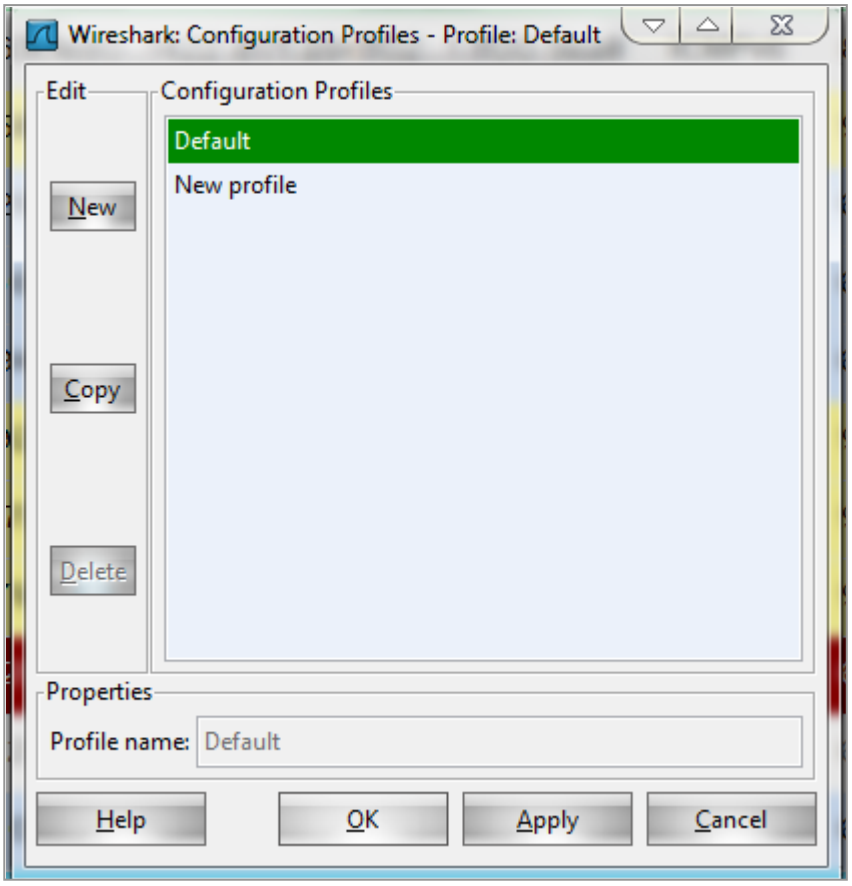


图4-2-6

当前我有两个配置文件，一个是正在使用的"Default"，还一个是没有做任何修改的配置文件（相当于初始配置）。假如我们选中“New Profiles”后点击“Aplly”后那么 wireshark 将恢复到默认配置；我们再次点击上面的“Default”应用后 wireshark 将恢复到我们所定义的显示以及操作方式；

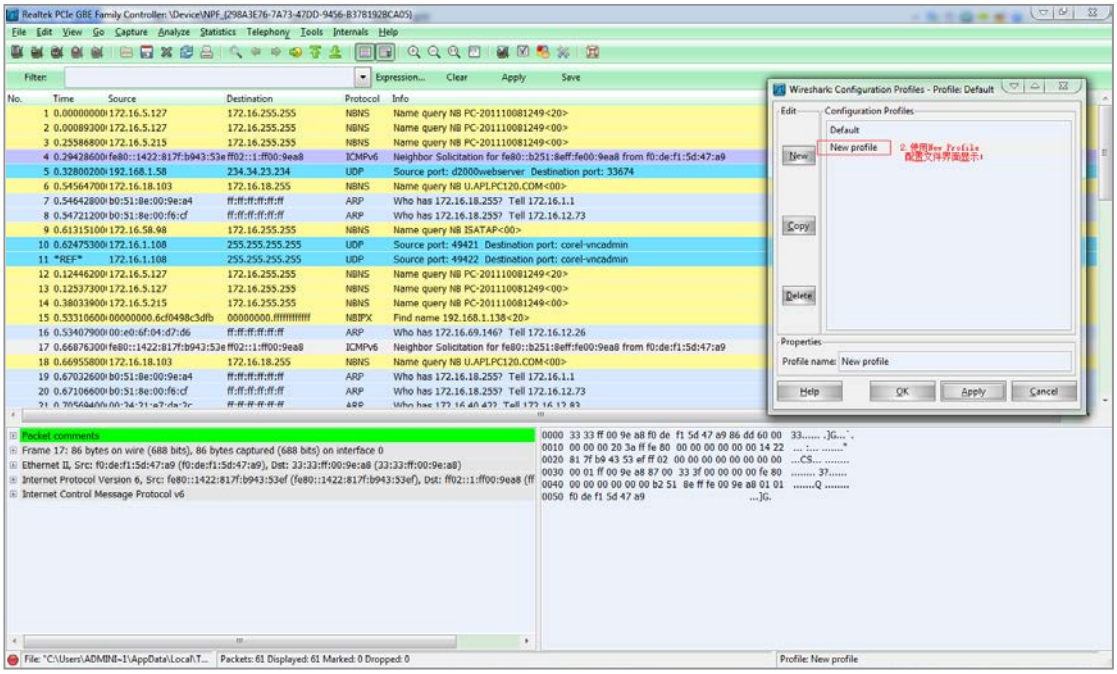


图4-2-7

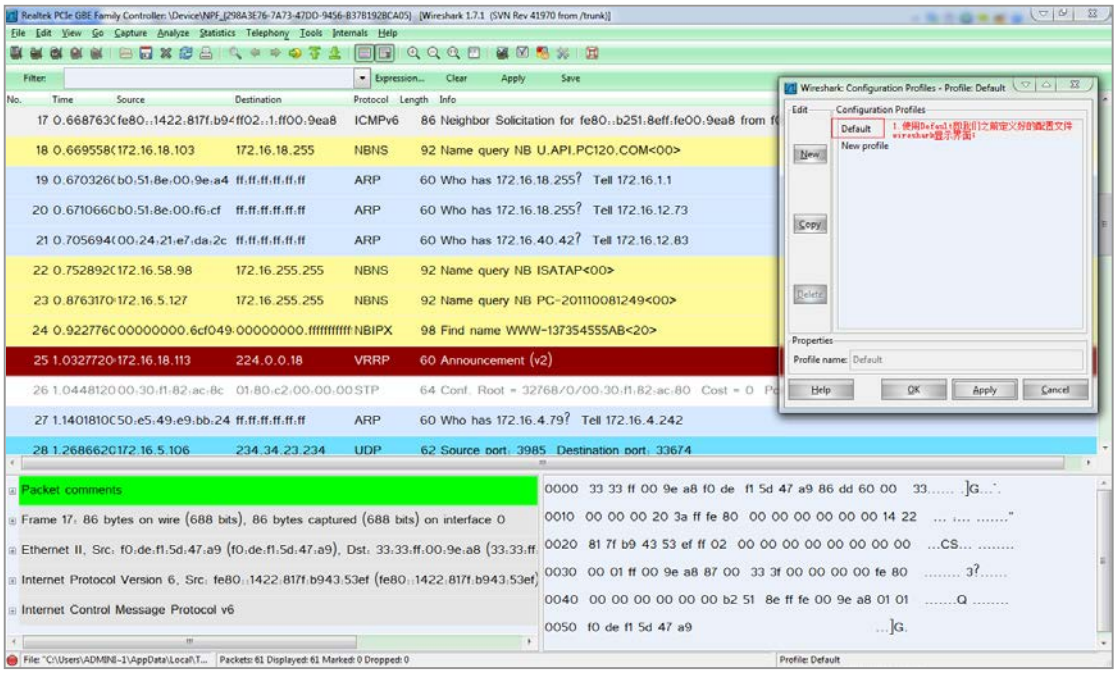


图4-2-8

Preferences（首选项设置）：该选项相同于以上3.1.6、首选项工具栏介绍的首选项设置；

4.2.3、View 菜单

该选项定义了 wireshark 的一些现实特性 ,我们可在此对 wireshark 的主界面做一些显示方面的修改操作。如图示4-2-9：

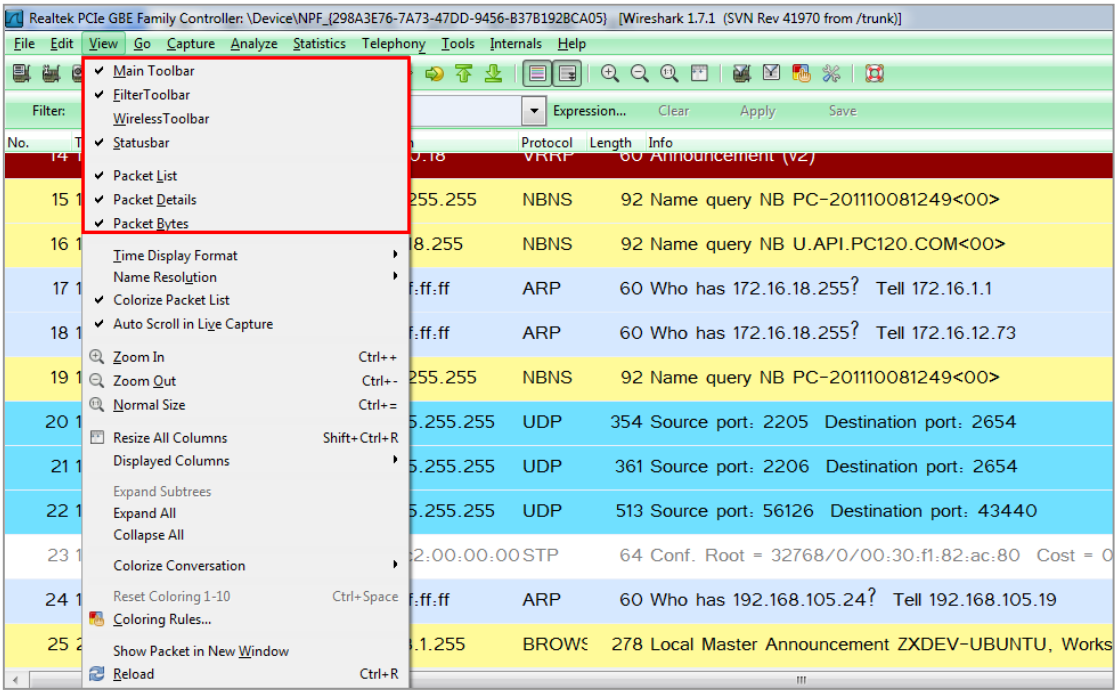


图4-2-9

如图4-2-9红色矩形框内所示几个选项定义了 wireshark 的菜单显示/不显示在主界面，“√”是显示出该栏，如图示显示了：Main Toolbar（主菜单栏）、Fileter Toolbar（过滤工具栏）等；

Time Display Format（时间显示格式）：如图4-2-10所示：

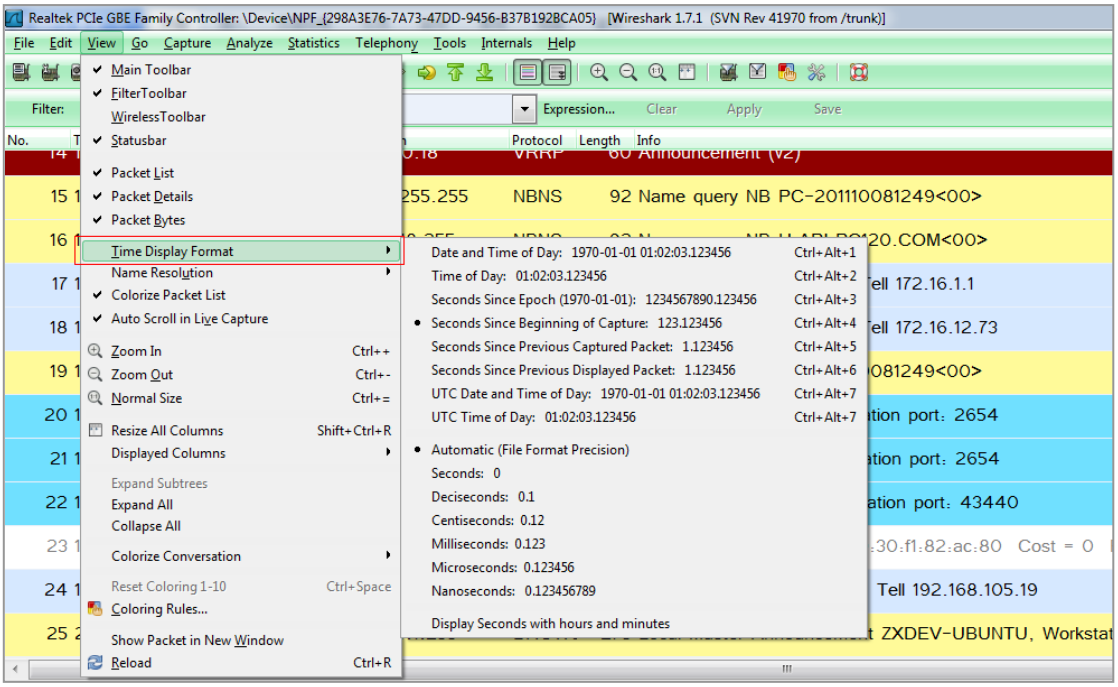


图4-2-10

如图4-2-10所示，在弹出的子菜单栏内有8种事件格式供我们选择。默认的显示方式是：Seconds Since Begingning of Capture：123.123456（距离开始抓取数据包时候的秒数）；

我们也可以根据需要定义其他显示方式，例如：Second Since Previous Capture Packet：1.123456（距离抓取的上一个包的秒数）；

Second Since Previous Display Packet：1.123456（距离上一个包显示的秒数）

除了以上所说的外我们还可以定义其他的显示方式，这个大家自己可以测试一下，看一看时间列的不同；我们在这里就不多做解释了。

Automatic（File Format Precision）（自动（文件格式优先））：该选项是 wireshark 默认的时间显示精度；除了默认值“Automatic”外 wireshark 还定义了另外几种供选择使用。例如：Seconds（秒）、Deciseconds（十分之一秒）、Millisconds（毫秒）以及 Microseconds（微秒）；我们普通抓包如果不选择“Automatic”手动选择到 Microseconds（微秒）级别就可以了。

Name Resolution（名称解析）：见 3.1.1、抓包工具栏；

Colorize Packet List（颜色显示数据包列表）：点击可以对数据包列表取消/应用颜色规则；

Auto Scroll in Live Capture（抓包时自动滚动数据包列表）：点击可取消/应用抓包时自动滚动数据包列表；相同于3.1.4、颜色定义工具栏；

Zoom In、Zoom Out、Normal Size 和 Resize All Colunns：相同于3.1.5、字体大小工具栏；

Displayed Columns（显示列数）:该功能可参考3.1.6、首选项工具栏--首选项设置--Culomns；

Expand Subtree、Expand All、Collapse All：分别是展开子树、展开所有子树和合并所有子树；如图4-2-11,4-2-12所示:

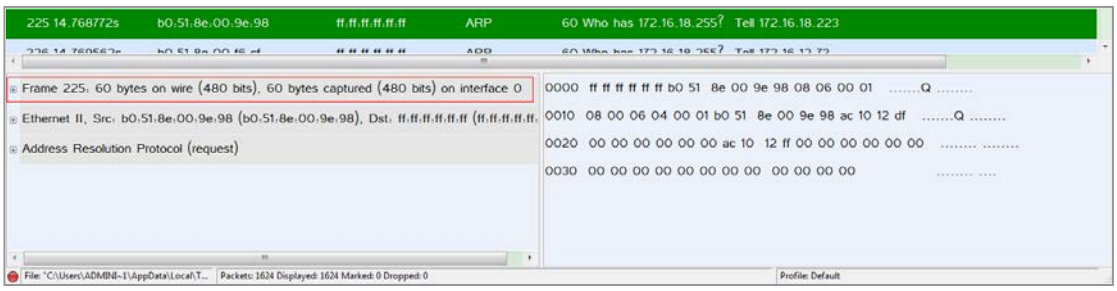


图4-2-11

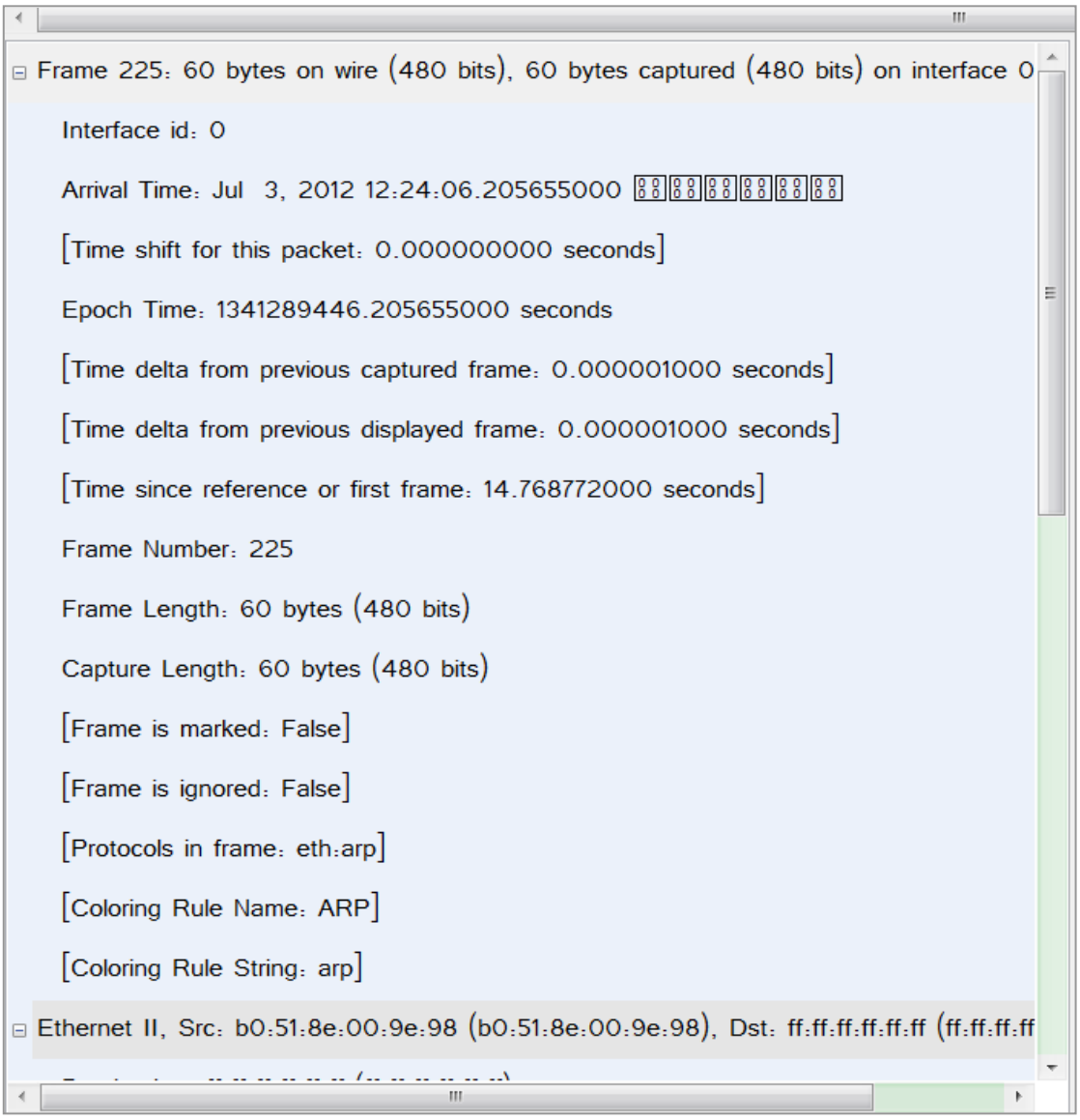


图4-2-12

Colorize Converation（会话流颜色规则）：重新标记一个会话流的显示颜色；

Reset Coloring 1-10（重置1-10颜色定义）：如果我们使用 Colorize Converation（颜色规则）定义了一组会话流的颜色后想要取消就是用该选项；

Coloring Rules（颜色规则）：参考3.1.6、首选项工具栏--颜色规则定义；

Show Packet in New Window（新窗口显示数据包）：鼠标对数据包列表的某一个数据包双击或者使用该选项将会在一个新的窗口打开数据包详细信息以及数据包字节数列表；

Reload（重新载入）：重新载入当前数据包列表；参考3.1.2、文件工具栏；

4.2.4、Go 菜单栏

如下图4-2-13示：

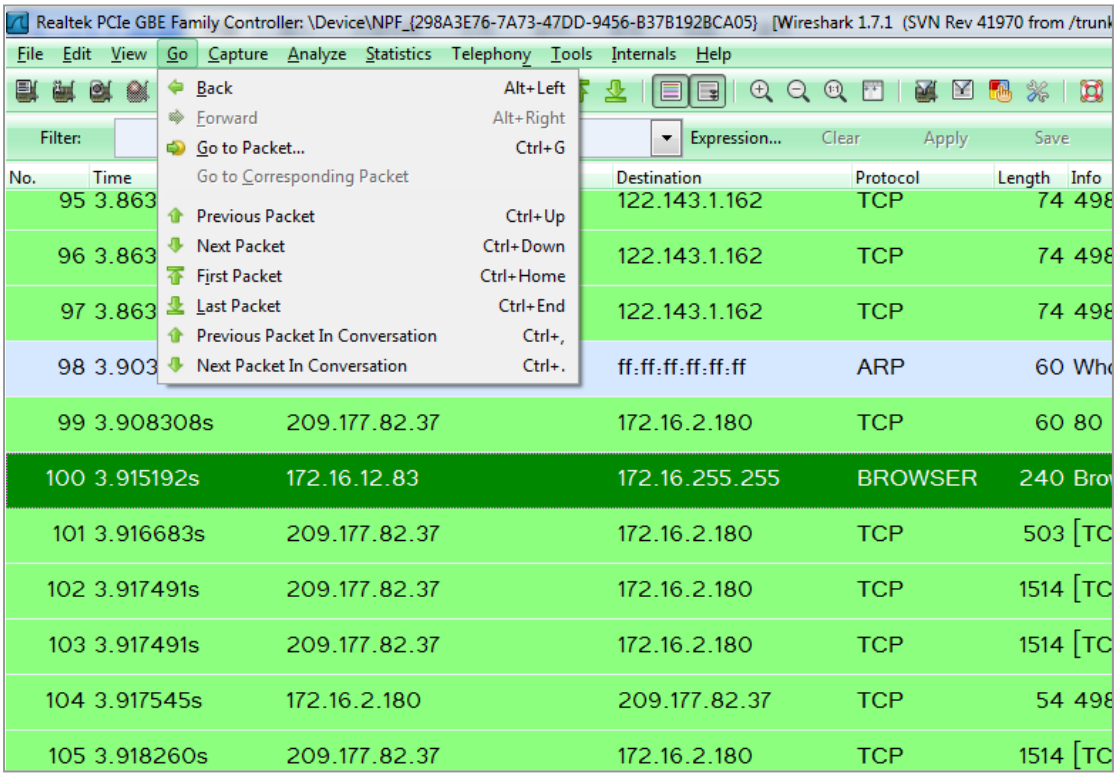


图4-2-13

该菜单栏可参考3.1.3、包查找工具栏；除了以上包查找工具栏里面所介绍的几个选项之外，该栏还额外多了一个：Go To Packet（定位到数据包）选项，点击该选项后弹出对话框。如图4-2-14示：

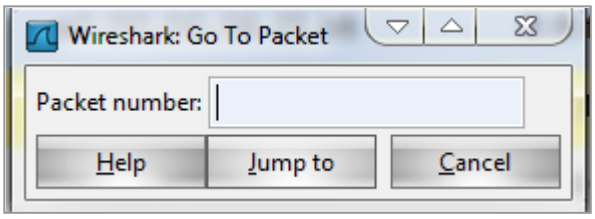


图4-2-14

在“Packet number”栏输入数据包相应编号即可快速定位到指定数据包；大家可以尝试下。例如我们想查看第100个捕捉到的数据包那么我们可在该栏输入100，点击“Jump to”即可跳至指定数据包100；

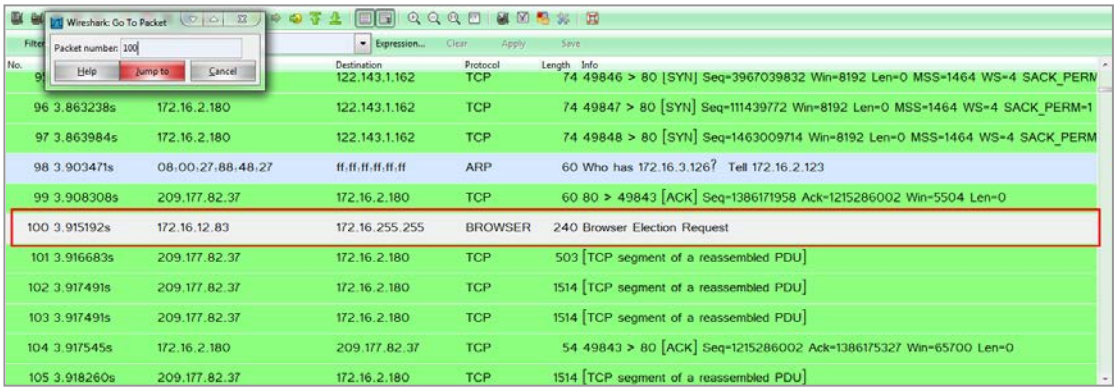


图4-2-15

如上图4-2-15所示，wireshark 定位到我们指定的100个数据包；

4.2.5、Capture 菜单栏

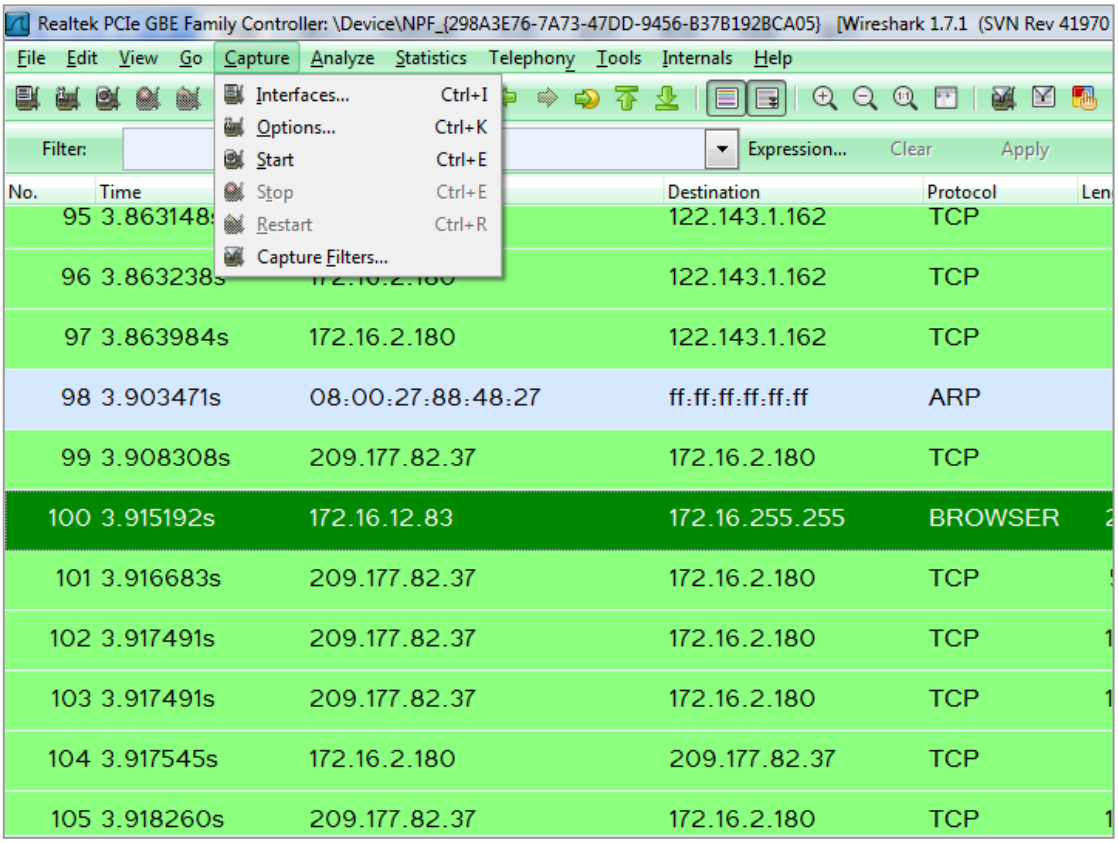


图4-2-16

该栏可参考 3.1.1、抓包工具栏；除了抓包工具栏所介绍过的几个选项外，我们还看到有一个：Capture Filter（抓包过滤器）选项，该选项我们放在后面与显示过滤器一起介绍；

4.2.6、Analyze 菜单栏

该栏主要用于数据包分析，我们可以使用该栏所定义的一些功能对数据包执行相应的操作。

如图 4-2-17 示：

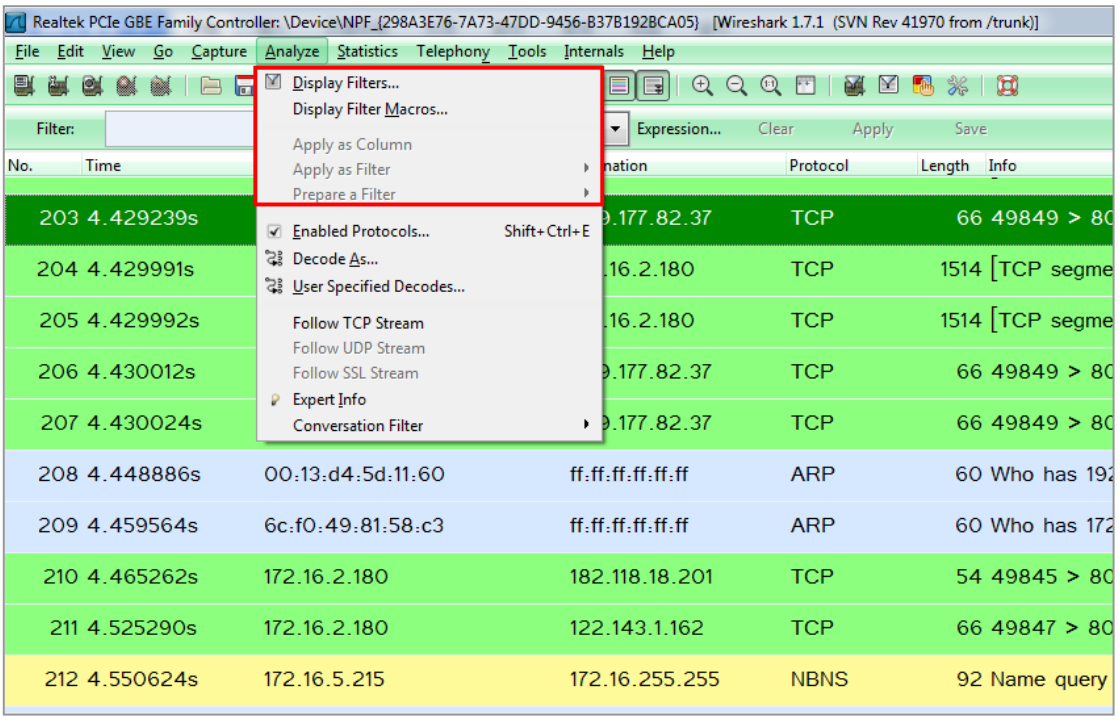


图 4-2-17

如上图 4-2-17 红色矩形框内的是显示过滤器，这个会在下面介绍到。我们先关注红色矩形框之外的几个常用选项；

Enabled Protocols（开启协议分析）：该选项用于告诉 wireshark 要分析哪些协议，也可以忽略一些协议让 wireshark 不对它进行分析以提高效率。点击该选项会弹出对话框，如图 4-2-18 所示：

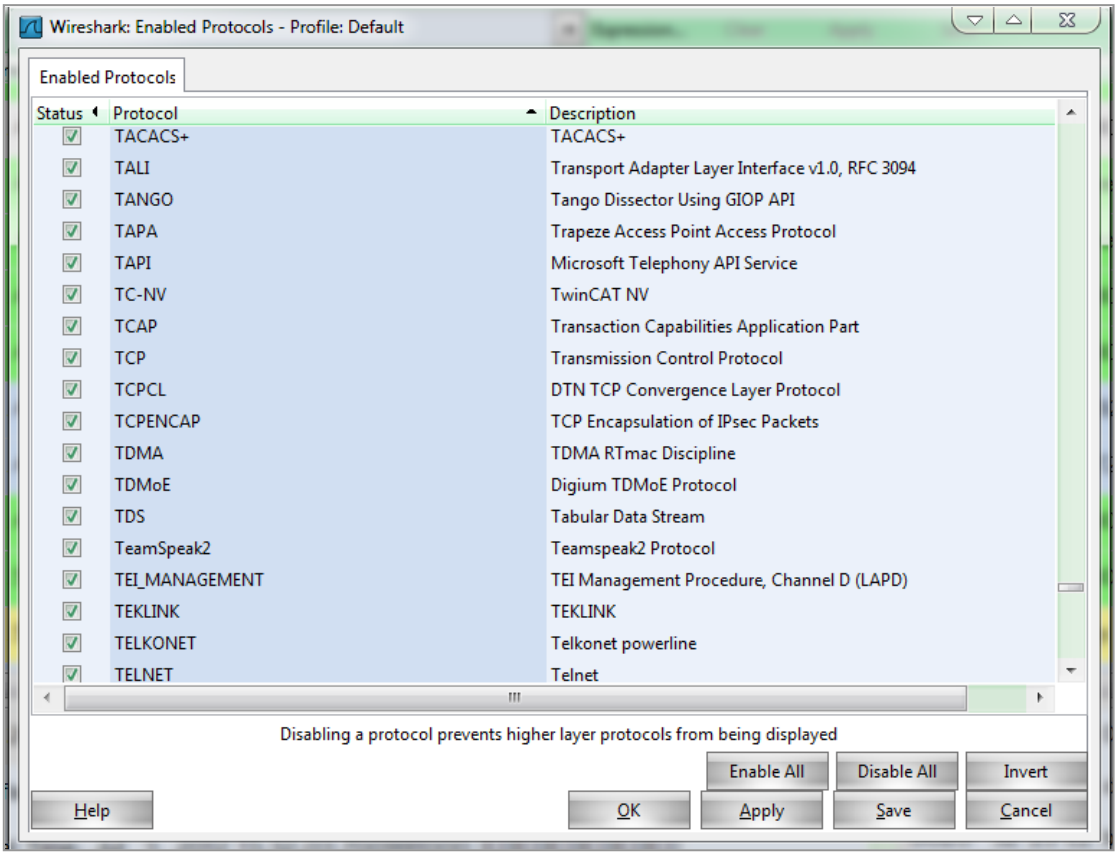


图 4-2-18

上图 4-2-18 几乎列出了所有的网路协议 ,但是我们常用到的也就那么几种(特殊行业应用除外) , 所以我们完全可以指定 wireshark 对特定的常用的协议来进行分析。

我们可以在该列表里面选择性的取消掉一些协议。例如我们取消掉对 TCP 协议数据包的分析后 wireshark 将不再显示 TCP 协议的数据包。如图 4-2-19,4-2-20 所示：

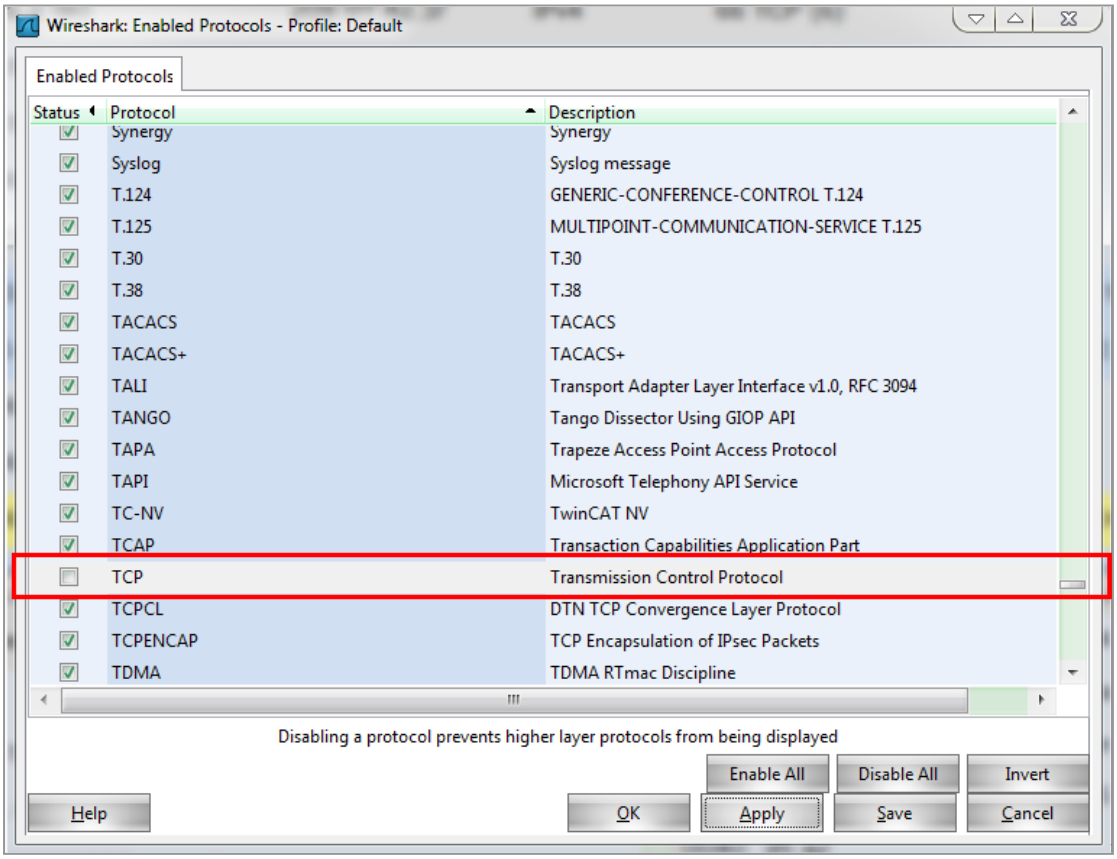

Wireshark: Enabled Protocols - Profile: Default
Enabled Protocols
Status
Protocol
Description
Synergy
Synergy
Syslog
Syslog message
T.124
GENERIC-CONFERENCE-CONTROL T.124
T.125
MULTIPOINT-COMMUNICATION-SERVICE T.125
T.30
T.30
T.38
T.38
TACACS
TACACS
TACACS+
TACACS+
TALI
Transport Adapter Layer Interface v1.0, RFC 3094
TANGO
Tango Dissector Using GIOP API
TAPA
Trapeze Access Point Access Protocol
TAPI
Microsoft Telephony API Service
TC-NV
TwinCAT NV
TCAP
Transaction Capabilities Application Part
TCP
Transmission Control Protocol
TCPCL
DTN TCP Convergence Layer Protocol
TCPENCAP
TCP Encapsulation of IPsec Packets
TDMA
TDMA RTmac Discipline
Disabling a protocol prevents higher layer protocols from being displayed
Enable All
Disable All
Invert
Help
OK
Apply
Save
Cancel

图 4-2-19

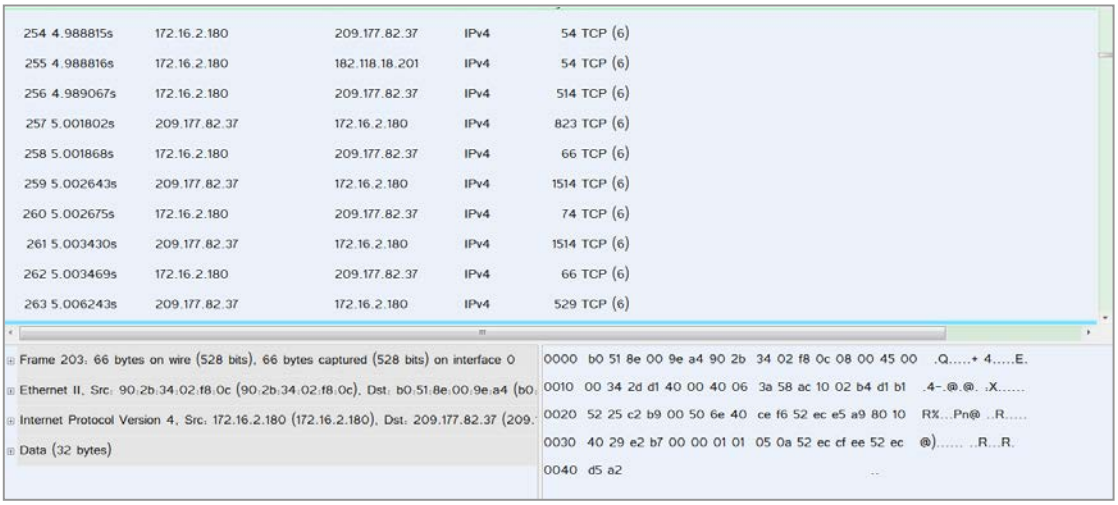

254 4.988815s 172.16.2.180 209.177.82.37 IPv4 54 TCP (6)
255 4.988816s 172.16.2.180 182.118.18.201 IPv4 54 TCP (6)
256 4.989067s 172.16.2.180 209.177.82.37 IPv4 514 TCP (6)
257 5.001802s 209.177.82.37 172.16.2.180 IPv4 823 TCP (6)
258 5.001868s 172.16.2.180 209.177.82.37 IPv4 66 TCP (6)
259 5.002643s 209.177.82.37 172.16.2.180 IPv4 1514 TCP (6)
260 5.002675s 172.16.2.180 209.177.82.37 IPv4 74 TCP (6)
261 5.003430s 209.177.82.37 172.16.2.180 IPv4 1514 TCP (6)
262 5.003469s 172.16.2.180 209.177.82.37 IPv4 66 TCP (6)
263 5.006243s 209.177.82.37 172.16.2.180 IPv4 529 TCP (6)
Frame 203: 66 bytes on wire (528 bits), 66 bytes captured (528 bits) on interface 0
Ethernet II, Src: 90:2b:34:02:f8:0c (90:2b:34:02:f8:0c), Dst: b0:51:8e:00:9e:a4 (b0:
Internet Protocol Version 4, Src: 172.16.2.180 (172.16.2.180), Dst: 209.177.82.37 (209.
Data (32 bytes)
0000 b0 51 8e 00 9e a4 90 2b 34 02 f8 0c 08 00 45 00 .Q.....+ 4.....E.
0010 00 34 2d d1 40 00 40 06 3a 58 ac 10 02 b4 d1 b1 .4-.@.@. :X.....
0020 52 25 c2 b9 00 50 6e 40 ce f6 52 ec e5 a9 80 10 R%...Pn@ ..R.....
0030 40 29 e2 b7 00 00 01 01 05 0a 52 ec cf ee 52 ec @)..... ..R...R.
0040 d5 a2 ..

图 4-2-20

点击应用后我们可以看到所有的 TCP 协议数据包将显示为白色，协议列为 IPv4 并且数据包详细信息列表将不再包含 TCP 协议的详细信息，取而代之的则是 IPv4 的 Data 字段。

除了取消掉某些协议的分析操作，我们也可以选择禁用所有，然后再选择我们想要分析的协议的数据包。Wireshark 将只会对于我们所选择的协议的数据进行捕捉分析。需要注意的是协议之间的关联，假如我们只选择 TCP 协议不选择对 IP、ethernet 协议的分析，那么我们将看到所有的数据包的协议列都是 UNKNOW。如图 4-2-21 所示：

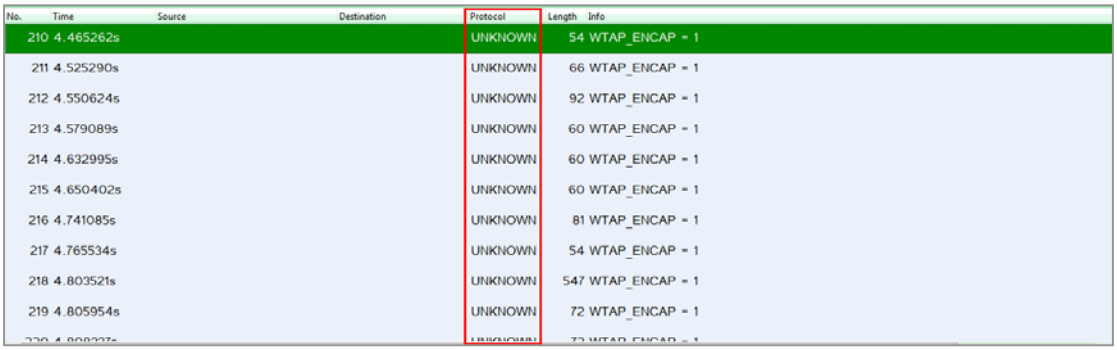


图 4-2-21

选择 ethernet 以及 IP 协议后我们看到数据包显示正常了，但是对于其他协议的数据包协议列将依然显示外一层封装的协议如图 4-2-22 所示：

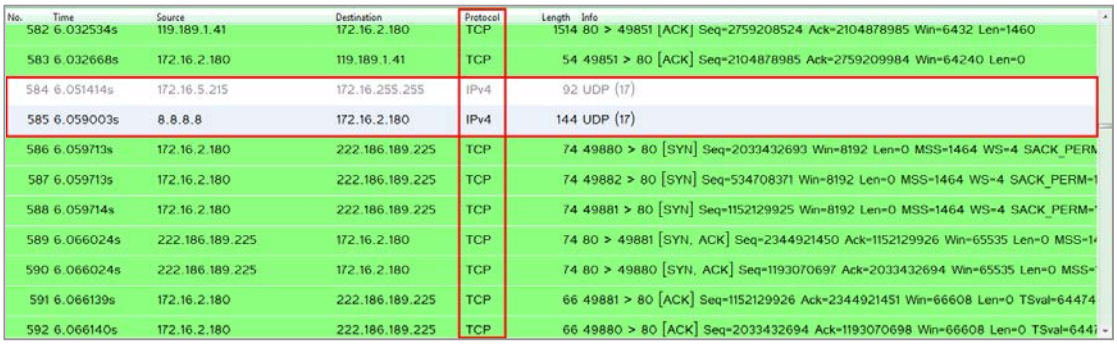


图 4-2-22

如上图 4-2-22 所示，由于我没有选择对 UDP 协议进行分析，所以显示的只是 UDP 的外层封装协议 IPv4；

Decode as（按指定协议规则解码）：该选项用于手动指定解码规则，wireshark 对抓取的数据包采用默认解码，但是有些时候我们需要自己来重新对数据包进行相应的解码，以方便我们的查看。例如，我们可以对有些手机用户访问的 TCP 数据流量指定按 HTTP 协议规则解码。这样我们可以看到数据包详细信息列表的 HTTP 请求，方便我们进行数据包分析；如图 4-2-23：

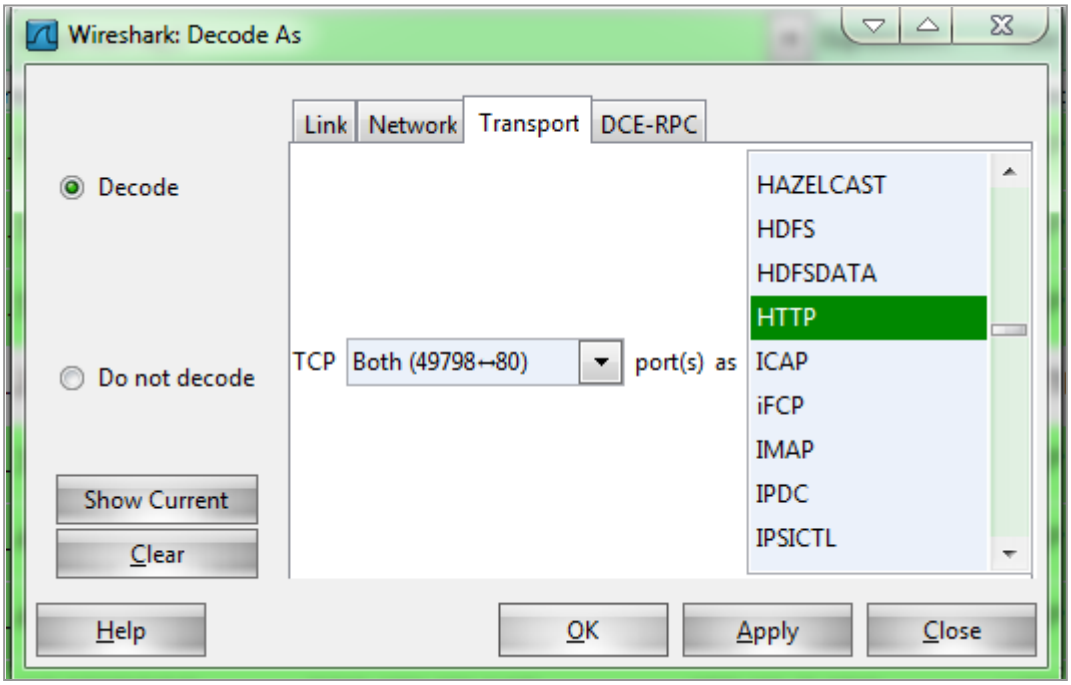
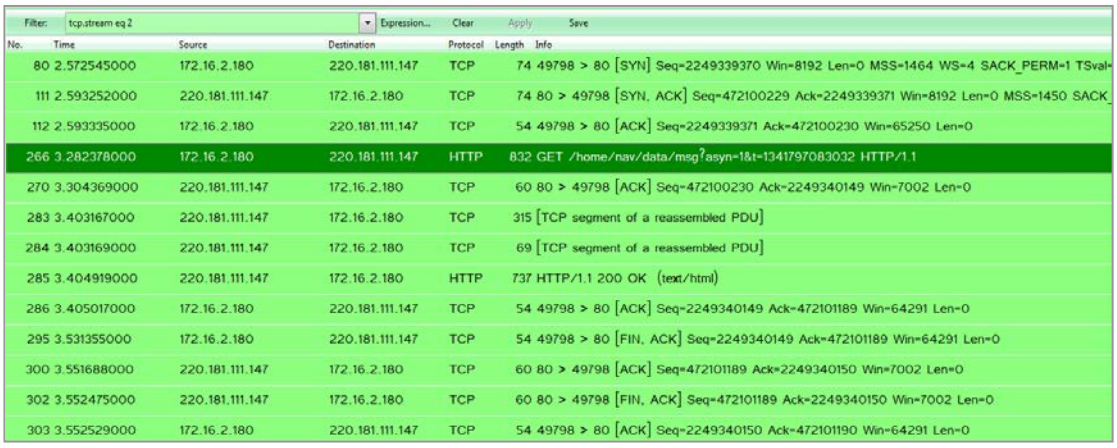
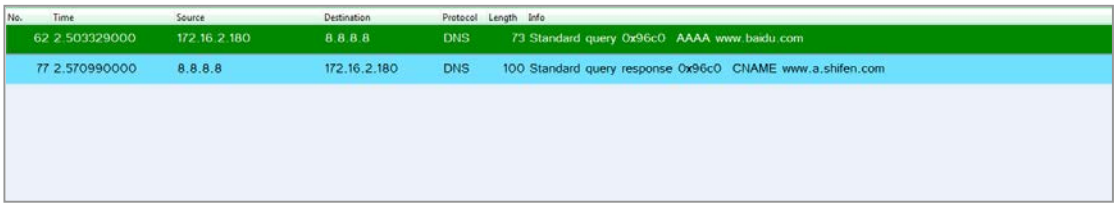


图 4-2-23

Follow TCP 和 Follow UDP：这两个选项在我们的数据包详细信息列表也有，鼠标选中某一个 TCP/UDP/SSL 的数据包之后点击该选项，wireshark 会自动列出属于该会话流的所有的数据包。如下图示：



执行 Follow TCP Stream 后 wireshark 自动筛选出该会话所有 TCP 数据包



执行 Follow UDP Stream 后 wireshark 自动筛选出该会话所有 UDP 数据包

执行 Follow SSL Stream 后 wireshark 自动筛选出该会话所有 SSL 数据包

Expert Info（综合专家信息）：该选项点击后会弹出一个对话框，对话框内列出了所有的会话所发送的各种数据包数量。如图 4-2-23 所示：

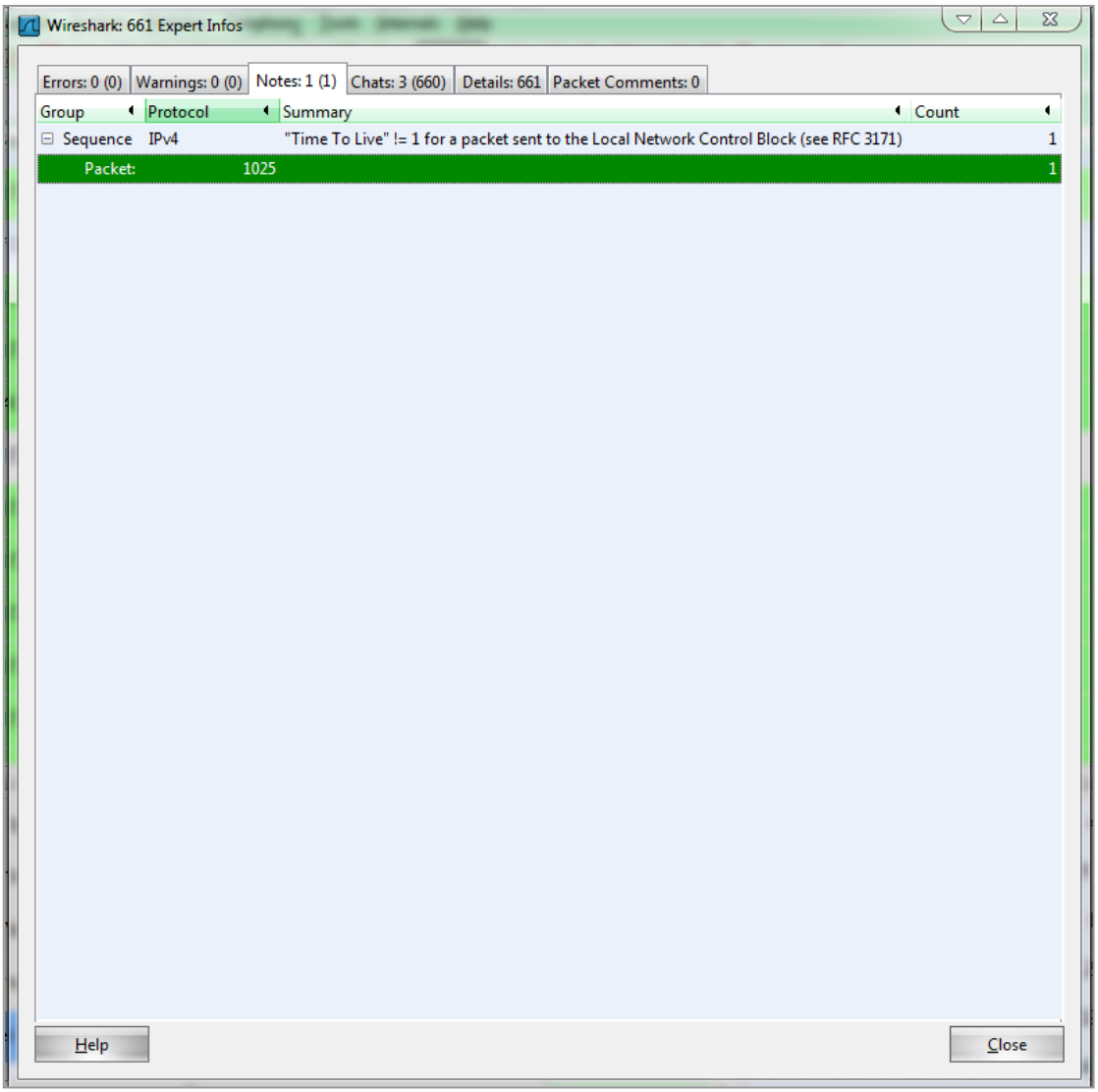


图 4-2-23

从上图 4-2-23 所示，我们可以查看数据包的一个大概统计信息。例如：Errors（错误）、Warning（告警）、Notes（通知）等信息；Chats（信息）以及 Details（详细）2 列列出了 TCP 数据包的会话流，我们可以看到 SYN、SYN ACK 和 GET 数据包的个数；详细信息一列列出了一个 TCP 会话流的大概情况。如图 4-2-24,4-2-25 所示：

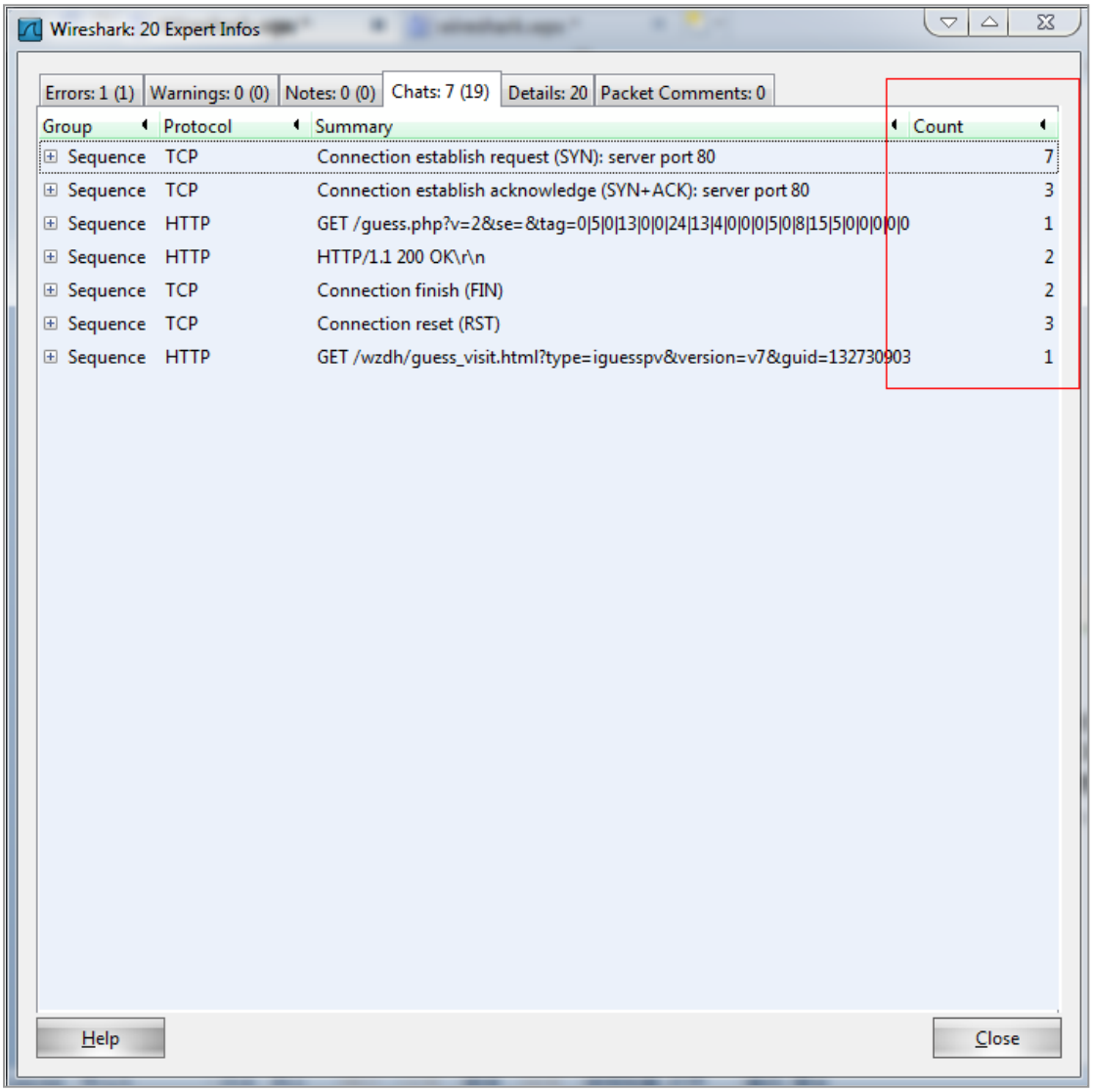

Wireshark: 20 Expert Infos
Errors: 1 (1)
Warnings: 0 (0)
Notes: 0 (0)
Chats: 7 (19)
Details: 20
Packet Comments: 0
Group
Protocol
Summary
Count
Sequence TCP Connection establish request (SYN): server port 80 7
Sequence TCP Connection establish acknowledge (SYN+ACK): server port 80 3
Sequence HTTP GET /guess.php?v=2&se=&tag=0|5|0|13|0|0|24|13|4|0|0|0|5|0|8|15|5|0|0|0|0|0 1
Sequence HTTP HTTP/1.1 200 OK\r\n 2
Sequence TCP Connection finish (FIN) 2
Sequence TCP Connection reset (RST) 3
Sequence HTTP GET /wzdh/guess_visit.html?type=iguesspv&version=v7&guid=132730903 1
Help
Close

图 4-2-24

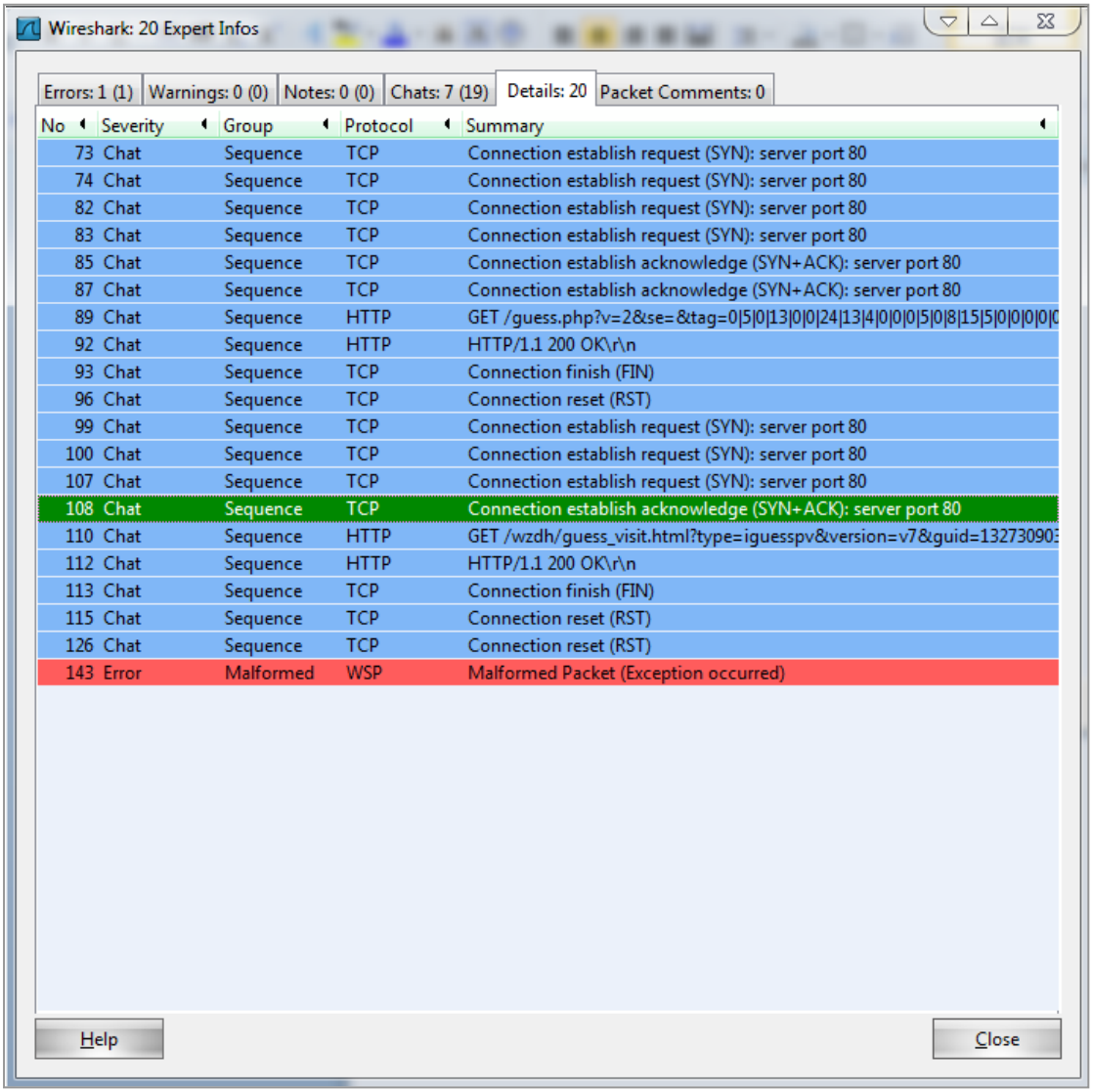


图 4-2-25

Packet comments（数据包描述）：由于我们没有添加任何数据包描述，所以这一列是空的。数据包描述的添加在上面 4.2.2、Edit 菜单栏我们有讲过了。您可以自己测试下。（添加之后需要保存否则重新加载后会消失）

4.2.7、Statistics 菜单栏

该栏主要是用于对数据包的一些统计操作，我们对该栏常用的选项进行一些基本介绍，该栏的功能之强大还望大家以后多多发现补充；如图示 4-2-26 所示：

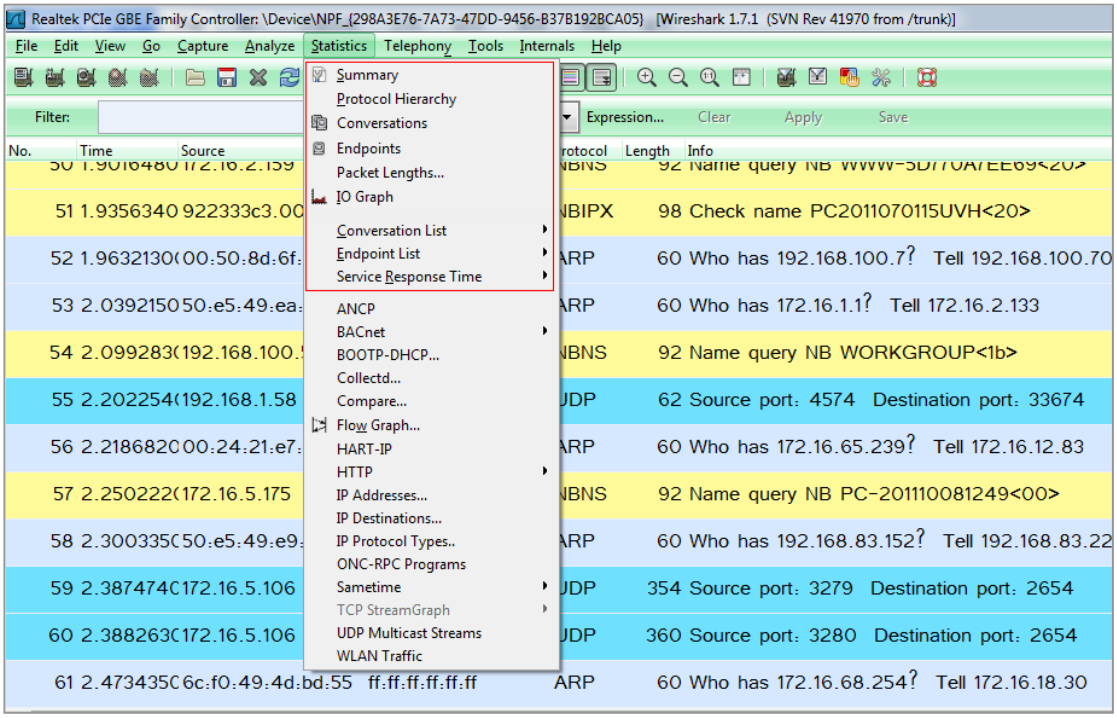


图 4-2-26

Summary（汇总）：点击 summary 弹出一个对话框，该对话框内统计了数据包文件以及数据包的一些概要信息。如图 4-2-27 所示：

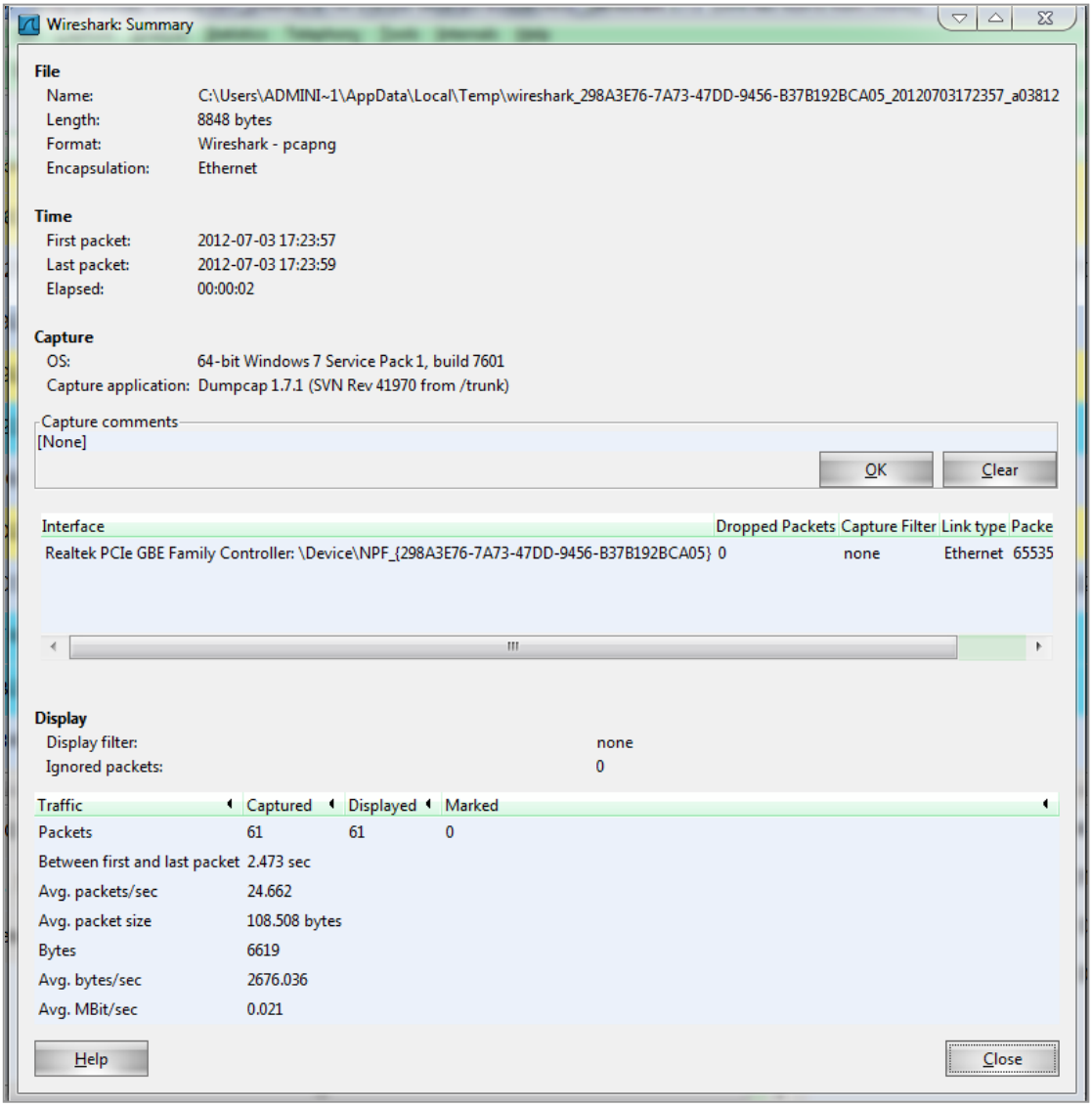


图 4-2-27

File（文件）：描述了数据包文件的一些信息：Name（名字）、Length（长度）以及 format（格式）等；

Time（时间）：数据包抓取时间；

Capture（抓取）：描述了本机的操作系统以及 wireshark 的版本号；

Display（显示）：通过该栏的信息我们可以知道数据包的接收频率、每秒钟接收多少数据包、数据包的平均大小、总字节数、平均每秒字节数以及平均每秒多少 M；

Protocol Hierarchy Statistics（协议分层汇总）：点击该选项弹出对话框，如图 4-2-28 示：

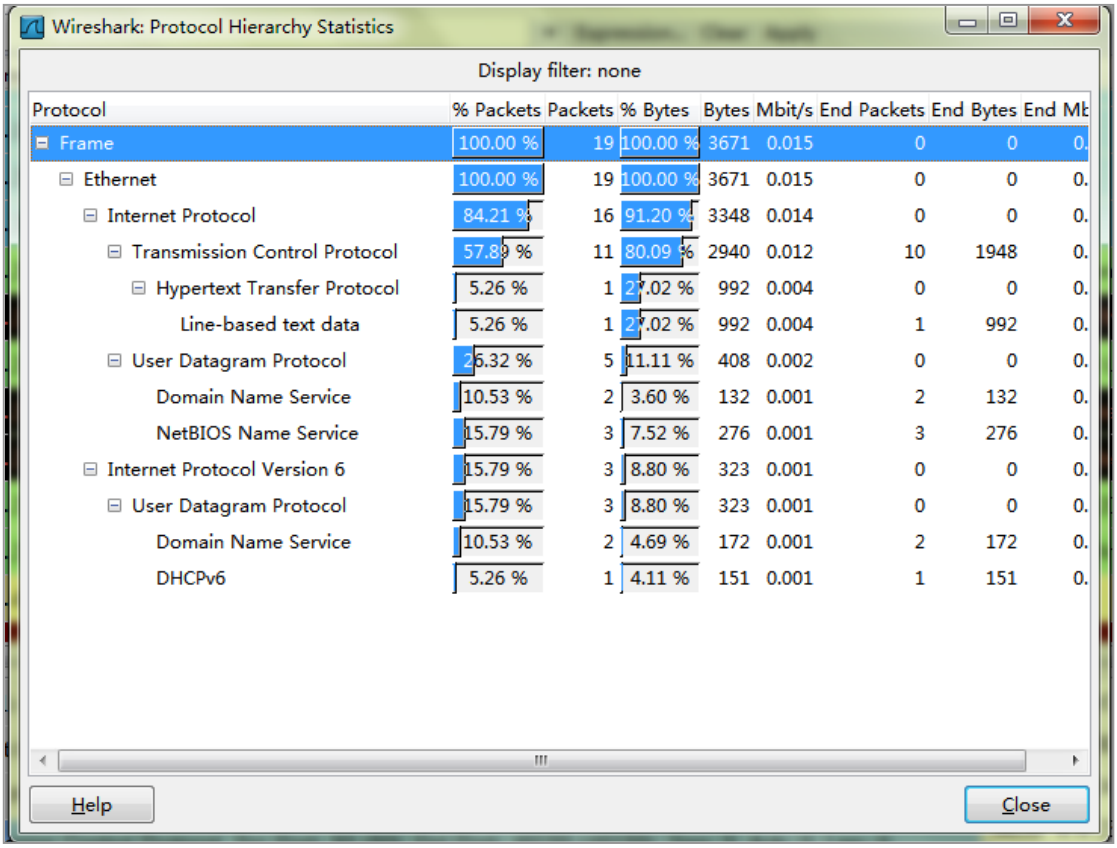
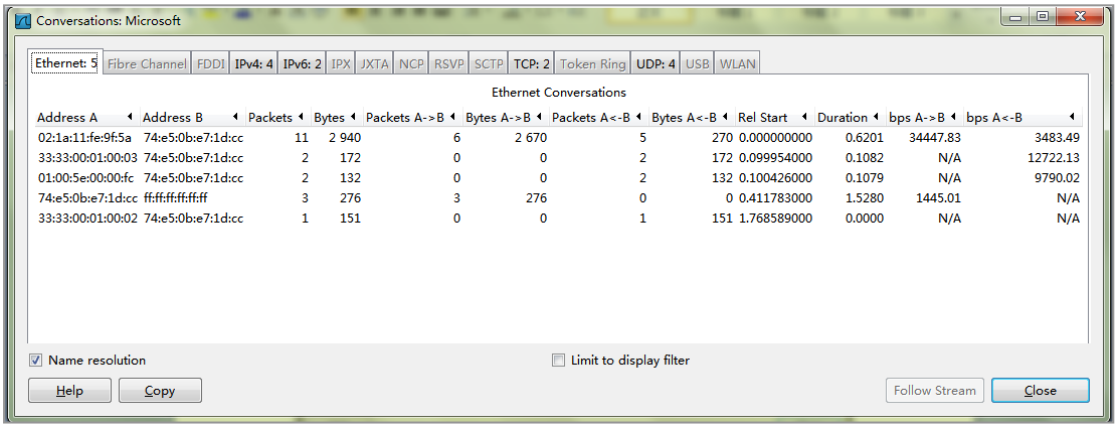


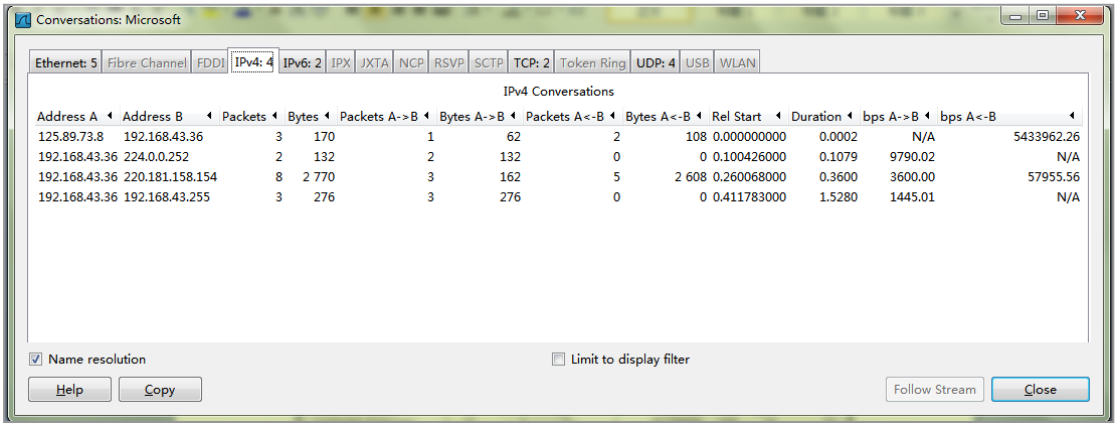
图 4-2-28

从 Protocol Hierarchy Statistics（协议分层汇总）对话框我们可以看出每层协议的数据包个数、字节数、兆等百分比信息；

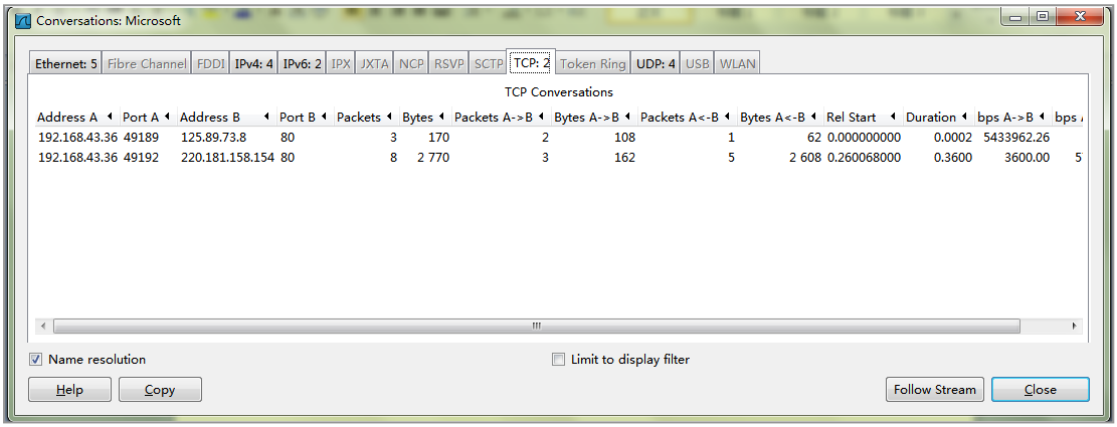
Conversation（会话）：点击该选项弹出对话框，从该对话从该对话框我们可以看出每层不同的所建立的会话个数；如图 4-2-29，4-2-30,4-2-31,4-2-32 所示：



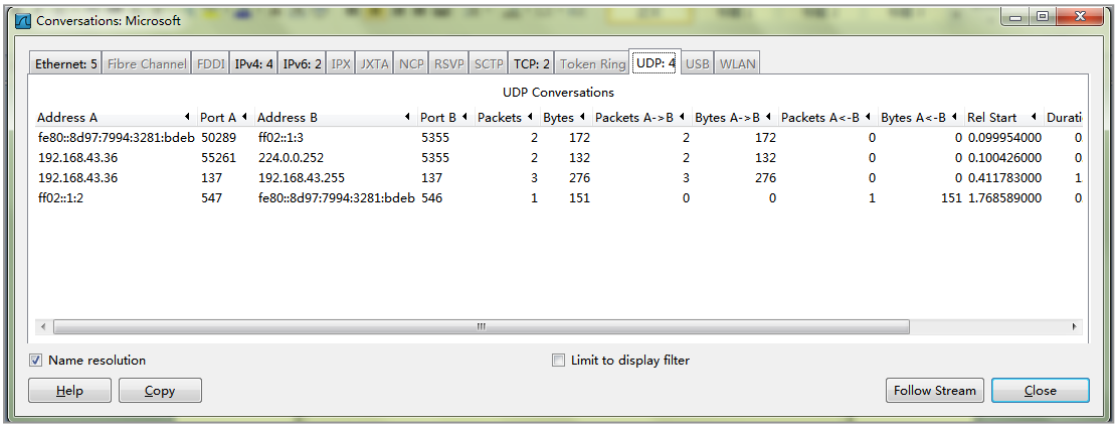
（Ethernet 层会话）图 4-2-29



（IP 层会话）图 4-2-30



（传输层）图 4-2-31



（传输层）图 4-2-32

End Point（结束点）：该选项栏与 conversation（会话）功能差不多，唯一差别在于会话显示源和目的地址，End Point（结束点）只显示每个会话的结束地址；如图 4-2-33 示（这里只选取了传输层的结束点）：

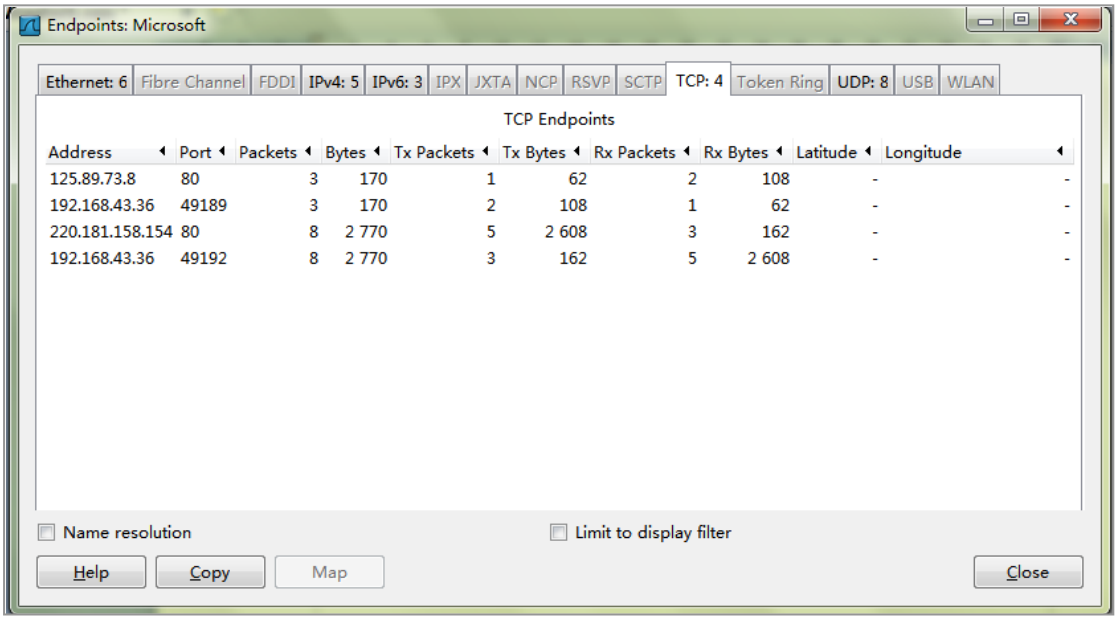


图 4-2-33

Packet Lengths（数据包长度）：点击选项弹出过滤对话框，在框内输入想要查找的特定数据包长度；如图 4-2-34 所示：

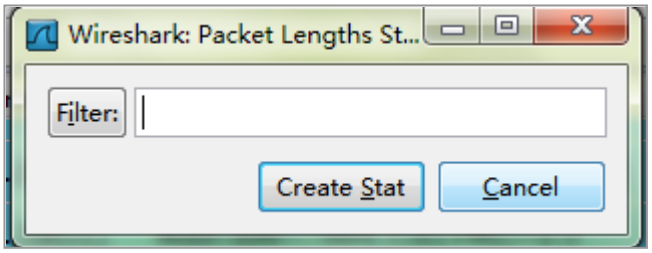


图 4-2-34

IO graphs（输入输出图表显示）：以图形化的显示方式显示出数据包单位时间内的个数、字节数或者位；如图 4-2-35 所示：

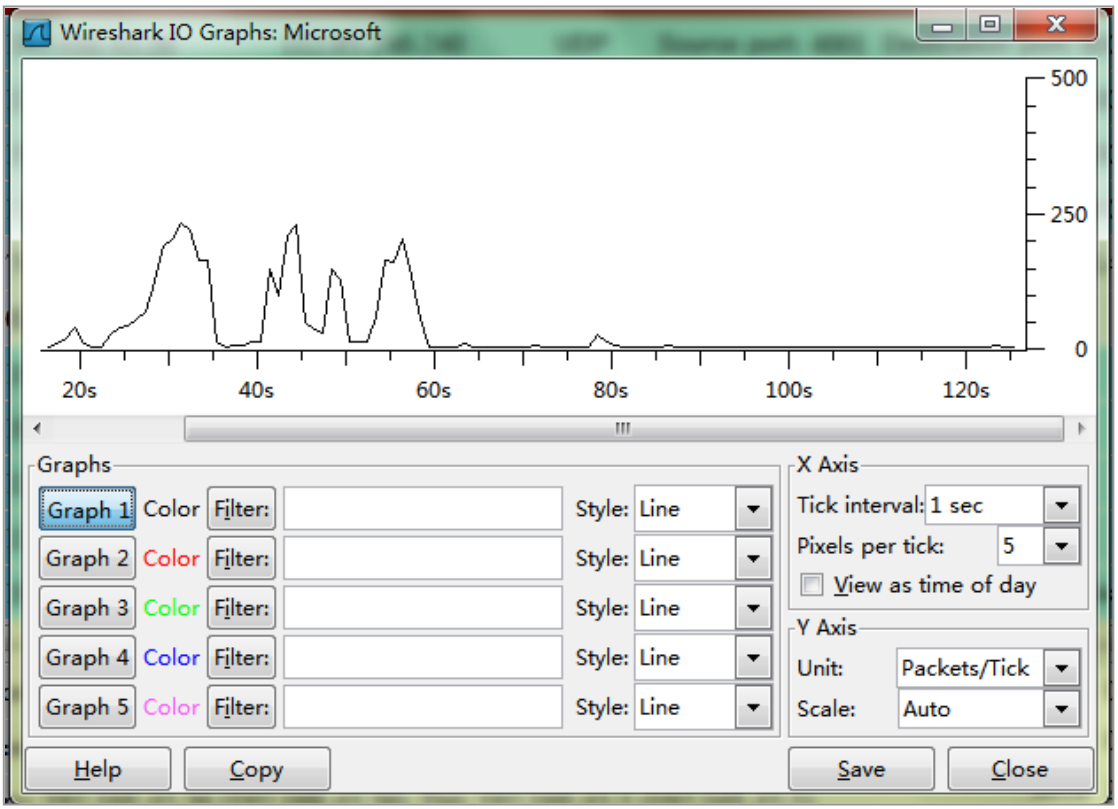
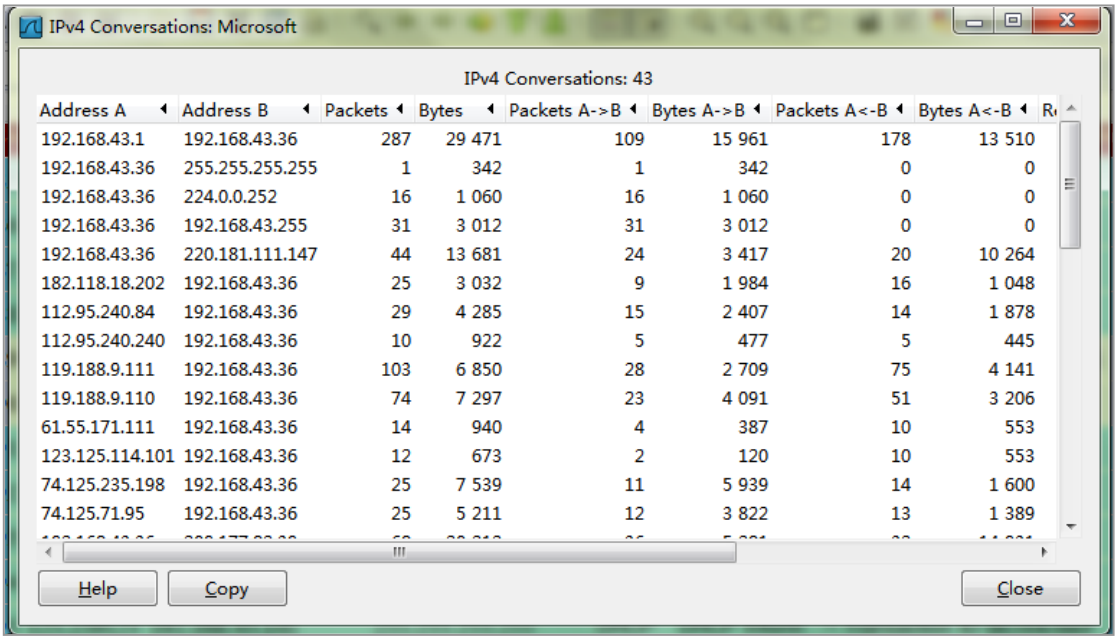


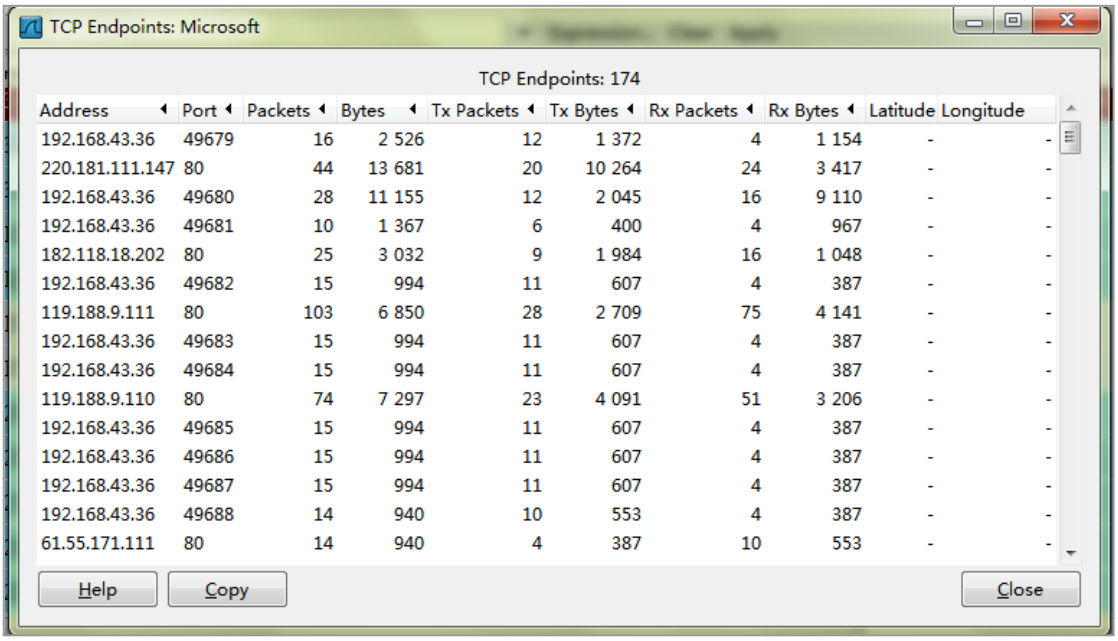
图 4-2-36

对于有兴趣的屌丝大家可以自己研究一下，在这里我们就不对该选项过多的介绍了。

Conversation List（会话列表）、End Point（结束列表）：这两个选项相同于上面介绍的 Conversation 以及 End Point，不同之处就是可以直接选择性的对不同的层会话进行查看；点击 Conversation 选项会弹出子菜单栏，在子菜单栏选择我们想要看的那层的会话信息即可；如图 4-2-37,4-2-38 所示：



（IP 层会话）图 4-2-37



（传输层结束点）图 4-2-38

Follow Graph（图形会话流）：以图形化的界面显示每个会话，如图 4-2-39，图 4-2-40 所示：

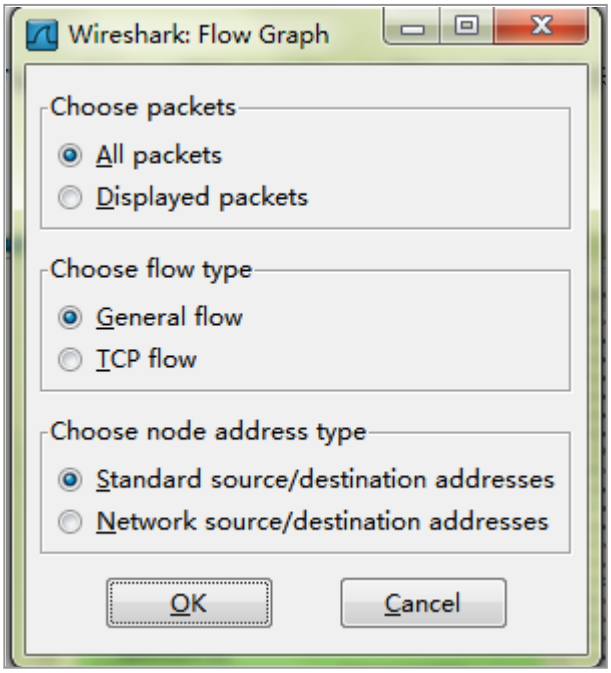


图 4-2-39

在上图 4-2-39 所示的 Choose packets 我们选择"All packets"（所有数据包）即对所有抓取到的数据包进行查看；Choose follow type（选择流类型）：我们选择 General flow（一般会话流）；

Choose node address type（节点地址类型）：选择标准的源目的地址；

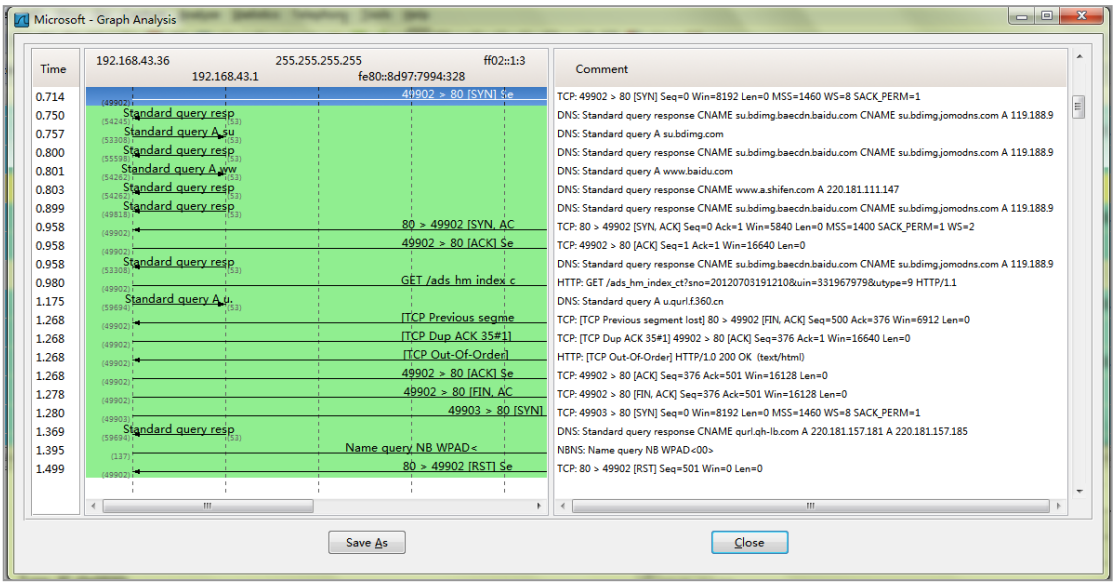


图 4-2-40

如上图所示，左边图形界面为每个 IP 地址所发送的数据包，右边是对该数据包的描述信息；看图可发现在 Time：0.714、0.958 以及 0.980 秒时所发送的数据包是一个标准的 TCP 三次握手的建立以及网页的请求动作；再下面还有该会话流的 TCP 分段以及分段的确认 ACK 等数据包；

UDP Multicast Stream（UDP 组播）：显示出所捕捉到的数据包的组播会话流；如图 4-2-41 所示：

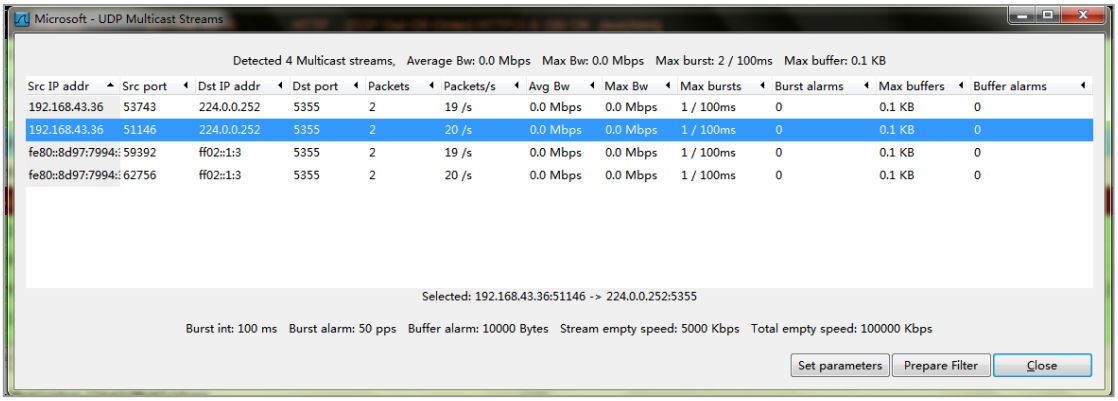


图 4-2-41

4.2.8、Telephony 菜单栏

由于 Telephony(电信)　一栏我们基本不会使用到,并且我们也没有电信的数据包来进行演示操作 ,所以在此呢我们就一跳而过对该栏不作任何介绍了.如果大家对该选项栏有兴趣也可以自己再研究一下.

4.2.9、Tools 菜单栏

该栏有一个 Fire Wall ACL Rules（防火墙访问控制列表规则），使用该功能为多种不同的防火墙创建命令行 ACL 规则(访问控制列表),支持 Cisco IOS, Linux Netfilter (iptables), OpenBSD pf and Windows Firewall (via netsh). Rules for MAC addresses, IPv4 addresses, TCP and UDP ports, 以及 IPv4+混合端口。以上假定所有规则用于外部接口；如图 4-2-42,4-2-43 所示：

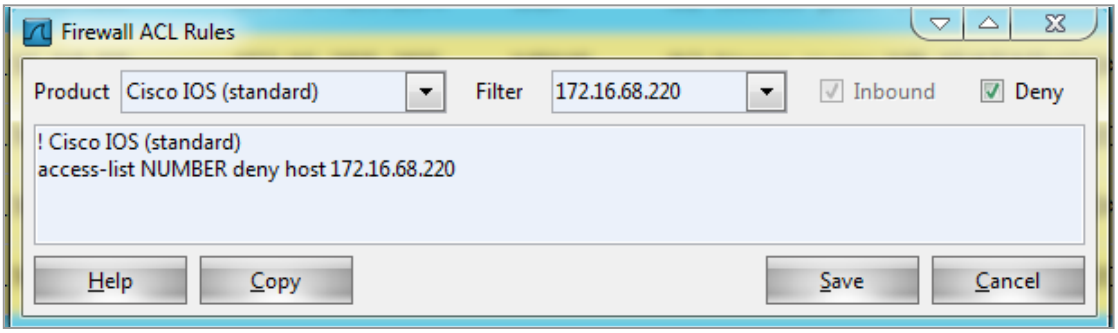


图 4-2-42

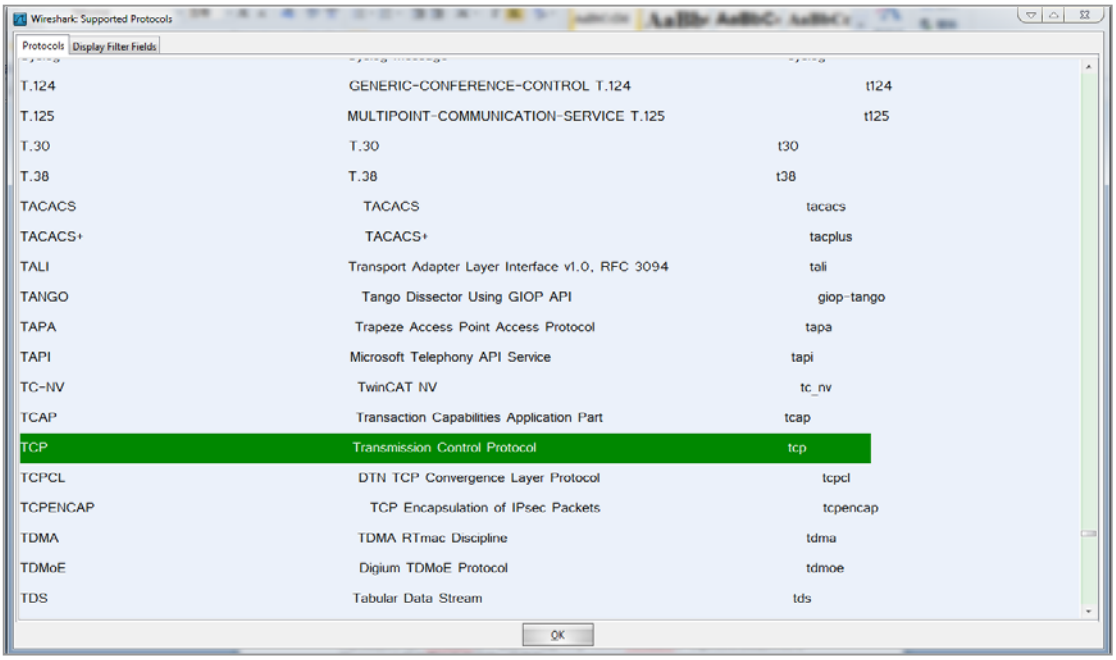
```
! Cisco IOS (standard)
access-list NUMBER deny host 172.16.68.220
```

图 4-2-44

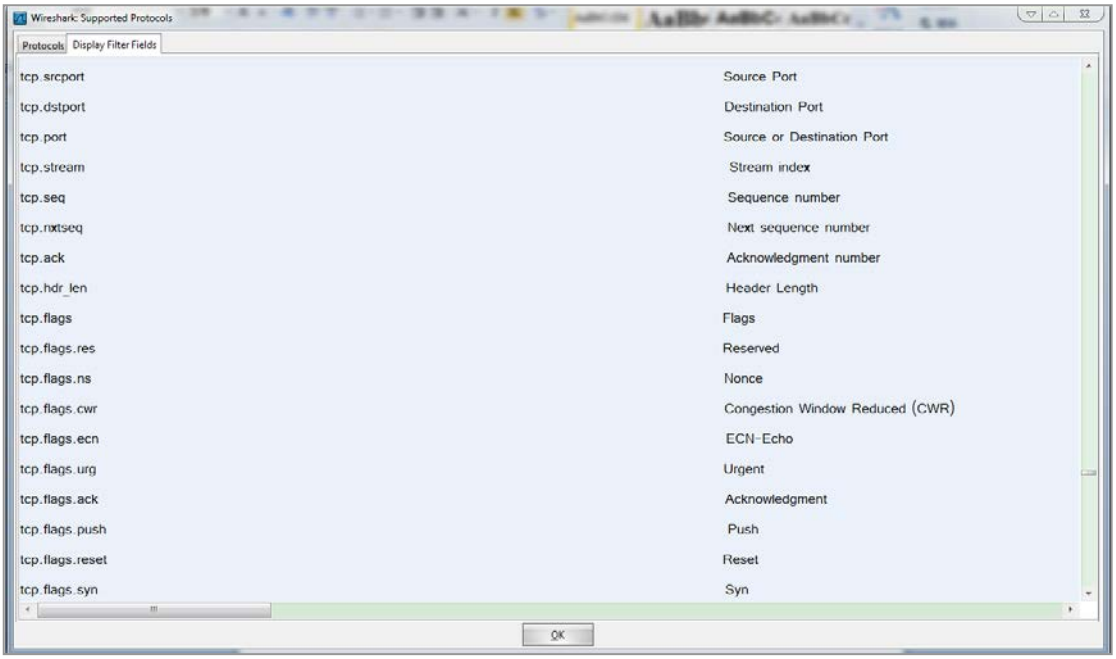
使用方法：Product（作品（翻译成这样不知道对不对）），该栏选择下拉菜单可看到由以上所讲的多种规则类型。选择想要使用的规则点击图 4-2-42“Copy”即可复制该规则。在我们的防火墙或者三层设备上面就可以直接使用了；

4.3.0、Internals（内部）菜单栏

这里面记录了一些解析列表以、wireshark 所支持的网络协议列表和支持显示的数据包字段；大家也可以有兴趣就查查。使用方法 :Internals——Dissector tables(解析列表)/Supported Protocol（支持的协议）；如图 4-3-1，4-3-2 所示所支持的协议类型以及 TCP 协议数据包 wireshark 所支持的显示字段（由于版本高低，可能所支持的协议类型以及显示字段可能会有差别）：



（wireshark 所持支持的协议类型）图 4-3-1



（wireshark 所支持各协议数据包字段的显示）图 4-3-2

4.3.1、Help 菜单栏

最后的一个是 help(帮助)一栏,从该栏我们可以访问到 wireshark 的官方网站,并且能获得一些 wireshark 的使用帮助文档,也可以在官方网站上面下载到各个版本的 wireshark.以及查看我们的当前使用的 wireshark 版本信息;方法是 Help--About wireshark 即可看到我们当前所使用的 wireshark 版本号以及其他的一些信息;如图 4-3-3：

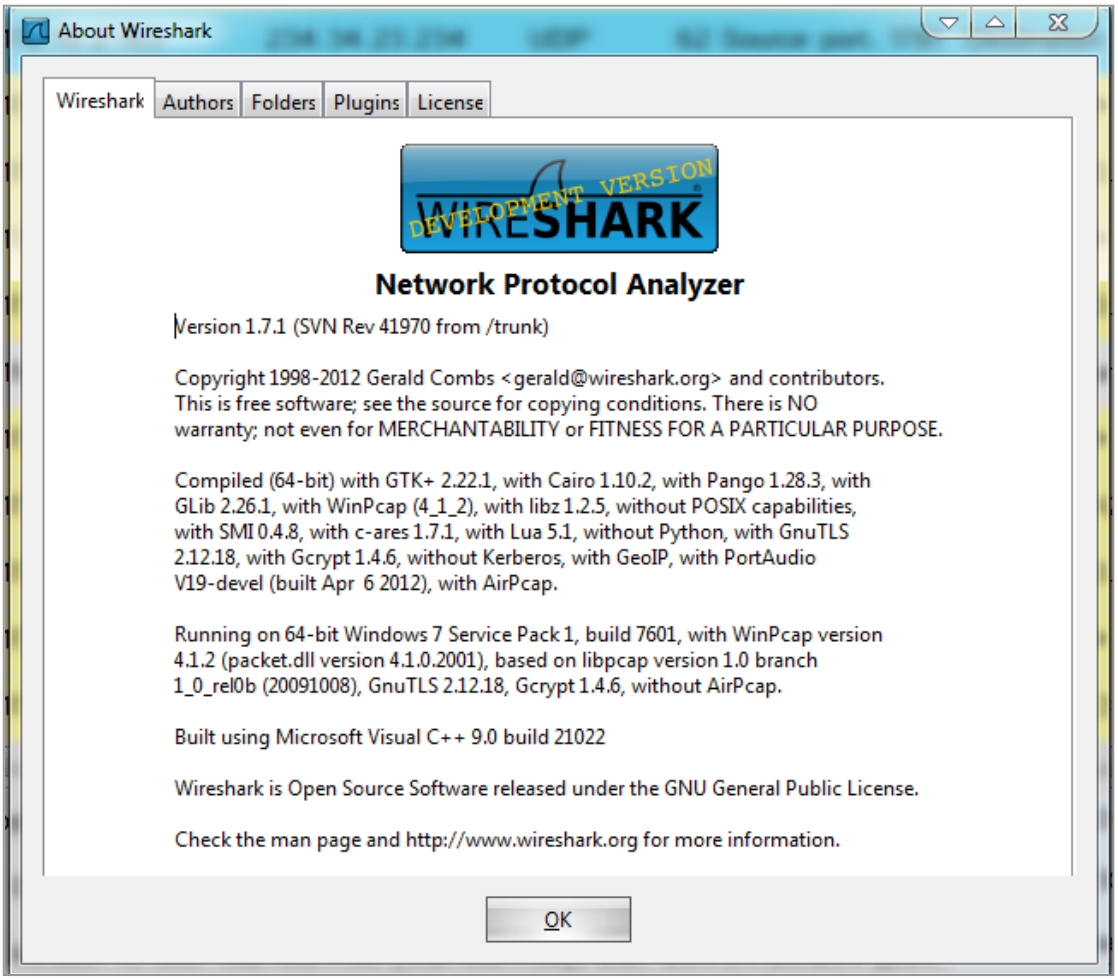
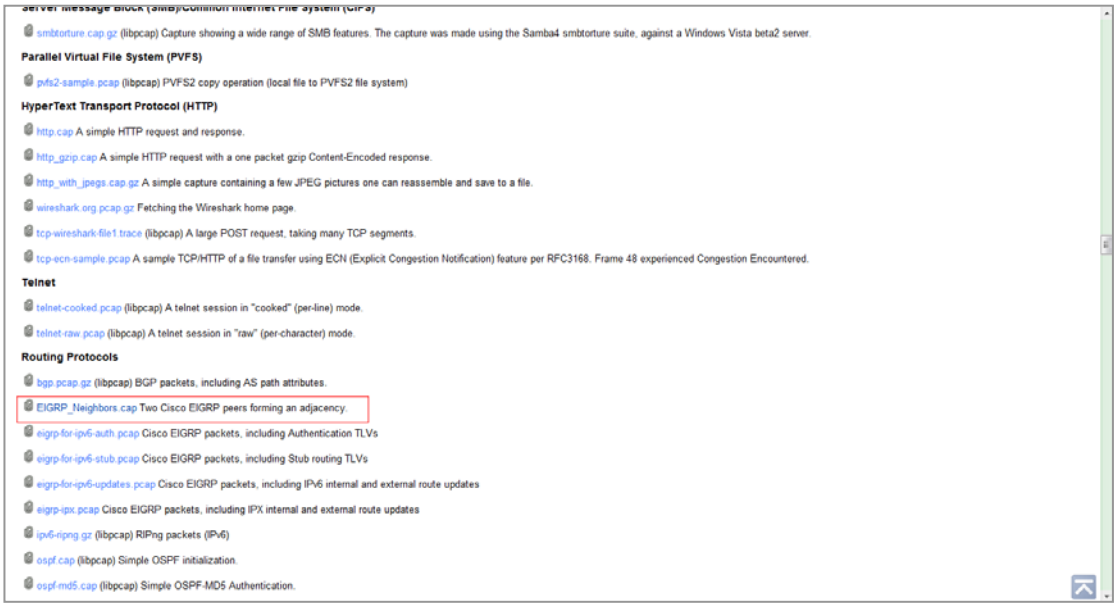


图 4-3-3

另外，Help 工具栏还有一个比较实用的功能就是能够通过官方的站来下载一些数据包，例如我们下载一个 Cisco 路由器 ERGIP 协议的邻居发现过程数据包；如下图所示：



（下载各种数据包：ERGIP 数据包）

（执行打开后，我们看到 ERGIP 的邻居建立过程）

综上，wireshark 的安装方式以及常用到的菜单选项我们也差不多介绍完了。除了上面所说的那些选项外，我们还有各个列表的右键菜单栏，所弹出的选项我们都在上面基本介绍完了。大家可以根据上面的介绍来比对使用右键菜单栏。我们就不在这里多做赘述了；

下面我们开始讨论 wireshark 的显示/抓包过滤器，这两个功能是 wireshark 最强大功能之一；我们可以在抓取数据包时或者是对已经抓取的数据包进行选择性抓取或者查看。下面我们开一个新的章节来介绍这两个功能；

◆ wireshark 显示/抓包过滤器

5.1、显示过滤器概括

我们可以使用菜单栏——Edit——Find Packet、菜单栏——Analyze——Display Filter 或者主菜单栏——首选项工具栏——  使用 wireshark 自带的一些过滤正则表达式来对数据包进行过滤；也可以在过滤规则栏直接输入正则表达式直接应用过滤器来启动显示过滤器。如图 5-1-1 所示：

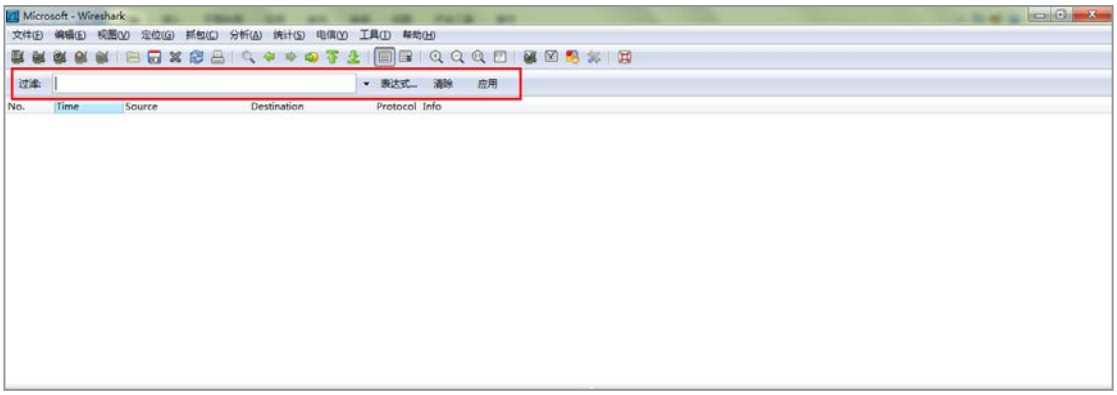


图 5-1-1

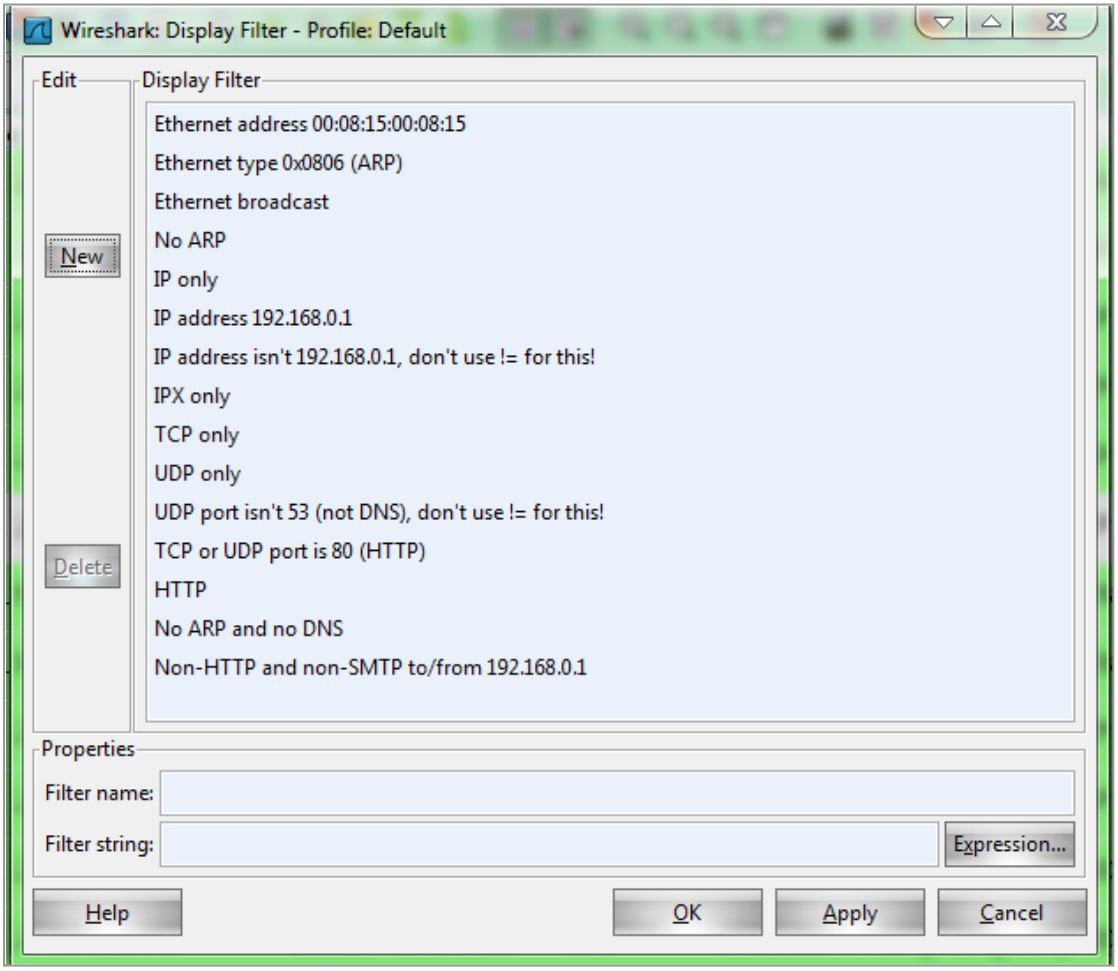


图 5-1-2

显示过滤器启动后的窗口界面如上图 5-1-2 所示，我们可以根据源/目的 IP 地址、以太网源/目的 IP 地址、TCP only、UDP only 和 IP only 等过滤规则来进行数据包简单的显示过滤；也可以使用更为复杂的正则表达式来进行过滤；下面我们介绍一些常用正则表达式的语法规范；

5.1.1、wireshark 规则编辑

假如 wireshark 自带的抓包过滤规则不能满足您的应用需要，那么我们可以手动来输入正则表达式来进行数据包过滤；点击如上图 5-1-2 所示的“New”按钮后 wireshark 会新建一个初始规则，我们对该规则进行编辑。Wireshark 常用的显示过滤器正则表达式个人整理了一下大概如下所示：

5.1.2、语法以及连接符

Wireshark 所用到的各种表示符号如下图所示：

语法：	Protocol	.	String 1	.	String 2	Comparison operator	Value	Logical Operations	Other expression
例子：	ftp		passive		ip	==	10.2.3.4	xor	icmp.type

英文写法：	C语言写法：	含义：
eq	==	等于
ne	!=	不等于
gt	>	大于
lt	<	小于
ge	>=	大于等于
le	<=	小于等于

英文写法：	C语言写法：	含义：
and	&&	逻辑与
or	||	逻辑或
xor	^^	逻辑异或
not	!	逻辑非

5.1.3、新建规则

过滤 IP：

指定来源 IP 或者目标 IP 等于某个 IP：

例：**ip.src eq 192.168.1.107 or ip.dst eq 202.102.192.68**

或者模糊书写：**ip.addr eq 202.102.192.68** 将能显示出源和目的等于该 IP 的会话；

过滤端口：

例：**tcp.port eq 80** 或者 **tcp.port==80** 不区分源或者目的端口只要数据包包含有 80 端口都将被过滤；

tcp.port eq 80 or udp.port eq 80 多条件匹配，匹配 TCP 或者 UDP 端口为 80 的数据包，不区分源或者目的端口；

tcp.dst port==80 显示目的端口为 80 的数据包；

tcp.src port==80 显示源端口为 80 的数据包；

tcp.port>=1and tcp.port<=80 过滤显示 TCP 端口大于等于 1 小于等于 80 范围内的数据包；

过滤协议：

例：在 Filter（过滤）框内直接输入 **tcp、udp、arp、icmp、http、smtp、ftp、dns、msnms、ip、ssl、oicq、bootp、** 等等；可直接过滤协议。假如想要排除哪个协议不显示我们可以直接在前面加上 ！来进行排除。例如：!arp 就对 ARP 协议的数据包不做显示；

过滤 MAC：

eth.dst==A0:00:00:04:C5:84 过滤目标 mac 为 A0:00:00:04:C5:84 的数据包；

eth.src eq A0:00:00:04:C5:84 过滤源 mac 为 A0:00:00:04:C5:84 的数据包；

eth.addr eq A0:00:00:04:C5:84 无论源或者目的地址为 A0:00:00:04:C5:84 的数据包都会进行过滤；

包长度过滤：

例：**udp.length==26** 这个长度是指 udp 本身固定长度 8 加上 udp 携带的数据字段的长度之和；

tcp.len>=7 指的是 ip 数据包(tcp 下面那块数据),不包括 tcp 协议本身的长度；

ip.len==94 除了以太网头固定长度 14,其它都算是 ip.len,即从 ip 本身到最后；

frame.len==119 整个数据包长度,从 eth 开始到最后；

eth--->ip or arp--->tcp or udp--->data 数据包长度层次区分；

http 模式过滤：

例：**http.request.method=="GET"** 过滤 HTTP 协议的 GET 请求数据包；

http.request.method=="POST" 过滤 HTTP 协议的 POST 请求数据包；

http.request.uri=="/img/logo-edu.gif" 过滤 HTTP 协议所请求的特定 URL 字段的数据包；

http contains"GET" 过滤包含 GET 请求字段的 HTTP 协议的数据包；

http contains"HTTP/1." 过滤包含 HTTP.1.1 字段的 HTTP 协议数据包；

GET 包:

http.request.method=="GET"&& http contains"Host:" 过滤 HTTP 请求类型为 GET 以及包含 Host: 字段的数据包；

http.request.method=="GET"&& http contains"User-Agent:" 过滤 HTTP 请求类型为 GET 并且携带 User-Agent: 字段的数据包；

POST 包：

http.request.method=="POST"&& http contains"Host:" 过滤 HTTP 请求类型为 POST 并且携带 Host: 字段的数据包；

http.request.method=="POST"&& http contains"User-Agent:" 过滤 HTTP 请求类型为 POST 并且携带 User-Agent: 字段的数据包；

响应包：

http contains"HTTP/1.1 200OK"&& http contains"Content-Type:" 过滤返回值为 200OK 并且携带 Content-Type: 字段的数据包；

TCP 参数过滤：

tcp.flags 显示包含 TCP 标志的封包；

tcp.flags.syn==0x02 显示包含 TCPSYN 标志的封包；

tcp.window_size==0&&tcp.flags.reset!=1 过滤 TCP 滑动窗口为 0 并且 tcp 标志位不等于 1 的数据包；

过滤内容：

语法：**tcp[offset,n]**（以下语法中的协议可变）

例：**tcp[0:20]**或者 **tcp[20]** 表示排除 TCP 头部长度后（即除去 20Bye 后），取 1 个携带的数据字段的字符；

tcp[20:]表示排除 TCP 头部字段后，取数据字段 1 个以上字符；

tcp[20:8]表示排除 TCP 头部长度后，取 8 个携带的数据字段的字符；

语法：**udp[offset:n]**

例：**udp[8:3]==81:60:03** 排除 UDP 头部（8 个 bytes）后,再取 3 个数据字段的数值，判断是否与==后面的数据相同；

语法：**ip[offset:n]**

例：**ip[20:4]==c0:e9:00:50** 除去 IP 头部 20 字节数后 IP 字段所携带的数据的前四个字节是否匹配==后面的数值；

语法：**eth.addr[offset:n]**（注意：该表达式用来匹配 MAC 地址，与前几个小有差异）

例：**eth.addr[0:3]==02:1a:11** 匹配 MAC 地址的前三个字段为==后面的值的数据包；

例：判断 upd 下面那块数据包前三个是否等于 0x20 0x21 0x22（0x20 0x21 0x22 因为是 16 进制所以在这里书写格式为 0xnn）

我们都知道 udp 固定长度为 8 所以排除 8 字节的首部长度后就是 UDP 协议的 Data 字段的值了；

格式：**udp[8:3]==20:21:22**

例：判断排除 tcp 头部长度后有哪些数据包前三个数值是否等于 0x20 0x21 0x22；

tcp 一般情况下，长度为 20,但也有不是 20 的时候。比如增加了 Option 字段后；

格式：**tcp[20:3]==20:21:22**

matches(匹配)和 contains(包含某字符串)：

matches（匹配）：语法：**tcp[8:n] matches "\\xnn\\xnn\\xnn"**

例：udp[8:2] matches "\\x00\\x35" 精确匹配除去 UDP 头部字段（即 UDP 协议携带的数据字段）载荷中十六进制数值为 x00 x35 的数据包；

例：**udp[0:2] matches "\\x00\\x35" and ip.src==192.168.43.1** 精确匹配 udp 协议载荷中前两个十六进制数值为 x00 x35 并且源地址是 192.168.43.1 的数据包；

tcp[20:] matches "\\x75\\x05\\x62\\x64\\x69\\x6d\\x67\\x06\\x62\\x61\\x65" 匹配包含（是包含，和上面的精确匹配是有差别的）除去 tcp 协议头部字段长度后，载荷中为以上数值的数据包；

Contains（包含）：语法：**udp contains nn;nn;nn;nn**

例：**udp contains 66:0c:81:80:00:01** 过滤包含 66:0c:81:80:00:01 十六进制数值的 udp 协议的数据包；

ip.src==192.168.1.107 and tcp contains "GET" 匹源地址为 192.168.1.107 并且 TCP 协议携带 GET 请求的数据包；

udp contains 7c:7c:7d:7d 匹配载荷中含有 0x7c7c7d7d 的 UDP 数据包；

DHCP：

以寻找伪造 DHCP 服务器为例，介绍 Wireshark 的用法。在显示过滤器中加入过滤规则，

显示所有非来自 DHCP 服务器并且 bootp.type==0x02（Offer/Ack）的信息：

bootp.type==0x02and not ip.src==192.168.1.1

综上我们 wireshark 的显示正则表达式的常用的几个输入语法我们基本介绍完了，下面我们在 wireshark 的过滤规则栏来进行几个示例；

例 1：假如我们想过滤 TCP 目的端口是 80 端口的数据包，我们可以再显示过滤栏输入：**tcp.dstport==80** 点击应用后 wireshark 将会把所有目的端口为 80 的数据包全部展示出来。如下图 5-1-3 所示：

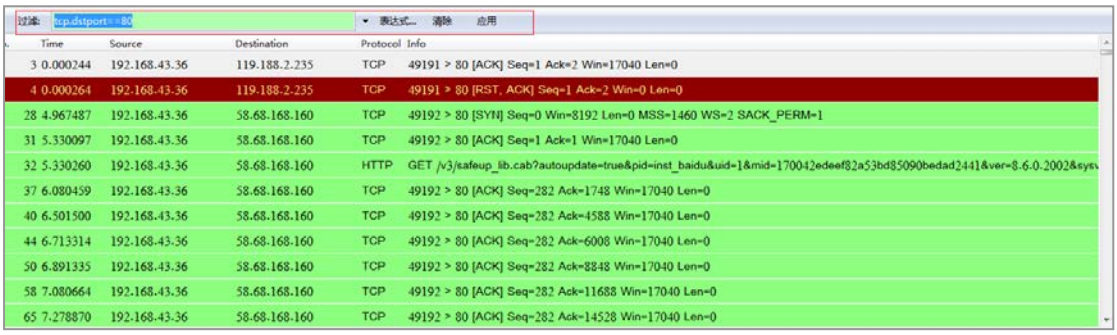


图 5-1-3

例 2：如果想要过滤特定 IP 地址的数据包，我们可以再 wireshark 输入：**ip.addr eq 192.168.43.36** ；点击应用后同样 wireshark 也会把所有源目的为 192.168.43.36 的数据包给过滤出来。如图 5-1-4 所示：

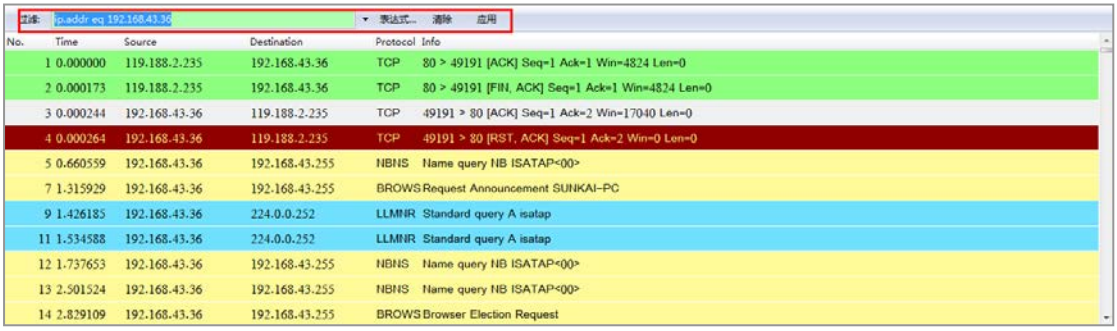


图 5-1-4

那假如我们想要只显示源地址为 192.168.43.26 的数据包不把目的地址是 192.168.43.36 的数据包显示出来。要实现这样的显示我们只需告诉 wireshark，我只想要源地址为 192.168.43.36 的数据包，不要目的地址为 192.168.43.36 的数据包。那么我们可以输入：**ip.src==192.168.43.36** ；即可实现。如下图 5-1-5 所示：

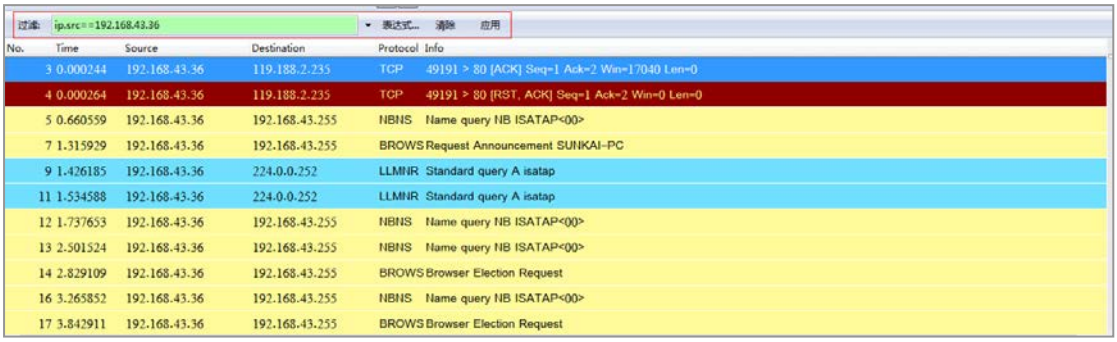


图 5-1-5

例 3：如果我们想要过滤 HTTP 协议的请求数据包，我们可以再 wireshark 过滤栏输入：**http.request** ；这样 wireshark 将会把所有的 http 协议的请求包 GET 以及 POST 等方式的请求数据包全部列出来。如图 5-1-6 所示：

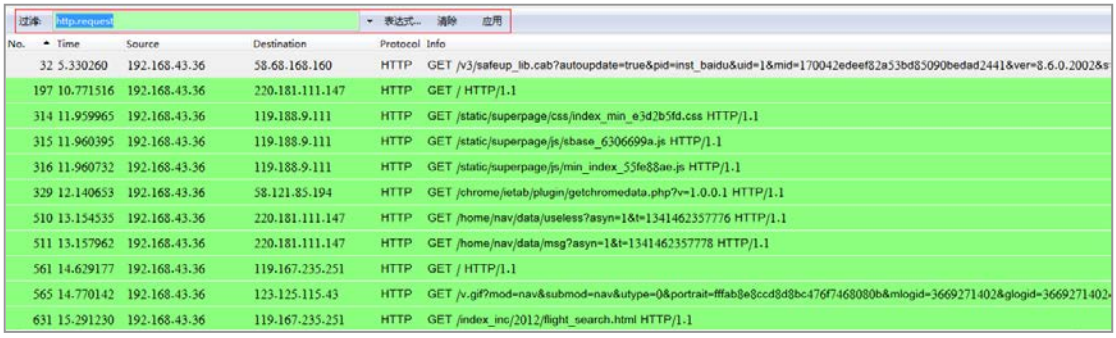


图 5-1-6

如果我们想要知道 HTTP 协议的请求方式的话我们可以输入：**http.request.method** ；应用后 wireshark 将把所有的 HTTP 请求方式列出来。如图 5-1-7 所示(由于我们所捕捉的数据包没有 POST 之类的请求方式，所以在图片上看不到 POST 请求）：

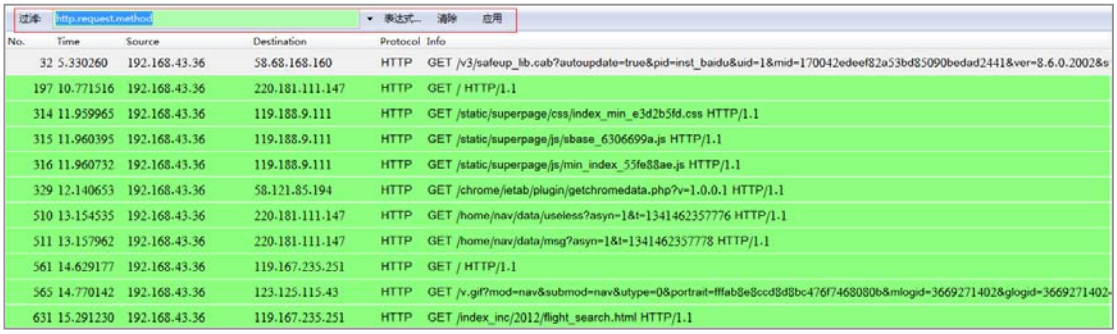


图 5-1-7

例 4：如果想要查看 HTTP 协议的 GET 请求的 host 那么我们可以使用：**http.host==www.baidu.com** ；来对特定的请求主机进行过滤。如下图 5-1-8 所示：

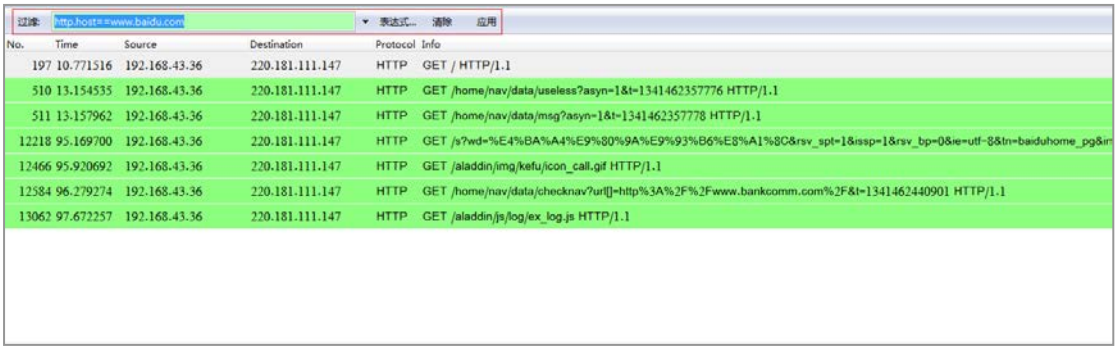


图 5-1-8

例 5：假如我们想要过滤出 TCP 标志位为 SYN 置一的 TCP 数据包，那么使用：**tcp.flags==0x02** ；将过滤出所有 TCP 标志位 SYN 的数据包。如下图 5-1-9 所示：

图 5-1-9

上图所示出的 0x02 为 TCP 头部 SYN 为是 1 时候的十六进制表示法，假如我们想要过滤其他标志位，如：ACK、SYN ACK 以及 PSH ACK 时我们只要在数据包详细信息列表查看一下 TCP 的各种标志位的十六进制表示方法。如下图 5-1-10 所示的 ACK 位被置 1 时的十六进制表示法为 0x01，那么我们只要把上面语句中的 0x02 换成 0x01 即可实现对 ACK 数据包的过滤；

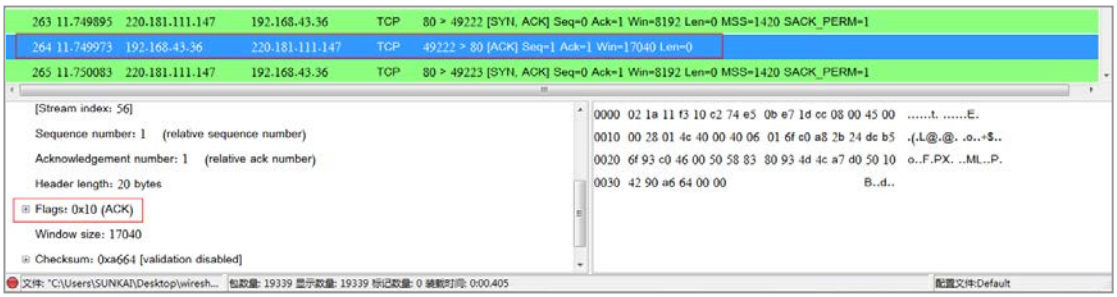


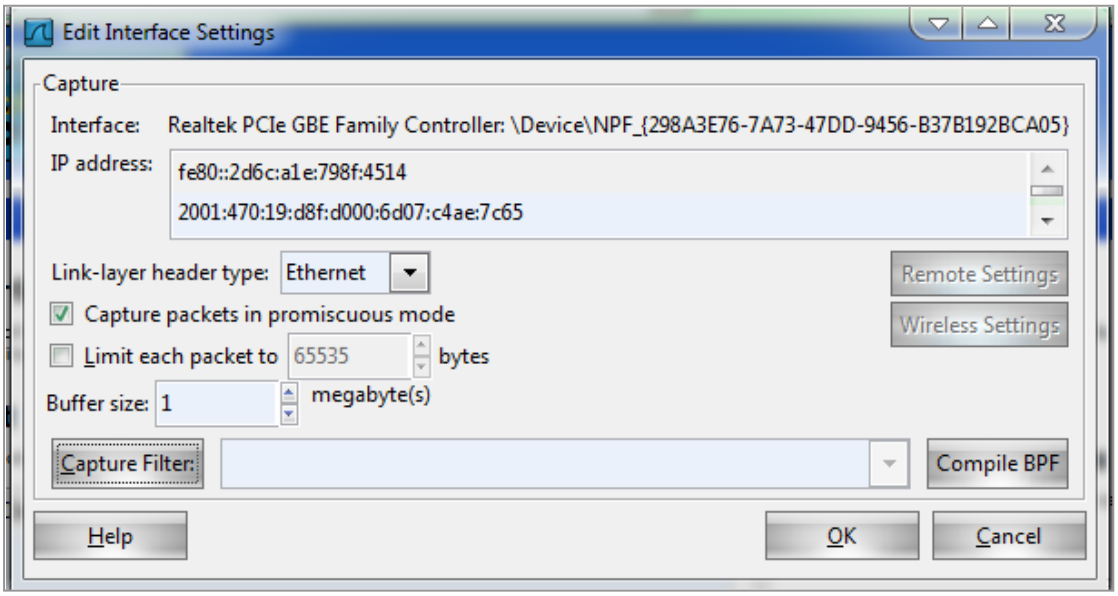
图 5-1-10

例子在这里我们就举这么多，如果大家比较有兴趣也可以自己参照我们上面在

5.1.1、wireshark 规则编辑 小结所介绍的规则编辑规则自己来进行尝试一下；显示过滤器的基本的常用内容在这我们也差不多介绍完了，下面我们进行抓包过滤器的介绍。

5.2、抓包过滤器概括

显示过滤器是对已经抓取的数据包进行过滤，而抓包过滤器是抓包过程中有选择的抓取数据包。即再抓包时候就实现了过滤；我们可以通过： Capture--Capture Filter，主菜单栏--  ，或者双击 wireshark 启动时的欢迎界面的网卡接口来进行打开；如下图 5-2-1，5-2-2 所示：



（点击 Capture Filter 来启动抓包过滤器）图 5-2-1

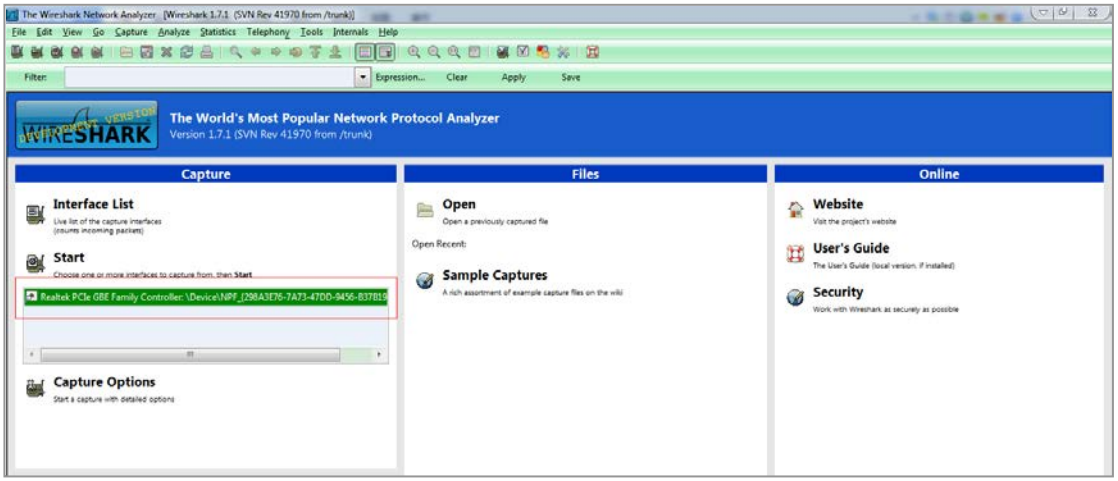


图 5-2-2

抓包过滤器启动后的弹出界面如下图 5-2-3 所示，我们可以像显示过滤器那样来进行手工的规则输入，也可以使用像显示过滤器那样自带的过滤规则来使用。抓包过滤器和显示过滤器的自带的规则是一样的，所以我们对于自带的规则就不进行详细介绍了。我们着重来说下抓包过滤器的手工规则输入语法；

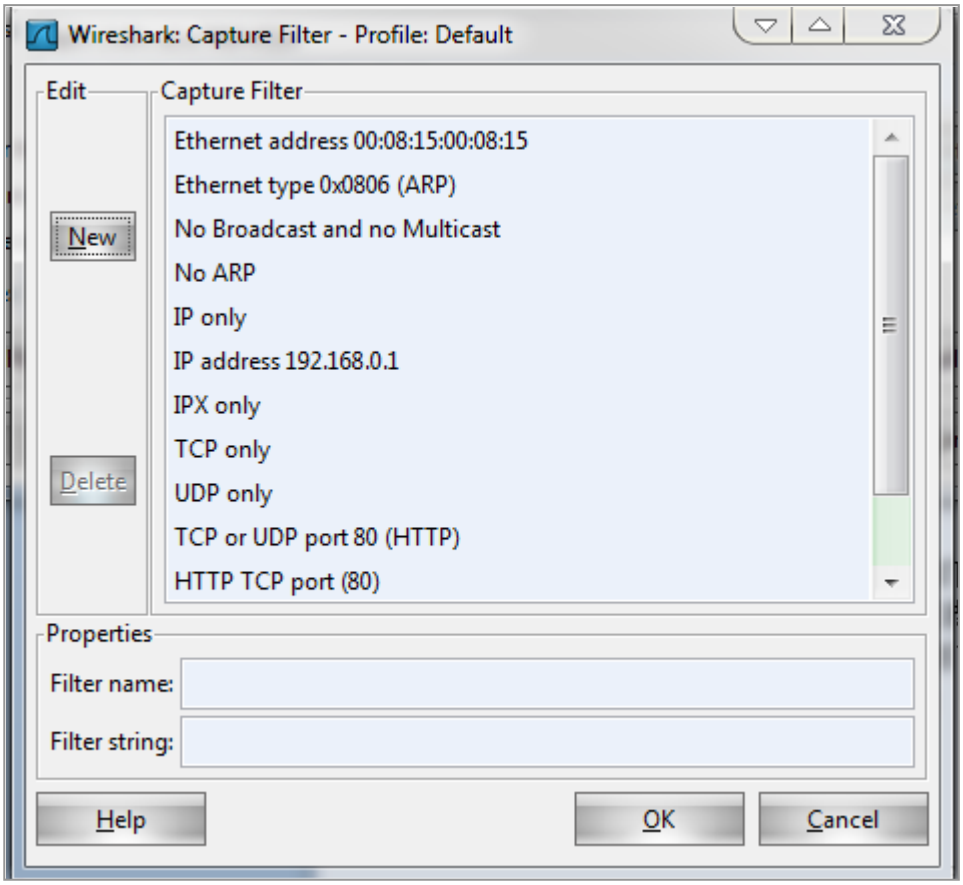


图 5-2-3

5.2.1、wireshark 规则编辑

假如 wireshark 自带的抓包过滤规则不能满足您的应用需要，那么我们可以手动来输入正则表达式来进行数据包过滤；点击如上图 5-1-2 所示的“New”按钮后 wireshark 会新建一个初始规则，我们对该规则进行编辑。Wireshark 常用的显示过滤器正则表达式个人整理了一下大概如下所示：

5.2.2、语法以及连接符

语法：	Protocol	Direction	Host(s)	Value	Logical Operations	Other expression
例子：	tcp	dst	10.1.1.1	80	and	tcp dst 10.2.2.2 3128

英文写法：	C语言写法：	含义：
eq	==	等于
ne	!=	不等于
gt	>	大于
lt	<	小于
ge	>=	大于等于
le	<=	小于等于

英文写法：	C语言写法：	含义：
and	&&	逻辑与
or	||	逻辑或
xor	^^	逻辑异或
not	!	逻辑非

5.2.3、新建规则

过滤 IP：

指定来源 IP 或者目标 IP 等于某个 IP：

例：**ip.src 192.168.1.107 or ip.dst 202.102.192.68**

或者模糊书写：**ip.host 202.102.192.68** 将能捕捉到源和目的等于该 IP 的会话；

src host 172.16.2.180 and not dst net 202.102.192.0/24 捕捉源地址为 172.16.2.180，但目的地址不是 202.102.192.0 网络的数据包；

过滤端口：

例：**tcp port 80** 不区分源或者目的端口只要数据包包含有 80 端口都将被捕捉；

tcp port 80 or udp port 80 多条件匹配，匹配 TCP 或者 UDP 端口为 80 的数据包，不区分源或者目的端口；

tcp dst port 80 捕捉目的端口为 80 的数据包；

tcp src port 80 捕捉源端口为 80 的数据包；

tcp port 1|| udp port 53 捕捉 TCP 端口等于 1 或者 UDP 端口等于 53 的数据包；

tcp src portrange 1-80 捕捉端口范围 1 到 80 的 TCP 数据包；

src net 172.16.0.0/16 and (src portrange 1-2000 or src portrange 2000-40000) &&(dst portrange 1-80) 捕捉所有源网络为 172.16.0.0 并且端口范围是 1-2000 或者是 2000-40000 并且目的端口是 1-80 端口范围的数据包；

过滤协议：

例：在 Filter（过滤）框内直接输入 **tcp、udp、arp、icmp、tcp port http、tcp port smtp、tcp port ftp、ip** 等等；可直接过滤协议。假如想要排除哪个协议不显示我们可以直接在前面加上！

来进行排除。例如：!arp 就对 ARP 协议的数据包不做显示；（由于捕捉过滤器和显示过滤器有些差异，我们再输入时要注意语法）

过滤 MAC：

ether dst A0-00-00-04-C5-84 捕捉目标 mac 为 A0-00-00-04-C5-84 的数据包；

ether src A0-00-00-04-C5-84 捕捉源 mac 为 A0-00-00-04-C5-84 的数据包；

ether host A0-00-00-04-C5-84 无论源或者目的地址为 A0-00-00-04-C5-84 的数据包都会进行过滤；

由于显示过滤器和抓包过滤器功能有差异，所以他们的语法输入规则也是不同的，并且抓包过滤器的功能也不如显示过滤器那么强大。我们在此就介绍这么多，假如有兴趣的读者您也可以自己研究一下该项的使用方式。我们下面对抓包过滤器的使用举几个简单实用的列子以方便大家的学习；

例 1：假如我们想要只抓取源地址为 172.16.2.180 的数据包，那么我们可以输入：**ip src host 172.16.2.180** ；如下图 5-2-4 示：

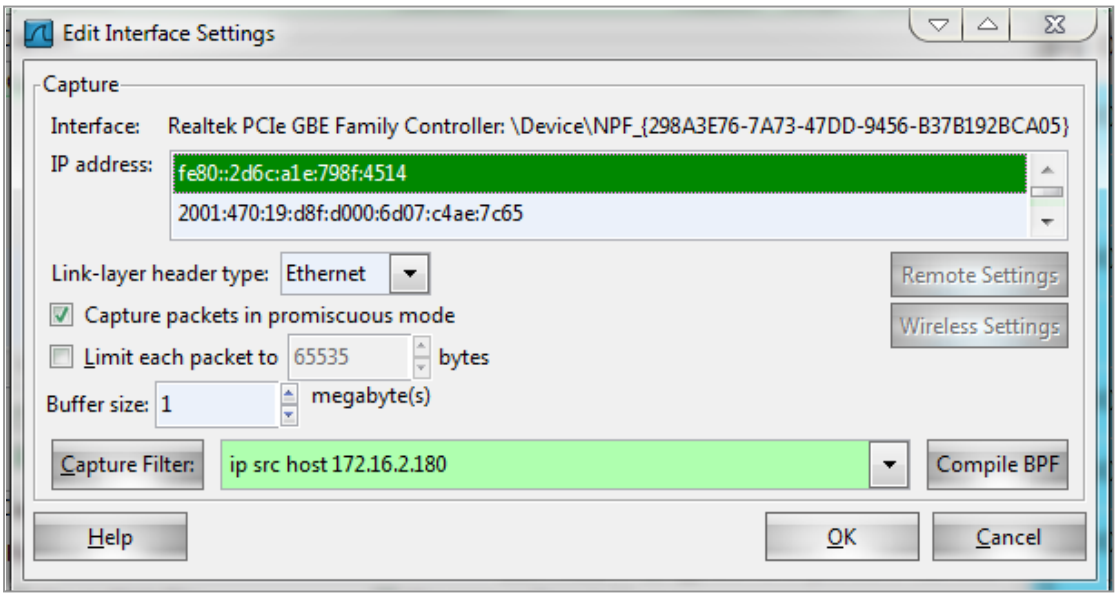


图 5-2-4

点击“OK”后点击“Start”wireshark 将开始抓包。并且只抓取源地址为 172.16.2.180 的数据包；如下图 5-2-5 所示：

No.	Time	Source	Destination	Protocol	Length	Info
20	3.6787720	172.16.2.180	183.60.49.88	UDP	137	Source port: 4007 Destination port: 8000
21	3.6788490	172.16.2.180	183.60.49.88	UDP	89	Source port: 4007 Destination port: 8000
22	3.6789180	172.16.2.180	183.60.49.88	UDP	89	Source port: 4007 Destination port: 8000
23	3.6794760	172.16.2.180	183.60.49.88	UDP	81	Source port: 4007 Destination port: 8000
24	3.7204010	172.16.2.180	183.60.49.88	UDP	89	Source port: 4007 Destination port: 8000
25	3.7365890	172.16.2.180	183.60.49.88	OICQ	81	OICQ Protocol
26	3.7374640	172.16.2.180	183.60.49.88	UDP	113	Source port: 4007 Destination port: 8000
27	3.7376970	172.16.2.180	183.60.49.88	UDP	113	Source port: 4007 Destination port: 8000
28	3.7379140	172.16.2.180	183.60.49.88	UDP	113	Source port: 4007 Destination port: 8000
29	3.7381420	172.16.2.180	183.60.49.88	UDP	113	Source port: 4007 Destination port: 8000
30	4.1504230	172.16.2.180	124.238.244.46	DNS	620	Standard query 0x0a04 [Malformed Packet]
31	4.8480270	172.16.2.180	183.60.49.88	OICQ	97	OICQ Protocol

图 5-2-5

例 2：假如想要抓取目的 TCP 端口范围是 1-80 的数据包，那么我们可以输入以上所介绍的 **tcp dst portrange 1-80** ；如下图 5-2-6 示：

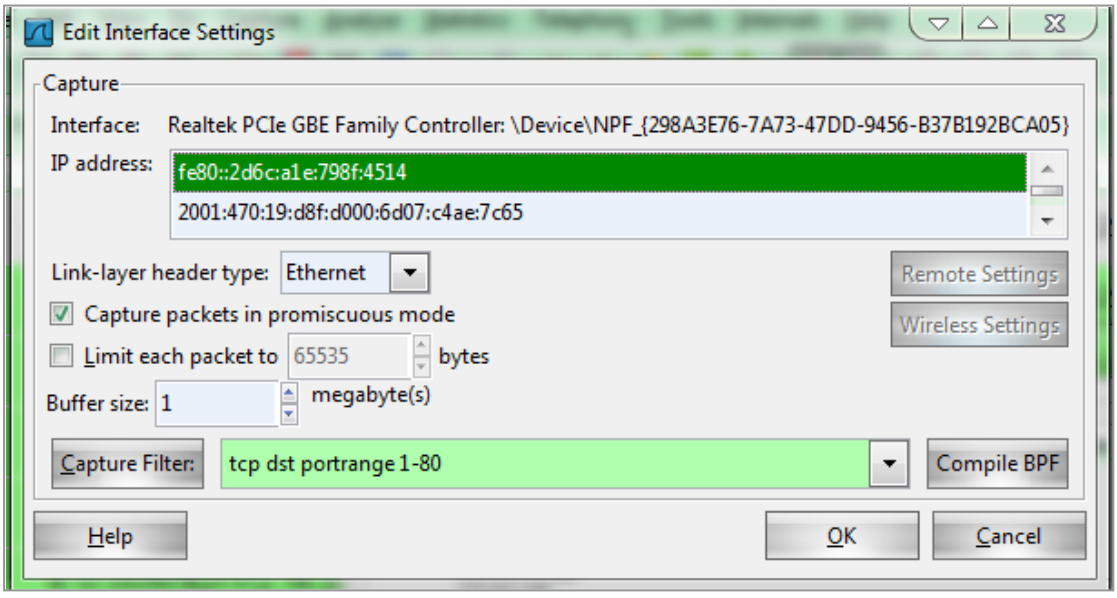


图 5-2-6

抓取的数据包如图 5-2-7 所示，目的端口全部是 80 端口的 HTTP 协议；

图 5-2-7

例 3 :假如我们想要抓取目的 MAC 地址是 b0-51-8e-01-80-8f ;的数据包，可输入 :**ether dst host b0-51-8e-01-80-8f** ；即可；如下图 5-2-8 示：

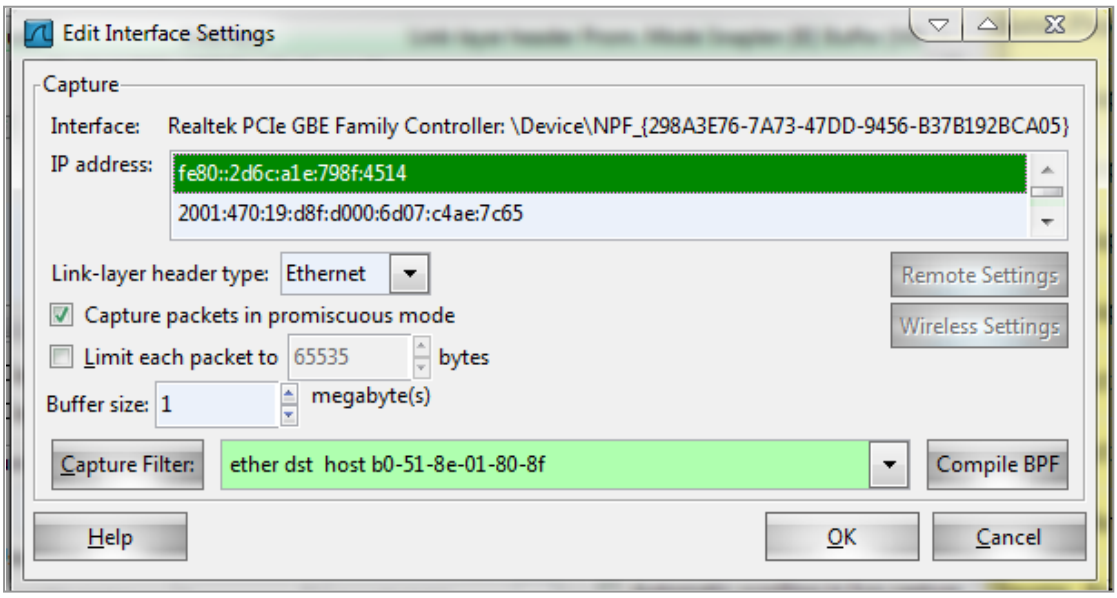


图 5-2-8

点击开始之后 wireshark 将只会捕捉目的 MAC 地址是 b0-51-8e-01-80-8f 的数据包。如下图 5-2-9 所示为我们抓取到的所有数据包：

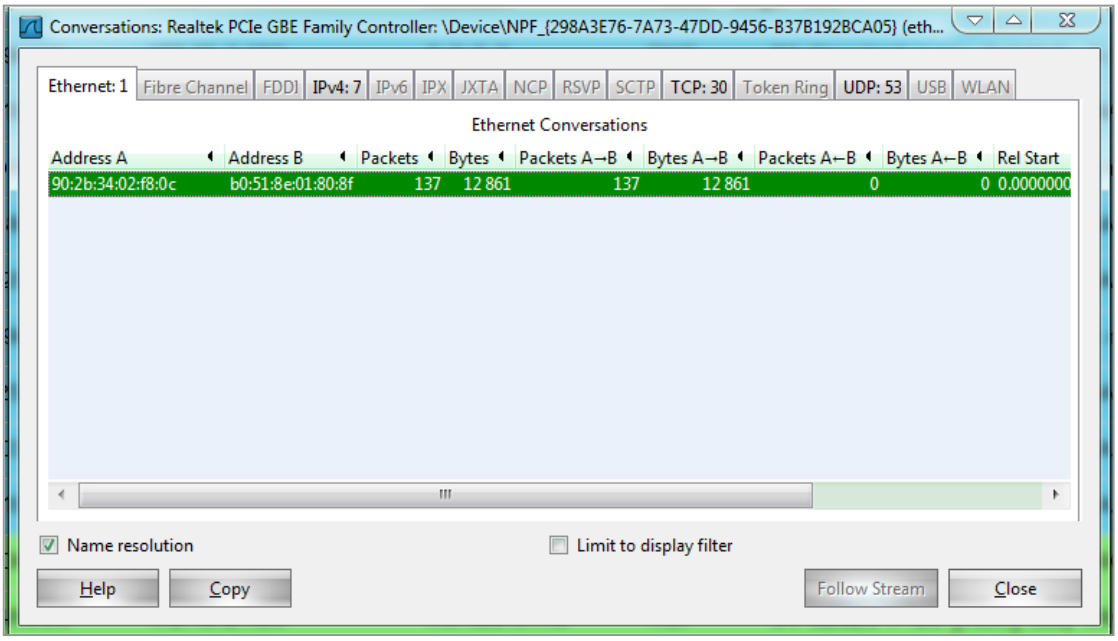


图 5-2-9

例 4：如果我们想要捕捉所有源网络为 172.16.0.0 并且 MAC 地址是 90-2B-34-02-F8-0C 并且源 TCP 端口范围是 1-65535 并且目的端口是 80 端口的数据包，那么我们可以输入：**src net 172.16.0.0/16 and (ether src host 90-2B-34-02-F8-0C&&tcp src portrange 1-65535) and tcp dst port 80** ；如下图 5-2-10 所示：

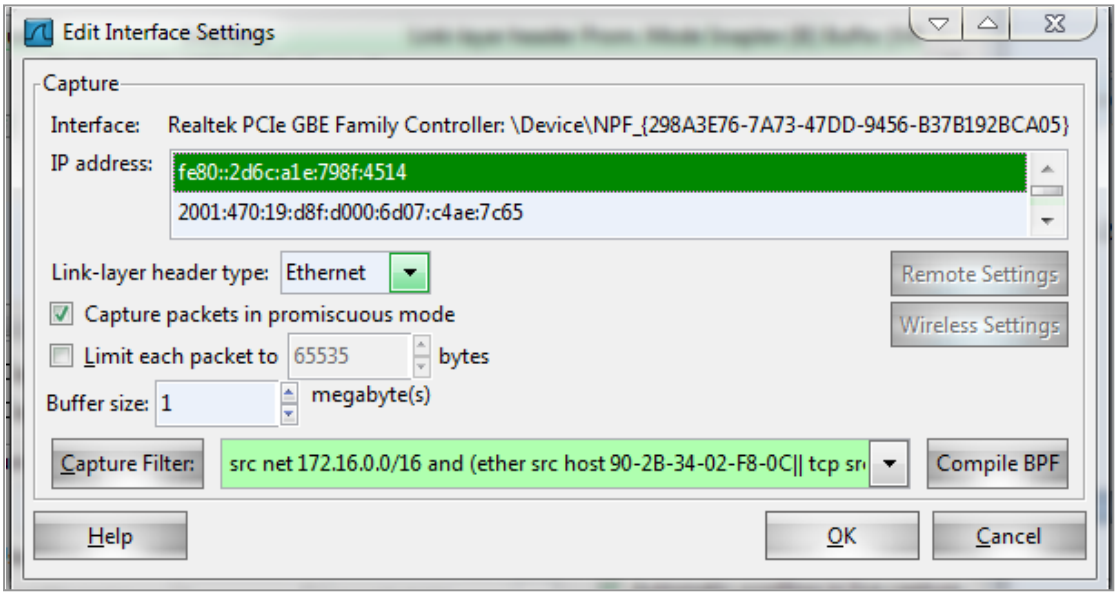


图 5-2-10

所捕捉到的数据包如下图 5-2-11，5-2-12 所示：

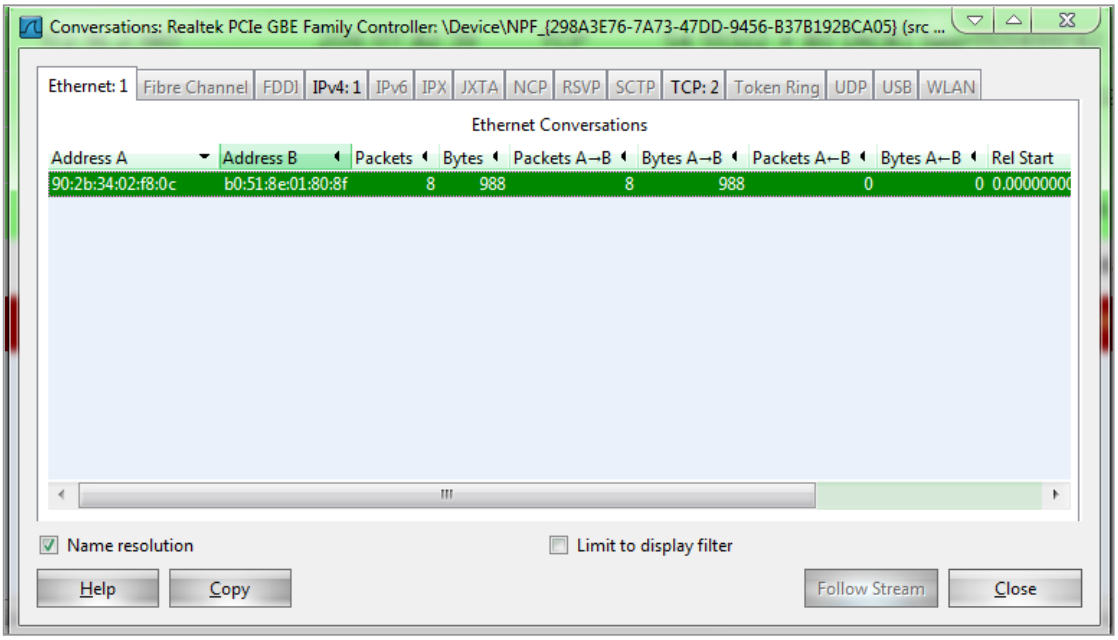


图 5-2-11

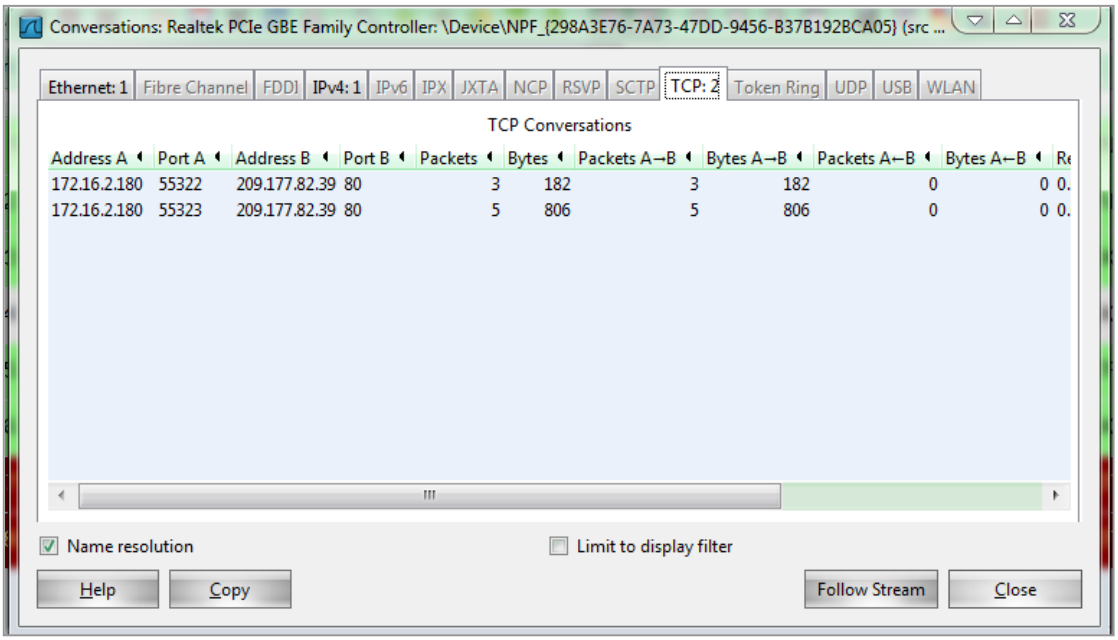


图 5-2-12

例子我们暂且就列举这么几个，大家以后自己可以尝试其他的过滤方式。在此我们的抓包过滤器我们就介绍这么多了。下面我们开始数据包分析的介绍。

◆ 协议分析

协议分析一章我们着重介绍 wireshark 所捕捉到的 TCP 以及 HTTP 正常数据流以及非正常的数据流，对于 UDP 我们只做简单的介绍，而对于这些协议的实现原理我们就稍微涉及到一点，毕竟现在有很多的书籍以及网络上对这些协议的介绍很多。大家如果对于协议的实现原理不是很清楚的话，直接上网下载一些文档以及自己买一些书籍都可以进行相关的学习。我们在此只对 wireshark 所捕捉到的各种协议的格式以及他们正常的工作过程做一些肤浅的讨论。

6.1、TCP 协议原理简介及分析

TCP/IP 在传输层指定了二个协议，UDP 和 TCP。在此我们将着重讨论 TCP 协议。TCP 与 UDP 类似位于网络层和应用层之间，是进程到进程的协议，但 TCP 是面向连接，可靠的运输协议，它给 IP 服务添加了面向连接和可靠性的特点。

6.1.1、TCP 协议原理简介

进程到进程的通信：

像 UDP 那样，TCP 也是使用端口号提供进程到进程的通信。与 UDP 不同的是 TCP 的端口号还可以复用，这是因为 TCP 使用 Socket（源 IP 地址源端口，目的 IP 地址目的端口）来标识一条连接。

流交付服务：

与 UDP 不同，TCP 是一种无边界的、面向数据流的协议，在 UDP 中进程把已有预先定义好边界的一些报文发送给 UDP 以便进行交付。但 TCP 刚允许发送进程以字节流的形式来传递数据，收进程也把数据作为字节流来接收。TCP 创建了一种环境，使得两个进程好像被一个假想的“管道”所连接。

全双工通信：

数据可在同一时间双向流动，每一个 TCP 都有发送缓存和接收缓存（滑动窗口），而两个方向都可以发送报文段。TCP 的捎带确认技术是全双工通信的一个很好应用。

面向连接的服务：

TCP 的工作时序为：

两个 TCP 在它们之间建立连接；

数据在两个方向进行交换；

连接被终止；

但有些连接并非连接，因为下层使用了不可靠的 IP 来进行交付服务；

可靠的服务：

TCP 使用确认机制来检查数据是否安全和完整地到达。这种机制的基础是带重传的肯定确认。

编号系统：

字节号：TCP 把在每一个方向传送的字节都进行编号，编号是从随机产生的数开始的；

序号：当字节都被编号后，TCP 就给每一个报文段指派一个序号。每个报文段的序号就是在这个报文段中的第一个字节数据的序号；

确认号：报文段中确认字段值定义了某一方所期望接收的下一个字节的编号。确认号是累计的。

滑动窗口：

TCP 在缓存上定义了一个窗口，缓存是用来暂存放从应用程序传递来并准备发送的数据。TCP 发送数据的多少由这个滑动窗口协议定义。在这个窗口中的字节是可以发送而不必考虑确认的，这个想象中的窗口有两个沿，一个在左边，另一个在右边，这个窗口叫滑动窗口左边的的字节是已发送且收

到确认的，右边字节是未发送的，窗口内的字节是发送了还未收到确认的，当收到确认后，窗口向前滑动。滑动窗口使传输更加有效，同时也控制了数据的流动，使得终点不致因数据过量而瘫痪。此滑动窗口是面向字节的。

窗口大小是 RWND 和 CWND 中较小的一个，源点并非必须要发送整个窗口大小的数据，接收端可以使窗口展开或合拢，但不应使它缩回，终点可以在任何时刻发送确认，只要不会引起窗口的缩回。接收端可以暂时关闭窗口，但发送端永远可以在窗口关闭后发送一个字节的报文段（防死锁）。

糊涂窗口综合症：

在滑动窗口的操作中可能出现一个严重的问题，这就是发送应用程序产生数据很慢，或者接收程序消耗数据很慢，或者两者都有。不管是上述哪种情况，都使得发送数据的报文段很小，这就非常低效的使用网络容量。这个问题叫糊涂窗口综合症。TCP 的慢启动状态会解决该问题。

发送端产生的症状：

为产生数据很慢的应用程序服务可能会产生糊涂窗口综合症，Nagle 算法找到了一个很好的解决方法。

1、发送端 TCP 经从发送应用程序收到的第一块数据发送出去，哪怕只有一个字节；

2、在发送第一个报文段以后，发送端 TCP 就在输出缓存中积累数据并行等待，直到或者接收端 TCP 发送出确认，或者已积累到足够的数据可以装成最大长度的报文段。这时，发送端 TCP 就可以发送这个报文段；

3、对剩下的传输，重复步骤 2，如果收到了对报文段 2 的确认，或者已积累到足够的数据可以装成最大长度的报文段，报文段 3 就必须发送出去。

接收端产生的症状：

如果接收端 TCP 为消耗数据很慢的应用服务，可能会产生糊涂窗口综合症。有两种建议的解决方法：

1、Clark 解决方法：只要有数据到达就发送确认，但在或者缓存已有足够大的空间放入最大长度的报文段之前，或者缓存空间的一半已经变空之前，一直都宣布窗口值为零；

2、推迟确认：当报文段到达时并不立即发送确认，接收端在对收到的报文段进行确认之前一直等待，直到缓存有足够的空间为止；

拥塞控制：

发送端的窗口大小不公取决于接收端，而且还取决于网络的拥塞。发送端有两种信息：

接收端通告的窗口大小和拥塞窗口大小。窗口的真正大小是两者中的较小的一个。真的窗口 = minimum(rwnd,cwnd)；

TCP 处理拥塞的一般策略是基于三个阶段：慢开始，拥塞避免和拥塞检测。在慢开始阶段，发送端从非常慢的发送速率开始，但很快就把速率增大到一个门限。当到达门限时，数据率的增长就放慢以避免拥塞。最后，如果检测到拥塞，发送端就又回到慢开始或拥塞避免阶段，这要根据拥塞是怎样检测到的。

在慢开始算法中，拥塞窗口的大小按指数规律增长，直到它到达一个门限为止。

在拥塞避免算法中，拥塞窗口的大小按照加法规律增长，直到拥塞被检测到。大多数的实现对拥塞检测的机制不一样：如果是用超时检测到拥塞，那么就开始一个新的慢开始阶段。如果是用三个 ACK 检测到拥塞，那么就开始一个新的拥塞避免阶段。

TCP 的报文段格式：

TCP 的分组叫做报文段，其格式如下图 6-1-1 所示：

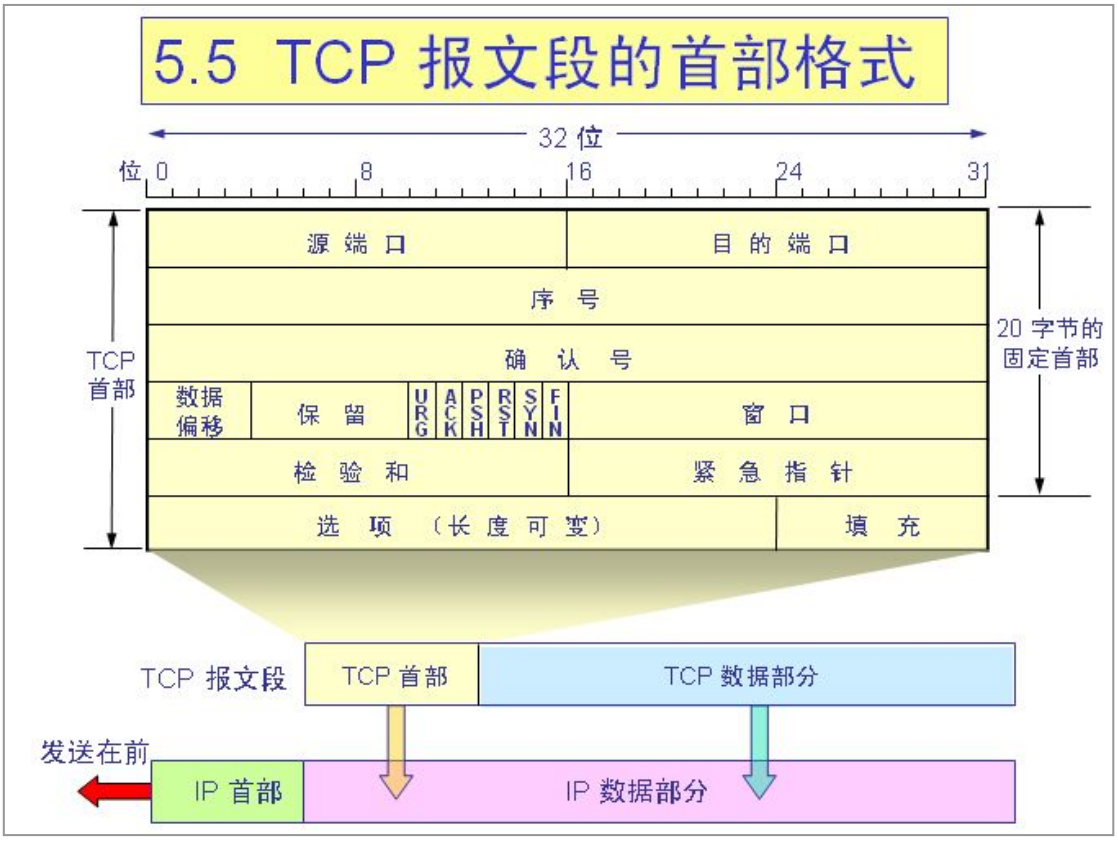


图 6-1-1

由两部分组成，首部和数据。首部在没有选项时是 20 字节，当有一些选项时可多达 60 字节。下面分别讨论每个字段的意思：

源端口：定义了在主机中发送这个报文段的应用程序的端口号；

目的端口：定义了在主机在接收这个报文应用程序的端口号；

序号：定义了指派给本报文段第一个数据字节的一个号。在连接建立时，每一方使用随机数产生器产生初始序号；

确认号：定义了报文段的接收端期望从对方接收的序号。确认号可以和数据捎带在一起发送；

数据偏移：这个 4 位字段指出 TCP 首部共有多少个 4 字节；

保留：保留为今后使用；

标志位：定义了 6 种不同的控制位或标志，在同一时间可设置一位或多位标志，

紧急指针有效：ACK:确认有效;PSH:请求推操作;RST:连接复位;SYN:同步序号;FIN:终止连接；

窗口：定义对方必须维持的窗口值（字节为单位）；

检验和：在 UDP 中是可选的，但在 TCP 中是强制的；

紧急指针：指明报文段中包含紧急数据；

选项：加入可选项，如 SACK；

TCP 连接：

TCP 连接是虚拟的，不是物理的，TCP 工作在高层。TCP 使用 IP 的服务把单个的报文段交付给接收端，但是 TCP 控制这个连接本身，如果一个报文段丢失了或受到损伤，那么它就要重传。但 IP 不知道这种重传，如果一个报文段没有按序到达，那么 TCP 保留它，直到丢失的报文段到达为止，IP 并不知道这种重新排序。TCP 面向连接的传输需要三个阶段：连接建立、数据传输和连接终止。

连接建立：

三次握手建立连接：

1、客户发送第一个报文段，SYN 报文段，这个报文段中只有 SYN 标志置 1，这个报文段的作用是使序号同步，不携带任何数据，但是它消耗一个序号；

2、服务器发送第二个报文段，即 SYN+ACK 报文段，服务器使用 SYN 报文段同步初始序号，以便从服务器发送字节。使用 ACK 标志确认已从客户端收到了 SYN 报文段，同时给出期望从客户端收到的下一个序号，服务器还必须定义客户端要使用的接收窗口。SYN + ACK 报文段不携带数据，但消耗一个序号；

3、发送第三个报文段仅仅是一个 ACK 报文段，此报文段序号与上一个 SYN 序号一样，ACK 不消耗任何序号，客户还必须定义服务器窗口值。（有些实现中第三个报文段可以携带数据块，这时报文段必须有一个新的序号来表示数据中的第一个字节的编号）

同时打开：

两个进程都发出主动打开，这种情况下，两个 TCP 都向对方发送 SYN+ACK 报文段，此时没有客户，也没有服务器，通信的双方是对等的。等到两个 SYN+ACK 报文段确认了 SYN 报文段后，连接就打开了。（四向握手）

数据传送：

连接建立后，双向的数据传送就可以开始。客户和服务器都可以在两个方向传送数据和确认，两端使用缓存提高了 TCP 的效率。

推送数据：

但有些时候一方的应用程序打算把它键入的字符发给对方并期望立即收响应，数据的延迟传输和延迟交付对应用程序来说是不可接受的，此时发送端可以请求推操作，表示发送端 TCP 不必等待窗

口被填满，它每创建一个报文段就立即发送，发送端 TCP 还必须设置推送位 PSH 以告诉接收端 TCP，这个报文段所包括的数据必须尽快地交付接收应用程序，而不要等待更多的数据的到来。

紧急数据：

有些时候发送应用程序希望某一块数据由接收应用程序不按序读出，解决这个问题的方法是发送 URG 位置 1 的报文段。发送应用程序告诉发送端 TCP 这块数据是紧急的，发送端 TCP 创建报文段，并把紧急数据放在报文段的开始。报文其余部分可以包括来自缓存的正常数据。首部中的紧急指针字段了紧急数据的结束和正常数据的开始。当接收端 TCP 收到 URG 位置 1 的报文段时，它就利用紧急指针的值从报文段中提取出紧急数据，并不按序把它交付给接收 应用程序。

连接终止：

参加数据交换的双方中的任何一方都可以关闭连接，当一个方向的连接被终止时，另外一方还可向对方发送数据，现今大多数的初到允许在终止时有两个选项：具有半关闭的三向握手和四向握手。但目前大多采用了四向握手。

四次挥手的连接结束：

1、客户 TCP 在收到客户进程发来的关闭命令后，就发送第一个报文段 FIN，此报文段可以包含客户发送的最后一块数据；

2、服务器 TCP 在收到这个 FIN 报文段后，将发送一个 ACK 已完成对客户端所发送的 FIN 报文段的确认；

3、服务器在传输完对客户端的数据后将发送一个 FIN 报文；

4、客户 TCP 发送最后一个报文段 ACK 来证实从 TCP 服务器收到了 FIN 报文段。

半关闭：

在 TCP，一方可以终止发送数据，但仍然可以接收数据，这叫半关闭。客户发送 FIN 报文，半关闭了这个连接，但服务器仍然可以发送数据，当服务器已经把所有处理的数据都发送完毕时，就发送 FIN 报文段。客户端收到后回送 ACK 完成关闭。

连接复位：

在一端的 TCP 可以拒绝一个连接请求，可以异常终止一条连接，或可以终止一条空闲的连接，所有这些都可以用 RST 标志来完成。

差错控制：

TCP 使用差错控制提供可靠性，差错控制包括以下一些机制：检测受到损伤的报文段、丢失的报文段、失序的报文段和重复的报文段。差错控制还包括检测出差错后纠正差错的机制。TCP 中的差错检测和差错纠正是通过使用这样三个简单工具得的：检验和、确认以及超时。

TCP 的计时器：

为了平滑地完成 TCP 的操作，大多数的 TCP 实现使用了至少 4 种计时器：重传计时器、持久计时器、保活计时器和时间等待计时器。

重传计时器：

为了重传丢失的报文段，TCP 使用了重传计时器，它处理重传超进 RTO，也就是对报文段的确认的等待时间。

持久定时器：

为了对付零窗口值的通知，避免死锁 TCP 为每一个连接使用一个持久计时器，以确定什么时候发送探测报文。

保活计时器：

为了避免某个空闲的连接持久处于打开状态，大多数的实现中都使用保活计时器，每当服务器收到客户的信息，就把计时器复位，待保活计时器超时就发送探测报文段，若发送了 10 个探测报文段还没收到响应就终止这个连接。

时间等待计时器：

为了防止关闭连接时最后一个 ACK 报文段丢失带来的问题和从一个连接来的重复的报文段可能会出现在下一个连接中，启用了时间等待计时器。

以上的 TCP 协议简介参考于 《TCP/IP 协议详解》 ，个人简化整理后写在这里。大家如果感觉有哪些地方错误以及不准确的地方还请大家多多参考些相关文档或者原书籍来进行学习。

6.1.2、TCP 协议数据包捕捉分析

首先我们先介绍下 wireshark 捕捉到的正常的 TCP 的数据包格式以及连接的建立与终止，然后我们再来对攻击数据包来进行分析。如下图 6-2-1 示，在 wireshark 的数据包详细信息栏查看 TCP 数据包的完整格式；

6.1.2.1、TCP 数据包头部格式

图 6-1-2

从上图中我们可以看到 TCP 的数据包和我们上面 TCP 协议简介中的格式完全一样。以上所示各个字段意义如下：

Source port（源端口）: 57598

Destination port（目的端口）: 80

Sequence number（序列号）: 4103881470

Header length（数据偏移也翻译为头部长度）: 40 bytes

Flags（标志位）: 0x002 (SYN)

000. .... = Reserved: Not set

...0 .... = Nonce: Not set

.... 0... .... = Congestion Window Reduced (CWR): Not set

.... .0.. .... = ECN-Echo: Not set

以上四个字段为保留；

.... ..0. .... = Urgent: Not set（URG 紧急指针）

....0 .... = Acknowledgment: Not set（ACK 标志）

.... 0... = Push: Not set（PUSH 标志）

.... .0.. = Reset: Not set（RST 标志）

....1. = Syn: Set（SYN 标志）

....0 = Fin: Not set（FIN 标志）

以上六个为 TCP 标志位；

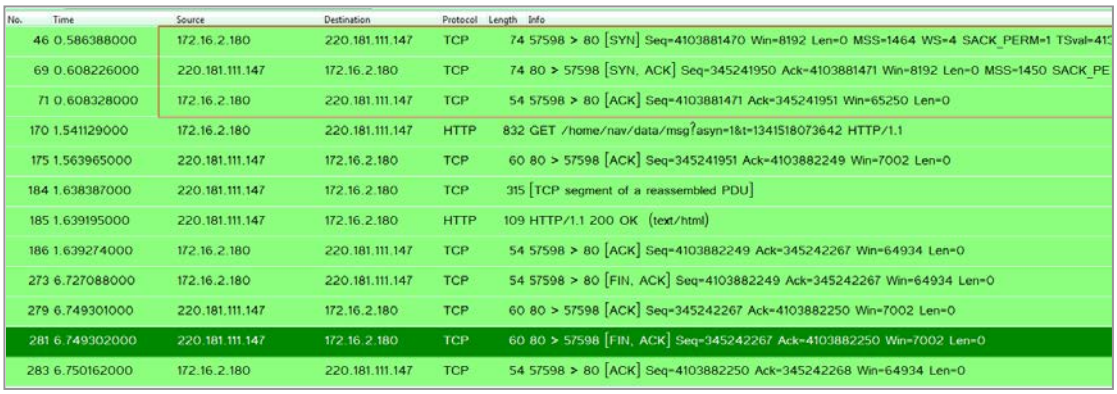
Window size value（窗口）: 8192

Checksum: 0xf6df（校验和）

Options: (20 bytes)（选项）

综上为 TCP 的首部字段；

6.1.2.2、TCP 连接建立的三次握手



No.	Time	Source	Destination	Protocol	Length	Info
46	0.586388000	172.16.2.180	220.181.111.147	TCP		74 57598 > 80 [SYN] Seq=4103881470 Win=8192 Len=0 MSS=1464 WS=4 SACK_PERM=1 TSval=41
69	0.608226000	220.181.111.147	172.16.2.180	TCP		74 80 > 57598 [SYN, ACK] Seq=345241950 Ack=4103881471 Win=8192 Len=0 MSS=1450 SACK_PE
71	0.608328000	172.16.2.180	220.181.111.147	TCP		54 57598 > 80 [ACK] Seq=4103881471 Ack=345241951 Win=65250 Len=0
170	1.541129000	172.16.2.180	220.181.111.147	HTTP		832 GET /home/nav/data/msg?asyn=1&t=1341518073642 HTTP/1.1
175	1.563965000	220.181.111.147	172.16.2.180	TCP		60 80 > 57598 [ACK] Seq=345241951 Ack=4103882249 Win=7002 Len=0
184	1.638387000	220.181.111.147	172.16.2.180	TCP		315 [TCP segment of a reassembled PDU]
185	1.639195000	220.181.111.147	172.16.2.180	HTTP		109 HTTP/1.1 200 OK (text/html)
186	1.639274000	172.16.2.180	220.181.111.147	TCP		54 57598 > 80 [ACK] Seq=4103882249 Ack=345242267 Win=64934 Len=0
273	6.727088000	172.16.2.180	220.181.111.147	TCP		54 57598 > 80 [FIN, ACK] Seq=4103882249 Ack=345242267 Win=64934 Len=0
279	6.749301000	220.181.111.147	172.16.2.180	TCP		60 80 > 57598 [ACK] Seq=345242267 Ack=4103882250 Win=7002 Len=0
281	6.749302000	220.181.111.147	172.16.2.180	TCP		60 80 > 57598 [FIN, ACK] Seq=345242267 Ack=4103882250 Win=7002 Len=0
283	6.750162000	172.16.2.180	220.181.111.147	TCP		54 57598 > 80 [ACK] Seq=4103882250 Ack=345242268 Win=64934 Len=0

图 6-1-3

从上图 6-1-3 红色矩形框内，我们可以观察出一个正常的 TCP 会话的建立以及终止：

1、首先 172.16.2.180 先发送了一个 TCP 标志位是 SYN 序列号为：Sequence number: 4103881470 的数据包用来新建一个 TCP 的连接；

2、然后对端的服务器回应了一个 SYN ACK 的响应包，服务器的 ACK 用来对客户端所发送的 SYN 数据包进行确认，确认序列号为：Acknowledgment number: 4103881471。由于客户端的 SYN 所占用了一个字节的长度所以服务器的确认数据包是客户端的序列号加 1。即通知客户端：我想收到的下一个数据包的序列号是 4103881471。同样服务器的 SYN 也是用来开始一个新的 TCP 会话，序列号为：Sequence number: 345241950；

3、客户端对服务器的 SYN 数据包进行确认，确认序列号：Acknowledgment number: 345241951。服务器的序列号加 1；

以上为一个正常的 TCP 的三次握手的连接建立方式；

6.1.2.3、TCP 四次挥手的连接终止

46 0.586388000	172.16.2.180	220.181.111.147	TCP	74 57598 > 80 [SYN] Seq=4103881470 Win=8192 Len=0 MSS=1464 WS=4 SACK_PERM=1 TSval=41
69 0.608226000	220.181.111.147	172.16.2.180	TCP	74 80 > 57598 [SYN, ACK] Seq=345241950 Ack=4103881471 Win=8192 Len=0 MSS=1450 SACK_PE
71 0.608328000	172.16.2.180	220.181.111.147	TCP	54 57598 > 80 [ACK] Seq=4103881471 Ack=345241951 Win=65250 Len=0
170 1.541129000	172.16.2.180	220.181.111.147	HTTP	832 GET /home/nav/data/msg?asyn=1&t=1341518073642 HTTP/1.1
175 1.563965000	220.181.111.147	172.16.2.180	TCP	60 80 > 57598 [ACK] Seq=345241951 Ack=4103882249 Win=7002 Len=0
184 1.638387000	220.181.111.147	172.16.2.180	TCP	315 [TCP segment of a reassembled PDU]
185 1.639195000	220.181.111.147	172.16.2.180	HTTP	109 HTTP/1.1 200 OK (text/html)
186 1.639274000	172.16.2.180	220.181.111.147	TCP	54 57598 > 80 [ACK] Seq=4103882249 Ack=345242267 Win=64934 Len=0
273 6.727088000	172.16.2.180	220.181.111.147	TCP	54 57598 > 80 [FIN, ACK] Seq=4103882249 Ack=345242267 Win=64934 Len=0
279 6.749301000	220.181.111.147	172.16.2.180	TCP	60 80 > 57598 [ACK] Seq=345242267 Ack=4103882250 Win=7002 Len=0
281 6.749302000	220.181.111.147	172.16.2.180	TCP	60 80 > 57598 [FIN, ACK] Seq=345242267 Ack=4103882250 Win=7002 Len=0
283 6.750162000	172.16.2.180	220.181.111.147	TCP	54 57598 > 80 [ACK] Seq=4103882250 Ack=345242268 Win=64934 Len=0

图 6-1-4

如上图 6-1-4 红色矩形框内所示，一个连接的终止的数据包的发送顺序：

1、首先角色为客户端的 172.16.2.180 这台主机发送一个 ACK FIN 数据包给服务器；其中确认序列号为：Acknowledgment number: 345242267；告知服务器端我期望的下一个数据包序列号为 345242267。并且 FIN 位被置 1.

2、服务器对客户端的 FIN 包进行确认，发送序列号为：Sequence number: 345242267；确认序列号为：Acknowledgment number: 4103882250 的结束确认数据包；

3、服务器对客户端数据传输完毕后发送 FIN 数据包，来结束服务器对客户端的通信。同样的发送 FIN 位置 1 的数据包；

4、客户端对服务器所发送的 FIN 数据包进行确认；

综上一个 TCP 连接的终止就完成了；

6.1.2.4、SYN Flood 攻击数据包

攻击原理：

恶意的攻击者利用 TCP/IP 协议的缺陷，向一个服务器发送大量的 SYN 报文段，而每一个这样的报文段来自不同的客户，并在 IP 数据报中使用虚假的源 IP 地址，服务器以为这些客户要发出主动打开，于是就分配必要的资源，如创建 TCB 表和设置一些定时器。TCP 的服务器就向这些虚假的客户发送 SYN+ACK 报文段，但他们都丢失了。如果客户端不发出确认，服务器会等待到超时，期间这些半连接状态都保存在一个空间有限的缓存队列中；如果大量的 SYN 包发到服务器端后没有应答，就会使服务器端的 TCP 资源迅速耗尽，导致正常的连接不能进入。如果在这样短暂的时间内 SYN 报文段的数量很大，服务器就会最终因资源耗尽而瘫痪。

SYN Flood 攻击数据包分析：

捕捉 SYN Flood 攻击数据包我们发现攻击机以极快的频率来进行发送，如上面所讲，在短暂的时间内耗尽服务器的资源，造成服务器的拒绝服务，从而达到攻击目的；

典型的 SYN Flood 攻击数据包如下图 6-1-5 所示：

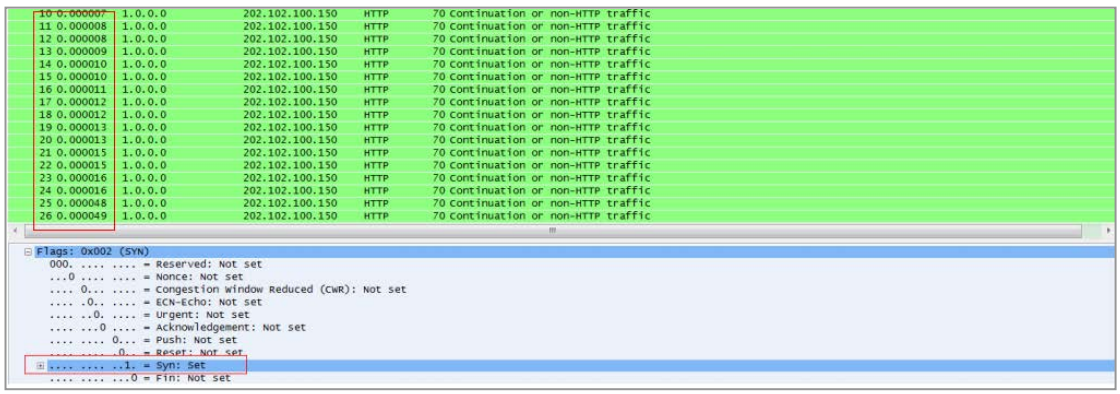


图 6-1-5

6.1.2.5、ACK Flood 攻击

攻击原理：

TCP 连接建立之后，所有的数据传输 TCP 报文都是带有 ACK 标志位的，主机在接收到一个带有 ACK 标志位的数据包的时候，需要检查该数据包所表示的连接四元组是否存在，然后决定是否回应。

对比主机以及防火墙在接收到 ACK 报文和 SYN 报文时所做动作的复杂程度，显然 ACK 报文带来的负载要小得多。所以在实际环境中，只有当攻击程序每秒钟发送 ACK 报文的速率达到一定的程度，才能使主机和防火墙的负载有大的变化。当发包速率很大的时候，主机操作系统将耗费大量的精力接收报文、判断状态，同时要主动回应 RST 报文，正常的数据包就可能无法得到及时的处理。这时候客户端（以 IE 为例）的表现就是访问页面反应很慢，丢包率较高。

目前 ACK Flood 并没有成为攻击的主流，而通常是与其他攻击方式组合在一起使用。回顾前面描述的 SYN Cookie 算法，其核心思想是主动回应 SYN/ACK 包，然后校验第 3 次握手的 ACK 报文是否合法，目前大多数实现中校验 ACK 报文的合法性都涉及到较为复杂的算法。当 SYN Flood 与 ACK Flood 一起发生的时候， 主机和防火墙将耗费大量的精力来计算 ACK 报文是否合法以致不堪重负。

ACK Flood 攻击数据包分析：

相同于 SYN Flood，ACK Flood 攻击的发送数据包的频率相对于 SYN Flood 有过之而无不及。如下图 6-1-6，6-1-7 所示：

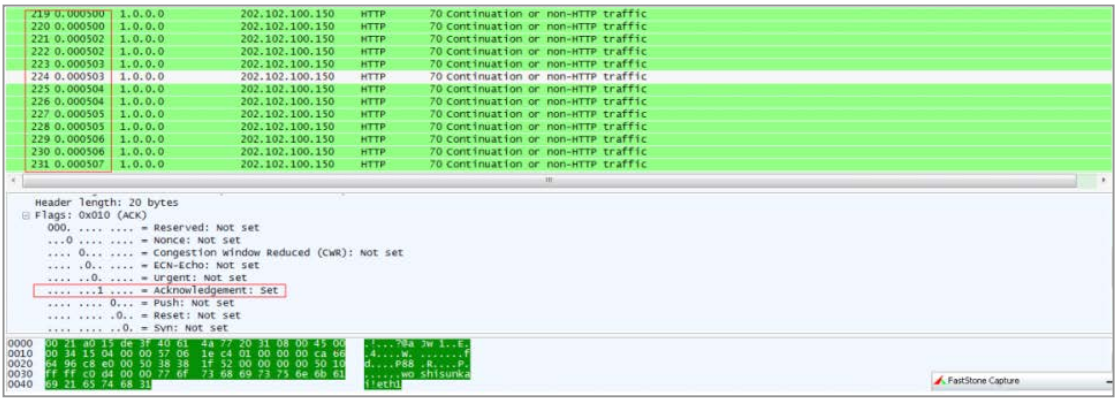


图 6-1-6

以上 TCP 协议的数据包分析我们就介绍这么多了，攻击数据包列举我们也就列举以上两种显而易见的。相对于流量型的攻击更为难防的 CC（Challenge Collapsar ）攻击我们将在下一章 HTTP 协议分析里面进行介绍；

6.2、HTTP 协议原理简介及分析

HTTP 协议相对于 TCP 协议简单了许多。HTTP 是一个属于应用层的面向对象的协议，由于其简捷、快速的方式，适用于分布式超媒体信息系统。经过几年的使用与发展，得到不断地完善和扩展。目前在 WWW 中使用的是 HTTP/1.1 版本。http（超文本传输协议）是一个基于请求与响应模式的、无状态的、应用层的协议，常基于 TCP 的连接方式，HTTP1.1 版本中给出一种持续连接的机制，绝大多数的 Web 开发，都是构建在 HTTP 协议之上的 Web 应用。

6.2.1、HTTP 协议工作原理简介

当我们打开浏览器，在地址栏中输入 URL，然后我们就看到了网页。 原理是怎样的呢？

实际上我们输入 URL 后，我们的浏览器给 Web 服务器发送了一个 Request, Web 服务器接到 Request 后进行处理 生成相应的 Response 然后发送给浏览器，浏览器解析 Response 中的 HTML, 这样我们就看到了网页，过程如下图6-2-1所示：

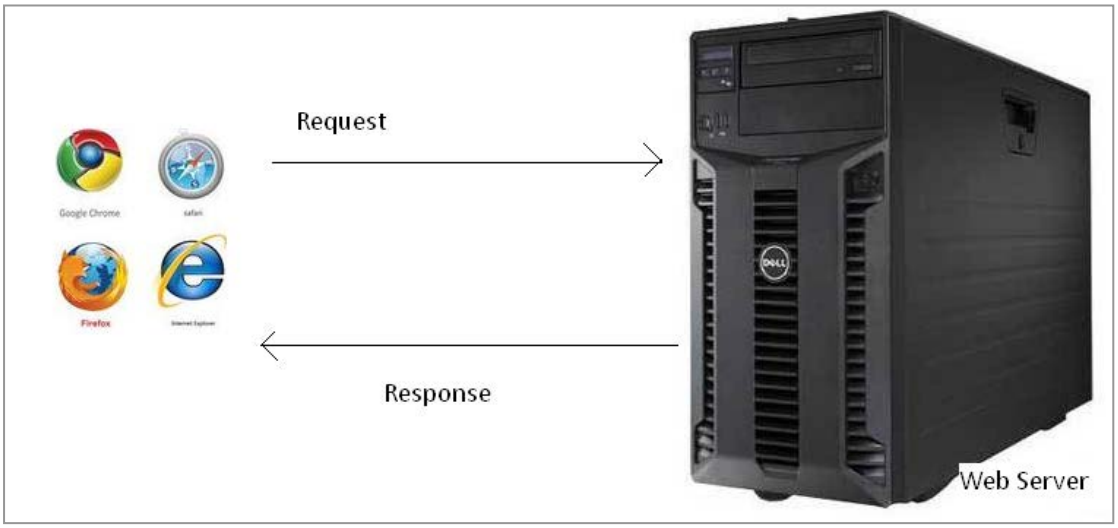


图6-2-1

我们的 Request 有可能是经过了代理服务器，最后才到达 Web 服务器的。过程如下图6-2-2所示：

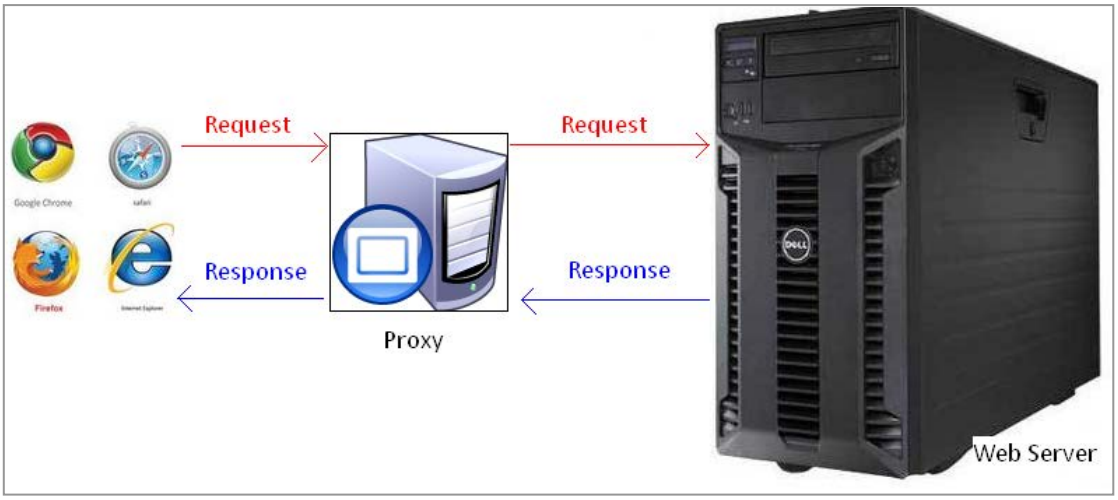


图6-2-2

代理服务器就是网络信息的中转站，有如下功能：

1、提高访问速度， 大多数的代理服务器都有缓存功能；

2、突破限制， 也就是翻墙了；

3、隐藏身份；

HTTP 协议（URL）

URL(Uniform Resource Locator) 地址用于描述一个网络上的资源， 基本格式如下

schema://host[:port#]/path/.../[?query-string][#anchor]

Scheme：指定低层使用的协议(例如：http, https, ftp)

Host：HTTP 服务器的 IP 地址或者域名

Port：HTTP 服务器的默认端口是80，这种情况下端口号可以省略。果使用了别的端口，必须指明，例如 http://www.cnblogs.com:8080/

Path：访问资源的路径

Query-string：发送给 http 服务器的数据

Anchor：锚

HTTP 协议的请求

http 请求由三部分组成，分别是：请求行（Request line ）、请求头（Request header ）、请求正文（body. Header）；

1、请求行以一个方法符号开头，以空格分开，后面跟着请求的 URI 和协议的版本，格式如下：**Method/Path-to-Resource HTTP/Version-Number**

其中 Method 表示请求方法,比如"POST","GET", Path-to-resoure 表示请求的资源，Http/version-number 表示 HTTP 协议的版本号；

例：**GET http://www.baidu.com/ HTTP/1.1**

Host: www.baidu.com

当使用的是"GET" 方法的时候，body 是为空的。请求方法（所有方法全为大写）有多种，各个方法的解释如下：

GET：请求获取 Request-URI 所标识的资源；

POST：在 Request-URI 所标识的资源后附加新的数据；

HEAD：请求获取由 Request-URI 所标识的资源的响应消息报头；

PUT：请求服务器存储一个资源，并用 Request-URI 作为其标识；

DELETE：请求服务器删除 Request-URI 所标识的资源；

TRACE：请求服务器回送收到的请求信息，主要用于测试或诊断；

CONNECT：保留将来使用；

OPTIONS：请求查询服务器的性能，或者查询与资源相关的选项和需求；

HTTP 协议的响应

在接收和解释请求消息后，服务器返回一个 HTTP 响应消息。HTTP 响应也是由三个部分组成，分别是：**状态码、消息报头、响应正文**

状态码用来告诉 HTTP 客户端,HTTP 服务器是否产生了预期的 Response.

HTTP/1.1中定义了5类状态码， 状态码由三位数字组成，第一个数字定义了响应的类别。

1XX：提示信息：表示请求已被成功接收，继续处理；

2XX：成功：表示请求已被成功接收，理解，接受；

3XX：重定向：要完成请求必须进行更进一步的处理；

4XX：客户端错误 - 请求有语法错误或请求无法实现；

5XX：服务器端错误 - 服务器未能实现合法的请求。

HTTP 协议的请求头

Cache 头域：

If-Modified-Since 作用： 把浏览器端缓存页面的最后修改时间发送到服务器去，服务器会把这个时间与服务器上实际文件的最后修改时间进行对比。如果时间一致，那么返回 304，客户端就直接使用本地缓存文件。如果时间不一致，就会返回 200 和新的文件内容。客户端接到之后，会丢弃旧文件，把新文件缓存起来，并显示在浏览器中。

例：**If-Modified-Since: Thu, 09 Feb 2012 09:07:57 GMT**

If-None-Match 作用: If-None-Match 和 ETag 一起工作，工作原理是在 HTTP Response 中添加 ETag 信息。当用户再次请求该资源时 将在 HTTP Request 中加入 If-None-Match 信息(ETag 的值)。如果服务器验证资源的 ETag 没有改变（该资源没有更新），将返回一个 304 状态告诉客户端使用本地缓存文件。否则将返回 200 状态和新的资源和 Etag. 使用这样的机制将提高网站的性能；

例：**If-None-Match: "03f2b33c0bfcc1:0"**

Pragma 作用：防止页面被缓存，在 HTTP/1.1 版本中，它和 Cache-Control:no-cache 作用一模一样；

Pragma 只有一个用法，例：**Pragma: no-cache。**

Cache-Control 作用: 这个是非常重要的规则。这个用来指定 Response-Request 遵循的缓存机制。各个指令含义如下：

Cache-Control:Public：可以被任何缓存所缓存；

Cache-Control:Private：内容只缓存到私有缓存中；

Cache-Control:no-cache：所有内容都不会被缓存；

Client 头域：

Accept 作用：浏览器端可以接受的媒体类型；

例：**Accept: text/html** 代表浏览器可以接受服务器回发的类型为 text/html 也就是我们常说的 html 文档；

如果服务器无法返回 text/html 类型的数据,服务器应该返回一个 406 错误(non acceptable)

通配符 * 代表任意类型；

例： **Accept: */*** 代表浏览器可以处理所有类型,(一般浏览器发给服务器都是发这个)；

Accept-Encoding：作用：浏览器申明自己接收的编码方法，通常指定压缩方法，是否支持压缩，支持什么压缩方法（gzip，deflate），（注意：这不是只字符编码）；

例：**Accept-Encoding: gzip, deflate**

Accept-Language 作用：浏览器申明自己接收的语言。

语言跟字符集的区别：中文是语言，中文有多种字符集，比如 big5，gb2312，gbk 等等；

例：**Accept-Language: en-us**

User-Agent 作用：告诉 HTTP 服务器，客户端使用的操作系统和浏览器的名称和版本。

我们上网登陆论坛的时候，往往会看到一些欢迎信息，其中列出了你的操作系统的名称和版本，你所使用的浏览器的名称和版本，这往往让很多人感到很神奇。实际上，服务器应用程序就是从 User-Agent 这个请求报头域中获取到这些信息 User-Agent 请求报头域允许客户端将它的操作系统、浏览器和其它属性告诉服务器。

例：**User-Agent: Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 5.1; Trident/4.0; CIBA; .NET CLR 2.0.50727; .NET CLR 3.0.4506.2152; .NET CLR 3.5.30729; .NET4.0C; InfoPath.2; .NET4.0E)**

Accept-Charset 作用：浏览器申明自己接收的字符集，这就是本文前面介绍的各种字符集和字符编码，如 gb2312，utf-8（通常我们说 Charset 包括了相应的字符编码方案）；

Cookie/Login 头域：

Cookie:作用：最重要的 header，将 cookie 的值发送给 HTTP 服务器；

Entity 头域：

Content-Length 作用：发送给 HTTP 服务器数据的长度；

例：**Content-Length: 38**

Content-Type 作用：当前类型；

例：**Content-Type: application/x-www-form-urlencoded**

Miscellaneous 头域：

Referer:作用：提供了 Request 的上下文信息的服务器，告诉服务器我是从哪个链接过来的，比如从我主页上链接到一个朋友那里，他的服务器就能够从 HTTP Referer 中统计出每天有多少用户点击我主页上的链接访问他的网站。

例：**Referer:http://translate.google.cn/?hl=zh-cn&tab=wT**

Transport 头域：

Connection 例如：Connection: keep-alive 当一个网页打开完成后，客户端和服务器之间用于传输 HTTP 数据的 TCP 连接不会关闭，如果客户端再次访问这个服务器上的网页，会继续使用这一条已经建立的连接；

例：**Connection: close** 代表一个 Request 完成后，客户端和服务器之间用于传输 HTTP 数据的 TCP 连接会关闭， 当客户端再次发送 Request，需要重新建立 TCP 连接。

Host（发送请求时，该报头域是必需的）作用：请求报头域主要用于指定被请求资源的 Internet 主机和端口号，它通常从 HTTP URL 中提取出来的；

例：**我们在浏览器中输入：http://www.guet.edu.cn/index.html**

浏览器发送的请求消息中，就会包含 Host 请求报头域，如下：

Host：http://www.guet.edu.cn 此处使用缺省端口号 80，若指定了端口号，则变成：Host：指定端口号；

HTTP 协议的应答头

Cache 头域：

Date 作用：生成消息的具体时间和日期；

例：**Date: Sat, 11 Feb 2012 11:35:14 GMT**

Expires 作用：浏览器会在指定过期时间内使用本地缓存；

例：**Expires: Tue, 08 Feb 2022 11:35:14 GMT**

Vary 作用：

例：**Vary: Accept-Encoding**

Cookie/Login 头域：

P3P 作用：用于跨域设置 Cookie，这样可以解决 iframe 跨域访问 cookie 的问题；

例：**P3P: CP=CURa ADMa DEVa PSAo PSDo OUR BUS UNI PUR INT DEM STA PRE COM NAV OTC NOI DSP COR**

Set-Cookie 作用：非常重要的 header，用于把 cookie 发送到客户端浏览器， 每一个写入 cookie 都会生成一个 Set-Cookie；

例：**Set-Cookie: sc=4c31523a; path=/; domain=.acookie.taobao.com**

Entity 头域：

ETag 作用：和 If-None-Match 配合使用。（实例请看上节中 If-None-Match 的实例）

例：**ETag: "03f2b33c0bfcc1:0"**

Last-Modifie：作用：用于指示资源的最后修改日期和时间。（实例请看上节的 If-Modified-Since 的实例）；

例：**Last-Modified: Wed, 21 Dec 2011 09:09:10 GMT**

Content-Type 作用：WEB 服务器告诉浏览器自己响应的对象的类型和字符集,

例：**Content-Type: text/html; charset=utf-8**

Content-Type:text/html;charset=GB2312

Content-Type: image/jpeg

Content-Length：指明实体正文的长度，以字节方式存储的十进制数字来表示。在数据下行的过程中，Content-Length 的方式要预先在服务器中缓存所有数据，然后所有数据再一股脑儿地发给客户端。

例：**Content-Length: 19847**

Content-Encoding：WEB 服务器表明自己使用了什么压缩方法（gzip，deflate）压缩响应中的对象。

例：**Content-Encoding：gzip**

Content-Language 作用： WEB 服务器告诉浏览器自己响应的对象的语言；

例：**Content-Language:da**

Miscellaneous 头域：

Server 作用：指明 HTTP 服务器的软件信息；

例：**Server: Microsoft-IIS/7.5**

X-AspNet-Version 作用：如果网站是用 ASP.NET 开发的，这个 header 用来表示 ASP.NET 的版本；

例：**X-AspNet-Version: 4.0.30319**

X-Powered-By 作用：表示网站是用什么技术开发的；

例：**X-Powered-By: ASP.NET**

Transport 头域：

Connection 例：Connection: keep-alive 当一个网页打开完成后，客户端和服务器之间用于传输 HTTP 数据的 TCP 连接不会关闭，如果客户端再次访问这个服务器上的网页，会继续使用这一条已经建立的连接

例：**Connection: close 代表一个 Request 完成后，客户端和服务器之间用于传输 HTTP 数据的 TCP 连接会关闭，当客户端再次发送 Request，需要重新建立 TCP 连接。**

Location 头域：

Location 作用：用于重定向一个新的位置，包含新的 URL 地址；

实例请看 304 状态实例；

6.2.2、HTTP 协议数据包捕捉分析

简单介绍了下 HTTP 协议的工作原理以及各个字段的意义后，我们开始使用 wireshark 来对所抓取的 HTTP 数据包进行分析。首先我们像上一章那样首先介绍下 HTTP 协议在 wireshark 中各个字段，然后再介绍攻击数据包；

6.2.2.1、HTTP 数据包头部格式

图 6-2-3

如上图 6-2-3 所示，HTTP 的头部数据格式每个字段的如下：

所请求的 URL 为：http://www.baidu.com/zxsoft/jszx/sk.asp?name=cisco#stuff

Schema（模式）：http

Host（主机）：www.baidu.com

Path（路径）：zxsoft/jszx/sk.asp

Query String（查询字符串）：name=cisco

Anchor（锚）：stuff

GET /index.php?tn=SE_desk HTTP/1.1

GET：

Request Method: GET：为请求方式；

/index.php?tn=SE_desk：为请求资源路径；

HTTP/1.1：为 HTTP 版本；

Host: www.baidu.com：请求主机为 www.baidu.com；

Connection: keep-alive：当一个网页打开完成后，客户端和服务器之间用于传输 HTTP 数据的 TCP 连接不会关闭，如果客户端再次访问这个服务器上的网页，会继续使用这一条已经建立的连接。

User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/536.5 (KHTML, like Gecko) Chrome/19.0.1084.52 Safari/536.5：浏览器类型以及操作系统型号；

Accept:text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8\r\n：浏览器端可以接受的媒体类型

Accept-Encoding: gzip,deflate,sdch：浏览器所接受的压缩方式；

Accept-Language: zh-CN,zh;q=0.8：浏览器所识别的语言；

Accept-Charset: GBK,utf-8;q=0.7,*;q=0.3：浏览器申明自己接收的字符集；

Cookie: BAIDUID=024CD1E917281DFC18D21E81324D9B09:FG=1; BDUT=kdmn024CD1E917281DFC18D21E81324D9B091383c8f9b2b0;

BDUSS=GYxZ3FlNmFKc3VuWEN2QWhVSjNONTFkN2xrY0liN3B0cEZ2OEEzc0tzSnRGTjFRQVFBQUFBJCQAAAAAAAAAAAoaJhr~-ggLuOjM2Ni8R290aAAA：缓存；

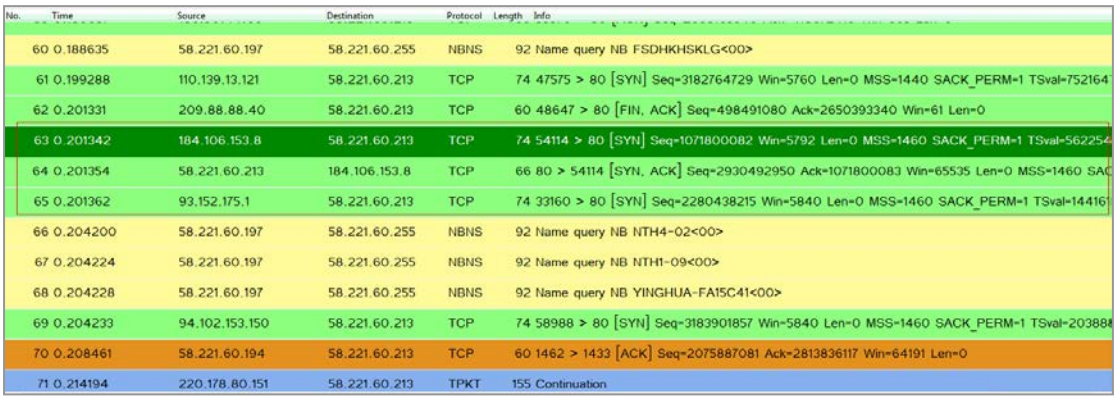
6.2.2.2、HTTP 协议的连接

一个正常的 HTTP 的请求数据包如图 5-2-2 所示，首先，TCP 先建立到对端的三次握手，成功后客户端首先发送 HTTP GET 请求来告知服务器端想要获取的 URL 以及其他一些信息。服务器发送客户端所请求的 URL 信息，完毕后发送 200 OK 消息告知客户端信息传送完毕；

从上图看 HTTP 的数据包结构正如我们之前所介绍的那样 ,现在大家对一个正常的 HTTP 的 GET 消息有了一个基本的认识。那么我们下面来看看异常的恶意的请求消息，即我们所讲的 CC 攻击。如下图 6-2-4 所示：

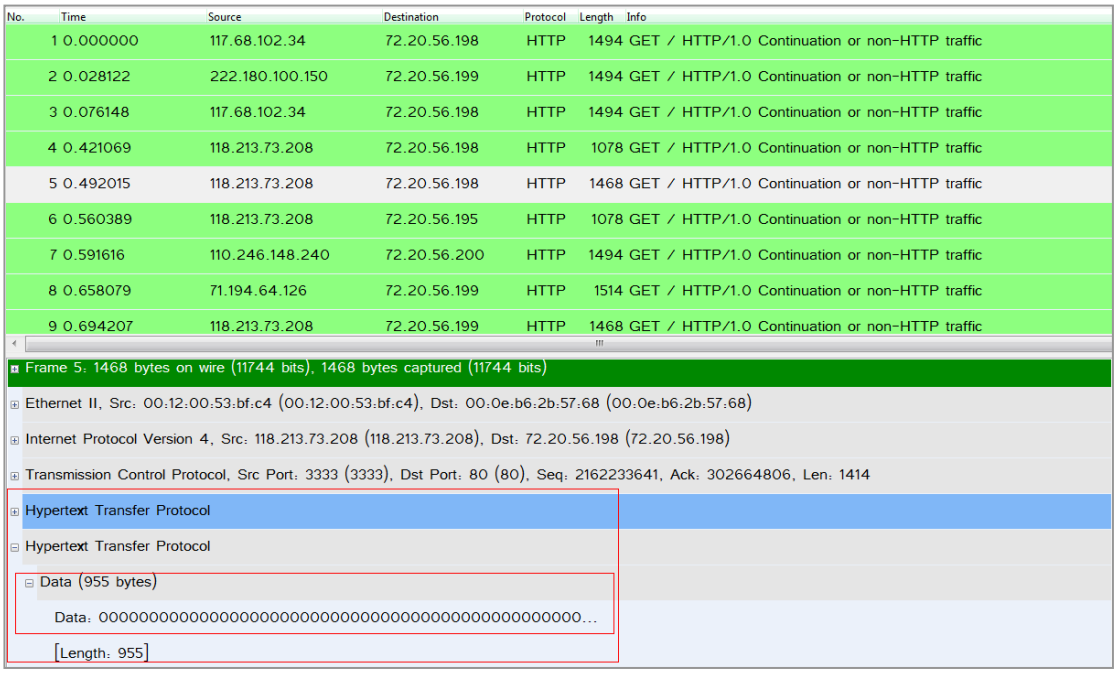
No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	211.160.122.92	98.126.64.28	TCP	60	44400 > 80 [SYN] Seq=920170908 Win=65535 Len=0 MSS=1380
2	0.000003	98.126.64.28	211.160.122.92	TCP	58	80 > 44400 [SYN, ACK] Seq=2077275677 Ack=920170909 Win=65535 Len=0 MSS=1460
3	0.000211	211.160.122.92	98.126.64.28	TCP	60	44400 > 80 [ACK] Seq=920170909 Ack=2077275678 Win=65535 Len=0
4	0.001245	211.160.122.92	98.126.64.28	HTTP	330	GET /favicon.ico HTTP/1.1
5	0.001281	98.126.64.28	211.160.122.92	HTTP	961	HTTP/1.1 200 OK (text/html)
6	0.368662	211.160.122.92	98.126.64.28	TCP	60	44400 > 80 [FIN, ACK] Seq=920171185 Ack=2077276585 Win=64628 Len=0
7	7.809313	211.160.122.92	98.126.64.28	TCP	60	23848 > 80 [SYN] Seq=3076971479 Win=65535 Len=0 MSS=1380
8	7.809318	98.126.64.28	211.160.122.92	TCP	58	80 > 23848 [SYN, ACK] Seq=2971786489 Ack=3076971480 Win=65535 Len=0 MSS=1460
9	7.809494	211.160.122.92	98.126.64.28	TCP	60	23848 > 80 [ACK] Seq=3076971480 Ack=2971786490 Win=65535 Len=0
10	7.809755	211.160.122.92	98.126.64.28	TCP	60	4335 > 80 [SYN] Seq=244033884 Win=65535 Len=0 MSS=1380
11	7.809760	98.126.64.28	211.160.122.92	TCP	58	80 > 4335 [SYN, ACK] Seq=1837418523 Ack=244033885 Win=65535 Len=0 MSS=1460
12	7.809958	211.160.122.92	98.126.64.28	TCP	60	4335 > 80 [ACK] Seq=244033885 Ack=1837418524 Win=65535 Len=0

（三次握手建立后发送 GET，典型 CC 攻击）图 6-2-4



No.	Time	Source	Destination	Protocol	Length	Info
60	0.188635	58.221.60.197	58.221.60.255	NBNS	92	Name query NB FSDHKHSKLG<00>
61	0.199288	110.139.13.121	58.221.60.213	TCP	74	47575 > 80 [SYN] Seq=3182764729 Win=5760 Len=0 MSS=1440 SACK_PERM=1 TSval=752164
62	0.201331	209.88.88.40	58.221.60.213	TCP	60	48647 > 80 [FIN, ACK] Seq=498491080 Ack=2650393340 Win=61 Len=0
63	0.201342	184.106.153.8	58.221.60.213	TCP	74	54114 > 80 [SYN] Seq=1071800082 Win=5792 Len=0 MSS=1460 SACK_PERM=1 TSval=562254
64	0.201354	58.221.60.213	184.106.153.8	TCP	66	80 > 54114 [SYN, ACK] Seq=2930492950 Ack=1071800083 Win=65535 Len=0 MSS=1460 SAC
65	0.201362	93.152.175.1	58.221.60.213	TCP	74	33160 > 80 [SYN] Seq=2280438215 Win=5840 Len=0 MSS=1460 SACK_PERM=1 TSval=14416
66	0.204200	58.221.60.197	58.221.60.255	NBNS	92	Name query NB NTH4-02<00>
67	0.204224	58.221.60.197	58.221.60.255	NBNS	92	Name query NB NTH1-09<00>
68	0.204228	58.221.60.197	58.221.60.255	NBNS	92	Name query NB YINGHUA-FA15C41<00>
69	0.204233	94.102.153.150	58.221.60.213	TCP	74	58988 > 80 [SYN] Seq=3183901857 Win=5840 Len=0 MSS=1460 SACK_PERM=1 TSval=20388
70	0.208461	58.221.60.194	58.221.60.213	TCP	60	1462 > 1433 [ACK] Seq=2075887081 Ack=2813836117 Win=64191 Len=0
71	0.214194	220.178.80.151	58.221.60.213	TPKT	155	Continuation

（三次握手不发送任何数据，典型 CC 攻击）图 6-2-5



（GET 消息包含多次请求并且包含垃圾数据，典型 CC 攻击）图 6-2-6

观察 6-2-4 我们发现，当一个 TCP 的三次握手建立成功后，客户端发送一个 GET 请求，服务器回应 ACK，但是在这时客户端突然发送了一个 FIN（结束）。服务器收到这个 FIN 消息后会结束这条连接，但是看图我们发现客户端以极快的频率来发送这种 GET 消息，服务器要不停的响应客户端的 GET 消息，然后客户端对自己所请求的消息不停地建立，结束，每一个请求消息里面的内容服务器都要花费资源去数据库执行查找操作，大量的 GET 消息以极快的频率就会造成会服务器 CPU、内存的使用率过高，就会造成服务器对后续的正常的 GET 请求也没有办法去执行响应。就会拒绝服务；

我们在此就只对 GET 请求攻击来进行一些简单的介绍，只为抛砖引玉想对大家有点提示作用。假如大家对这方面有兴趣也可以自己去百度一下各种 DDOS 攻击方式以及原理。我们在这里对于其他的一些不常见的攻击类型就不多做介绍了；

◆ 常见问题

Wireshark 在安装过程中以及使用过程中有时候因为各种原因会造成各种问题，我们在这里对常见的安装以及使用问题做一些简单的说明，如果大家在使用中遇到了问题可以进行参照，希望对大家有点提示作用。如果您的问题在这里没有说到，那么您打开浏览器输入 www.baidu.com 然后输入您的问题描述即可。

7.1、wireshark 安装问题

7.1.2、找不到接口

我们安装好了 wireshark 后，运行时候发现 wireshark 提示如下图 7-1-1 所示的界面，那么原因是什么呢？其实在 wireshark 的错误弹出对话框里面的文字已经告诉我们了。

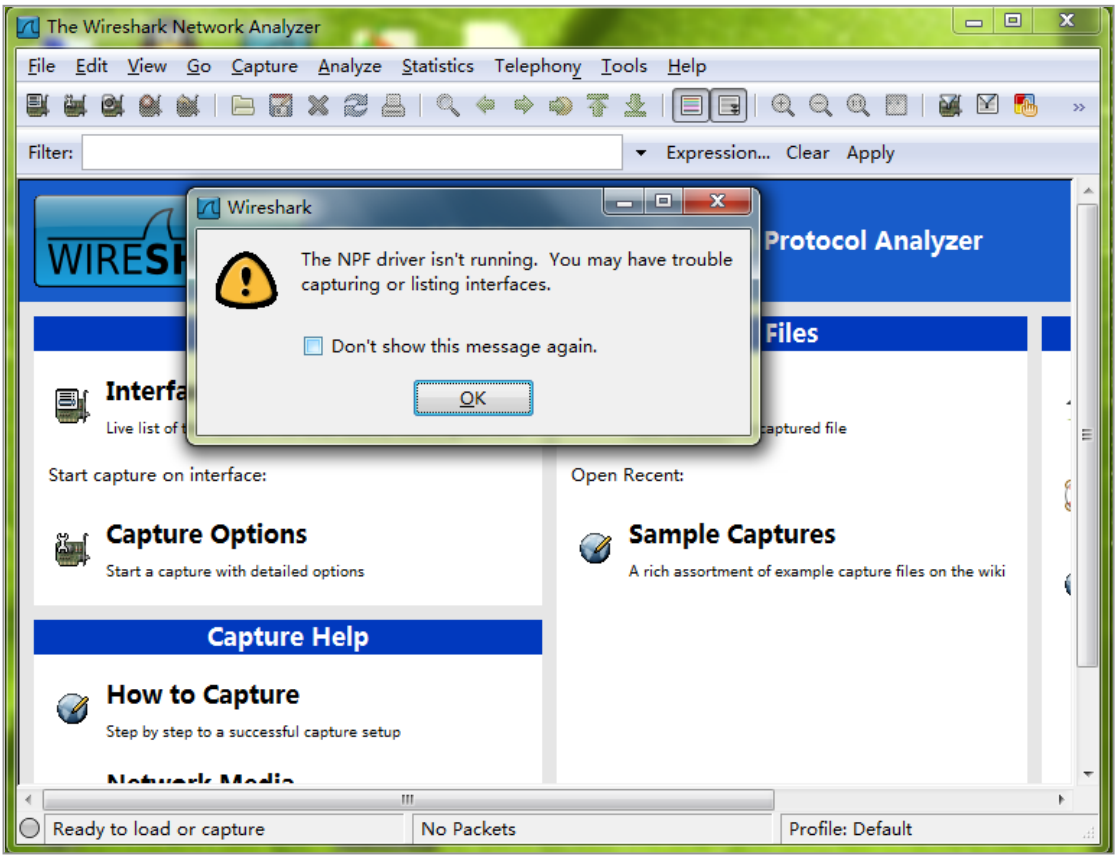


图 7-1-1

如上图所示，我们在接口列表看不到有网卡。点击 Interface List 后弹出如下对话框：

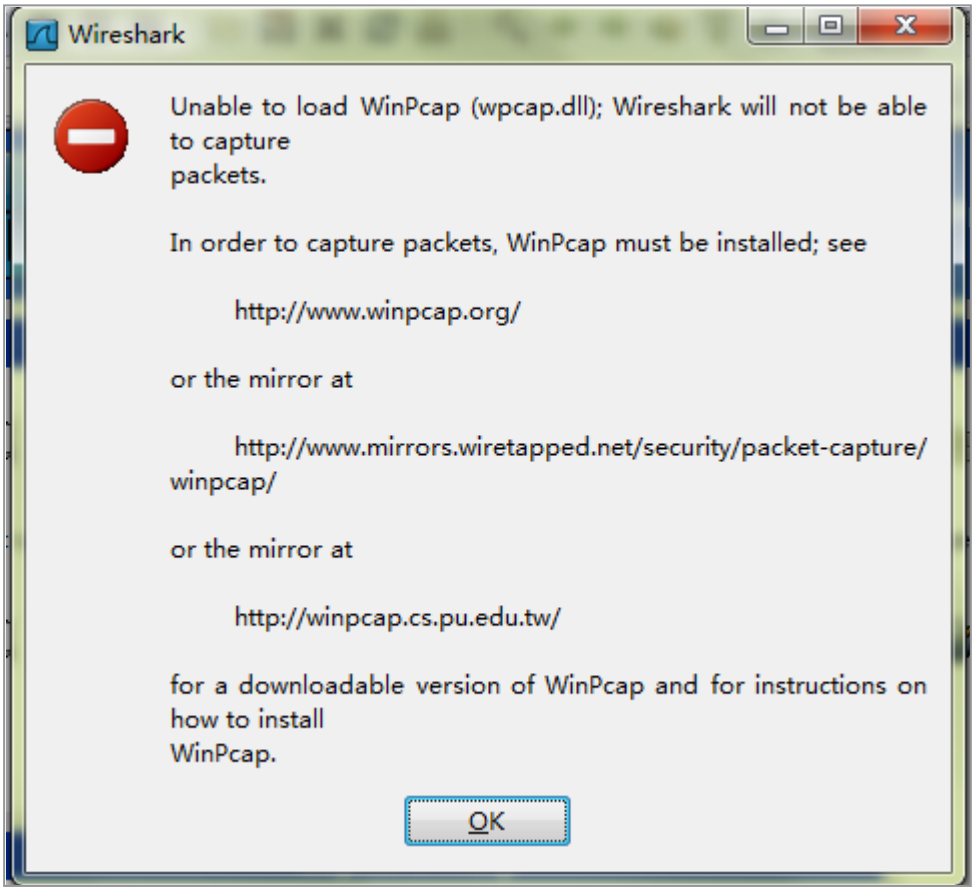


图 7-1-2

未加载 Winpcap，wireshark 将不能去捕捉；这时候我们只需要下载重新安装 Winpcap 这个软件就可以了。

除了没有安装 Winpacap 软件会造成接口列表看不到以外，还有另一个问题也会造成 wireshark 的接口失踪。如上图 7-1-1 所示那样，在 wireshark 安装好之后，我们启动 wireshark，发现依然找不到网卡。但是 wireshark 又不会像上一个问题一样弹出对话框告诉我们问题原因，如下图 7-1-3,。这时候怎么办呢？

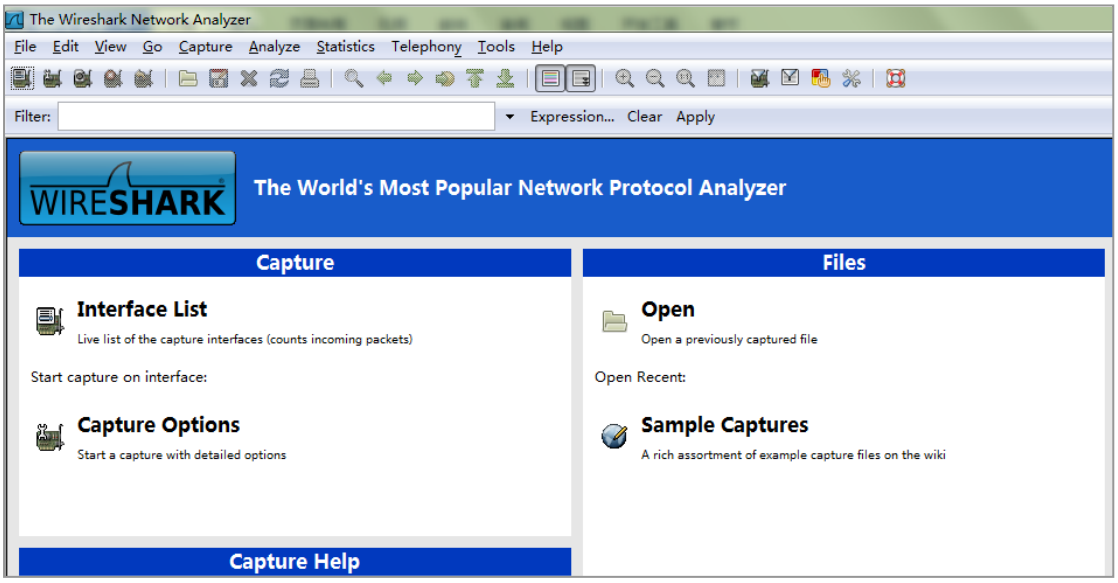


图 7-1-3

其实该问题的原因是系统的一个文件导致的，在我们的系统盘即 C 盘的 System32 文件夹下面 Driver 下有一个 Nvmini.sys 文件，我们只要重启系统，然后进入到安全模式里面（必须进入到安全模式才能看到该文件）找到该文件把它删除后照着原来的文件名字新建一个记事本文件，然后把该记事本文件的后缀名字也改成和原来的文件一模一样的。但是值得注意的一点就是在文件建立好后我们要把该文件的属性改成只读属性，这样系统就不会自动更改此文件的内容了。否则的话在下一次我们重启电脑后发现 wireshark 会依然找不到接口；更改此文件后我们重启电脑，wireshark 将会恢复正常。

对于 windows 7 以及 windows 2008 操作系统，我们是找不到 Nvmini.sys 文件的，这时候我们只需要在 CMD 命令行界面，输入“net start npf”并回车即可。XP/WIN7/WIN2008 下都是这个命令，只是 WIN7/WIN2008 需要以管理员身份运行 CMD。

7.2、wireshark 显示问题

7.2.1、数据包序列号问题

在使用 wireshark 抓取数据包时候，我们观察 TCP 的数据包发现所有的数据包的序号以及确认序号全是一样的，全是 1。如下图 7-2-1 所示那样：

图 7-2-1

如果我们想要观察数据包的序列号的话该怎么办呢？出现该问题是因为我们有一个选项要去掉，点击首选项工具栏或者点击数据包详细信息列表的 TCP 协议一行，右击——Protocol Preferences——Analyze TCP Sequence Numbers；取消该选项后再重新载入数据包列表后，我们再看 wireshark 就会显示正常了；如下图 7-2-2 所示：

图 7-2-2

7.2.2、校验和问题

启动 wireshark 抓取数据包后，我们观察数据包发现有大量的校验和错误的数据包存在，如下图 7-2-3 所示那样，但是相应的应用程序并没有没有问题。造成这个原因的问题是因为我们的网卡，网卡默认在工作时候是校验数据包的校验和的，系统将 CheckSum 的计算工作交由网卡去计算，在高速网络交换的情况下可以大大减轻 CPU 的工作负荷。

在 windows 系统中的 Checksum Offload 过程如下：

如果网卡支持，在高级选项里可以设置 Checksum Offload 是否对 Rx 或 Tx 有效，也可以设置为对两者都有效。

1、对于 Tx，设置 Checksum Offload 有效之后，Windows 的传输层将随机填充 TCP 校验和，因此在本机上抓取的数据包是 Bad CheckSum。然后，网卡会自动计算正确的校验码然后发送，因此对方收到的仍然是正确的 TCP 包。

2、对于 Rx，设置 Checksum Offload 有效之后，网卡在接收数据时，会填充一个 NDIS_TCP_IP_CHECKSUM_PACKET_INFO 结构并设置标志位；如果由于某种原因失败，则不设置标志位，由 Windows 里的 TCP/IP 协议栈来完成数据校验。

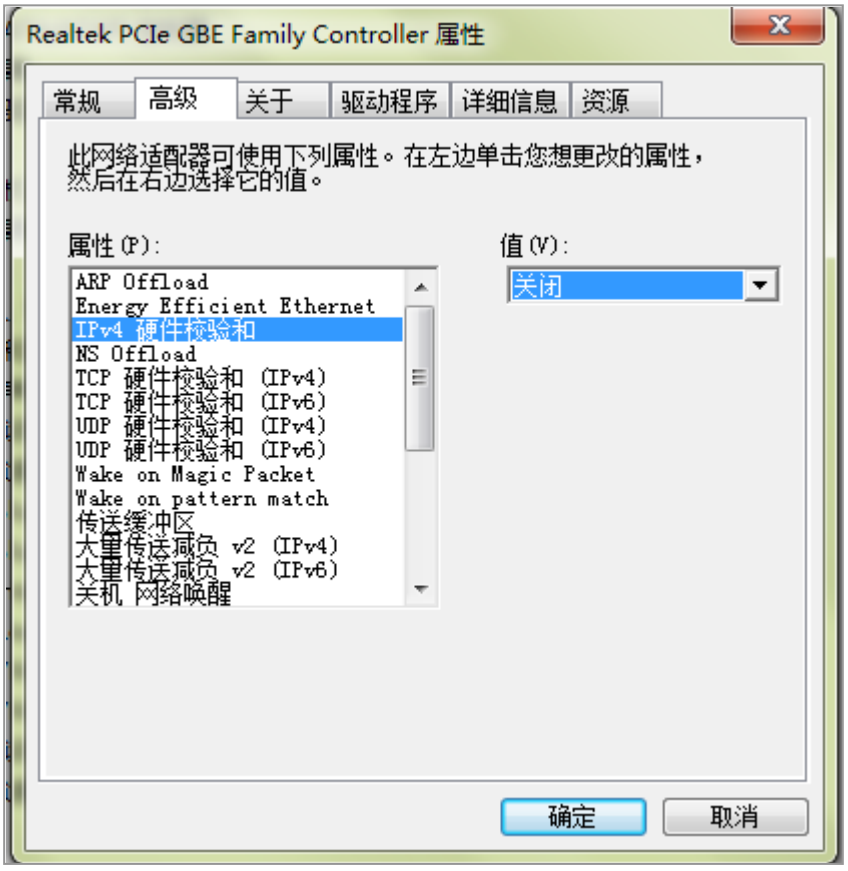


图 7-2-4

CheckSum Offload 实际上是将传输层的一部分工作交给了硬件完成，以节约系统的 CPU 资源。微软的测试表明它可以最多节约 30%的 CPU 资源。IBM 里 AIX 的文档则指出：对于 PCI 接口的千兆网卡来说还不如让 400Mhz 以上的 CPU 来计算校验和，而 PCI-X 的千兆网卡启用此项后可以达到线路速度，从而节约 CPU 资源。如果我们想要界面清爽一点的话，在这里我们把网卡的校验关闭就可以了。

至此，我们关于 Wireshark 简单的日常使用的一些功能，基本就介绍完了。但是 Wireshark 软件不止这么强大，对于更多的高级功能我相信肯定还没有接触到。在此呢由于作者能力有限就暂时介绍这么些简单的功能实现。