《网络与通信》课程实验报告

实验三：数据包结构分析

姓名	胡才郁	院系	计算机学院	学号	20121034
任课教师	刘通		指导教师	刘通	
实验地点	计 708		实验时间	2022.10.14	
实验课表现	出勤、表现得分(10)		实验报告得分(40)		实验总分	
	操作结果得分(50)					

实验目的：

1. 了解 Sniffer 的工作原理，掌握 Sniffer 抓包、记录和分析数据包的方法；
2. 在这个实验中，你将使用抓包软件捕获数据包，并通过数据包分析每一层协议。

实验内容：

使用抓包软件捕获数据包，并通过数据包分析每一层协议。

实验要求：（学生对预习要求的回答）（10 分）　　得分：

- **常用的抓包工具**

1. Wireshark

是一个功能十分强大的开源的网络数据包分析器，可实时从网络接口捕获数据包中的数据。它尽可能详细地显示捕获的数据以供用户检查它们的内容，并支持多协议的网络数据包解析。

2. Hping

Hping是最受欢迎和免费的抓包工具之一。它允许你修改和发送自定义的ICMP，UDP，TCP和原始IP数据包。此工具由网络管理员用于防火墙和网络的安全审计和测试。

HPing可用于各种平台，包括Windows，MacOS，Linux，FreeBSD，NetBSD，OpenBSD和Solaris。

3. Fiddler

Fiddler工具非常经典且强大，它可以提供电脑端、移动端的抓包、包括 http 协议和 https 协议都可以捕获到报文并进行分析；可以设置断点调试、截取报文进行请求替换和数据篡改，也可以进行请求构造，还可以设置网络丢包和延迟进行APP弱网测试等。

Fiddler 的第一个优点，就是功能强大并齐全；第二个优点就是Fiddler是开源免费的，所有的电脑只要安装就可以直接使用所有的功能！

但是Fiddler只能在Windows下安装使用。不巧的是我的操作系统是macOS，所以在这次实验中Fiddler我就不考虑了。

4. TCPdump

TCPdump 是专门作用于 Linux 命令行的抓包工具，它可以提供非常多的参数来对网络数据包进行过滤和定义。它抓取到的报文可以直接打印在 Linux 的命令行界面，也可以进行保存成文件。

不过由于是在命令行中完成全部的操作，没有图形化界面，对于初学者可能会使用起来不方便，不过对于熟悉 CLI 开发者应该而言更加灵活，这一点见仁见智。

顺带一提，大部分常用的抓包工具在 Kali Linux 下都有默认安装，本实验中有一部分也是使用 Kali Linux 进行实验，Kali Linux 确实是网络安全人员最适合使用的 Linux 发行版。

实验过程中遇到的问题如何解决的？（10 分）	得分：

问题 1：在抓取 HTTPS 请求的过程中无法正常处理

HTTP 是超文本传输协议，信息是明文传输，HTTPS 则是具有安全性的 SSL 加密传输协议。HTTP 和 HTTPS 使用的是完全不同的连接方式，HTTP 的连接很简单,是无状态的。HTTPS 协议是由 SSL+HTTP 协议构建的可进行加密传输、身份认证的网络协议，由于其安全性，需要对 Wireshark 进行手动配置才可以抓取。

通过设置 SSLKEYLOGFILE 环境变量，可以指定浏览器在访问 SSL/TLS 网站时将对应的密钥保存到本地文件中，有了这个日志文件之后 Wireshark 就可以将报文进行解密了。

问题 2：抓取 IP 数据包时，首部和校验字段一直显示为失效

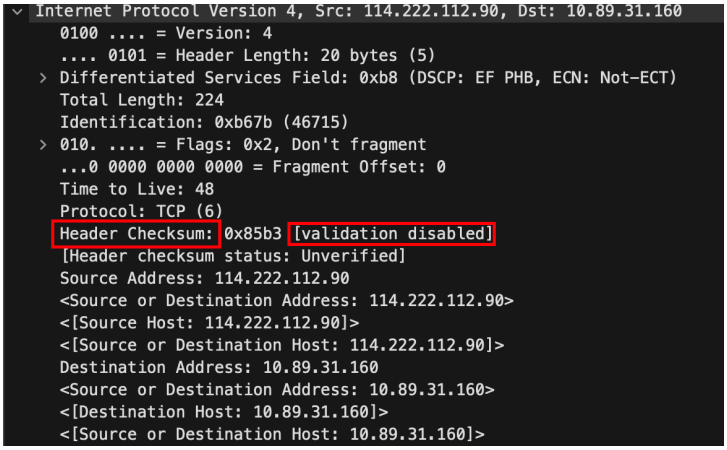

图 1. Wireshark 中默认不提供 IP 首部校验和服务

使用 Header Checksum 字段数据无误的情况下，Wireshark 默认不提供校验。这是由于目前很多网卡已经支持 IP 片以及 IP/TCP/UDP 等协议的校验和计算，当协议层发现网卡支持相应的特性时，会将相应的处理交给网卡操作。如上面提到的校验和，正常情况下，校验和由对应的协议层处理，但在网卡使能情况下会将其推迟到网卡层面处理，网卡处理结束后直接发送，这就是为什么 Wireshark 抓到的报文里面的校验和会提示不正确的原因。

解决方式为在协议的默认设置中勾选下图中的选项。

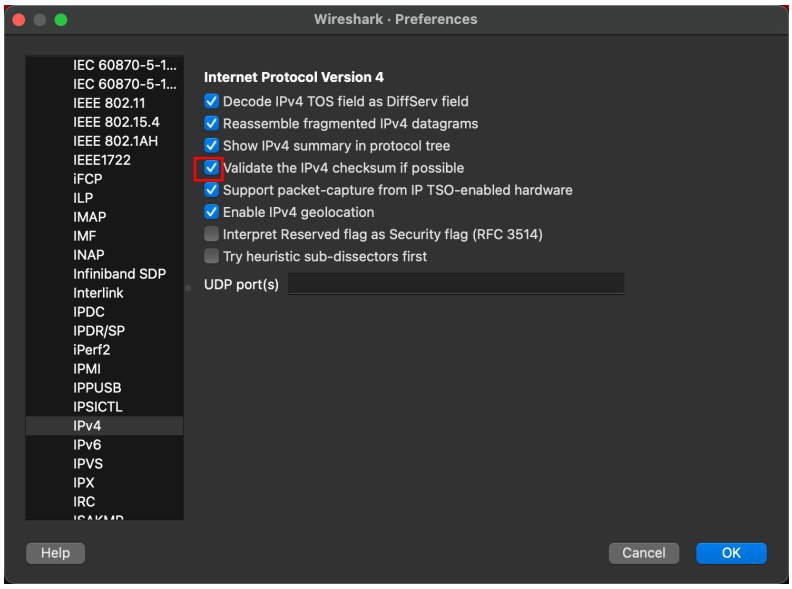

图 2. 勾选 Validate 选项

问题 3：Ping 命令可以 Ping 通却抓不到相关的数据

原因是实验时一直挂着 VPN。理想情况下，VPN 会通过加密的，安全的专用网络来处理你的所有流量，使得第三方很难监控到你的网络浏览数据。Ping 返回的数据到达了代理服务器使得本地的 Wireshark 不可以直接通过筛选目的地址的方式找到相关数据。

解决方式为关闭 VPN 代理后再使用 Wireshark 抓取，就可以正常的使用了。

本次实验的体会（结论）（10 分）	得分：

在本次实验中，我使用 Wireshark 对于多种数据包进行了抓包分析。

使用 Wireshark 进行抓包可以手动设置是否使用“混杂”模式还是普通模式，对于分析数据包而言，使用普通模式，关闭混杂模式不会出现多余的数据包，更有利于分析数据包结构。下图为 Wireshark 中对于是否使用“混杂”模式的选项设置。

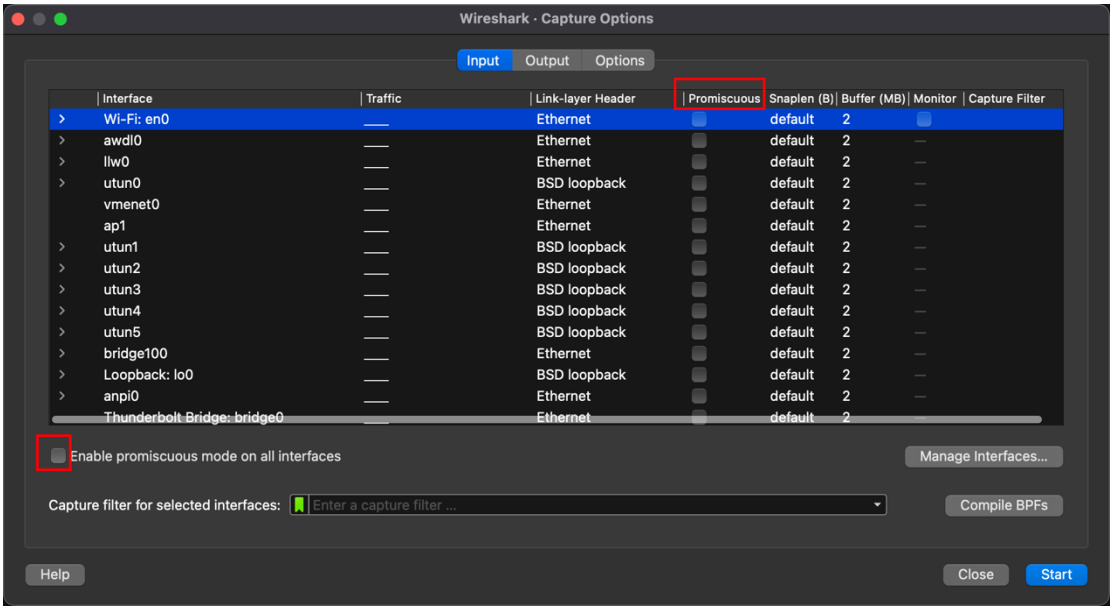

图 3. Wireshark 中关于是否使用混杂模式的设置

并且，上图也很好的体现了 Wireshark 可以抓流经不同网卡的数据包。在 macOS 操作系统下，默认网卡名称为 en0，而 Linux 下默认网卡名称为 eth0。

需要注意的是，此次实验我在两种操作系统下进行实验，一个为 macOS，另一个为虚拟机中的 Kali Linux，这样做的原因是使用虚拟机中的网络减少无关包的广播。

本次我主要做了两个实验，一个主要测试 ping 的 DNS 与 ICMP，另一个测试 curl 的 HTTP 等等。由于我的宿主机连在校园网中并挂有 VPN 中，抓到的无关数据包较多，而 Wireshark 没有为 DNS 提供整体的流筛选功能，但是为 HTTP 提供了整体的流筛选功能。所以做 ping 实验时我使用虚拟机中的 kali Linux 减少无关的包抓取，做 curl 实验时使用 macOS 的流筛选功能方便观察。

数据在网络上是以很小的称为帧（Frame）的单位传输的，帧由几部分组成，不同的部分执行不同的功能。以数据链路层的“帧”为例。不同的“帧”是由对应于不同信息功能的多个部分组成。帧的类型和格式根据通信各方的数据链路层使用的协议确定，由网络驱动程序按照一定的规则生成，通过网络接口卡发送到网络，并通过网络传输到目的主机。

在正常情况下，网络接口卡会读取一个帧并进行检查。如果帧中携带的目的 MAC 地址与自己的物理地址一致或为广播地址，网络接口卡就会产生一个硬件中断，以引起操作系统的注意。然后，帧中包含的数据被发送到系统中进行处理，否则，帧被丢弃。但如果某块网卡被设置为“混杂”模式，该网卡将接收网络上传输的所有帧，这就形成了监控。

思考题：（10 分）

思考题 1：（4 分）	得分：

写出捕获的数据包格式。

此处以捕获 HTTP 数据包为例介绍，对于各个协议数据报的解释，将在下文中介绍。

对于本次实验操作，这里对抓取过程中所出现的内容进行了注释：

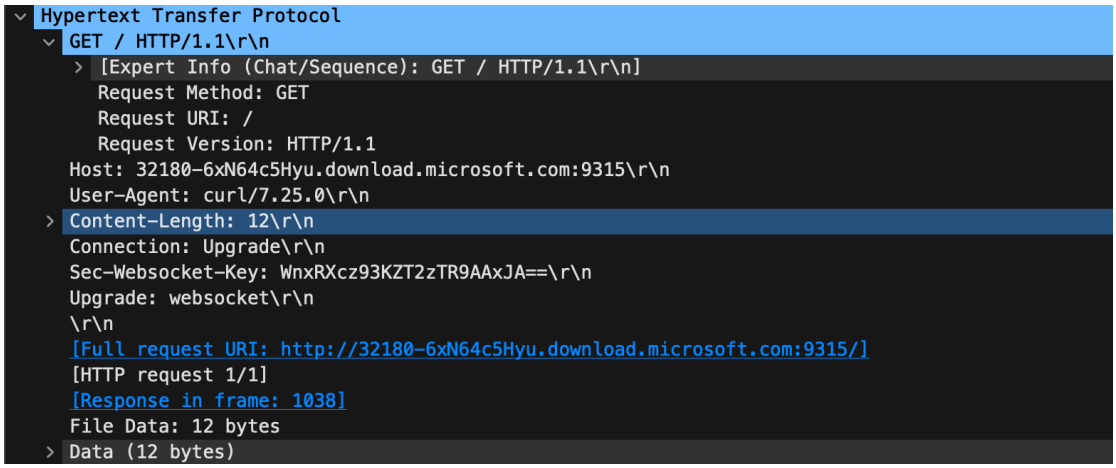

图 4. Wireshark 中捕获的 HTTP 协议包

上图中展示了此HTTP协议包的详细信息，例如请求方式为GET，协议版本为HTTP/1.1，请求URL为根目录“/”等等。

对于完成一次HTTP请求而言，具体可以分为以下几个步骤：

1. 发送一个HTTP的请求
2. 服务器收到我们的请求返回了一个SEQ/ACK进行确认
3. 服务器讲HTTP的头部信息返回给客户端，在HTTP协议中，状态码200表示正常
4. 客户端收到服务器返回的头部信息，向服务器发送SEQ/ACK进行确认
5. 当发送完成之后，客户端就会发送FIN/ACK来进行关闭链接的请求。

HTTP 常用字段解释解释如下：

表 1. HTTP 常用字段

#	抓取 HTTP Request 的顺序，从 1 开始，以此递增
Result	HTTP 状态码
Protocol	请求使用的协议，如 HTTP/HTTPS/FTP 等
Host	请求地址的主机名
URL	请求资源的位置
Body	该请求的大小
Caching	请求的缓存过期时间或者缓存控制值
Content-Type	请求响应的类型
Process	发送此请求的进程：进程 ID
Comments	允许用户为此回话添加备注

HTTP 请求报文主要由三部分组成，即请求行、请求头部、请求数据。其中，这三个部分分别包含如下常见字段：

1. 请求行：

- 请求方法，GET 和 POST 是最常见的 HTTP 方法，除此以外还包括 DELETE、HEAD、OPTIONS、PUT、TRACE。
- 请求对应的 URL 地址，它和报文头的 Host 属性组成完整的请求 URL。
- 协议名称及版本号。

2. 请求头：

- HTTP 的报文头，报文头包含若干个属性，格式为“属性名:属性值”，服务端据此获取客户端的信息。
- 与缓存相关的规则信息，均包含在 header 中

3. 请求体：

- 报文体，它将一个页面表单中的组件值通过 param1=value1&param2=value2 的键值对形式编码成一个格式化串，它承载多个请求参数的数据。

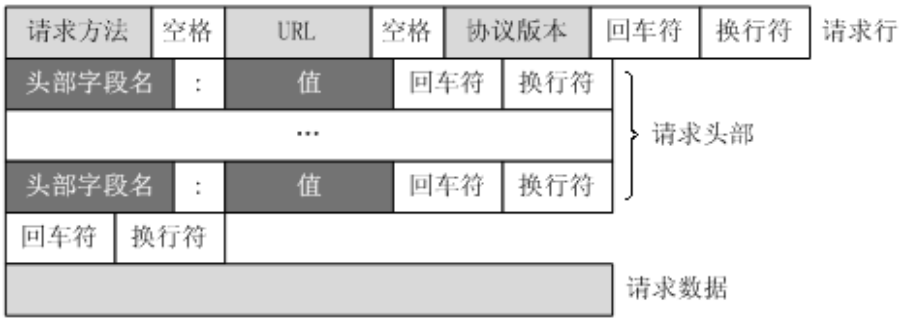

图 5. HTTP 数据包格式

思考题2：（6分）	得分：

写出实验过程并分析实验结果。

本次实验采用抓包工具Wireshark进行，Wireshark会在捕获报文的时候自动记录捕获的时间，在解码显示时显示出来，分析问题时提供了很好的时间记录。

本实验中着重做了两个实验进行抓包分析，分别为Ping百度与curl请求百度数据的实验。这两个实验除了IP、Ethernet等相同协议外，第一个实验涉及DNS的请求与相应、ICMP，而第二个涉及HTTP与TCP的三次握手、四次挥手。

实验一：Ping百度

Ping百度，观察请求与相应信息。为了更好的反应IP数据包的分片操作，此处Ping使用-S参数发送4000Bytes.此处是用虚拟机中的Kali Linux和Kali原生安装的Wireshark进行实验。如下图所示，其中发送4000Bytes主要是为了观察IP报文长度大于MTU时的分片操作。

```
┌──(root㉿kali)-[/]
└─# ping -c 1 -s 4000 www.baidu.com
PING www.baidu.com (198.18.0.191) 4000(4028) bytes of data.
4008 bytes from 198.18.0.191 (198.18.0.191): icmp_seq=1 ttl=62 time=0.560 ms

--- www.baidu.com ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.560/0.560/0.560/0.000 ms
```

图 6. Ping 命令请求百度

Ping命令结束后，返回Wireshark查看结果。可以看到在ping百度这个过程中，对于较高

层网络协议而言，用到了DNS、IP、ICMP。接下来从这些数据包中选取典型的进行分析。

No.	Time	Source	Destination	Protocol
1	0.000000000	192.168.64.4	192.168.64.1	DNS
2	0.000019546	192.168.64.4	192.168.64.1	DNS
3	0.000765143	192.168.64.1	192.168.64.4	DNS
4	0.000765226	192.168.64.1	192.168.64.4	DNS
5	0.001139317	192.168.64.4	198.18.0.191	IPv4
6	0.001149111	192.168.64.4	198.18.0.191	IPv4
7	0.001149944	192.168.64.4	198.18.0.191	ICMP
8	0.001571630	198.18.0.191	192.168.64.4	IPv4
9	0.001571755	198.18.0.191	192.168.64.4	IPv4
10	0.001571838	198.18.0.191	192.168.64.4	ICMP

图 7. Ping 命令请求后，多种协议类型结果

观察图7可知，访问www.baidu.com网址，针对DNS协议，共有四条数据。两次DNS请求与两次DNS响应（请求与响应配对存在）。

出现两对DNS请求响应的原因是分别对www.baidu.com请求查询IPv4与IPv6地址。第一次DNS请求查询百度域名对应的IPv4地址，第二次DNS请求查询百度域名对应的IPv6地址。因为两次查询协议过程相同，故以第一次查询内容来进行分析，如下图所示。

```
▸ Frame 3: 73 bytes on wire (584 bits), 73 bytes captured (584 bits) on interface eth0, id 0
▸ Ethernet II, Src: f2:2f:4b:c0:02:64 (f2:2f:4b:c0:02:64), Dst: 6a:71:9a:d7:0f:c0 (6a:71:9a:d7:0f:c0)
▸ Internet Protocol Version 4, Src: 192.168.64.1, Dst: 192.168.64.4
▸ User Datagram Protocol, Src Port: 53, Dst Port: 56832
▸ Domain Name System (response)
```

图 8. 各层协议分层

第一次查询的相应返回结果如上图所示，Wireshark帮助开发者进行各个协议之间的分层。从上到下分别对应计算机网络层次结构。

数据链路层 – Ethernet II
网络层 – IPv4
传输层 – UDP
应用层 – DNS

接下来针对这一条对于DNS查询内容，自底向上进行分析。

数据链路层：

对于图6中的第一行，Frame 3: 73 bytes on wire (584 bits), 73 bytes captured (584 bits) on interface eth0.它的意思是在0号网卡上数据帧号码3，捕获了73字节，即584位。需要注意的是，以太帧工作在数据链路层，在数据链路层中有MTU，也就是最大传输单元。任何一个以太帧都不能超过MTU设置的字节数（一般为1500字节），如果超过了，比如IP数据报太大了，就需要对IP数据报进行分片处理。这种分片操作会在下面分析IP数据报时具体解释。

以太帧有多种格式。其中最为常用的就是此处的Ethernet II帧格式，也就是Wireshark抓包最容易看到的。

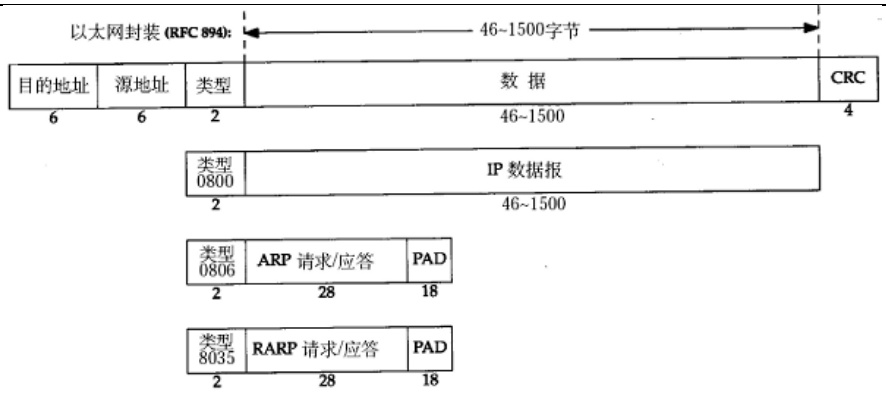

图 9. Ethernet II 帧格式

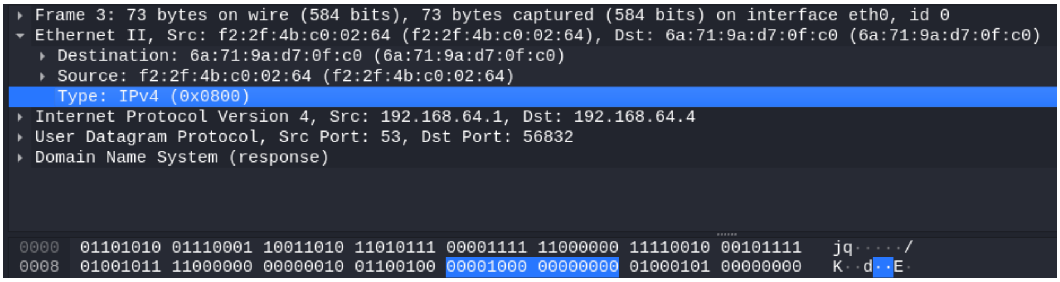

图 10. Wireshark 捕获 Ethekrnet II 帧

抓包Ethernet II可以清楚的看到里面包含了三个信息，Destination, Source, Type依次对应着图7中的目的地址、源地址、类型，而Type类型为IPv4说明它是一个IP包，版本是v4。Type类型对应的就是这个以太帧的上层层协议，其位数为16，即两个字节，对应图8中蓝色高亮部分，值为0x0800也与以太帧结构图吻合。

以太帧结构图上面还标明了Type还可以对应APR协议，也就是说，以太帧即可以与网络层相关联（IP协议），也可以与第二层数据链路层相关联（ARP协议）。此处的type，常用的有0x0806 ARP, 0x0835 RARP等等。

网络层

在网络层中分析IP协议的内容。IP数据报分为首部与数据部分，数据部分即传递的数据，比如TCP协议的数据，UDP协议的数据等等。

图 11. IP 数据报结构图

对首部而言，它是由不可变部分与可变部分来组成的。不可变部分长度固定，20个字节，也就是上图的前五行，每行从0-31共32位，也就是四个字节。4 * 5 = 20个字节，这20个字节是固定不变的。Wireshark捕获的IP数据报如下：

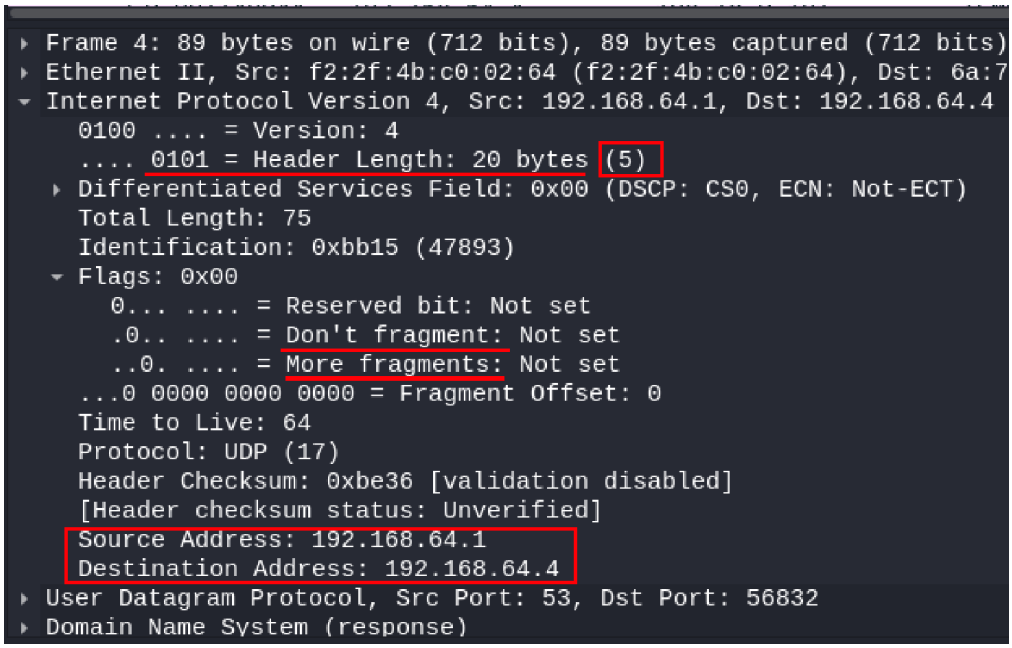

图 12. Wireshark 中 IP 数据报

Header Length是首部的长度，0101转换成十进制就是5，由于它的一个单位代表32位，也就是四个字节。所以5 * 4 = 20正好是20个字节。对于Total length，即IP数据报的总长度，首部加上数据部分的长度。Protocol字段为这个IP数据包包含的高层协议，此处为UDP，对应值为17。Source Address与Destination Address字段封装了源IP地址与目标IP地址。此处的Flags共3位，对应DF、MF的是否分片。由于这是DNS上层的IP数据报，所以长度较小，不需要分片，FLAG为000，之后会对于长度较大，需要分片的情况做讨论。

传输层

UDP的报文结构较为简单，分为源端口、目的端口、长度和校验和四个字段。四个字段每个占用16位，即两个字节。UDP的首部固定为8个字节，前两个字节Source Port 表示源端口，Destination Port表示目标主机的端口。长度Length是首部加上数据部分的长度，在下图值为55。55减去首部长度8，也就是图的最下方UDP payload中47bytes，表示此UDP数据报的数据部分为47字节。

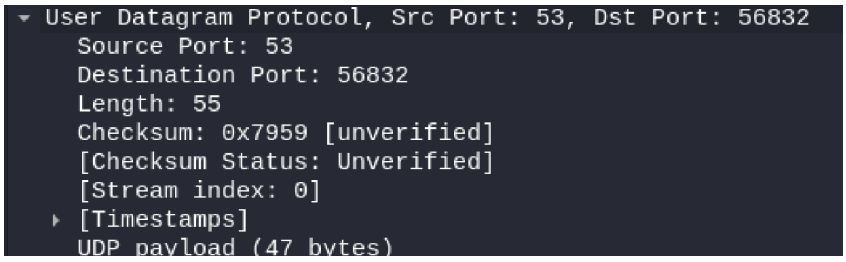

图 13. 传输层 UDP 部分

应用层

此处应用层比较重要的字段即queries与answers，type A即查询IPv4数据，若type AAAA则查询IPv6数据，此处返回百度的IPv4地址为198.18.0.191。通过对于数据包整体的分析，也可以明确DNS是基于UDP协议的。

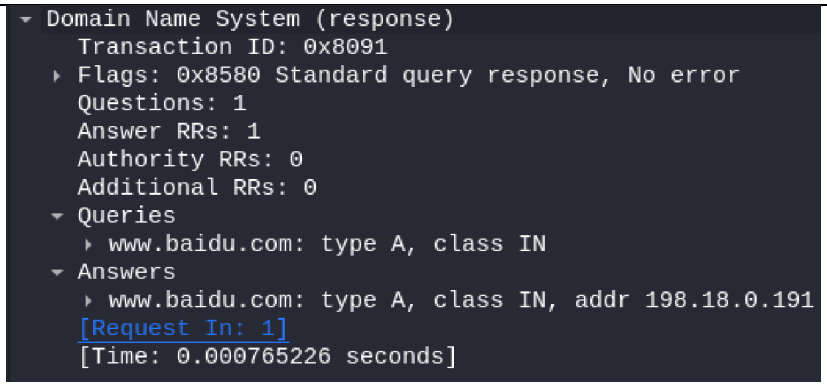

图 14. 应用层 DNS 部分

分析结束敲下ping -S 4000 www.baidu.com命令后的一次DNS请求各个层的协议后，分析IP数据报的分片操作。由于数据链路层限制了MTU，因此此处ping的4000bytes超出了MTU限制，需要进行分片操作。观察下图的ping操作返回的数据中IP数据报的部分：

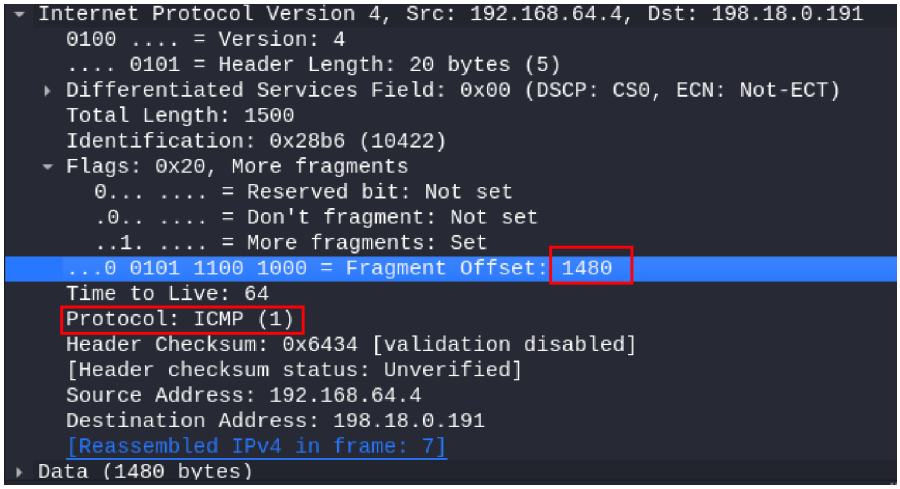

图 15. IP 数据报分片操作

观察到Flags字段中，DF = 0，MF = 1。如果MTU为1500Bytes，原IP数据报可分为3个部分，4000Bytes的数据段，20Bytes的IP首部，8Bytes的ICMP首部。可以分为3个IP数据报分片，由于IP数据报首部占用了20字节，因此这三个分片分别为20 + 1480，20 + 1480，20 + 1048。由于此处使用Ping指令，因此需要用到ICMP协议首部的8个字节，存在了最后一个分片中，这也与此处Protocol字段为ICMP，值为1对应。所以最后一个分片大小为IP数据报首部长20+数据字段长1040+ICMP首部字段。实验一分析到此结束。

实验二：curl请求百度数据

由于HTTP协议是TCP协议的上层协议，下面以发送HTTP，抓取HTTP与TCP协议包为例，详细介绍：

在命令行中使用curl访问百度，使用-I参数只请求HTTP协议中的请求头，来模拟此次操作。

```
➜  ~ curl -I www.baidu.com
HTTP/1.1 200 OK
Accept-Ranges: bytes
Cache-Control: private, no-cache, no-store, proxy-revalidate, no-transform
Connection: keep-alive
Content-Length: 277
Content-Type: text/html
Date: Sun, 23 Oct 2022 01:36:48 GMT
Etag: "575e1f74-115"
Last-Modified: Mon, 13 Jun 2016 02:50:28 GMT
Pragma: no-cache
Server: bfe/1.0.8.18
```

图 16. 使用 curl 发送 HTTP 请求

执行完curl命令后切换到Wireshark观察抓包情况，并且筛选tcp.stream可以完整的捕获到此次操作的全部数据包。如下图所示，包含了TCP的三次握手，四次挥手的过程。

tcp.stream eq 20

No.	Time	Source	Destination	Protocol	Length	Info
325	18.202711	10.89.11.148	36.152.44.95	TCP	78	52113 → 80 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=64 TSv
326	18.216169	36.152.44.95	10.89.11.148	TCP	78	80 → 52113 [SYN, ACK] Seq=0 Ack=1 Win=8192 Len=0 MSS=1328
327	18.216433	10.89.11.148	36.152.44.95	TCP	54	52113 → 80 [ACK] Seq=1 Ack=1 Win=262144 Len=0
328	18.216578	10.89.11.148	36.152.44.95	HTTP	132	HEAD / HTTP/1.1
330	18.230600	36.152.44.95	10.89.11.148	TCP	60	80 → 52113 [ACK] Seq=1 Ack=79 Win=29056 Len=0
331	18.233163	36.152.44.95	10.89.11.148	HTTP	386	HTTP/1.1 200 OK
332	18.233329	10.89.11.148	36.152.44.95	TCP	54	52113 → 80 [ACK] Seq=79 Ack=333 Win=261760 Len=0
333	18.233671	10.89.11.148	36.152.44.95	TCP	54	52113 → 80 [FIN, ACK] Seq=79 Ack=333 Win=262144 Len=0
334	18.246187	36.152.44.95	10.89.11.148	TCP	60	80 → 52113 [ACK] Seq=333 Ack=80 Win=29056 Len=0
335	18.246188	36.152.44.95	10.89.11.148	TCP	60	80 → 52113 [FIN, ACK] Seq=333 Ack=80 Win=29056 Len=0
336	18.246347	10.89.11.148	36.152.44.95	TCP	54	52113 → 80 [ACK] Seq=80 Ack=334 Win=262144 Len=0
534	21.300029	36.152.44.95	10.89.11.148	TCP	60	80 → 52113 [RST] Seq=334 Win=0 Len=0

图 17. 此次操作 Wireshark 抓取的数据包

第一次挥手：服务端发送一个［FIN＋ACK］，表示自己没有数据要发送了，想断开连接，并进入FIN_WAIT_1状态。

第二次挥手：客户端收到FIN后，知道不会再有数据从服务端传来，发送ACK进行确认，确认序号为收到序号＋1（与SYN相同，一个FIN占用一个序号），客户端进入CLOSE_WAIT状态。

第三次挥手：客户端发送［FIN＋ACK］给对方，表示自己没有数据要发送了，客户端进入LAST_ACK状态，然后直接断开TCP会话的连接，释放相应的资源。

第四次挥手：服务户端收到了客户端的FIN信令后，进入TIMED__WAIT状态，并发送ACK确认消息。服务端在TIMED_WAIT 状态下，等待一段时间，没有数据到来，就认为对面已经收到了自己发送的ACK并正确关闭了进入CLOSE状态，自己也断开了TCP连接，释放所有资源。当客户端收到服务端的ACK回应后，会进入CLOSE状态并关闭本端的会话接口，释放相应资源。

使用Wireshark提供的Flow图，如下图所示，可以将整个过程观察的更加直观。

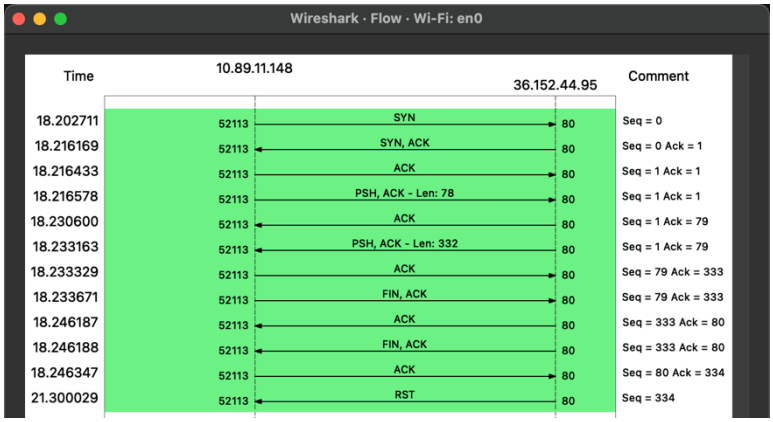

图 18. 使用 Wireshark 抓取 TCP、HTTP 协议包 Flow 图

详细观察此次TCP建立过程中的ACK信息，如下图所示。其中数据包中的字段包括了源端口号为80端口，目标端口号为52113端口。Acknowledgement为1，即ACK，并且SEQ/ACK分析中，更说明了是对328帧的ACK。

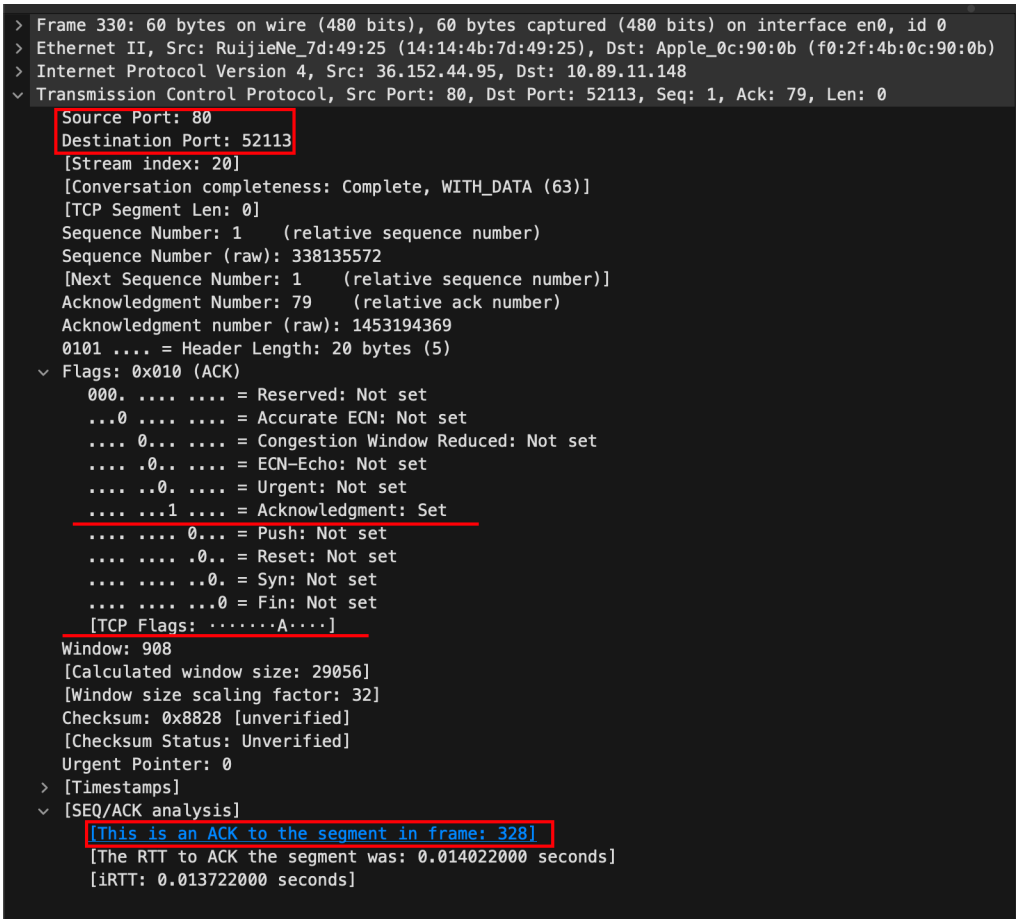

图 19. 使用 Wireshark 抓取 TCP 报文

指导教师评语：

日期：