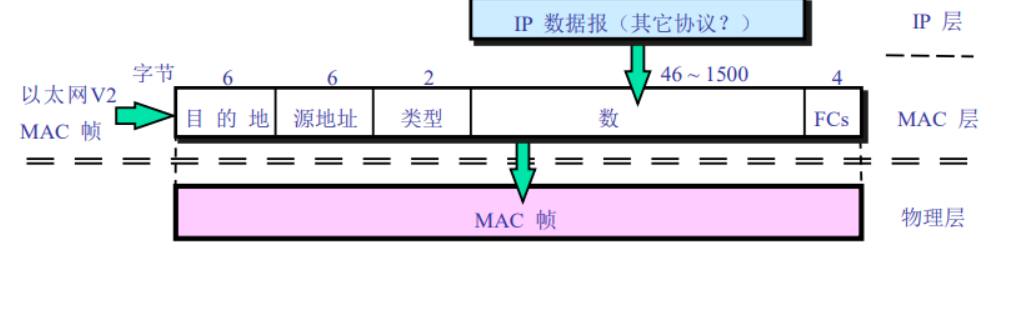
2152118 史君宝 计算机网络 第三次课程报告

**以太网MAC帧：**

以太网数据帧简称以太帧，起始部分由前同步码和帧开始定界符组成，后面紧跟着一个以太网报头，以 MAC 地址说明目的地址和源地址。以太帧的中部是该帧负载的包含其他协议报头的数据包，最常见的如 IP 协议。另外以太帧由一个 32 位冗余校验码结尾，用于检验数据传输是否出现损坏。

以太网MAC帧的地址格式:



前同步码：用来使接收端的网络适配器在接收 MAC 帧时能够迅速调整时钟频率，使它和发送端的频率相同。前同步码为7个字节，其值为1和0交替，即10101010...

帧开始定界符：帧的起始符，为1个字节。其值为：10101011，标志着一个帧的开始，告诉接收端适配器：帧要来了，准备接收。

目的地址：接收帧的网络适配器的物理地址（MAC 地址），为 6 个字节（48 比特）。作用是当网卡接收到一个数据帧时，首先会检查该帧的目的地址，是否与当前适配器的物理地址相同，如果相同，就会进一步处理；如果不同，则直接丢弃。

源地址：发送帧的网络适配器的物理地址（MAC 地址），为 6 个字节（48 比特）。

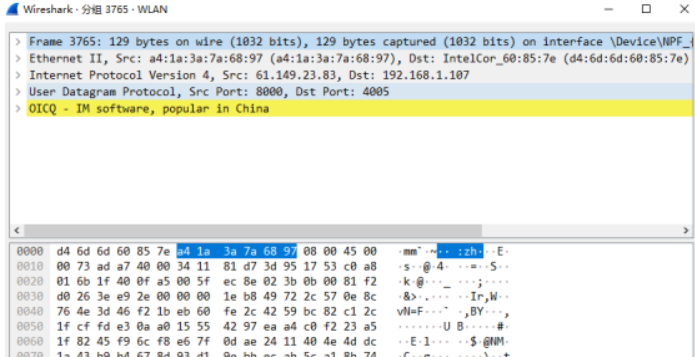
类型：上层协议的类型，占2个字节。由于上层协议众多，所以在处理数据的时候必须设置该字段，标识数据交付哪个协议处理。例如，字段为 0x0800 时，表示将数据交付给 网络层的IP 协议。

数据：也称为效载荷，表示交付给上层的数据。以太网帧数据长度最小为 46 字节，最大为 1500 字节，最大值也叫最大传输单元（MTU）。如果不足 46 字节时，会填充到最小长度。

帧检验序列 FCS：检测该帧是否出现差错，占 4 个字节（32 比特）。发送方计算帧的循环冗余码校验（CRC）值，把这个值写到帧里。接收方计算机重新计算 CRC，与 FCS 字段的值进行比较。如果两个值不相同，则表示传输过程中发生了数据丢失或改变。这时，就需要重新传输这一帧。

因此，以上可知以太网数据帧的整体大小在 64～1518 字节之间（不含前导字段7字节和帧起始符1字节）。

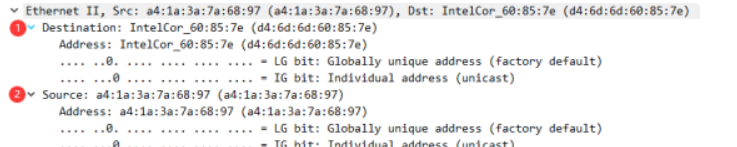
我们采用wireshark进行抓包分析，找到自己ip对应的网卡抓取一些数据包，这里以QQ程序传输的数据包为例，截图如下：



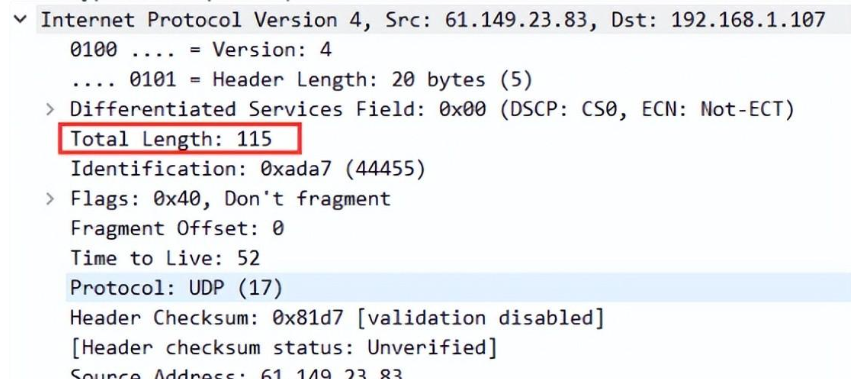
对上面进行说明：

Frame 3765: 129 bytes on wire (1032 bits), 129 bytes captured (1032 bits) on interface \Device\, id 0： 这句话的意思就是说数据帧号码3765，捕获了129个字节，也就是1032位（一字节等于八位），在interface 0上面，也就是在网卡0上面（一个机器可能有多个网卡）。

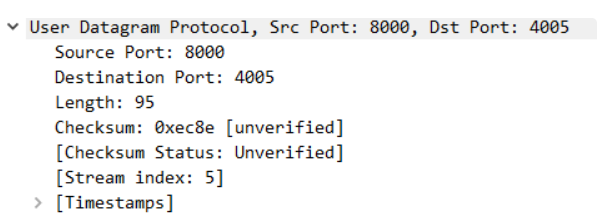
Ethernet II：以太帧的包头， 可以清楚地看到里面包含三个信息：Destination, Source,Type依次对应前面介绍以太帧数据结构的目的地址、源地址、类型，而type类型值为0x0800表示IPv4，也就是说它是一个IP包。



Internet Protocol Version 4：第三个就是IP数据包。关于IP数据包格式我们以后在进行介绍，这里可以看到，这个IP数据包的总长度为115字节。

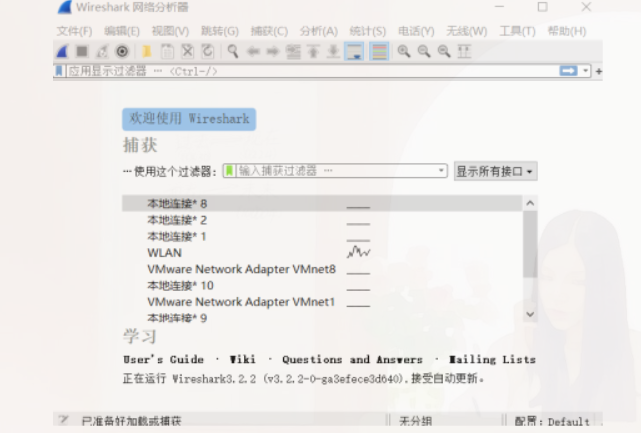


User Datagram Protocol：UDP数据包。这个IP数据包高层传输层协议为UDP，也就是说QQ这个应用程序选择UDP作为通信协议，而不是TCP。关于传输层协议也以后再进行介绍。



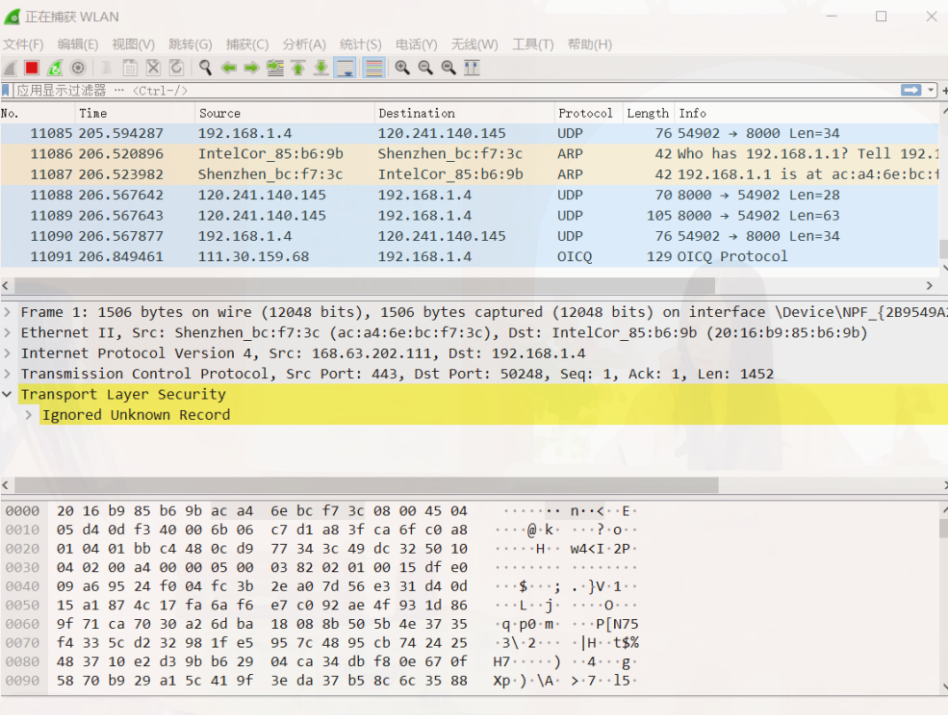
那么，最后我们可以得出捕获到的这个以太帧的总长度为129字节，IP数据包的total length是115字节，加上以太帧的包头（6+6+2=14）14字节,115+14就等于129字节，然而，根据前面介绍以太帧尾部还有四个字节的FCS校验和。后面的FCS四个字节哪里去了呢？是不是我们计算错误了？显然我们的计算是正确的，这是因为数据包经过网络设备，如路由器、交换机等硬件已经把以太帧的校验做过了，它返回给操作系统的只有前面的部分，FCS没有返回给操作系统，所以wireshark也抓不到后面校验的四个字节，当然，校验错误的以太帧也不会被wireshark所捕获，所以，我们看到的以太帧都是通多校验的正确的以太帧。

**无线局域网MAC帧：**

****

打开软件后可以直接在这里选择相应的网卡，也可以在菜单中的捕获->选项，在弹出窗口中选择对应网络接口，这里我选择 WLAN。

选择相应接口后就可以点击开始，开始捕获数据：



顶部是Packet List Pane（数据包列表窗格），显示了捕获的每个数据包的摘要信息。单击此窗格中的数据包可控制另外两个窗格中显示信息

中间是Packet Details Pane（数据包详细信息窗格），更加详细的显示了了“数据包列表”窗格中所选的数据包。

底部是Packet Bytes Pane（数据包字节窗格），显示了“数据包列表”窗格中所选数据包的实际数据（以十六进制形式表示实际的二进制），并突出显示了在“数据包详细信息”窗格中所选的字段。

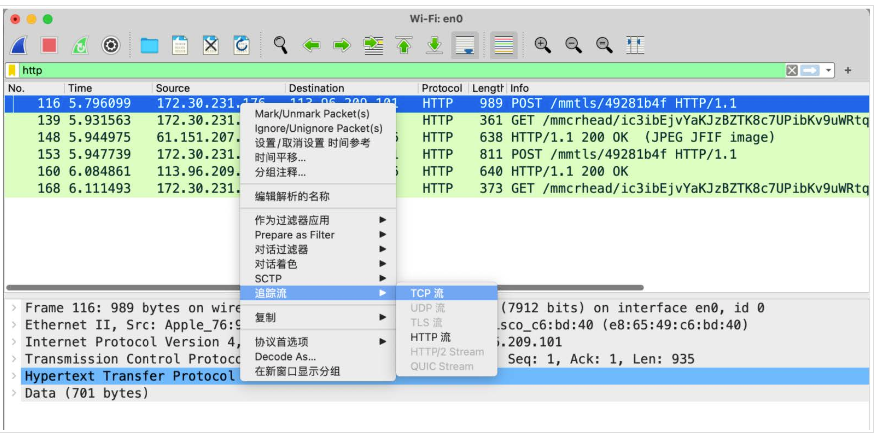
打开cmd窗口，ping网关，然后去软件中抓取相应的包：

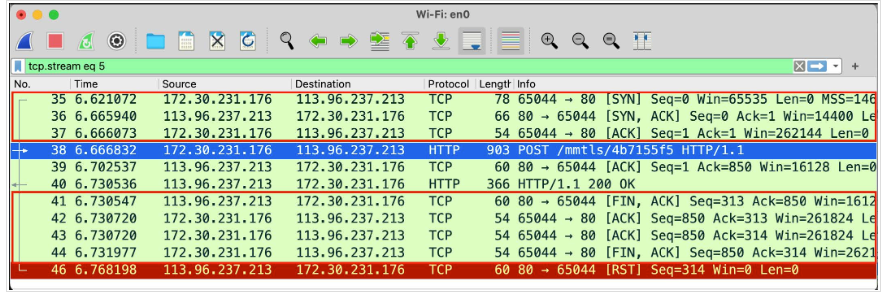


**TCP协议：**

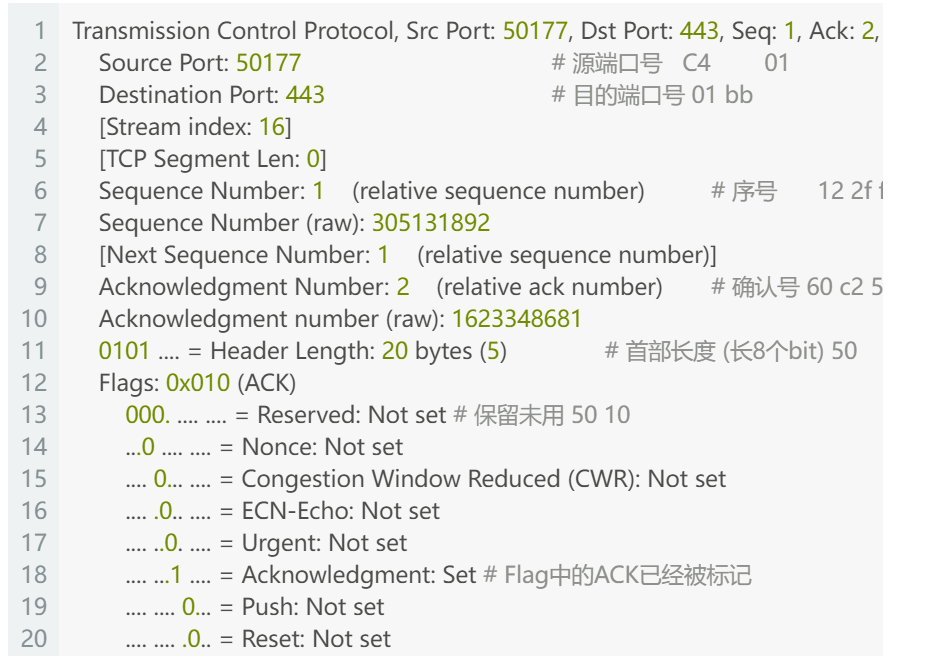
TCP 被称为是面向连接的(connection oriented), 这是因为在一个应用进程可以开始向另一个应用进程发送数据之前，这两个进程必须互相先“握手”, 即它们必须相互发送某些预备报文段，以建立确保数据传输的参数。

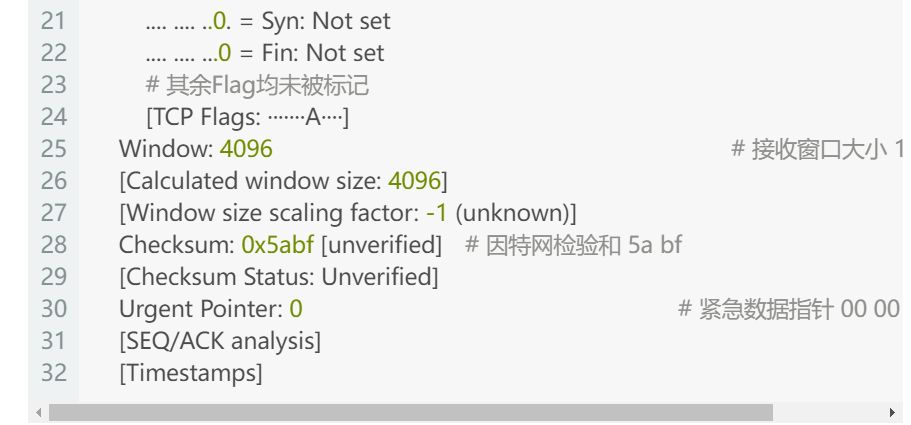
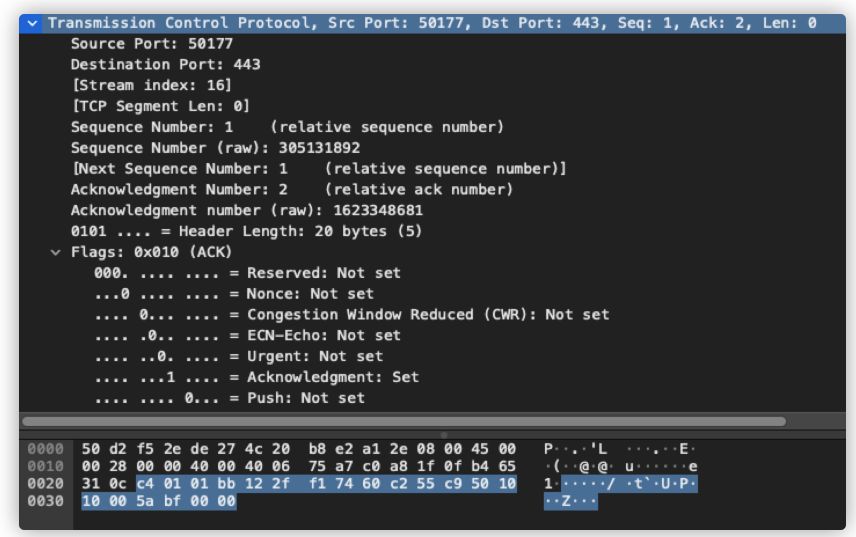




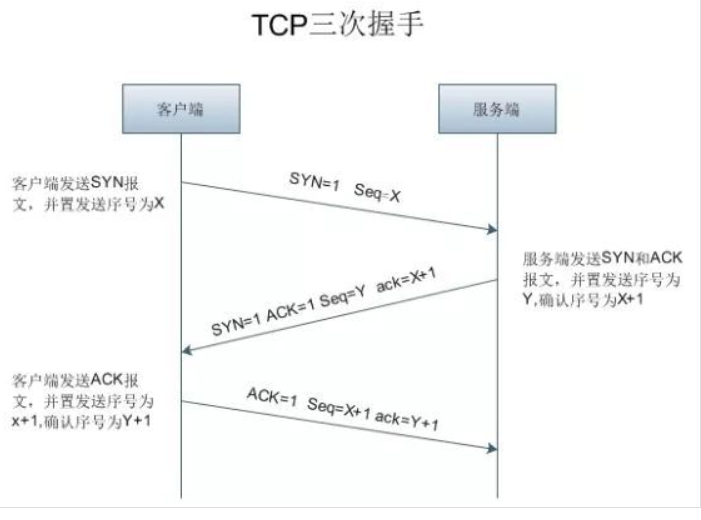


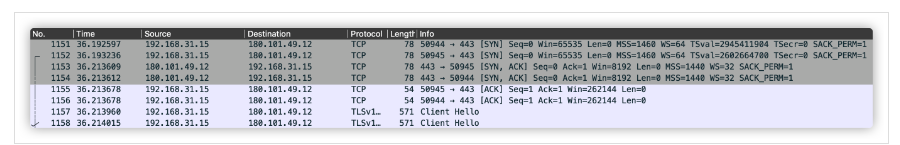
利用Wireshark 抓取一个TCP抓取数据包， 查看其具体数据结构和实际的数据：



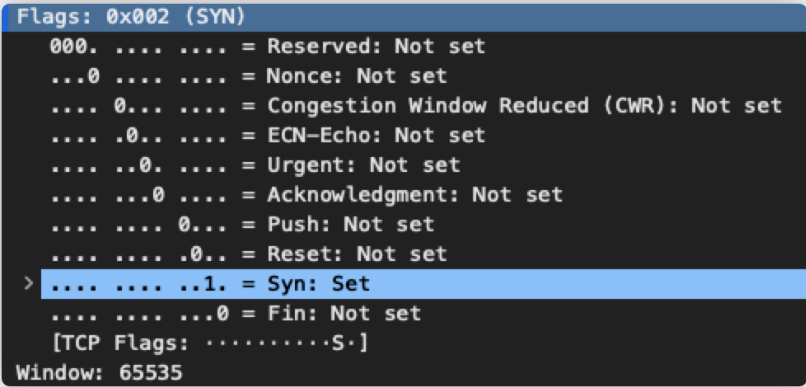
下面我们说一下TCP的三次握手：



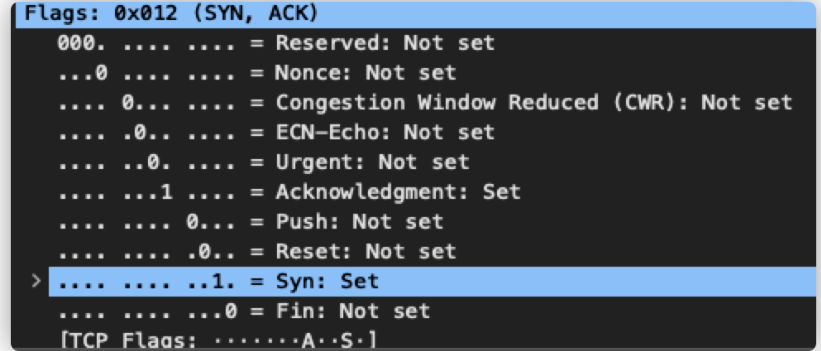


这就是本地电脑和百度三次握手的过程。 每次握手的过程这里重复了两次，但内容是一样的。

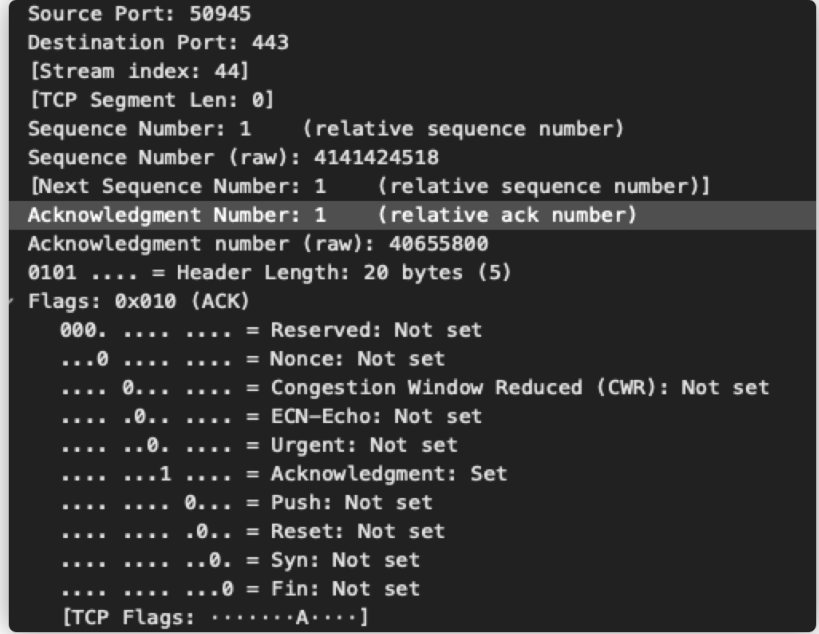
第一次握手：首先，是从本地客户端发向百度服务器的一个数据报。在这个数据报中，Seq=0，说明一开始是从序号为0的包好事发送的。我们看到这里SYN 这一位已经被设为1了，因为还没收到来自百度的确认信息，因此这里ACK为设为0。



第二次握手：这是百度服务器给本地电脑发送的报文，其中 Seq=0, ACK = 1 ,因为TCP是全双工通信的，因此从百度发送来的第一个报文段也是从 seq=0 开始的。但是这个报文段中包含了对我发给百度的包的确认信息，因此这里ACK被设置了，且值为1，这说明 1 以前的包我全部都收到了，请本地发送1以后的包。

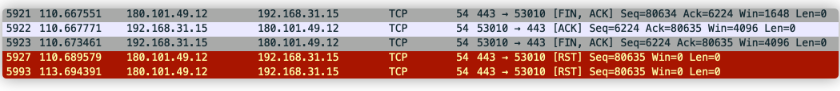


第三次握手：第三次握手是本地发送给百度服务器的，此时， Seq=1, 说明这是本地发送的第二个包了(第一个包Seq=0) ； ACK = 1 说明已经收到了来自服务端的 1以前的所有包。

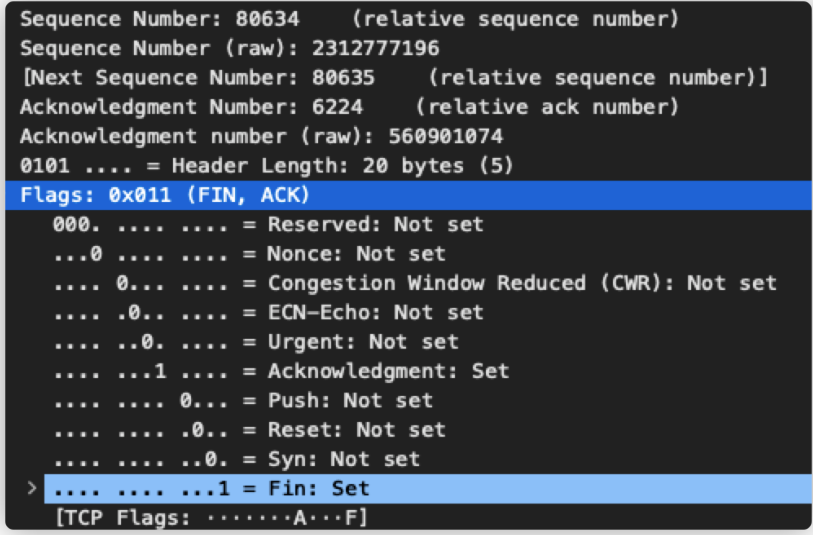


下面我们说一下TCP的四次挥手：

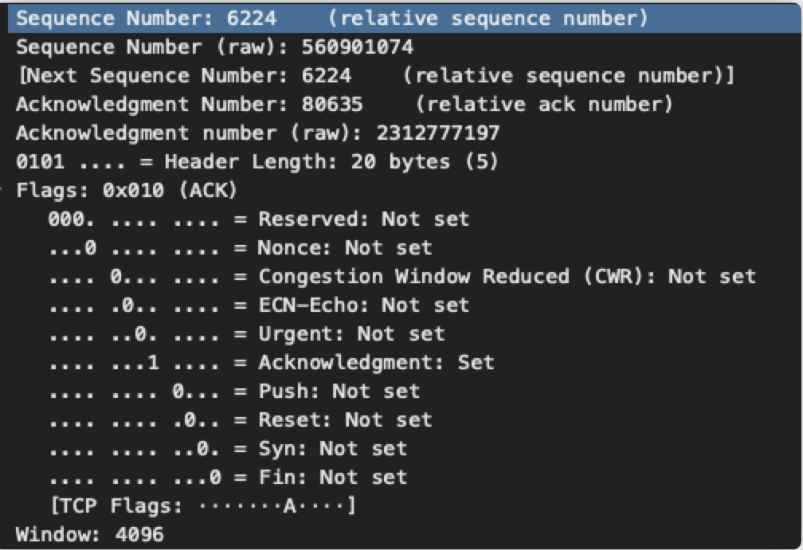
当通信双方完成数据传输，需要进行TCP连接的释放，由于TCP连接是全双工的，因此每个方向都必须单独进行关闭。这个原则是当一方完成它的数据发送任务后就能发送一个FIN来终止这个方向的连接。收到一个FIN只意味着这一方向上没有数据流动，一个TCP连接在收到一个FIN后仍然能发送数据。首先进行关闭的一方将执行主动关闭，而另一方执行被动关闭。因为正常关闭过程需要发送4个TCP帧，因此这个过程也叫四次挥手。



第一次挥手：第一次挥手是百度向本地发送的一个包， Seq=80634 Ack =6224. 同时，设定了Fin位，表示服务器单方面想和本地断开联系。

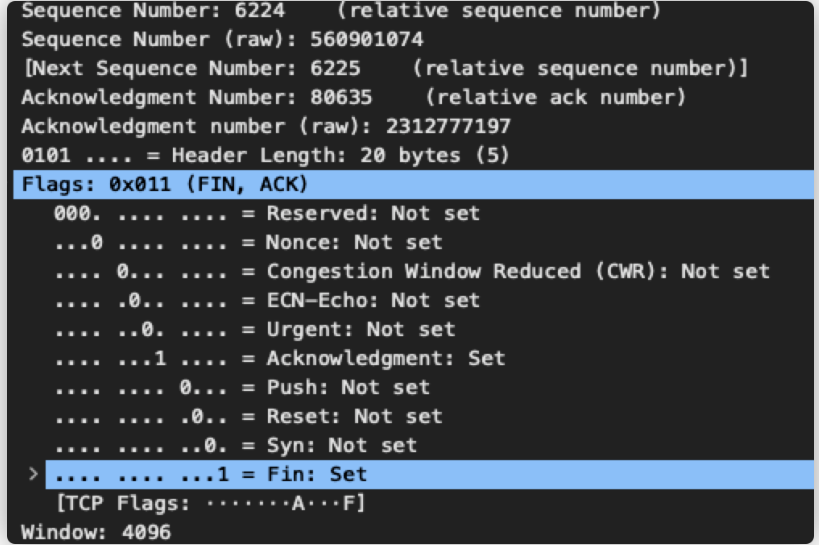


第二次挥手：第二次挥手是本地收到了百度想要结束的Fin之后，返回了一个带有ACK的包。同时告诉服务器，ACK=80635,说明80635以前的数据段都已经收到了。



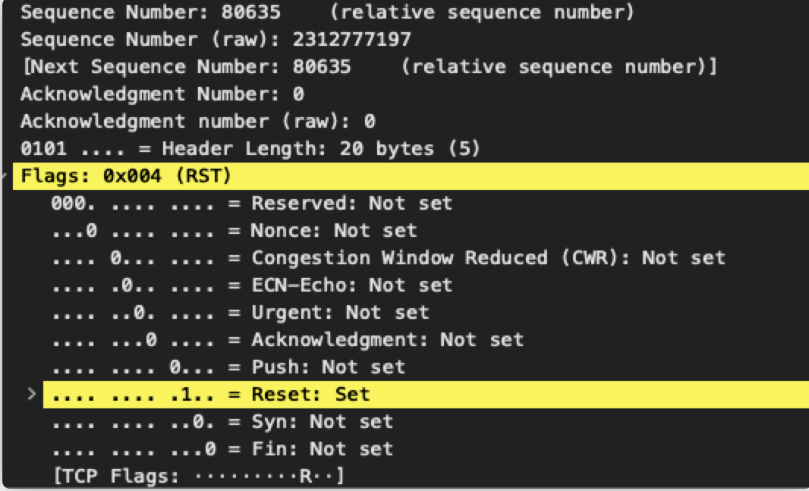
第三次挥手：前两次挥手是主动关闭方断开连接，但是只是单方面关闭连接。现在要被动关闭方来断开连接，才能实现真正的断连。

我们看到这个是 本地向百度发送的包，设置了Fin和ACK，注意到这个包的ACK和第二次挥手发送的ACk的值是一模一样的，因为在这两次挥手之间没有收到新的来自百度的包。

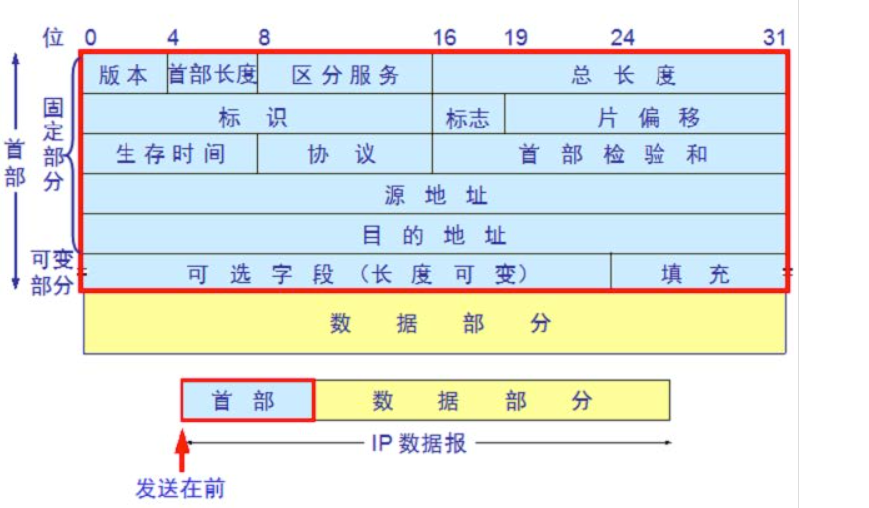


第四次挥手：第四次挥手，是百度收到了本地发送的带有FIN 标志的包后，返回的确认报文。

这个报文比较特殊，因为返回的报文只设置了RST位，没有设置 Seq，Ack。 RST位被设置以后，接收端收到之后不必发送ACK包。



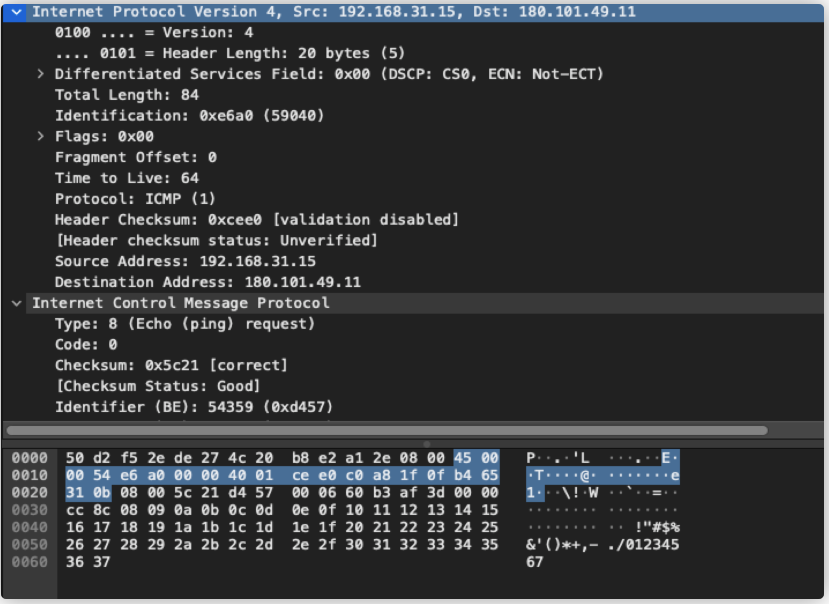
**IP协议：**



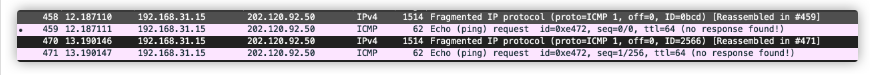
根据上课学习知道IP报文要交给数据链路层封装后才能发送，理想状况下，每个IP报文正好能放在同一个物理帧中发送。如果一个数据包超过1500字节(以太网的帧中最多可容纳1500字节的数据). 就需要将该包进行分片发送，这个上限被称为物理网络的最大传输单元(MTU. Maxium Transfer Unit)

TCP/IP 协议在发送IP数据报文时，一般选择一个合适的初始长度。当这个报文要从一个MTU大的子网发送一个MTU 小的网络时，IP协议就把这个报文的数据部分分割成能被目的子网锁容纳的较小的数据分片，组成较小的报文发送。每个较小的报文被称为一个分片。 每个分片都有一个IP 报文头，分片后的数据包的IP报头和原式IP报头分片偏移、MF标志位和校验字段不同外，其他都一样。

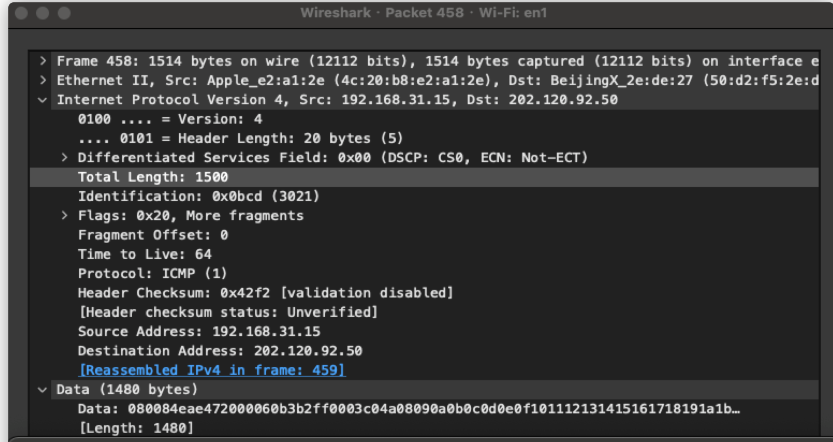
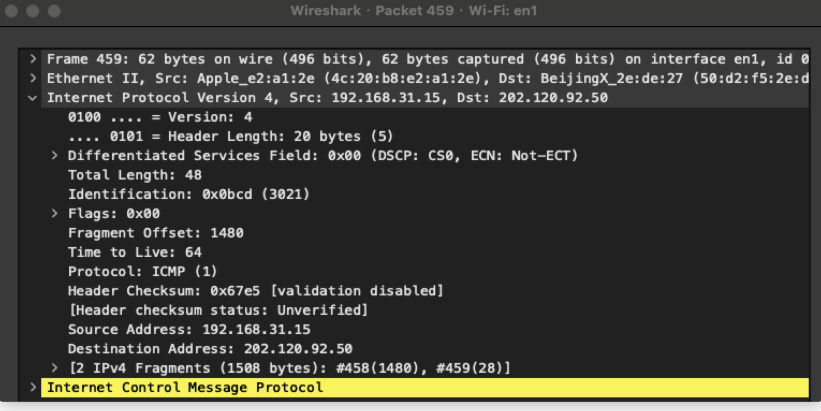
我们取一个有IP协议的ICMP数据报来分析，并根据该报文分析IP协议的报文格式。

我们对截获的报文进行分析，将属于同一个ICMP报文的分片找出来，并分析其字节长度特点。



上面产生了两个报文：

我们发现他们的 Identification 都是一样的，为 0x0bcd. 由此我们可以断定这两个分片属于同一个ICMP报文。

第一段报文是只有IP层，没有ICMP层的。总长度为1500，即一个MTU。但是因为IP层头长度为20，所以data的总长度只有1480。 因为这是第一个报文，因此并没有数据偏移量，因此这时 Fragment Offset : 0 ，同时在Flags中也有注明，说有其他分片。

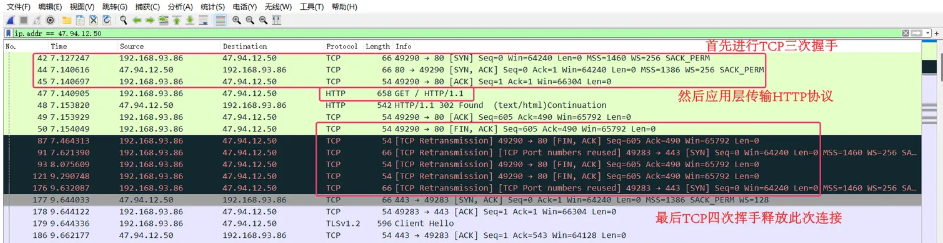
第二段报文报文的总长度为48，头长度为20，data是28，那么为什么不是20而是28呢？因为ICMP的头部占了8个字节。因为这是最后一段报文，因此Flag为0，前面只有一个满的报文，因此 Fragment Offset为1480.

因为说不管ping的字节有多长，一定都是填满一个报文之后再分出来，所以其Offset 一定都是1480的整数倍。

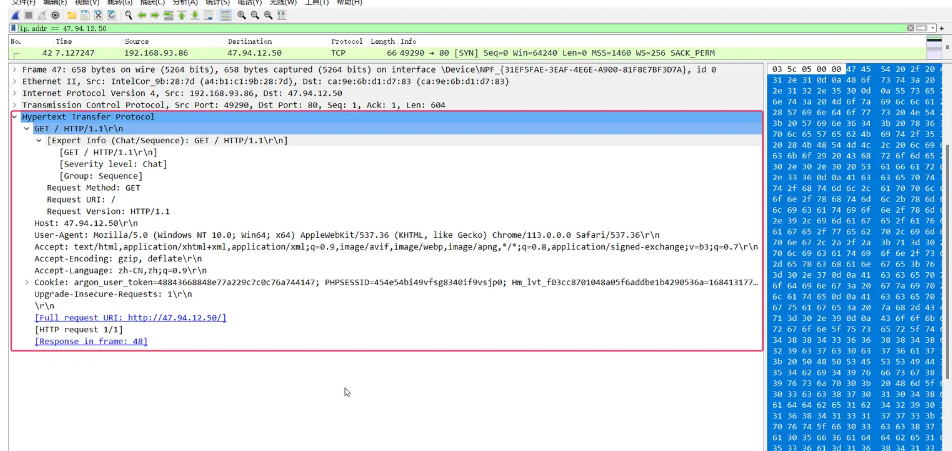
**HTTP协议：**

我们首先访问web云服务器，以个人网站 http://47.94.12.50 为例，后台打开 Wireshark嗅探，显示过滤器输入 ip.addr == 47.94.12.50。

收到的报文如下:



首先主机先与云服务器进行第一次TCP连接，完成三次握手，然后应用层使用HTTP协议进行传输，最后传输层使用TCP四次挥手释放第一次 TCP 连接。下面根据自下而上地分析从MAC层协议、IP协议、TCP协议一直到 HTTP 协议的过程。这里我们主要看HTTP协议：



由此HTTP报文可知主机通过HTTP/1.1协议使用GET的请求方法向服务器 资源发起请求。

具体的内容分析如下：

Host：47.94.12.50。代表了请求资源所属的主机。

User-Agen：Mozilla/5.0(Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/113.0.0.0 Safari/537.36。代表主机使用的 HTTP 协议的客户端类型。

Accept:text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,\*/\*;q=0.8,application/signedexchange;v=b3;q=0.7。 代表客户端这边支持任何类型的资源。