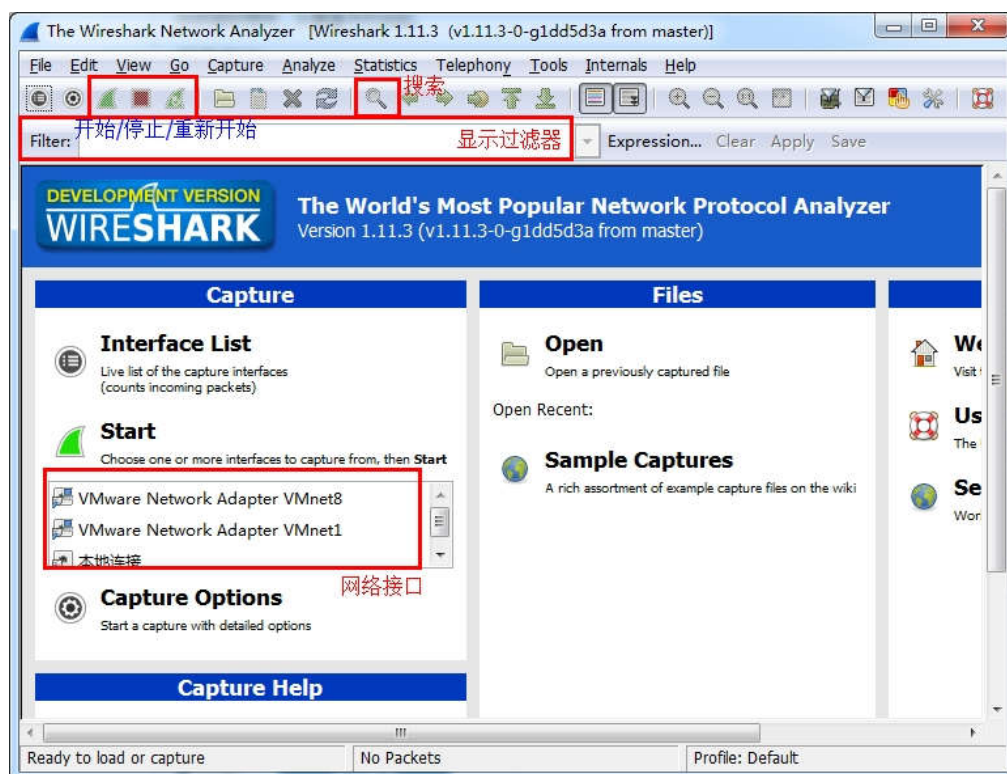


# 分析 wireshark 捕获的数据包

## 1. 抓取封包

启动 wireshark，在接口列表中选择网卡接口名，然后点击开始，则在此接口上抓包。菜单：Capture -> Options 可以配置高级属性，但是我们现在先不管。



选择一个接口，然后点击工具栏的"Start"，就可以看到捕获的封包。

Wireshark 默认会捕获该接口所有发送和接收的封包。

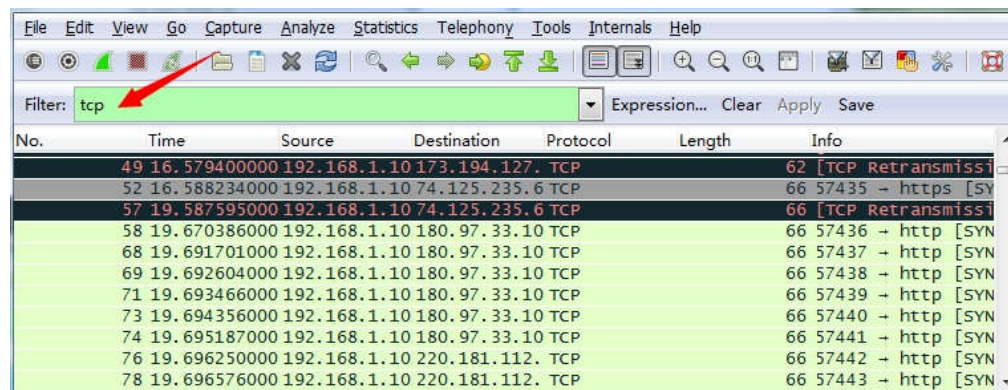
如果要停止捕获，点击工具栏的"Stop"（红色的小方块），即可。如果不停止，一直会不断的抓取数据包，可能导致我们的内存吃紧。



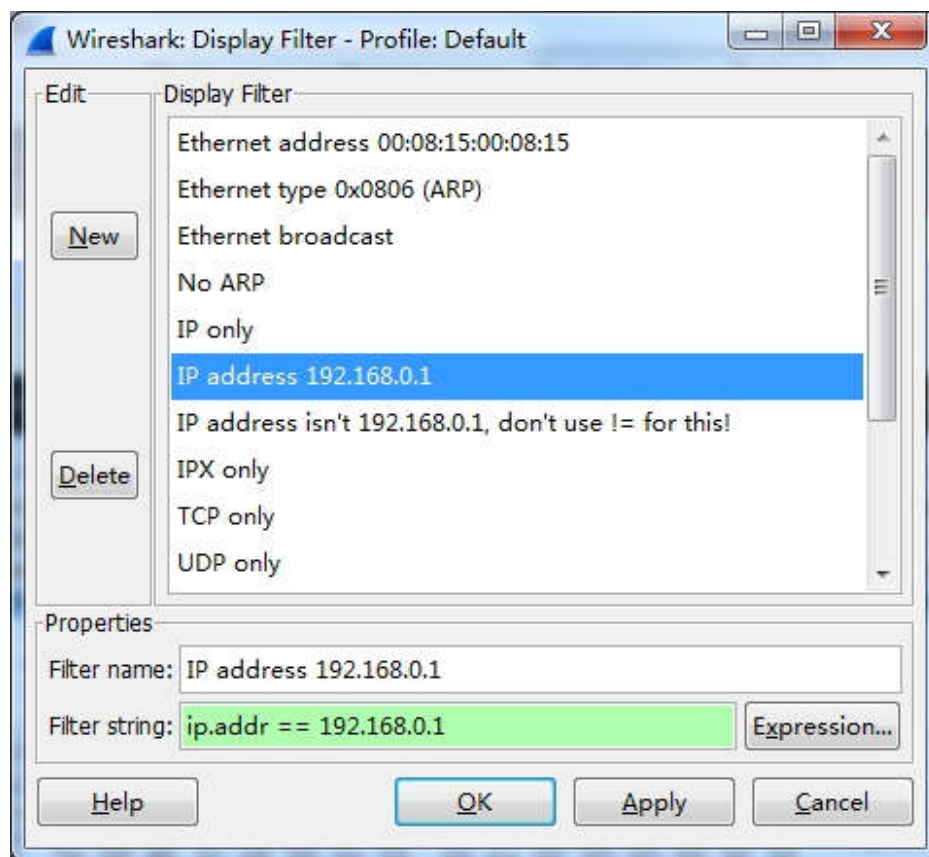
## 2.过滤封包

默认情况下, Wireshark 会捕获大量的封包, 以至于我们无法找到所需要的封包。这时就要用到 wireshark 过滤器。wireshark 提供两种类型的过滤器, 一种是捕获过滤器, 一种是显示过滤器。

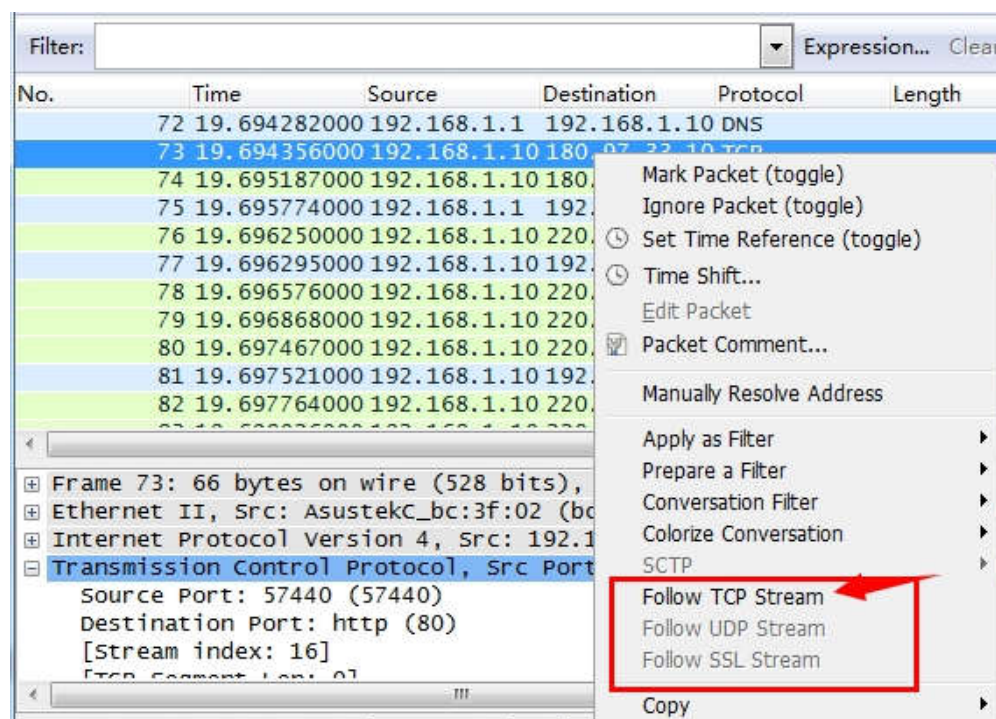
最基本的方式就是使用窗口顶端的显示过滤器, 输入过滤表达式 (或者点击 Expression 按钮, 然后选择一个过滤表达式) 并点击输入框后的"Apply"或者按下回车。



同样, 我们也可以点击菜单中的 Analyze -> Display Filters 来自定义我们的过滤条件。

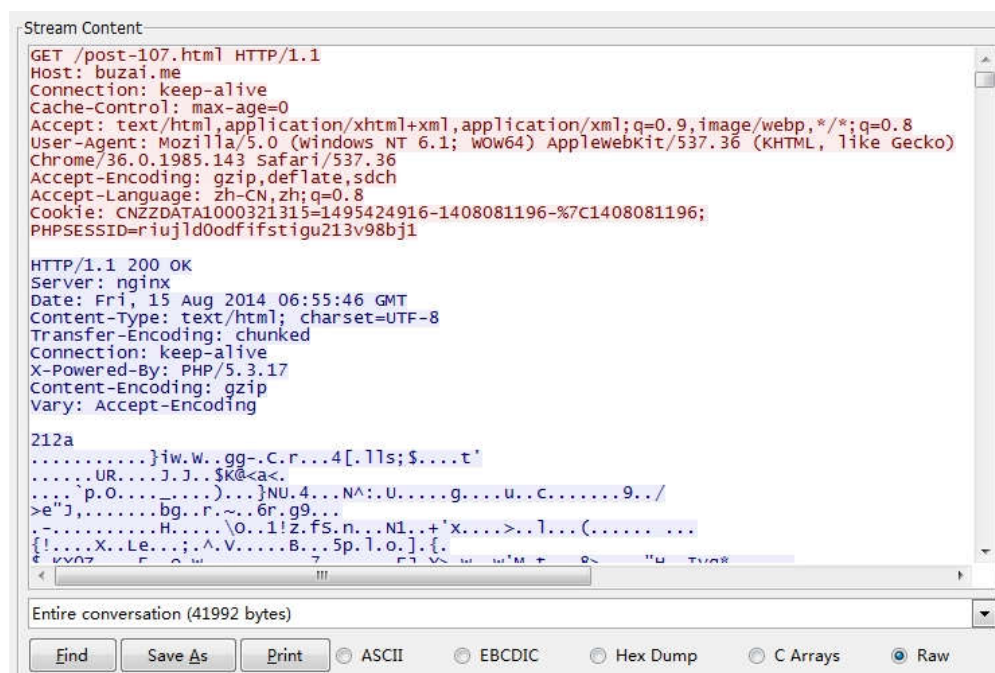


另一个很有趣的事情是，在封包列表窗口，选择一个封包然后右键，会出现 follow TCP stream 或 follow UDP stream (SSL 我是没见过) 可选，然后 follow 进去。

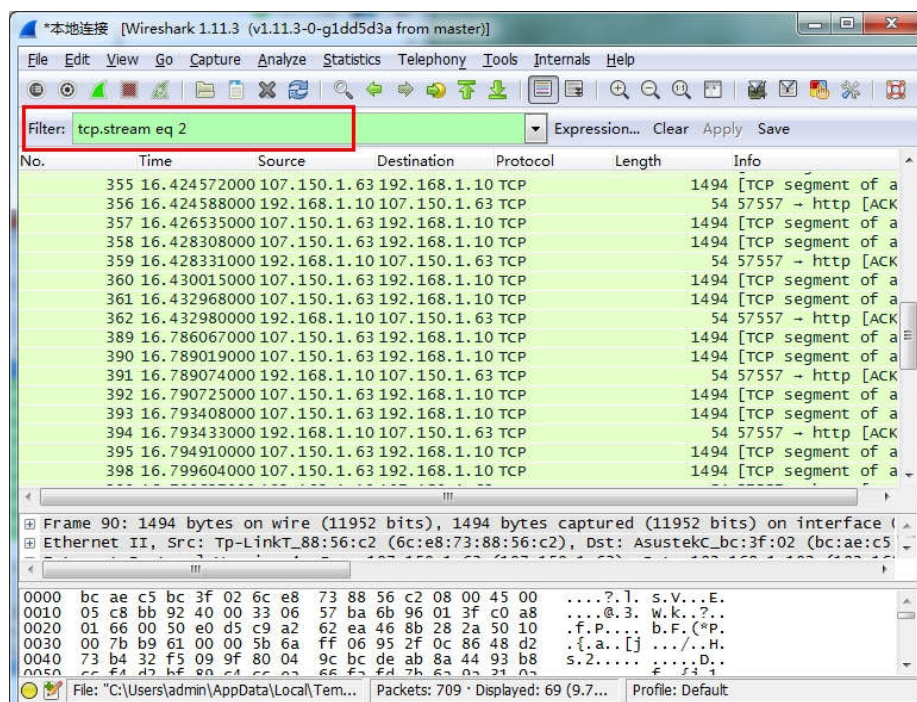




解释一下：Follow Stream 是什么意思呢？也就是会话记录，服务器和客户端之间的全部会话记录。

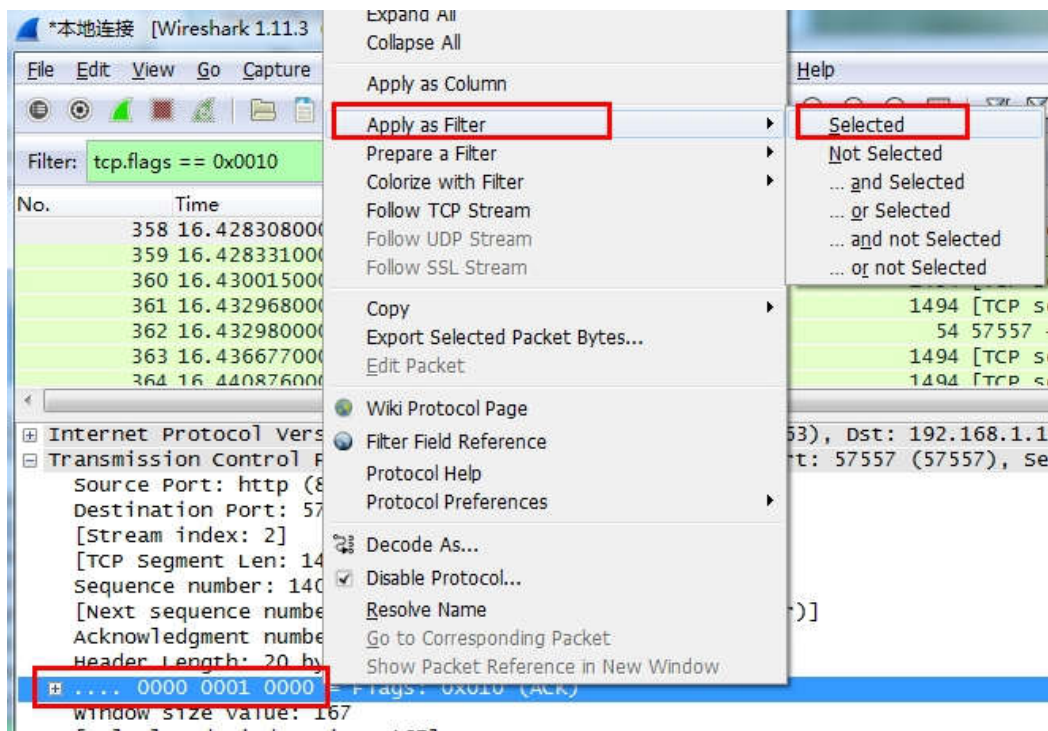


新手都会疑问，这么多请求我应该 follow 哪个封包？实际看哪个都是一样。follow 进入就是一个历史页面，进入了再慢慢找你需要的数据。你会发现显示过滤表达式被自动生成，wireshark 仅显示构成此会话的所有封包。

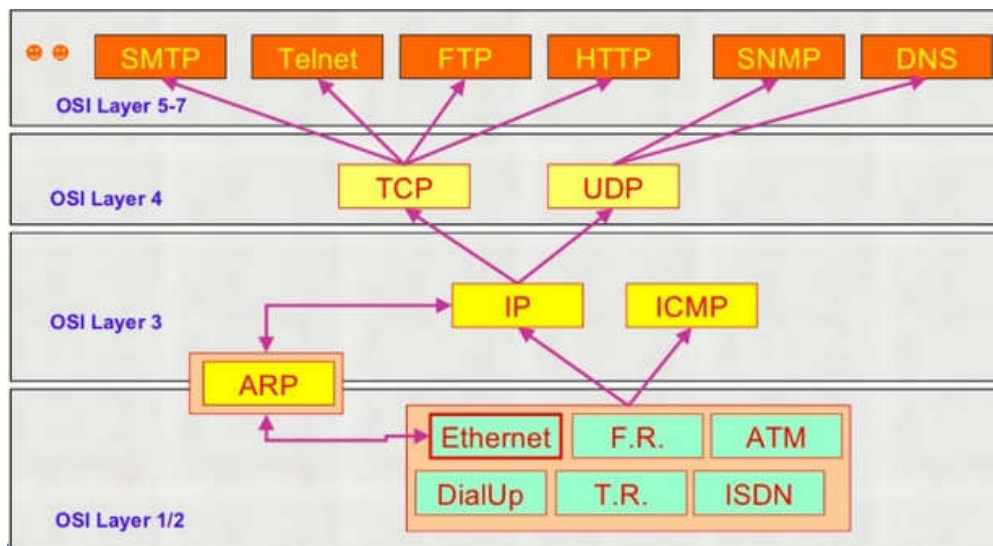


如何构造显示过滤表达式。

假设我要捕获所有以首部 Flags 标志为 ACK 的封包，怎么办呢？



TCP/IP 协议栈 (DOD 模型)



wireshark-以太网帧示例

wireshark 是严格遵循 OSI 模型规范的。我们知道以太网是实现在数据链路层和物理层的。

No.	Time	Source	Destination	Protocol	Length	Info
357	16.426535000	107.150.1.63	192.168.1.10	TCP	1494	[TCP segment of a
358	16.428308000	107.150.1.63	192.168.1.10	TCP	1494	[TCP segment of a
359	16.428331000	192.168.1.10	107.150.1.63	TCP	54	57557 → http [ACK

Frame 357: 1494 bytes on wire (11952 bits), 1494 bytes captured (11952 bits) on interface 0

Interface id: 0 (\Device\NPF\_{658085F8-5CA9-46D2-9572-67D8A60891C5})

Encapsulation type: Ethernet (1)

Arrival Time: Aug 15, 2014 14:55:50.632254000 [E][E][E][E][E][E]

[Time shift for this packet: 0.000000000 seconds]

Epoch Time: 1408085750.632254000 seconds

[Time delta from previous captured frame: 0.001947000 seconds]

[Time delta from previous displayed frame: 0.001947000 seconds]

[Time since reference or first frame: 16.426535000 seconds]

Frame Number: 357

Frame Length: 1494 bytes (11952 bits)

Capture Length: 1494 bytes (11952 bits)

[Frame is marked: False]

[Frame is ignored: False]

[Protocols in frame: eth:ethertype:ip:tcp]

[Coloring Rule Name: HTTP]

[Coloring Rule String: http || tcp.port == 80 || http2]

Ethernet II, Src: Tp-LinkT\_88:56:c2 (6c:e8:73:88:56:c2), Dst: AsustekC\_bc:3f:02 (bc:ae:c5:bc:3f:02)

Destination: AsustekC\_bc:3f:02 (bc:ae:c5:bc:3f:02)

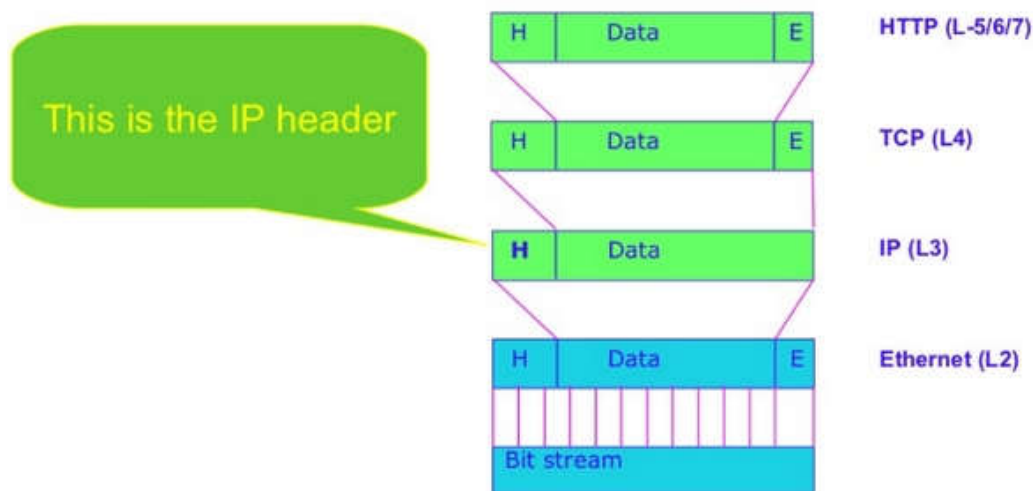
Source: Tp-LinkT\_88:56:c2 (6c:e8:73:88:56:c2)

Type: IP (0x0800)

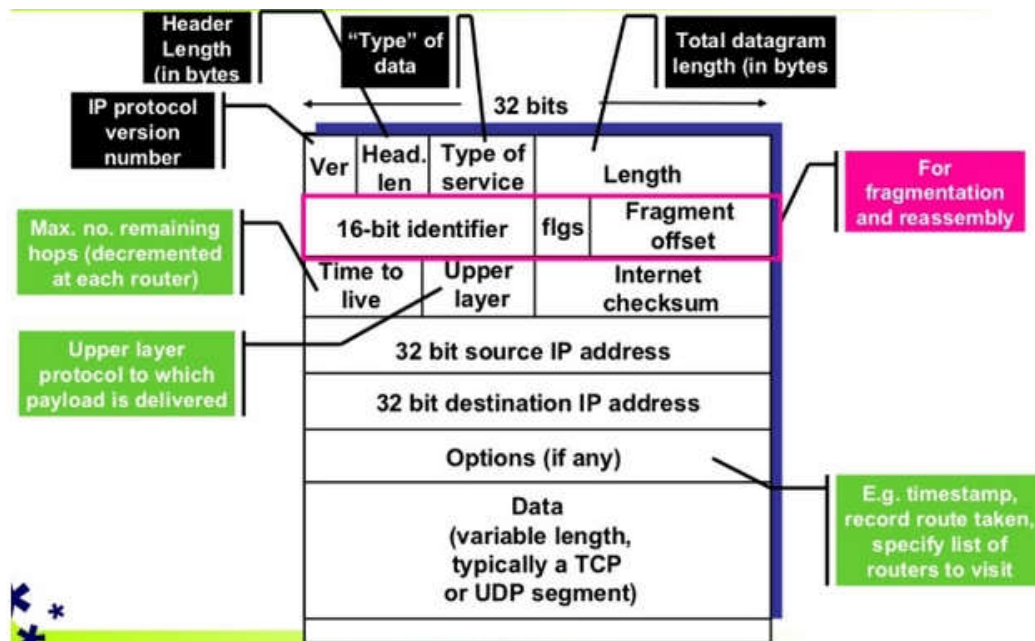
Internet Protocol Version 4, Src: 107.150.1.63 (107.150.1.63), Dst: 192.168.1.102 (192.168.1.102)

Transmission Control Protocol, Src Port: http (80), Dst Port: 57557 (57557), Seq: 12600, Ack

## IP 数据报







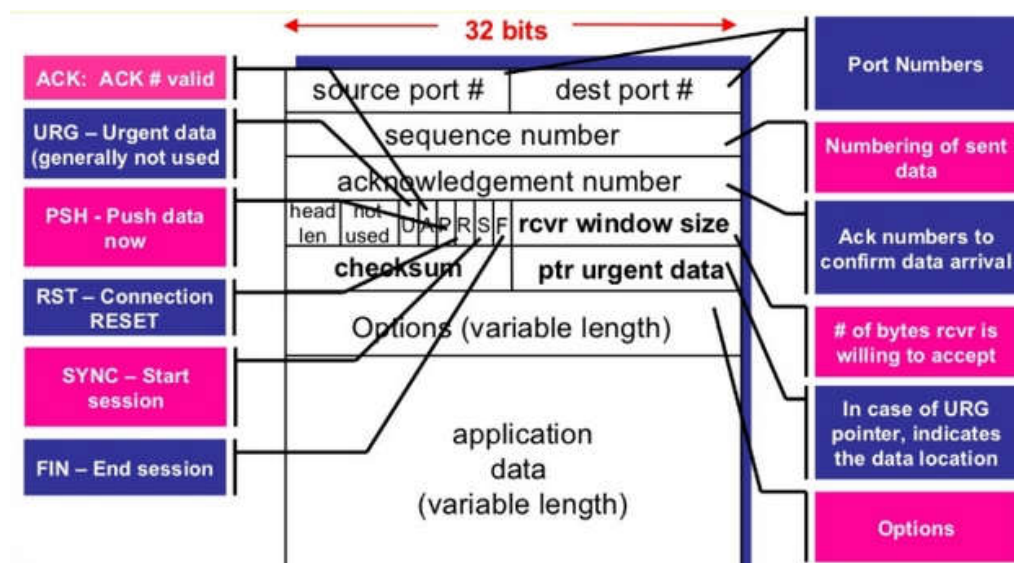
wireshark-IP 数据报示例

No.	Time	Source	Destination	Protocol	Length	Info
674	84.782177000	192.168.1.10	106.120.151.1	TCP	759	TCP segment of a
675	84.782225000	192.168.1.10	106.120.151.1	HTTP	601	POST /q HTTP/1.1
676	84.848458000	106.120.151.1	192.168.1.10	TCP	60	http → 57592 [ACK

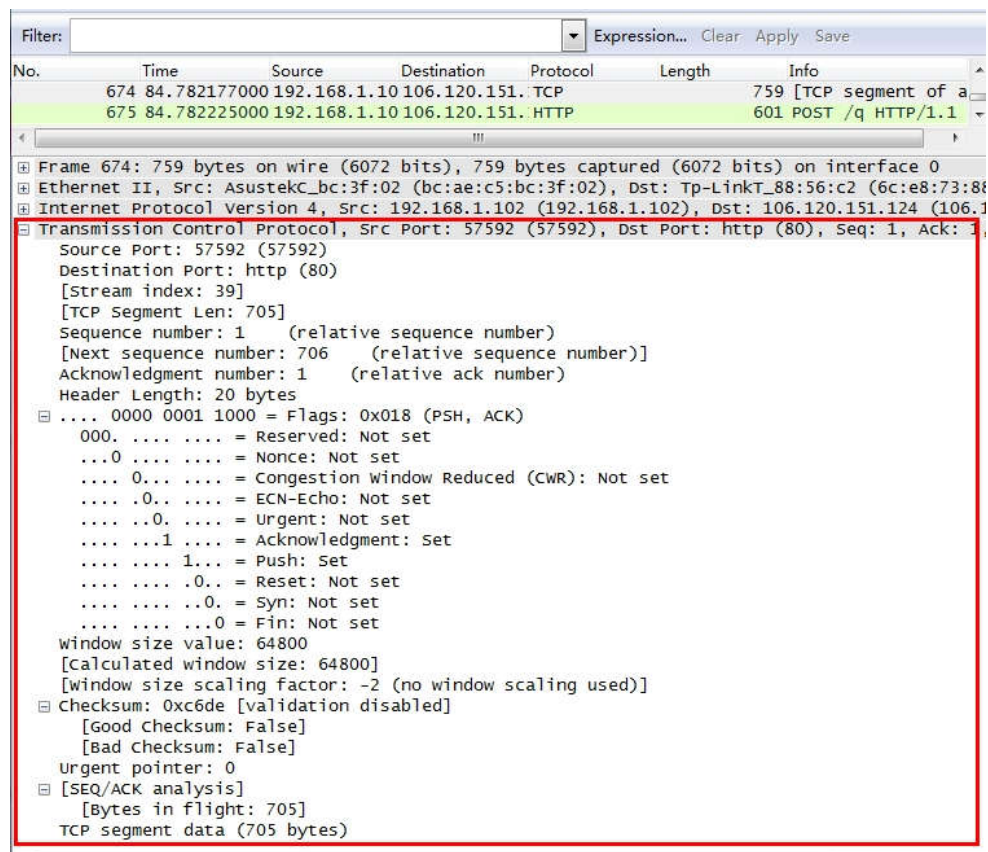
  

<p>Frame 674: 759 bytes on wire (6072 bits), 759 bytes captured (6072 bits) on interface 0</p> <p>Ethernet II, Src: AsustekC_bc:3f:02 (bc:ae:c5:bc:3f:02), Dst: Tp-LinkT_88:56:c2 (6c:e8:73:88)</p> <p>Internet Protocol Version 4, Src: 192.168.1.102 (192.168.1.102), Dst: 106.120.151.124 (106.120.151.124)</p> <p>Version: 4</p> <p>Header Length: 20 bytes</p> <p>Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00: Not-ECT (Not ECN-Capable Transport))</p> <p>0000 00.. = Differentiated Services Codepoint: Default (0x00)</p> <p>.... ..00 = Explicit Congestion Notification: Not-ECT (Not ECN-Capable Transport) (0x00)</p> <p>Total Length: 745</p> <p>Identification: 0x0d86 (3462)</p> <p>Flags: 0x02 (Don't Fragment)</p> <p>0... .... = Reserved bit: Not set</p> <p>.1.. .... = Don't fragment: Set</p> <p>..0. .... = More fragments: Not set</p> <p>Fragment offset: 0</p> <p>Time to live: 64</p> <p>Protocol: TCP (6)</p> <p>Header checksum: 0x0000 [validation disabled]</p> <p>[Good: False]</p> <p>[Bad: False]</p> <p>Source: 192.168.1.102 (192.168.1.102)</p> <p>Destination: 106.120.151.124 (106.120.151.124)</p> <p>[Source GeoIP: Unknown]</p> <p>[Destination GeoIP: Unknown]</p> <p>Transmission Control Protocol, Src Port: 57592 (57592), Dst Port: http (80), Seq: 1, Ack: 1,</p>
--

TCP 报文段

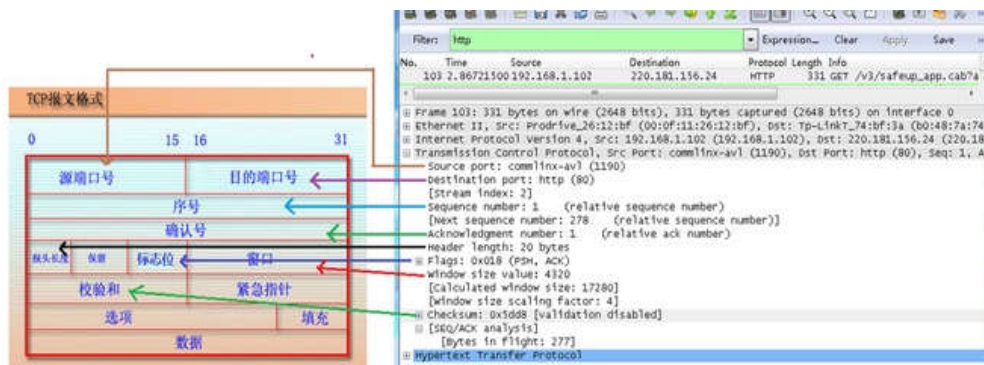


wireshark-TCP 报文段示例



我们可以很清晰的看到, wireshark 中的首部数据与 OSI 模型中各协议的首部格式是相对应的。所以, 利用 wireshark 来学习网络, 那是非常直观有效的。



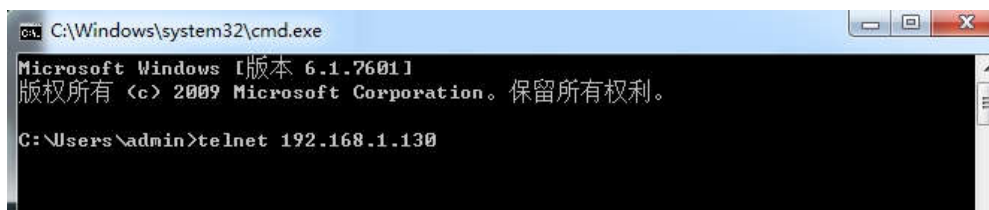


## 3.wireshark 实战演练

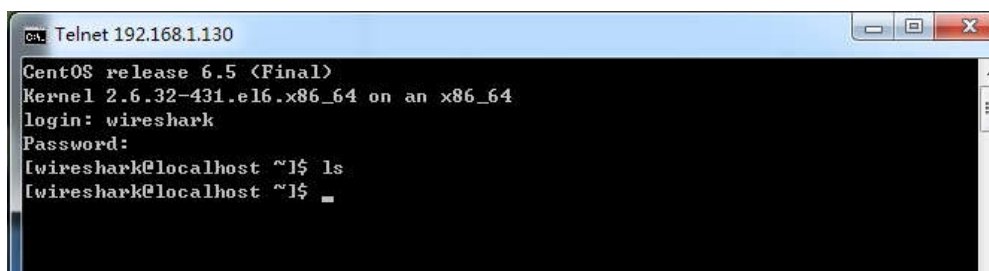
### 3.1 实验一：网络中明文传输的危险性

通过明文传输的 protocol 和工具相当多，典型的就是 telnet,ftp,http。我们拿 telnet 做这次实验。假设我以 telnet 方式登录到我的 linux 服务器，然后通过 wireshark 抓包，以抓取账号和密码信息。

1、首先启动 wireshark，并处于 Capture 状态。然后通过 telnet 远程登录我们的 linux 服务器。

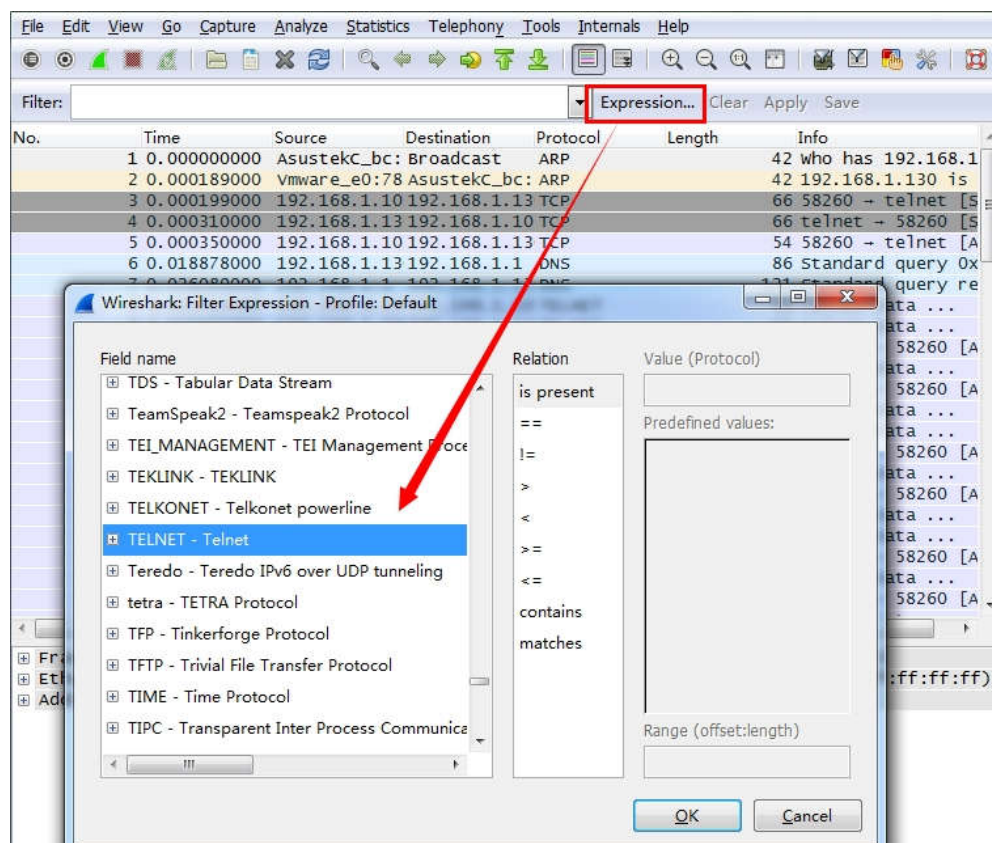


进入登录界面后，输入账号和密码登入系统。

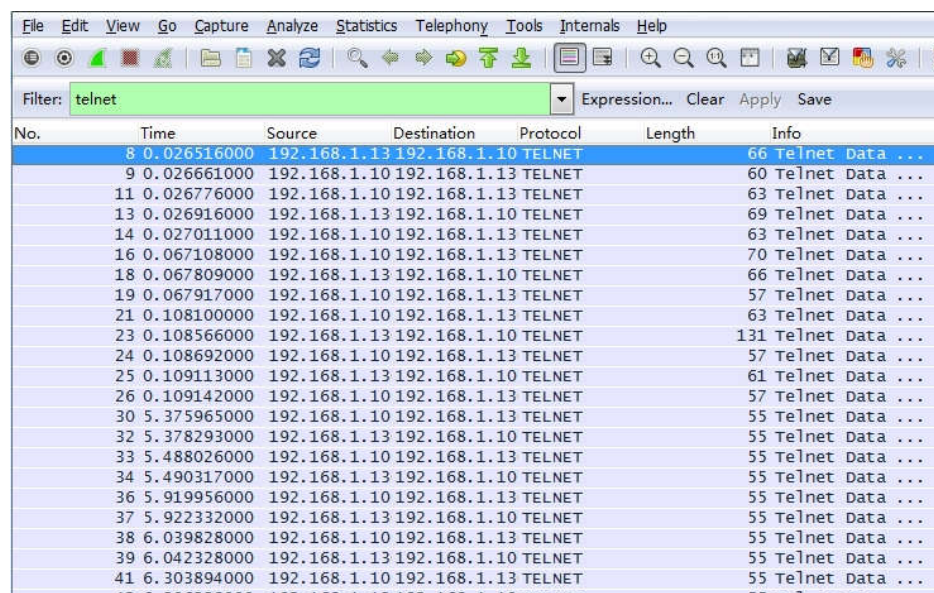


2、接下来停止 wireshark 的截取封包的操作,执行快捷方式的"Stop"即可。

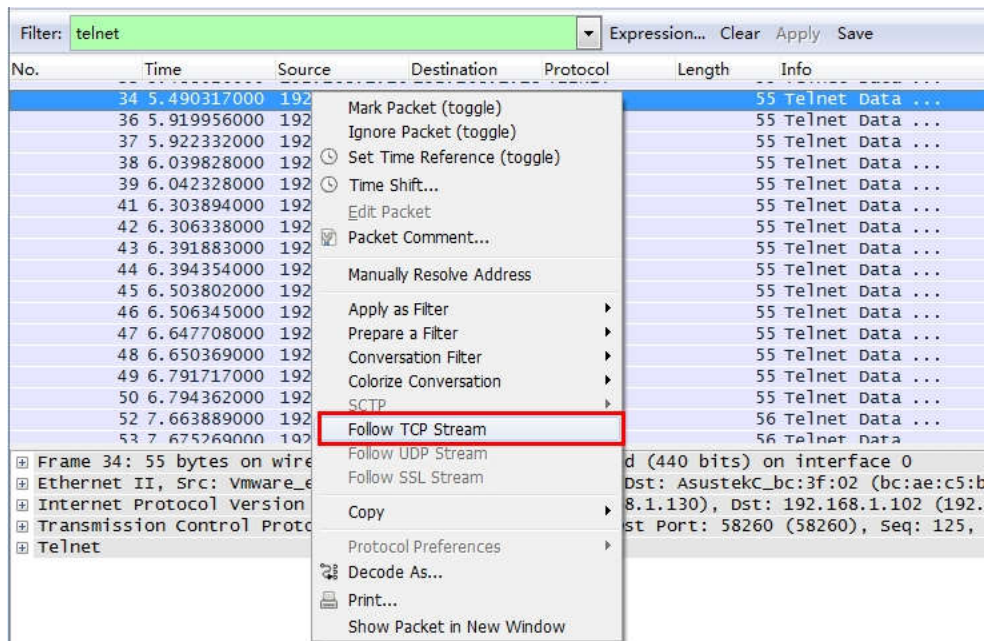
不过，捕获的信息非常多，这个时候可以利用 Display Filter 功能，过滤显示的内容，如下图所示，点击 Expression,然后选择过滤表达式。这里，我们选择 TELNET 即可。



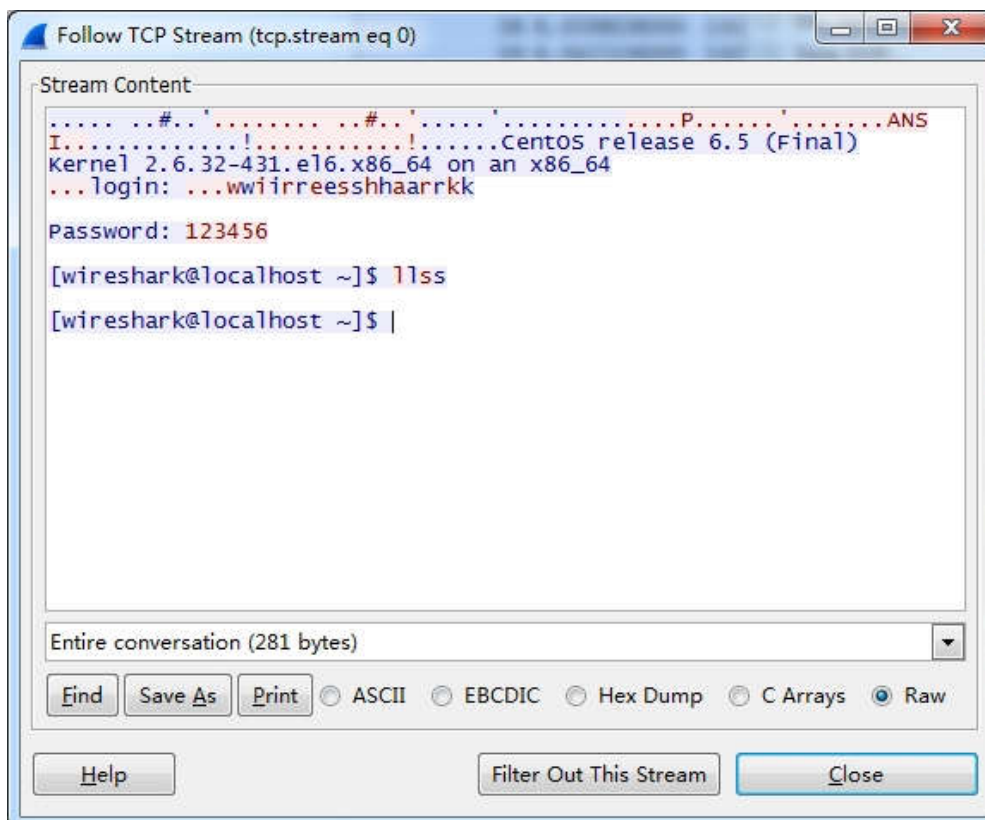
表达式确定之后，选择"Apply",就可以过滤出只包含 TELNET 的封包



来，我们查看一下整个 telnet 会话的所有记录， wireshark 可以记录会话记录（就像我们聊 QQ 时，"QQ 聊天记录"一样），任意找到一个 telnet 封包，右键找到"Follow TCP Stream"， wireshark 就会返回整个会话记录。

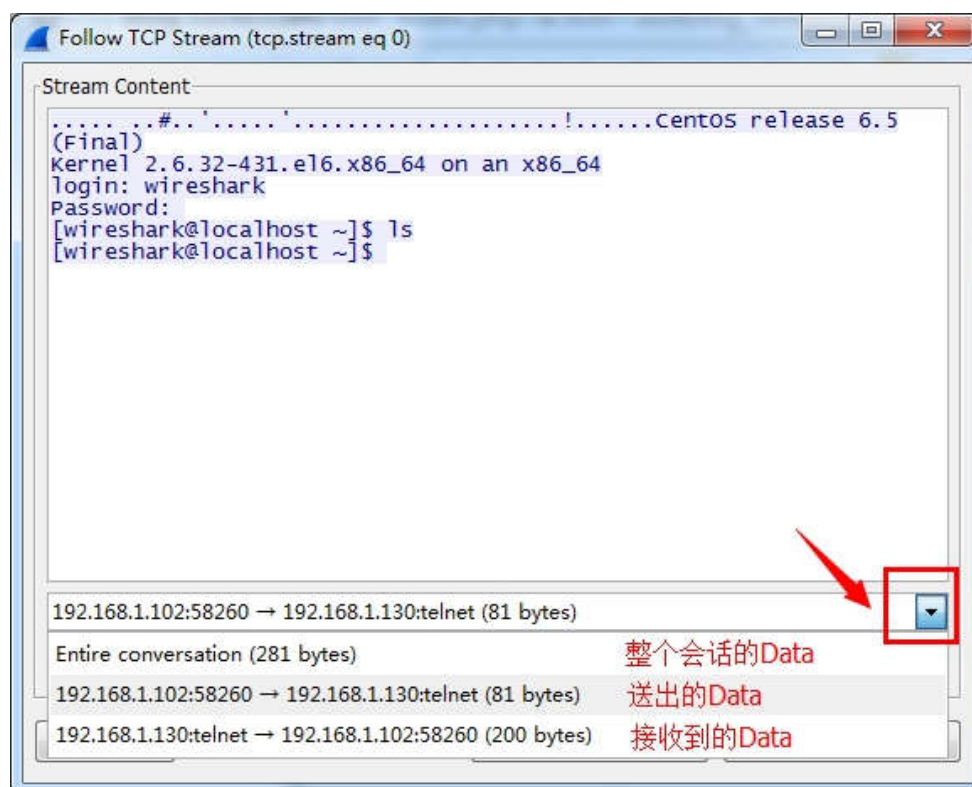


OK， 我们看到以下这些数据信息，红色的部分是我们发送出去的 DATA，蓝色的部分是我们接收到的 DATA。



为了更准确的看清楚，我们再次仅筛选出我们发送出去的 DATA。或者仅接收到的 DATA。





从这里，我们可以确切的抓到账号和密码信息。

login:wireshark Password:123456，除了这些，我们还可以更进一步知道别人在看什么网站，或是私人文件，隐私将毫无保障。

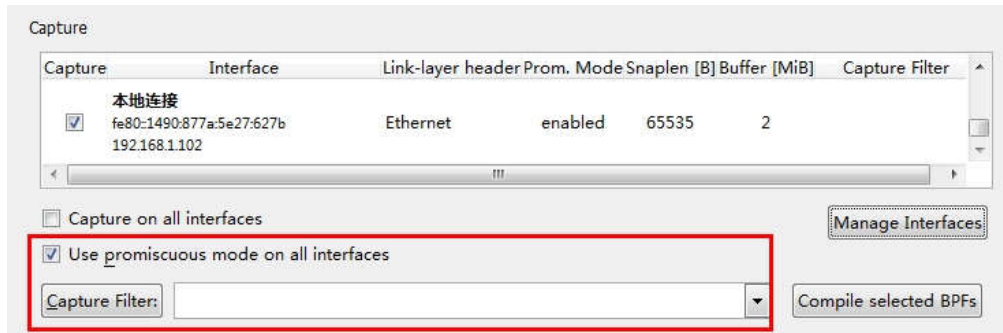
(注：为了避免这些情况，防止有心人监测到重要信息，可以使用SSH,SSL,TSL,HTTPS 等加密协议对重要数据进行加密，然后再到网络上传输，如果被人截取下来，看到的内容也是被加密的。)

### 3.2 实验二：HTTP Protocol

目的：通过观察 HTTP 协议，练习如何过滤出自己需要的数据，并能够清楚的知道 TCP/IP 实际的运作方式。

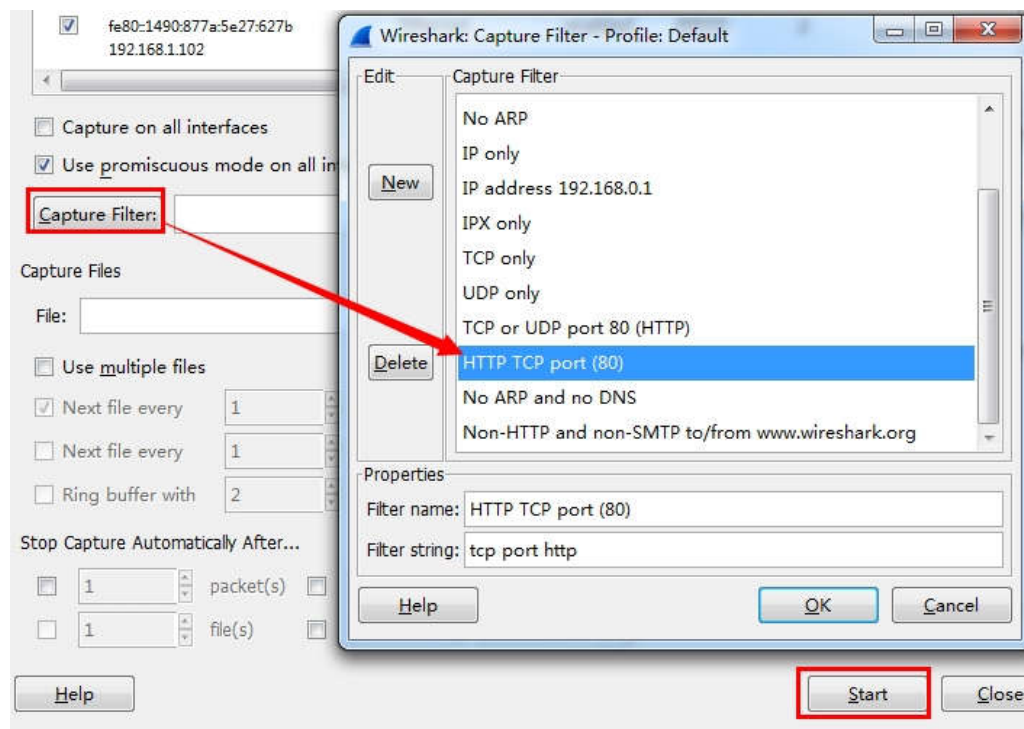
这里，我们采用"(Capture Filter)捕获过滤"，菜单栏 Capture -> Options

跟我们相关设定的有如下部分，Use promiscuous mode on all interfaces, Capture Filter, 其实我们需要设定的仅仅是 Captrue Filter



Capture Filter 使用 libpcap filter 语言, 详细的语法可以参考 tcpdump 的 man 页面 ( [http://www.tcpdump.org/tcpdump\\_man.html](http://www.tcpdump.org/tcpdump_man.html) )

我们这里不需要使用这么复杂的过滤表达式, 直接用 wireshark 已经建立好的常用功能 HTTP TCP port(80)即可。首先点击"Capture Filter"按钮, 然后选择 HTTP TCP port(80), OK 之后选择"Start"开始捕获封包。



这里以浏览器访问 [www.baidu.com](http://www.baidu.com) 为例, 来看看访问一个网页会发生什么?

回到 wireshark, 看到已经抓取了很多封包。OK, 前面我们已经交代了 TCP/IP 协议栈, 这里就不再累述。

看下面这幅图中，红色的框框，这就是 TCP 在做三次握手，建立连接。

The image shows a Wireshark packet capture window. A red box highlights the first three packets, which are part of a TCP three-way handshake between 192.168.1.10 and 180.97.33.10. The first packet is a SYN packet from the client to the server. The second packet is a SYN-ACK packet from the server to the client. The third packet is an ACK packet from the client to the server. The fourth packet is an HTTP GET request from the client to the server. The rest of the packets in the list are TCP segments of a reassembled PDU.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	192.168.1.10	180.97.33.10	TCP	66	61948 → http [SYN] Seq=0 win=8192 Len=0
2	0.034244000	180.97.33.10	192.168.1.10	TCP	66	http → 61948 [SYN, ACK] Seq=0 Ack=1 Win=0 Len=0
3	0.034311000	192.168.1.10	180.97.33.10	TCP	54	61948 → http [ACK] Seq=1 Ack=1 Win=16384 Len=0
4	0.034488000	192.168.1.10	180.97.33.10	HTTP	473	GET / HTTP/1.1
5	0.075115000	180.97.33.10	192.168.1.10	TCP	60	http → 61948 [ACK] Seq=1 Ack=420 Win=2048 Len=0
6	0.077236000	180.97.33.10	192.168.1.10	TCP	516	[TCP segment of a reassembled PDU]
7	0.079424000	180.97.33.10	192.168.1.10	TCP	1494	[TCP segment of a reassembled PDU]
8	0.079474000	192.168.1.10	180.97.33.10	TCP	54	61948 → http [ACK] Seq=420 Ack=1903 Win=0 Len=0
9	0.080913000	180.97.33.10	192.168.1.10	TCP	1494	[TCP segment of a reassembled PDU]
10	0.083182000	180.97.33.10	192.168.1.10	TCP	1494	[TCP segment of a reassembled PDU]
11	0.083221000	192.168.1.10	180.97.33.10	TCP	54	61948 → http [ACK] Seq=420 Ack=4783 Win=0 Len=0
12	0.084859000	180.97.33.10	192.168.1.10	TCP	1494	[TCP segment of a reassembled PDU]
13	0.087546000	180.97.33.10	192.168.1.10	TCP	1494	[TCP segment of a reassembled PDU]
14	0.087592000	192.168.1.10	180.97.33.10	TCP	54	61948 → http [ACK] Seq=420 Ack=7663 Win=0 Len=0
15	0.089055000	180.97.33.10	192.168.1.10	TCP	1494	[TCP segment of a reassembled PDU]
16	0.091768000	180.97.33.10	192.168.1.10	TCP	1494	[TCP segment of a reassembled PDU]
17	0.091792000	192.168.1.10	180.97.33.10	TCP	54	61948 → http [ACK] Seq=420 Ack=10543 Win=0 Len=0

接下来这一行，就是 Client 向 server 作出 request。

我们可以在封包详细信息列表框中，针对于每一层查看，更清楚的了解每一层的运作模式，同理也可以使用 Follow TCP Stream

