实验四、应用层协议分析实验报告

组号：_____

姓名：_____ 学号：_____ 班级：_____

姓名：_____ 学号：_____ 班级：_____

一、 实验目的

分析应用层协议（如 FTP，HTTP）的工作过程，理解应用层与传输层及下层协议的关系。

二、 实验内容

（1）每组同学利用现有实验室网络及云服务器搭建内网、外网环境；

（2）用 Wireshark 截获 HTTP 报文，分析报文结构及浏览器和服务器的交互过程；分析 HTTP 协议的缓存机制。分析应用层协议跟 TCP/DNS 等协议的交互关系。

（3）用 Wireshark 截获 FTP 的报文，分析 FTP 协议的连接；分析被动模式，普通模式的区别；分析 NAT 对 FTP 的影响。使用 netcat 工具模拟 FTP 的客户端。

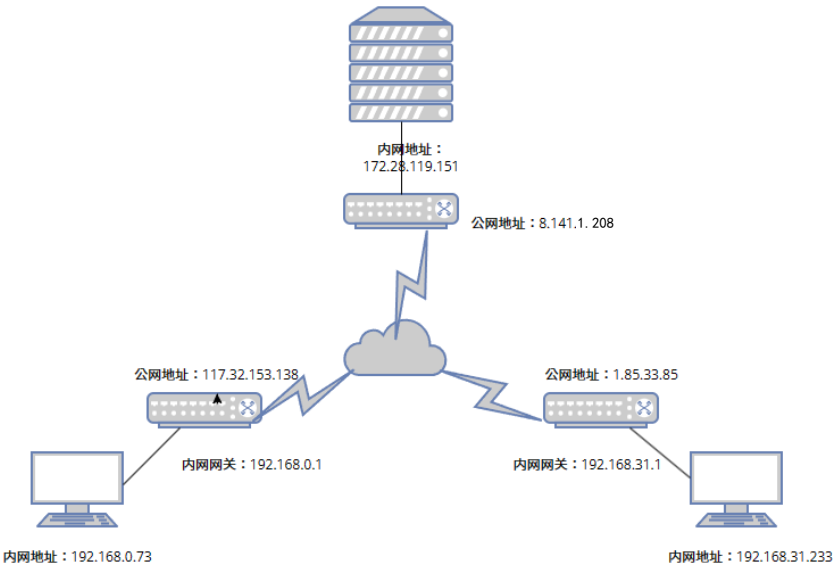
注：HTTP 和 FTP 两个协议二选一。

三、 实验环境与分组

每 2 名同学一组，以现有校园网络环境及云服务器搭建内网、外网网络。

四、 实验组网

以各组现有网络实际情况为准，标注内网、公网地址。



五、 实验过程及结果分析

1、HTTP 协议分析

（一）清空缓存后的 ARP，DNS 和 HTTP 协议分析

步骤 1：在计算机终端上运行 Wireshark 截获所有的报文。

步骤 2：清空 ARP，DNS 和 HTTP 浏览器的缓存：

浏览器缓存的清除以 Chrome 浏览器为例，地址栏中输入 chrome://settings/,找到高级选项中的“隐私设置和安全性”,清除浏览数据。

执行“ipconfig /flushdns”清除本地 DNS 缓存。

执行“arp －d”命令清空 arp 缓存。

步骤 3：在浏览器中访问 3 个网址，即 www.xjtu.edu.cn（202.117.1.13）, www.github.com（20.205，243，166）, www.unb.br（164.41.102.70）;

步骤 4：执行完之后，Wireshark 停止报文截获，分析截获的报文。

观察几个协议的配合使用，注意访问的延迟情况。特别分析 HTTP 的请求和应答。注意一个网址的访问中：

- 有几次 DNS 解析（包含协议的配合使用分析）;

通过以下指令过滤 wireshark 报文记录，我们可以让报文顺序更加清晰，所得报文如图 4.1.1.1。

```
(dns or http) and ip.addr == 192.168.0.73 or arp
```

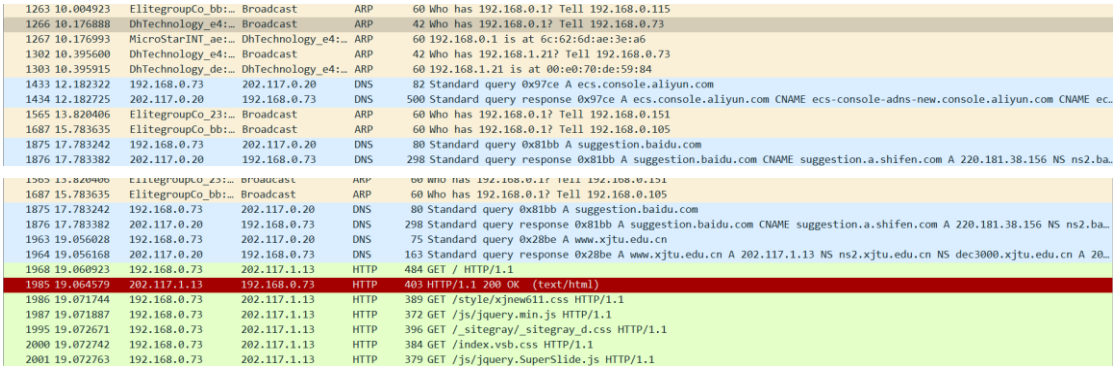


图 4.1.1.1

显然，在执行命令清除 ARP 与 DNS 缓存后，在我们访问一个域名时，其协议访问顺序为 ARP—DNS—HTTP，接下来我们将一步步分析各个部分。

1. 启动后 10s 左右

如图 4.1.1.2，首先，主机使用 ARP 协议广播查询 192.168.0.1（默认网关）的位置并获得回答，主机由此确认了网关的 MAC 地址。紧接着主机继续查询 192.168.1.21 位置并获得回答，此应该为另一主机位置。

图 4.1.1.2

2.启动后 19s 左右

在获得网关位置后，进入 DNS 访问阶段，由于我们访问的三个域名没有分别清除 ARP 协议记录，因此 ARP 协议不再出现。简要分析对三个网站分别的 DNS 协议记录：

截获报文如图4.1.1.3，首先，主机发送DNS协议到默认DNS服务器（202.117.0.20）查询 www.xjtu.edu.cn 的IP地址，共一次，并得到回应202.117.1.13（如图4.1.1.4）。然后进入了 HTTP 协议直接访问 202.117.1.13 的部分。

图 4.1.1.3

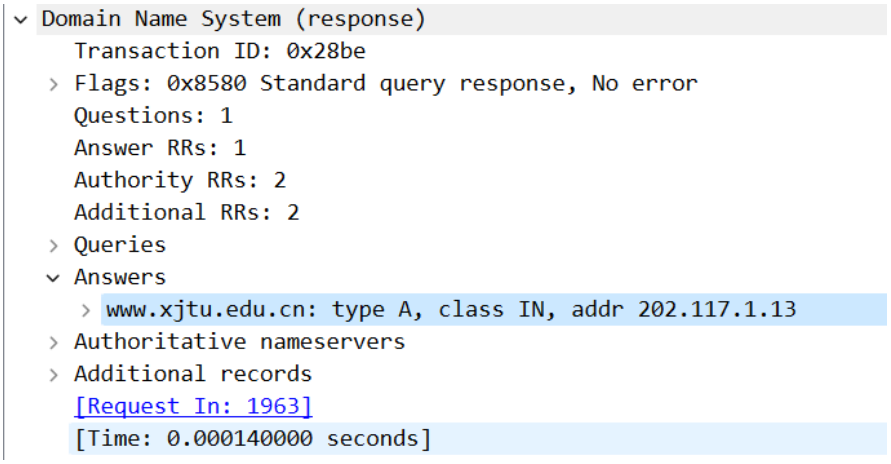


图 4.1.1.4

然后，值得注意的是，在一定量 HTTP 协议进行完成后，主机又开始发送 DNS 报文，这次是在大量地进行与 xjtu.edu.cn 域名“高度”相关的查询，如图 4.1.1.5。

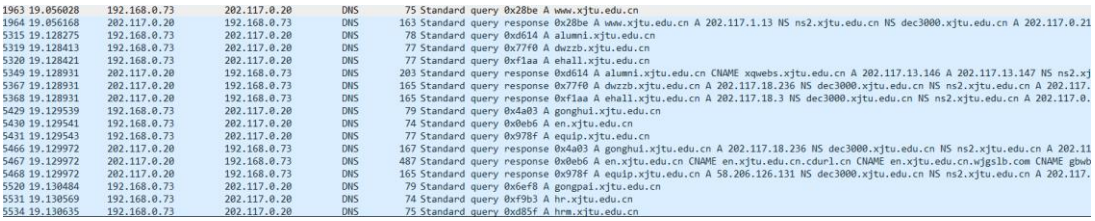


图 4.1.1.5

对 www.github.com 和 www.unb.br 的访问也只发现进行了一次 DNS 查询。但似乎没有出现与该域名“高度”有关的查询，如图 4.1.1.6 和图 4.1.1.7。

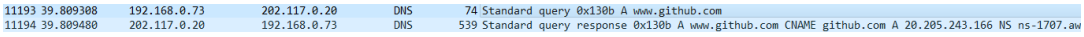


图 4.1.1.6

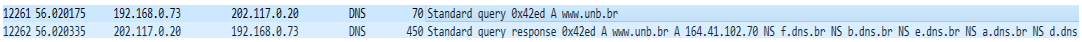


图 4.1.1.7

之所以会出现对 alumni.xjtu.edu.cn 这样域名的查询，是因为在服务器返回的 html 文件中出现了该域名，因此客户端自动帮我们进行了查询。由于 www.xjtu.edu.cn 中出现的域名基本以 xjtu.edu.cn 结尾，因此很容易观察到（即所述“高度”有关）。事实上其它域名也引发了这样的情况，例如 www.unb.br 就引起了对 www.youtube.com 等的查询，原因是相同的。这样的机制将加速我们可能的域名访问。

总而言之，访问网页时会采取 ARP-DNS-HTTP 的顺序，在本次实验中 ARP 协议的使用明显远早于 DNS 和 HTTP 协议（提前了 9s），而 DNS 和 HTTP 协议都在第 19 秒内才开始，这是因为在运行时没有及时关闭其他使用网络的进程，导致其他进程访问网络时过早使用 ARP 协议，这是以后需要注意的影响因素。

- 用了几个连接（建立连接时，本地的端口不同，HTTP 服务常用端口是 80，HTTPS 服务常用端口是 443）；

在进行 DNS 协议查询时，其使用的是基于 UDP 的无连接服务，端口号并没有指定规律，只需要提供服务方的固定端口，例如：

查询 www.xjtu.edu.cn 时主机端口 52445，目的端口 53；

查询 alumni.xjtu.edu.cn 时主机端口 58225，目的端口 53。

而对于 HTTP 协议，由于其是运行在 TCP 上的有连接服务，因此会有一定规则。在访问 www.xjtu.edu.cn 的过程中，首先是发送 GET 命令获得基本 html 文件，如图 4.1.2.1。

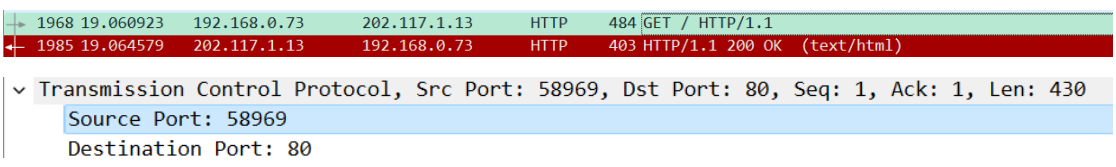


图 4.1.2.1

客户机从主机的 58969 端口发送至服务方 80 端口，在获得基本 html 文件后采用流水线的方式连续发送数条 GET 命令，每一条都从客户机不同端口发出而向服

务器同一端口，以进行流水线作业，如图 4.1.2.2。

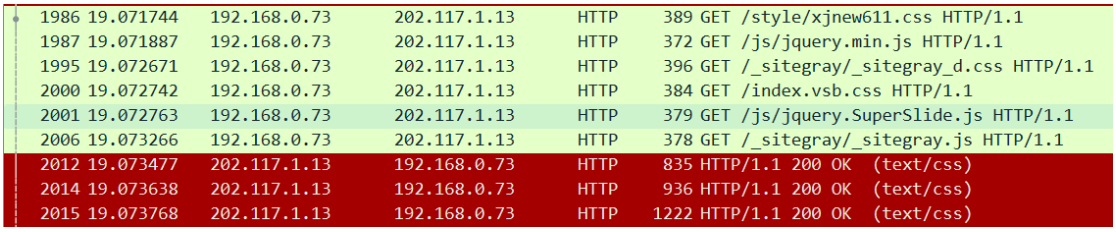


图 4.1.2.2

请求报文的发送端口从 58969 到 58974，而对应回复报文也会采用相同的端口号作为目的端口，通过对之后报文查询，发现主机对此最高同时开 6 个端口（即 58969 到 58974）。以端口 58970 发送接收的报文如图 4.1.2.3 和图 4.1.2.4。

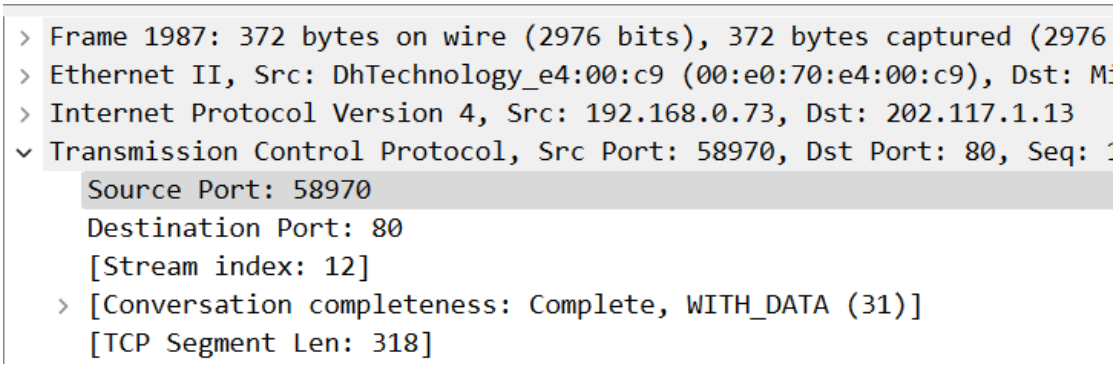


图 4.1.2.3

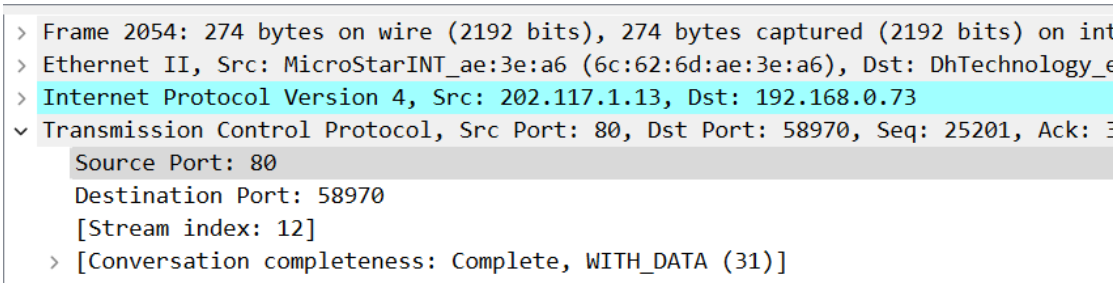
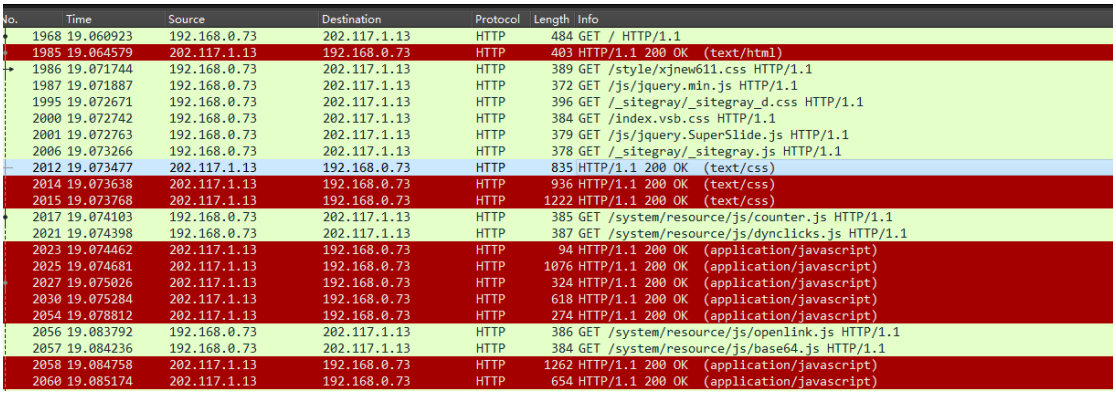


图 4.1.2.4

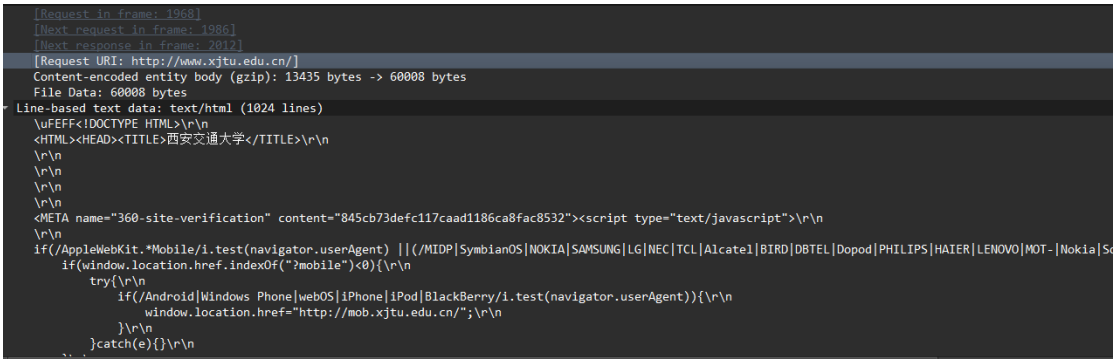
- 取了几个对象（GET 的对象，如：HTML，CSS，JS，图片等）；

对 www.xjtu.edu.cn 的访问中共取了 7 种对象，分别是：HTML、css、javascript、PNG、JPEG JFIF image、GIF89a 以及 x-icon。部分可见图 4.1.3.1。
HTTP 取对象是这样运行的：首先请求主页（GET / HTTP/1.1），得到回复(HTTP/1.1 200 OK (text/html))，该回复包含主页的 html 文件（如图 4.1.3.2）。客户端在解析 html 文件时会发现有超链接，此时即会根据超链接递归地向服务器发送请求，服务器收到请求后会发送相应资源并作回复。例如客户端发送 GET /style/xjnew611.css HTTP/1.1\r\n，服务器响应 HTTP/1.1 200 OK (text/css)。



No.	Time	Source	Destination	Protocol	Length	Info
1968	19.060923	192.168.0.73	202.117.1.13	HTTP	484	GET / HTTP/1.1
1985	19.064579	202.117.1.13	192.168.0.73	HTTP	403	HTTP/1.1 200 OK (text/html)
1986	19.071744	192.168.0.73	202.117.1.13	HTTP	389	GET /style/xjnew611.css HTTP/1.1
1987	19.071887	192.168.0.73	202.117.1.13	HTTP	372	GET /js/jquery.min.js HTTP/1.1
1995	19.072671	192.168.0.73	202.117.1.13	HTTP	396	GET /_sitegray/_sitegray_d.css HTTP/1.1
2000	19.072742	192.168.0.73	202.117.1.13	HTTP	384	GET /index.vsb.css HTTP/1.1
2001	19.072763	192.168.0.73	202.117.1.13	HTTP	379	GET /js/jquery.SuperSlide.js HTTP/1.1
2006	19.073266	192.168.0.73	202.117.1.13	HTTP	378	GET /_sitegray/_sitegray.js HTTP/1.1
2012	19.073477	202.117.1.13	192.168.0.73	HTTP	835	HTTP/1.1 200 OK (text/css)
2014	19.073638	202.117.1.13	192.168.0.73	HTTP	936	HTTP/1.1 200 OK (text/css)
2015	19.073768	202.117.1.13	192.168.0.73	HTTP	1222	HTTP/1.1 200 OK (text/css)
2017	19.074103	192.168.0.73	202.117.1.13	HTTP	385	GET /system/resource/js/counter.js HTTP/1.1
2021	19.074398	192.168.0.73	202.117.1.13	HTTP	387	GET /system/resource/js/dynclicks.js HTTP/1.1
2023	19.074462	202.117.1.13	192.168.0.73	HTTP	94	HTTP/1.1 200 OK (application/javascript)
2025	19.074681	202.117.1.13	192.168.0.73	HTTP	1076	HTTP/1.1 200 OK (application/javascript)
2027	19.075026	202.117.1.13	192.168.0.73	HTTP	324	HTTP/1.1 200 OK (application/javascript)
2030	19.075284	202.117.1.13	192.168.0.73	HTTP	618	HTTP/1.1 200 OK (application/javascript)
2054	19.078812	202.117.1.13	192.168.0.73	HTTP	274	HTTP/1.1 200 OK (application/javascript)
2056	19.083792	192.168.0.73	202.117.1.13	HTTP	386	GET /system/resource/js/openlink.js HTTP/1.1
2057	19.084236	192.168.0.73	202.117.1.13	HTTP	384	GET /system/resource/js/base64.js HTTP/1.1
2058	19.084758	202.117.1.13	192.168.0.73	HTTP	1262	HTTP/1.1 200 OK (application/javascript)
2060	19.085174	202.117.1.13	192.168.0.73	HTTP	654	HTTP/1.1 200 OK (application/javascript)

图 4.1.3.1



```
[Request in frame: 1968]
[Next request in frame: 1986]
[Next response in frame: 2012]
[Request URI: http://www.xjtu.edu.cn/]
Content-encoded entity body (gzip): 13435 bytes -> 60008 bytes
File Data: 60008 bytes
Line-based text data: text/html (1024 lines)
\uFEFF<!DOCTYPE HTML>\r\n
<HTML><HEAD><TITLE>西安交通大学</TITLE>\r\n
\r\n
\r\n
\r\n
\r\n
<META name="360-site-verification" content="845cb73defc117caad1186ca8fac8532"><script type="text/javascript">\r\n
\r\n
if(/AppleWebKit.*Mobile/i.test(navigator.userAgent) ||(/MIDP|SymbianOS|NOKIA|SAMSUNG|LG|NEC|TCL|Alcatel|BIRD|DBTEL|Dopod|PHILIPS|HAIER|LENOVO|MOT-|Nokia|Sc
    if(window.location.href.indexOf("?mobile")<0){\r\n
        try{\r\n
            if(/Android|Windows Phone|webOS|iPhone|iPod|BlackBerry/i.test(navigator.userAgent)){\r\n
                window.location.href="http://mob.xjtu.edu.cn/";\r\n
            }\r\n
        }catch(e){}\r\n
```

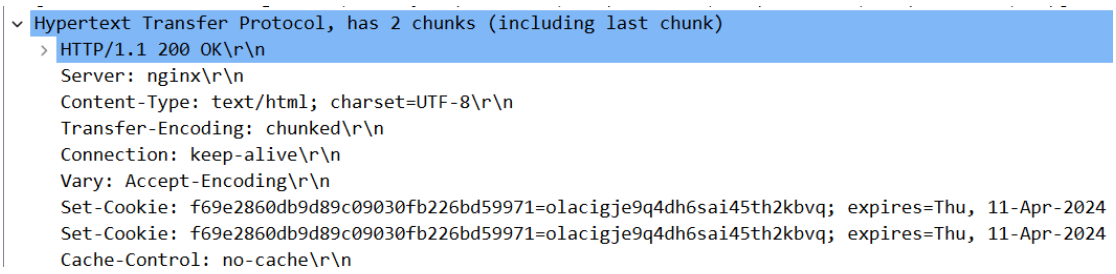
图 4.1.3.2

- 有没有 Cookie 及其工作过程（Set-Cookie、Cookie、304 Not Modified 相关数据包）等；

HTTPS 的加密内容不好分析，可观察 TLS 加密传输的建立过程，传输端口等。

注：TLS 加密传输的分析见进阶自设计处。

第一次未成功捕捉到 Cookie 工作过程，事实上，最多捕获到建立 Cookie 的过程，因为我们已经清除了相应缓存。图 4.1.4.1 所示包为之后捕捉，set cookie 由服务器分配给主机，里面包含了数个参数：name-value 用于标识主机，expires 为此 cookie 的过期时间，max-age 为其有效时间，path 为这个 cookie 在该路径下有效，domain 指该 cookie 的有效域名，最后标注仅在 HTTP 下有效（HTTPONLY）。



```
Hypertext Transfer Protocol, has 2 chunks (including last chunk)
  HTTP/1.1 200 OK\r\n
  Server: nginx\r\n
  Content-Type: text/html; charset=UTF-8\r\n
  Transfer-Encoding: chunked\r\n
  Connection: keep-alive\r\n
  Vary: Accept-Encoding\r\n
  Set-Cookie: f69e2860db9d89c09030fb226bd59971=olacigje9q4dh6sai45th2kbvq; expires=Thu, 11-Apr-2024
  Set-Cookie: f69e2860db9d89c09030fb226bd59971=olacigje9q4dh6sai45th2kbvq; expires=Thu, 11-Apr-2024
  Cache-Control: no-cache\r\n
```

图 4.1.4.1

（二）带缓存的 ARP，DNS 和 HTTP 协议分析

照着 1.7.1 中的步骤 1-4 再次执行一遍，但不执行步骤 2。观察缓存的使用和带来的好处。

再次执行，发现网址打开速度变快，抓包如图 4.2.1（只特地给出 HTTP 报文）。发现出现了 Cookie 信息，如图 4.2.2。

仍然出现了相关的 ARP 和 DNS 查询报文，但 ARP 查询时间较为靠后且 DNS 查询数量较先前略有减少。事实上，即使有缓存，网络通信仍然需要进行有效的地址解析和域名解析，因为缓存可能会过期，或者某些数据可能在缓存中不存在。尽管缓存可以减少对网络协议的查询频率，但它并不能完全消除对 ARP 和 DNS 查询包的需要。

从图 4.2.1 中可见客户端发出的 HTTP 请求数量大大减少，这是因为所需要的资源未修改，可以从缓存中获取。

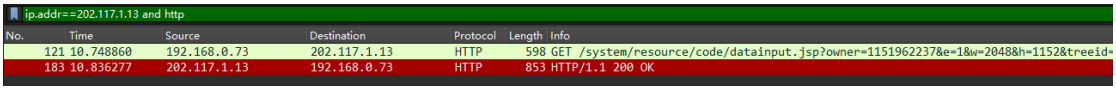


图 4.2.1

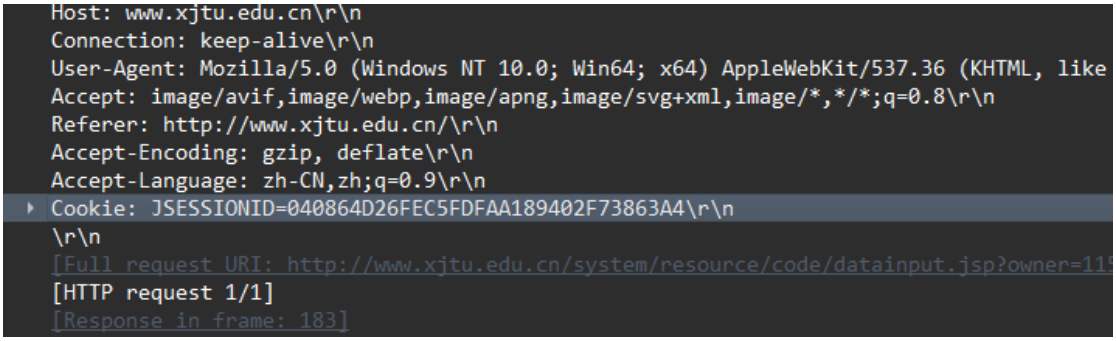


图 4.2.2

有时会在请求时抓到 304 Not Modified，表示所请求的资源自上次请求以来未被修改（此处在宿舍再次测试），见图 4.2.3。具体说明可以参见图 4.2.4。

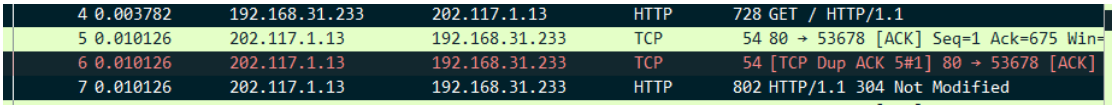


图 4.2.3

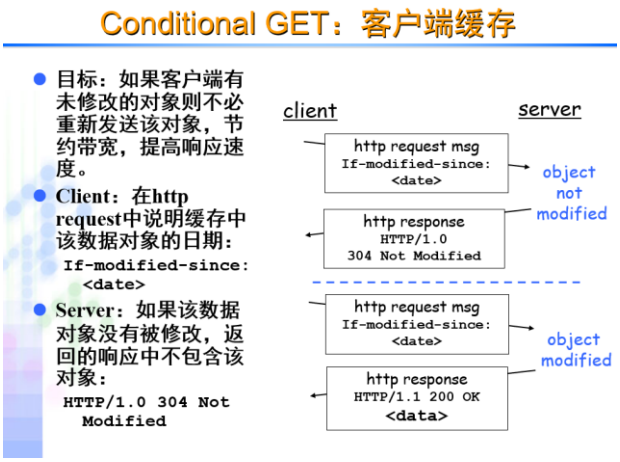


图 4.2.4

（三）使用 ncat 工具访问 HTTP 服务

参考 1.7.1 中的步骤 1-4 和分析结果，在命令窗口执行 ncat -C xxx.xxx.xxx.xxx 80，ncat 连接上 HTTP 服务器后，根据协议输入合适的请求。其中 xxx.xxx.xxx.xxx 为服务器地址。

访问 www.xjtu.edu.cn，请求主页.值得注意的是在 HTTP/1.1 中，必须提供 Host:标头。命令如下：

```
ncat -C www.xjtu.edu.cn 80
GET / HTTP/1.1
Host: www.xjtu.edu.cn
```

同时，我们使用 TCPDUMP 命令捕获了报文便于分析，命令如下：

```
sudo tcpdump -w record.pcapng
```

相关内容见图 4.3.1。

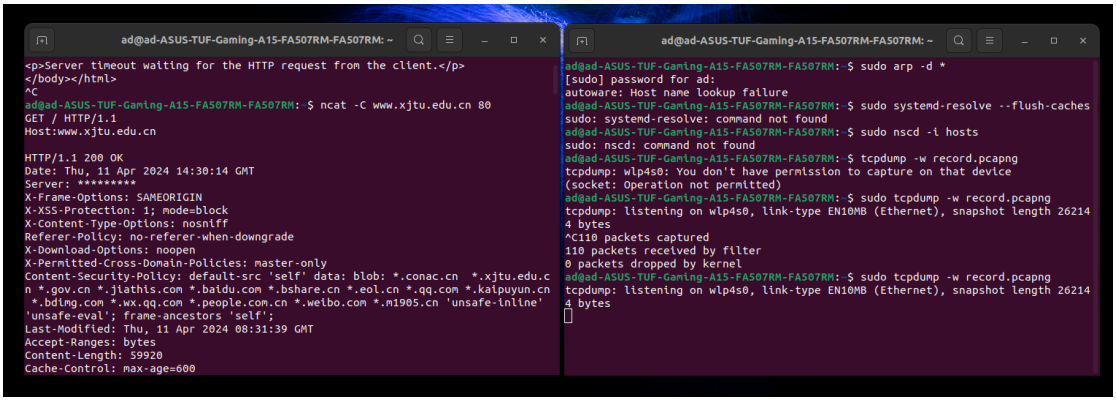


图 4.3.1

使用 ncat 工具访问的数据包和使用浏览器的数据包类似，也先进行了 DNS 查询见图 4.3.2。

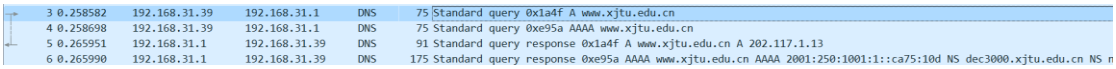


图 4.3.2

之后客户端发送 GET 请求，服务器回复。图 4.3.3 仅给出请求内容。

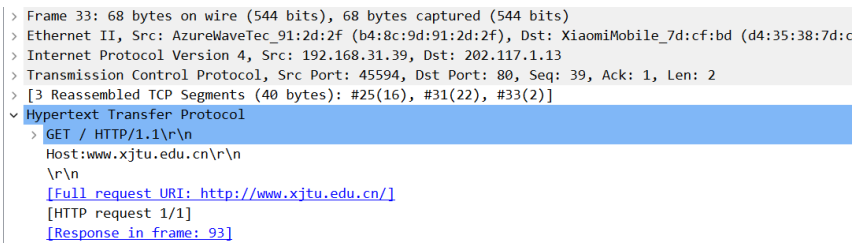


图 4.3.3

六、 互动讨论主题

1、HTTP 协议的缓存，DNS 的缓存；缓存对网络访问速度的影响。

（a）HTTP 协议的缓存，当浏览器或客户端请求某个资源时，服务器可以在响应中添加缓存控制头来指示该资源是否可以被缓存以及缓存的有效期。如果资源被缓存，并且在缓存有效期内再次请求该资源时，客户端可以直接从本地缓存获取该资源，而不需要再次请求服务器。这样可以减少网络传输的数据量和请求的响应时间，提高页面加载速度。

（b）DNS 的缓存则是指在解析域名时，DNS 服务器会将解析结果缓存在本地缓存中。当再次解析相同的域名时，就可以直接从本地缓存获取解析结果，而不需要再次进行完整的域名解析过程。这样可以减少域名解析的时间，加快访问速度。

（c）两种缓存虽然原理不同，但操作上都可以提高网络访问速度，减少响应时间和数据传输量。但同时，缓存也可能导致数据的更新延迟或者从缓存中获取到过期的数据。

2、NAT 对 FTP 传输的影响，比较 HTTP 与 FTP 的特点；

（a）NAT 对 FTP 传输的影响：

（i）**主动模式 FTP**：FTP 使用两个连接来传输文件，一个用于命令和控制（控制连接），另一个用于文件数据传输（数据连接）。在主动模式下，客户端连接到 FTP 服务器的控制端口（通常是端口 21），并向服务器发出命令来打开数据连接。然后，服务器将连接回客户端的数据端口（通常是端口大于 1023），以传输文件数据。在 NAT 环境中，FTP 服务器尝试连接回客户端的数据端口时可能会遇到问题，因为 NAT 设备不知道如何将外部连接转发到内部客户端，除非事先进行了端口映射。

（ii）**被动模式 FTP**：在被动模式下，客户端连接到 FTP 服务器的控制端口（通常是端口 21），并发送 PASV 命令。服务器响应并告知客户端要连接的数据端口（通常是一个高端口）。然后，客户端发起数据连接到服务器指定的端口。被动模式 FTP 更适合在 NAT 环境中使用，因为数据连接是由客户端发起的，不需要服务器连接回客户端的端口。

（b）HTTP 与 FTP 的特点比较：

（i）**HTTP**：

- 无状态协议，每个 HTTP 请求和响应都是独立的，不保留会话状态。
- 通常使用单个 TCP 连接来传输请求和响应。
- 通常在端口 80 上运行，并且易于在 NAT 环境中进行配置。
- 传输的数据通常是短期请求的网页、图像等文档，适合传输小型文件。

（ii）**FTP**：

- 有状态协议，它使用控制连接和数据连接来传输文件，并保持会话状态。

• 通常需要使用两个 TCP 连接来传输文件，这在 NAT 环境中可能会更具挑战性。
• 服务器通常在端口 21 上运行，数据连接则在不同的端口上。
• 更适用于大文件传输和需要复杂操作的文件管理，如上传、下载、重命名等。

七、 进阶自设计

1、用 nmap 的 ncat 来模拟 https 客户端，访问 1-2 个网站。

为了使用 ncat 模拟 https 客户端，在前文 http 的基础上我们还需要加入 ssl 协议，且相比于 HTTP 所在的 80 端口，HTTPS 迁移到了 443 端口，因此命令为：

```
ncat -ssl -C www.casdn.net 443
GET / HTTP/1.1
Host:www.csdn.net
```

使用 tcpdump 工具进行抓包分析：同样是先使用 DNS 查询，见图 7.1.1。

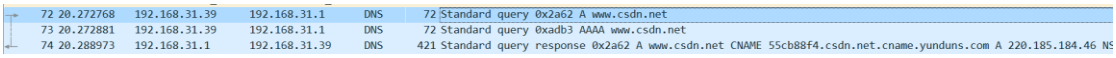


图 7.1.1

在获得对应 IP 地址后，主机通过 TLS（安全传输）向目标发送数据包，可以看到是 client hello，即请求连接，见图 7.1.2。

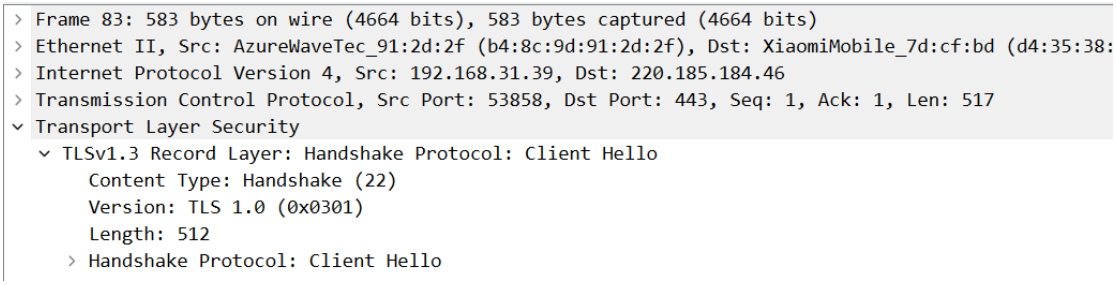


图 7.1.2

然后服务器使用同协议回复了连接请求，其中包含 HTTPS，并设置了一些内容，见图 7.1.3。

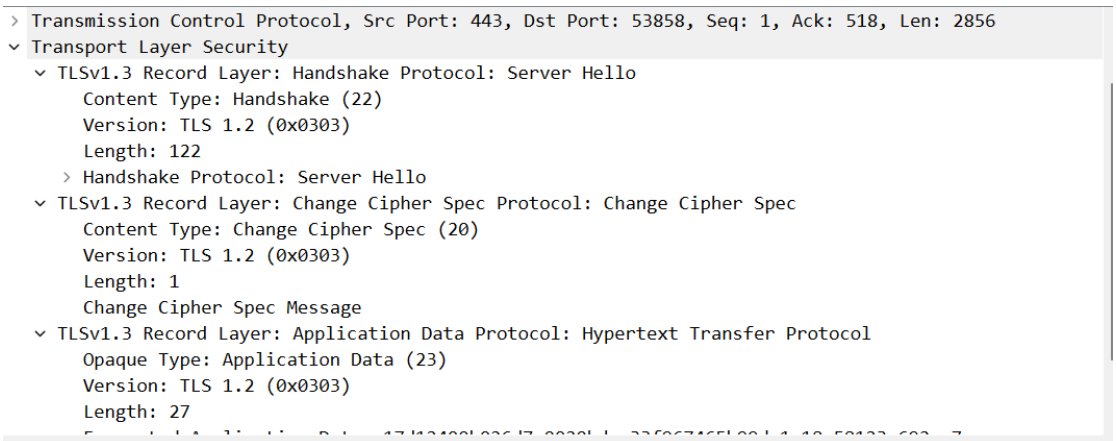


图 7.1.3

之后主要内容也通过 TLS 传输，见图 7.1.4。

85	20.416629	220.185.184.46	192.168.31.39	TLSv1…	2922	Server Hello, Change Cipher Spec, Application Data
86	20.416683	192.168.31.39	220.185.184.46	TCP	66	53858 → 443 [ACK] Seq=518 Ack=2857 Win=65536 Len=0 TSval=3886002901 TSecr=3161384477
87	20.416699	220.185.184.46	192.168.31.39	TLSv1…	786	Application Data, Application Data, Application Data
88	20.416709	192.168.31.39	220.185.184.46	TCP	66	53858 → 443 [ACK] Seq=518 Ack=3577 Win=65536 Len=0 TSval=3886002901 TSecr=3161384477
89	20.418979	192.168.31.39	220.185.184.46	TLSv1…	146	Change Cipher Spec, Application Data
90	20.475509	220.185.184.46	192.168.31.39	TCP	66	443 → 53858 [ACK] Seq=3577 Ack=598 Win=1536 Len=0 TSval=3161384543 TSecr=3886002903
91	20.475548	220.185.184.46	192.168.31.39	TLSv1…	353	Application Data
92	20.475565	220.185.184.46	192.168.31.39	TLSv1…	353	Application Data
93	20.475842	192.168.31.39	220.185.184.46	TCP	66	53858 → 443 [ACK] Seq=598 Ack=4151 Win=65536 Len=0 TSval=3886002960 TSecr=3161384543
94	21.235444	192.168.31.39	220.185.184.46	TLSv1…	104	Application Data
95	21.277301	220.185.184.46	192.168.31.39	TCP	66	443 → 53858 [ACK] Seq=4151 Ack=636 Win=1536 Len=0 TSval=3161385347 TSecr=3886003719

图 7.1.4

在终端上，我们能获得加密前的信息，见图 7.1.5。

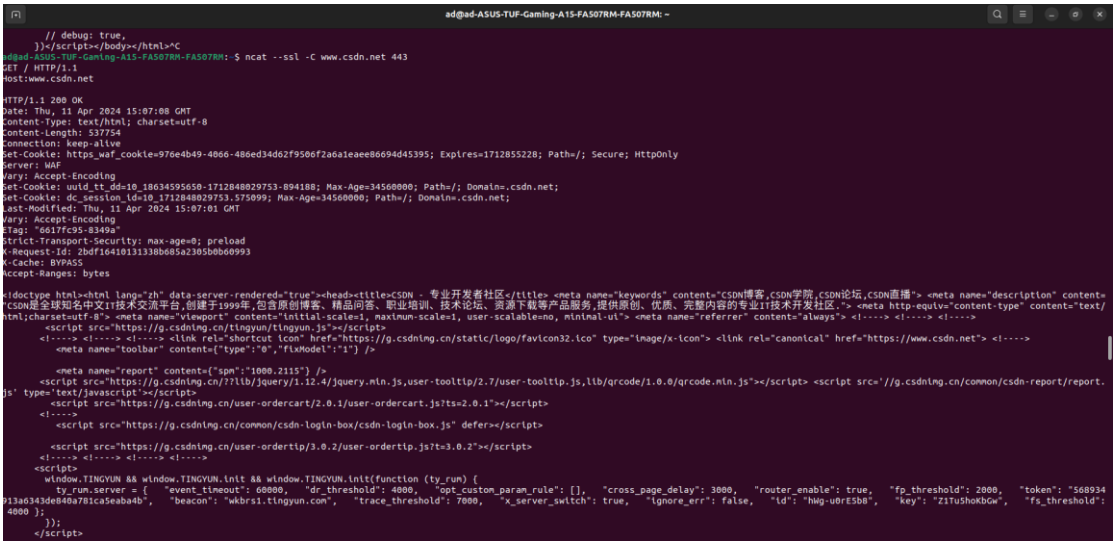


图 7.1.5

其内部信息仍然是 HTTP1.1 格式，可以看到报文前面的协议格式，set-cookie 和主体内容的 HTML 格式内容。

使用相同方法我们还访问了另一个内容更少，因此结构清晰的网页(ys.mihoyo.com)，终端得到的内容见图 7.1.6，抓包得到的内容见图 7.1.7。

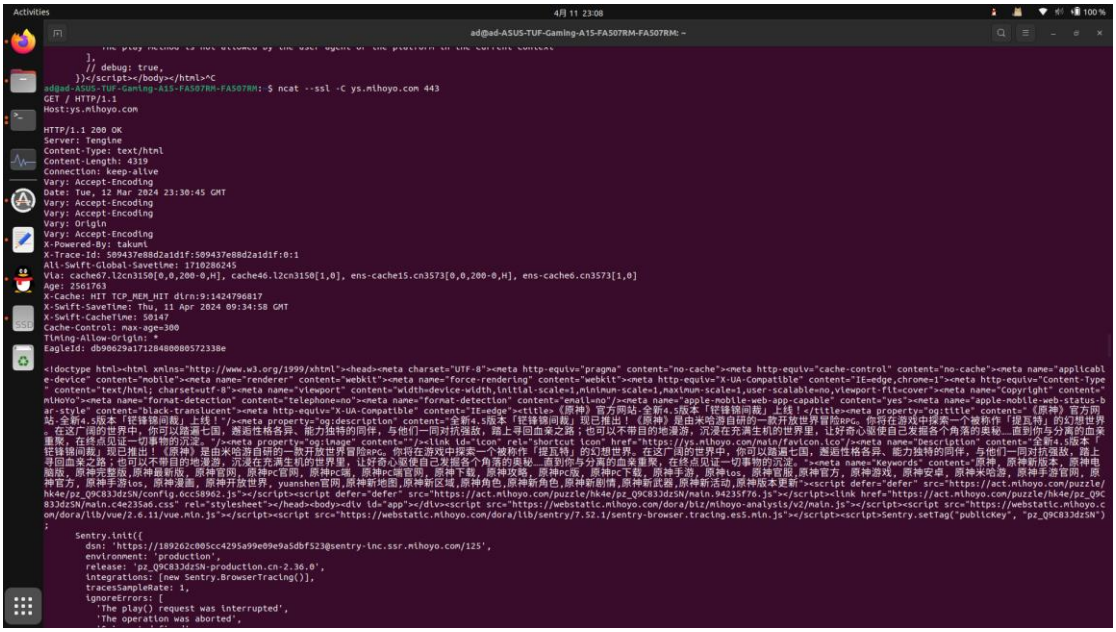


图 7.1.6

图 7.1.6

13	3.652929	192.168.31.39	219.144.98.217	TLSv1…	571 Client Hello (SNI=ys.mihoyo.com)
14	3.659106	219.144.98.217	192.168.31.39	TCP	54 443 → 34962 [ACK] Seq=1 Ack=518 Win=41984 Len=0
15	3.659138	219.144.98.217	192.168.31.39	TLSv1…	1494 Server Hello, Change Cipher Spec, Application Data
16	3.659162	192.168.31.39	219.144.98.217	TCP	54 34962 → 443 [ACK] Seq=518 Ack=1441 Win=65536 Len=0
17	3.668975	219.144.98.217	192.168.31.39	TLSv1…	2199 Application Data, Application Data, Application Data
18	3.669022	192.168.31.39	219.144.98.217	TCP	54 34962 → 443 [ACK] Seq=518 Ack=3586 Win=65536 Len=0
19	3.670598	192.168.31.39	219.144.98.217	TLSv1…	134 Change Cipher Spec, Application Data
20	3.681889	219.144.98.217	192.168.31.39	TLSv1…	349 Application Data
21	3.681927	219.144.98.217	192.168.31.39	TLSv1…	349 Application Data
22	3.682191	192.168.31.39	219.144.98.217	TCP	54 34962 → 443 [ACK] Seq=598 Ack=4176 Win=65536 Len=0
23	4.823378	192.168.31.39	219.144.98.217	TLSv1…	92 Application Data
24	4.870520	219.144.98.217	192.168.31.39	TCP	54 443 → 34962 [ACK] Seq=4176 Ack=636 Win=42496 Len=0

图 7.1.7

2、在云服务器上搭建 Apache2（或其他 WEB 服务器），并测试修改 HTML 或图片文件，看客户端能否及时访问到更新的内容。注意抓包分析。

访问 Apache2 服务器（8.141.1.208）如图 7.2.1 所示，修改 HTML 后访问客户端，见图 7.2.2，发现修改已显示。抓包如图 7.2.3 所示。

可以发现进入 Apache2 服务器主页后客户端向服务器请求了主页内容，服务器返回状态码 200，表示请求已成功。之后二者的交互与前文的分析别无二致。值得注意的是后续进入了/my 文件夹，因此客户机发送 GET /my/ HTTP/1.1 请求，而服务器返回 301 Moved Permanently 状态码，并在响应头中包含一个 Location 字段，指向资源的新位置。以上所述发生在 5~9s 间。

28s 后，重新访问网页时，网页已发送修改。共 3 轮 HTTP 请求与回复。第一轮尝试请求主页，客户机通过回复发现还需要 p.jpeg，于是第二轮即请求该文件（见图箭头）。最后一轮验证是否有更新或错误。

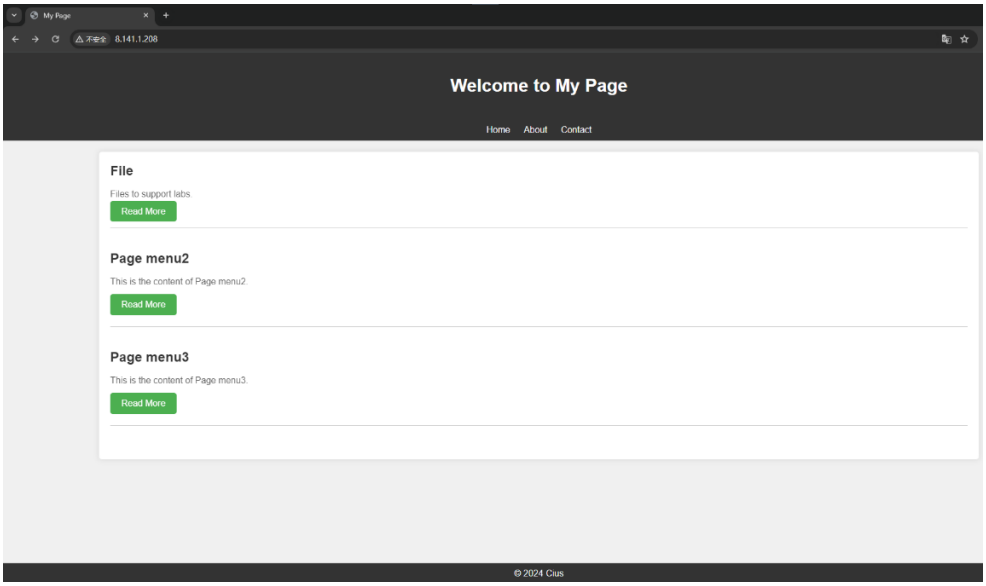


图 7.2.1

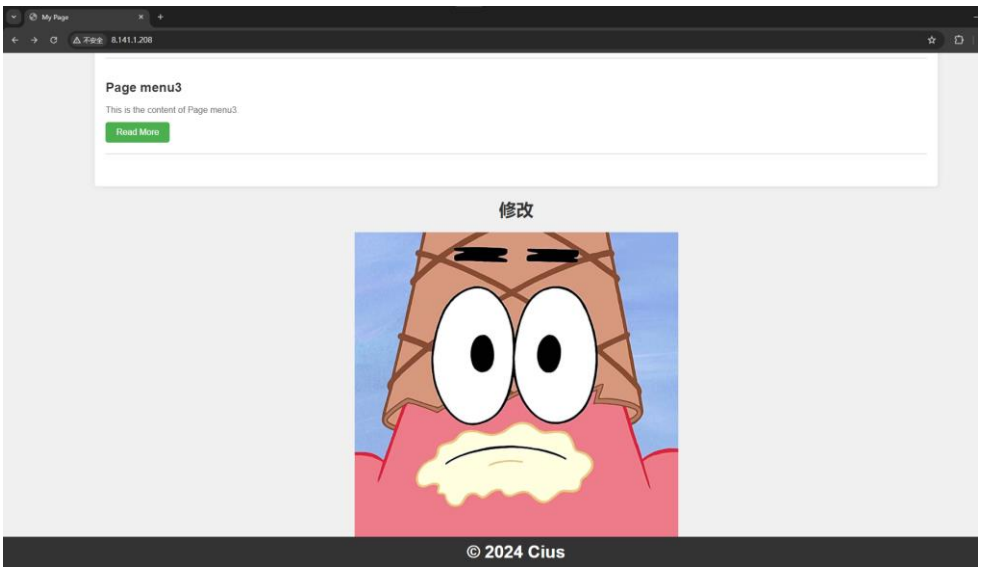


图 7.2.2

ip.addr ==8.141.1.208 and http

No.	Time	Source	Destination	Protocol	Length	Info
117	5.506700	192.168.31.233	8.141.1.208	HTTP	464	GET / HTTP/1.1
130	5.527166	8.141.1.208	192.168.31.233	HTTP	837	HTTP/1.1 200 OK (text/html)
318	5.870049	192.168.31.233	8.141.1.208	HTTP	404	GET /favicon.ico HTTP/1.1
319	5.890924	8.141.1.208	192.168.31.233	HTTP	488	HTTP/1.1 404 Not Found (text/html)
655	8.367360	192.168.31.233	8.141.1.208	HTTP	496	GET /my HTTP/1.1
663	8.389317	8.141.1.208	192.168.31.233	HTTP	564	HTTP/1.1 301 Moved Permanently (text/html)
664	8.393654	192.168.31.233	8.141.1.208	HTTP	497	GET /my/ HTTP/1.1
675	8.413848	8.141.1.208	192.168.31.233	HTTP	853	HTTP/1.1 200 OK (text/html)
676	8.433615	192.168.31.233	8.141.1.208	HTTP	411	GET /icons/blank.gif HTTP/1.1
682	8.453434	8.141.1.208	192.168.31.233	HTTP	430	HTTP/1.1 200 OK (GIF89a)
685	8.453928	192.168.31.233	8.141.1.208	HTTP	410	GET /icons/back.gif HTTP/1.1
692	8.467787	192.168.31.233	8.141.1.208	HTTP	412	GET /icons/binary.gif HTTP/1.1
693	8.467818	192.168.31.233	8.141.1.208	HTTP	413	GET /icons/unknown.gif HTTP/1.1
694	8.467850	192.168.31.233	8.141.1.208	HTTP	412	GET /icons/image2.gif HTTP/1.1
697	8.468882	192.168.31.233	8.141.1.208	HTTP	410	GET /icons/text.gif HTTP/1.1
699	8.473430	8.141.1.208	192.168.31.233	HTTP	498	HTTP/1.1 200 OK (GIF89a)
705	8.487939	8.141.1.208	192.168.31.233	HTTP	527	HTTP/1.1 200 OK (GIF89a)
707	8.487939	8.141.1.208	192.168.31.233	HTTP	592	HTTP/1.1 200 OK (GIF89a)
709	8.487939	8.141.1.208	192.168.31.233	HTTP	528	HTTP/1.1 200 OK (GIF89a)
712	8.489005	8.141.1.208	192.168.31.233	HTTP	511	HTTP/1.1 200 OK (GIF89a)
1484	28.965568	192.168.31.233	8.141.1.208	HTTP	556	GET / HTTP/1.1
1487	28.987993	8.141.1.208	192.168.31.233	HTTP	894	HTTP/1.1 200 OK (text/html)
1489	29.015881	192.168.31.233	8.141.1.208	HTTP	402	GET /my/p.jpeg HTTP/1.1
1714	29.113290	8.141.1.208	192.168.31.233	HTTP	232	HTTP/1.1 200 OK (JPEG JFIF image)
1727	32.423440	192.168.31.233	8.141.1.208	HTTP	497	GET /my/ HTTP/1.1
1729	32.446803	8.141.1.208	192.168.31.233	HTTP	853	HTTP/1.1 200 OK (text/html)

请求 p.jpeg

图 7.2.3