
哈尔滨工业大学

<<计算机网络>>

实验报告

(2019 年度秋季学期)

姓 名	张景润
学 号	1172510217
学 院	计算机科学与技术学院
教 师	刘亚维
实验名称	利用 Wireshark 进行协议分析

目录

一、	实验目的	1
二、	实验环境	1
三、	实验内容与要求	1
四、	实验工具与准备	1
1.	分组嗅探工具	1
2.	Wireshark工具	2
3.	Wireshark使用	2
五、	实验过程与分析	4
1.	HTTP分析	4
2.	TCP分析	7
3.	IP分析（选择知乎官网 www.zhihu.com 用于Ipv4分析）	13
4.	Ethernet抓包分析	15
5.	ARP抓包分析	16
6.	UDP抓包分析	18
7.	DNS协议解析	19
六、	协议总结	21
6.1	HTTP协议	21
6.2	TCP协议	22
6.3	IP协议	22
6.4	Ethernet协议	23
6.5	ARP协议	23
6.6	UDP协议	24
6.7	DNS协议	24
七、	实验心得	25
1.	知识收获	25

一、 实验目的

1. 熟悉并掌握 Wireshark 的基本操作；
2. 了解网络协议实体间进行交互以及报文交换的情况。

二、 实验环境

1. Windows 9x/NT/2000/XP/2003；
2. 与因特网连接的计算机网络系统；
3. Wireshark。

三、 实验内容与要求

1. 学习 Wireshark 的使用；
2. 利用 Wireshark 分析 HTTP 与 TCP 协议；
3. 利用 Wireshark 分析 IP 协议与 Ethernet 数据帧；
4. 利用 Wireshark 分析 DNS 与 UDP 协议（选作内容）；
5. 利用 Wireshark 分析 ARP 协议（选作内容）。

四、 实验工具与准备

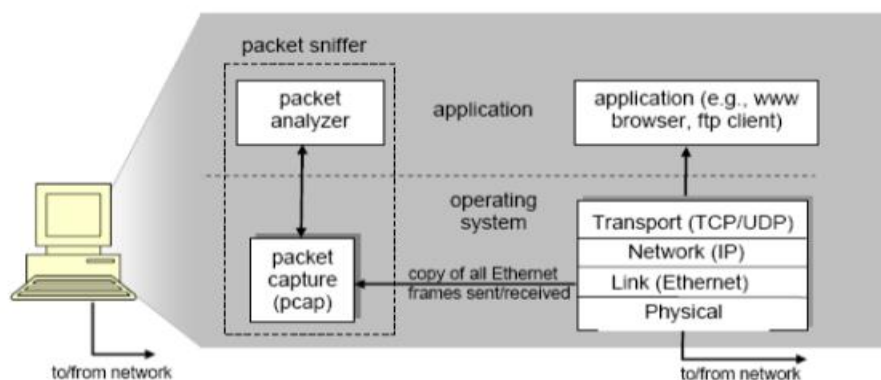
1. 分组嗅探工具

(1) 要深入理解网络协议，需要仔细观察协议实体之间交换的报文序列。为探究协议操作细节，可使协议实体执行某些动作，观察这些动作及其影响。这些任务可以在仿真环境下或在如因特网这样的真实网络环境中完成；

(2) 观察在正在运行协议实体间交换报文的基本工具被称为分组嗅探器。一个分组嗅探器俘获（嗅探）计算机发送和接收的报文。一般情况下，分组嗅探器将存储和显示出被俘获报文的各协议头部字段的内容。

(3) 分组嗅探器是附加计算机普通软件上的，主要有两部分组成。分组俘获库接收计算机发送和接收的每一个链路层帧的拷贝。高层协议（如：HTTP、FTP、TCP、UDP、DNS、IP 等）交换的报文都被封装在链路层帧中，并沿着物理媒体（如以太网的电缆）传输。

(4) 分组嗅探器的第二个组成部分是分析器。分析器用来显示协议报文所有字段的内容。为此，分析器必须能够理解协议所交换的所有报文的结构。分组分析器理解以太网帧格式，能够识别包含在帧中的 IP 数据报。分组分析器也要理解 IP 数据报的格式，并能从 IP 数据报中提取出 TCP 报文段。然后，它需要理解 TCP 报文段，并能够从中提取出 HTTP 消息。最后，它需要理解 HTTP 消息。



分组嗅探器工具

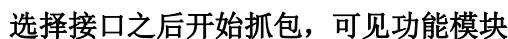
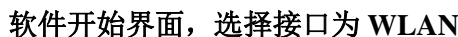
2. Wireshark 工具

(1) Wireshark 是一种可以运行在 Windows, UNIX, Linux 等操作系统上的分组分析器。Wireshark 是免费的，可以从 <https://www.wireshark.org/download.html> 得到，Wireshark 的用户指导可以从 <https://www.wireshark.org/docs/> 获得；

(2) Wireshark（前称 Ethereal）是一个网络封包分析软件。网络封包分析软件的功能是撷取网络封包，并尽可能显示出最为详细的网络封包资料。Wireshark 使用 WinPCAP 作为接口，直接与网卡进行数据报文交换。

3. Wireshark 使用

(1) 启动 Wireshark 软件，打开 Google 浏览器，选择网络接口 WLAN；



The image shows a Wireshark packet capture analysis of an HTTP GET request. The top pane displays a list of captured packets, with packet 35 selected. The middle pane shows the details of this packet, including Ethernet II, Internet Protocol Version 4, and Hypertext Transfer Protocol sections. The bottom pane shows the raw packet data in hexadecimal and ASCII.

No.	Time	Source	Destination	Protocol	Length	Info
35	4.380102	172.20.20.19	219.217.226.17	HTTP	408	GET / HTTP/1.1
78	4.380708	172.20.20.19	219.217.226.17	HTTP	597	GET /_js/_portlet/plugins/simplehues/css/simplehues.css HTTP/1.1
79	4.380939	172.20.20.19	219.217.226.17	HTTP	568	GET /_css/tel2/system.css HTTP/1.1
80	4.380962	172.20.20.19	219.217.226.17	HTTP	591	GET /_js/_portlet/plugins/sudyhavi/css/sudyhavi.css HTTP/1.1
81	4.380983	172.20.20.19	219.217.226.17	HTTP	587	GET /_js/_portlet/plugins/jquery/js/jquery.datapicker.js HTTP/1.1
84	4.380981	172.20.20.19	219.217.226.17	HTTP	571	GET /_css/system/system HTTP/1.1
97	4.380816	219.217.226.17	172.20.20.19	HTTP	189	HTTP/1.1 304 Not Modified
98	4.380816	219.217.226.17	172.20.20.19	HTTP	190	HTTP/1.1 304 Not Modified
101	4.380816	219.217.226.17	172.20.20.19	HTTP	191	HTTP/1.1 304 Not Modified
111	4.380455	219.217.226.17	172.20.20.19	HTTP	191	HTTP/1.1 304 Not Modified

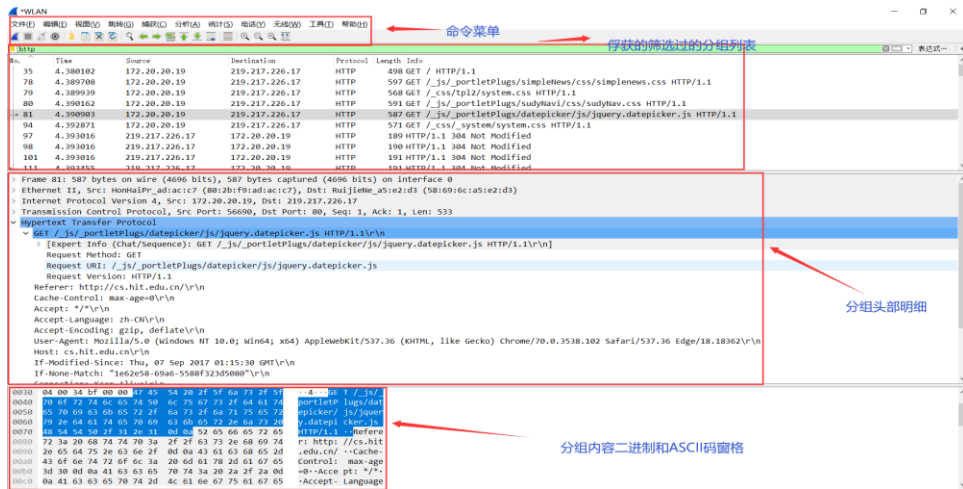
Frame 35: 408 bytes on wire (3084 Bits), 408 bytes captured (3084 Bits) on interface 0
 Ethernet II, Src: RealtekUplinkAdapter (08-00-27-00-00-00), Dst: RealtekUplinkAdapter (58-09-6c-a5-e2-d3)
 Internet Protocol Version 4, Src: 172.20.20.19, Dst: 219.217.226.17
 Transmission Control Protocol, Src Port: 56688, Dst Port: 80, Seq: 1, Ack: 1, Len: 444
 Hypertext Transfer Protocol, Method: GET, Path: /
 Destination Port: 80
 [Stream index: 1]
 [TCP segment len: 444]
 Sequence number: 1 (relative sequence number)
 [next sequence number: 445 (relative sequence number)]
 Acknowledgment number: 1 (relative ack number)
 0181 --- => Header length: 20 bytes (5)
 Flags: 0x018 (PSH, ACK)
 Window size value: 1024
 [calculated window size: 262144]
 [window size scaling factor: 256]
 Checksum: 0xa7ad [unverified]
 [checksum Status: unverified]
 Urgent pointer: 0
 [raw data bytes: 408]

```

0020  e2 11 dd 70 00 80 33 37 81 9d 6f 59 82 1d 18 -- -->P-P- ->ov-
0021  08 00 27 00 00 00 00 00 00 00 00 00 00 00 00 -- -->K-G-E T / H
0022  2f 31 2e 31 0e 0a 43 61 63 68 65 2d 43 6f 6e 7a /1.1 -Ca che-Cont
0023  72 6f 6e 3a 2e 6d 63 78 20 61 67 05 3d 80 8d 0a rol: max -agenu-
0024  41 63 65 70 7a 3a 20 7a 65 78 7a 2f 68 7a 6d Accept: text/html
0025  6e 2c 61 70 2e 6d 63 7a 61 69 6f 6e 2f 78 68 1,applicat/onsh
0026  7a 6e 3a 78 6d 6e 2c 61 70 70 6e 69 63 61 7a text/xml; applicat
0027  69 6f 6e 2f 78 6d 6e 3b 71 3d 30 20 39 2c 2a 2f ion/xml; q=0.9,/
0028  2a 20 71 3d 30 2e 38 8d 01 41 63 65 05 7a 2d *q=0.8 -Accept-
0029  4c 03 6e 67 7e 61 67 65 3a 20 7a 68 0d 43 4d Language: zh-CN
  
```

抓包得到相应的报文

(3) **详细用户界面**：命令菜单、俘获分组列表、分组头部明细、分组内容窗口、筛选俘获分组。

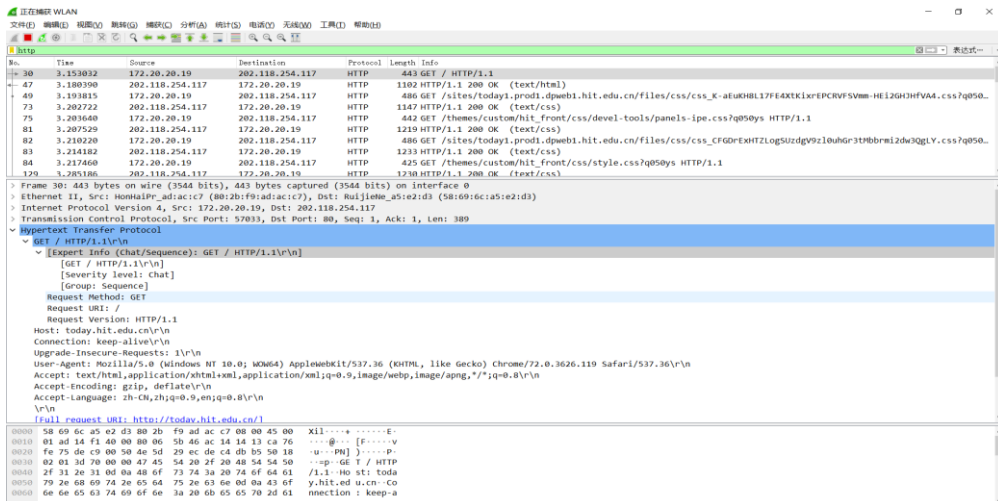


详细用户界面

五、实验过程与分析

1. HTTP 分析

注：浏览器中输入的网址为：<http://today.hit.edu.cn/>（因参考书中网址无效，所以选择了一个哈工大的门户网站）



访问网站抓取的 HTTP 包

(1) HTTP 的 GET/response 交互

a) Google 浏览器运行 [HTTP1.1](#)，服务器运行 HTTP 版本号为 [HTTP1.1](#);

```

Hypertext Transfer Protocol
  GET / HTTP/1.1\r\n
    [Expert Info (Chat/Sequence): GET / HTTP/1.1\r\n]
    Request Method: GET
    Request URI: /
    Request Version: HTTP/1.1
    Host: today.hit.edu.cn\r\n

```

HTTP 请求报文反应浏览器的协议版本

```

Hypertext Transfer Protocol
  HTTP/1.1 200 OK\r\n
    [Expert Info (Chat/Sequence): HTTP/1.1 200 OK\r\n]
    Response Version: HTTP/1.1
    Status Code: 200
    [Status Code Description: OK]
    Response Phrase: OK
    Date: Mon, 04 Nov 2019 06:34:03 GMT\r\n
    Server: Apache/2.4.18 (Ubuntu)\r\n
    X-Content-Type-Options: nosniff\r\n

```

HTTP 响应报文反应服务器的协议版本

b) Google 浏览器向服务器指出可接受语言版本为 [zh-CN](#)，[简体中文](#);

```

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8\r\n
Accept-Encoding: gzip, deflate\r\n
Accept-Language: zh-CN,zh;q=0.9,en;q=0.8\r\n
\r\n

```

接受的语言为 zh-CN

c) 本机 IP 地址为 [172.20.20.19](#)，服务器<http://today.hit.edu.cn/>的IP地址为 [202.118.254.117](#);

```

Internet Protocol Version 4, Src: 172.20.20.19, Dst: 202.118.254.117
  0100 .... = Version: 4
  .... 0101 = Header Length: 20 bytes (5)
  > Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
  Total Length: 429
  Identification: 0x14f1 (5361)
  > Flags: 0x4000, Don't fragment
  Time to live: 128
  Protocol: TCP (6)
  Header checksum: 0x5b46 [validation disabled]
  [Header checksum status: Unverified]
  Source: 172.20.20.19
  Destination: 202.118.254.117

```

本机 IP 和目的 IP

d) 浏览器返回的状态代码为 [200](#)。

```

Hypertext Transfer Protocol
  HTTP/1.1 200 OK\r\n
    [Expert Info (Chat/Sequence): HTTP/1.1 200 OK\r\n]
    Response Version: HTTP/1.1
    Status Code: 200
    [Status Code Description: OK]
    Response Phrase: OK

```

服务器返回状态码 200

(2) HTTP 条件 GET/response 交互

a) 第一个 GET 请求在请求报文中无 IF-MODIFIED-SINCE，服务器明确返回了文件内容，可以通过状态码和数据段感知；

```

Hypertext Transfer Protocol
  GET / HTTP/1.1\r\n
    [Expert Info (Chat/Sequence): GET / HTTP/1.1\r\n]
      [GET / HTTP/1.1\r\n]
      [Severity level: Chat]
      [Group: Sequence]
    Request Method: GET
    Request URI: /
    Request Version: HTTP/1.1
    Host: today.hit.edu.cn\r\n
    Connection: keep-alive\r\n
    Upgrade-Insecure-Requests: 1\r\n
    User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/72.0.3626.119 Safari/537.36\r\n
    Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8\r\n
    Accept-Encoding: gzip, deflate\r\n
    Accept-Language: zh-CN,zh;q=0.9,en;q=0.8\r\n
    \r\n
  
```

无: IF-MODIFIED-SINCE请求行

无相应的请求行

```

Hypertext Transfer Protocol
  HTTP/1.1 200 OK\r\n
    [Expert Info (Chat/Sequence): HTTP/1.1 200 OK\r\n]
      [HTTP/1.1 200 OK\r\n]
      [Severity level: Chat]
      [Group: Sequence]
    Response Version: HTTP/1.1
    Status Code: 200
  
```

状态码为200表明返回

明确返回了信息

```

Content-encoded entity body (gzip): 20955 bytes -> 195775 bytes
File Data: 195775 bytes
Line-based text data: text/html (4510 lines)
  <!DOCTYPE html>\n
  [truncated]<html lang="zh-hans" dir="ltr" prefix="content: http://purl.org/rss/1.0/modules/content/"
  <head>\n
  
```

响应报文中携带了数据

携带了网页文件内容

b) 向发出的较晚 GET 请求中，有该行: IF-MODIFIED-SINCE，且该行后面的信息是本地缓存文件中 Last-Modified 字段的最后修改时间；

```

Hypertext Transfer Protocol
  GET /sites/today1.prod1.dpweb1.hit.edu.cn/files/images/2019/04/10/zqyw.jpg HTTP/1.1\r\n
    [Expert Info (Chat/Sequence): GET /sites/today1.prod1.dpweb1.hit.edu.cn/files/images/
      [GET /sites/today1.prod1.dpweb1.hit.edu.cn/files/images/2019/04/10/zqyw.jpg HTTP/1.
      [Severity level: Chat]
      [Group: Sequence]
    Request Method: GET
    Request URI: /sites/today1.prod1.dpweb1.hit.edu.cn/files/images/2019/04/10/zqyw.jpg
    Request Version: HTTP/1.1
    Referer: http://today.hit.edu.cn/\r\n
    User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like G
    Cache-Control: max-age=0\r\n
    Accept: image/png,image/svg+xml,image/*;q=0.8,*/*;q=0.5\r\n
    Accept-Language: zh-CN\r\n
    Accept-Encoding: gzip, deflate\r\n
    Host: today.hit.edu.cn\r\n
    If-Modified-Since: Tue, 09 Apr 2019 23:52:22 GMT\r\n
    If-None-Match: "19b92-58621a4e57844"\r\n
    Connection: Keep-Alive\r\n
  
```

请求报文中该行


```

Hypertext Transfer Protocol
  HTTP/1.1 304 Not Modified\r\n
    [Expert Info (Chat/Sequence): HTTP/1.1 304 Not Modified\r\n]
    [HTTP/1.1 304 Not Modified\r\n]
    [Severity level: Chat]
    [Group: Sequence]
    Response Version: HTTP/1.1
    Status Code: 304
    [Status Code Description: Not Modified]
    Response Phrase: Not Modified
    Date: Mon, 04 Nov 2019 07:28:55 GMT\r\n
    Server: Apache/2.4.18 (Ubuntu)\r\n
    ETag: "19b92-58621a4e57844"\r\n
    X-Content-Type-Options: nosniff\r\n
    Last-Modified: Tue, 09 Apr 2019 23:52:22 GMT\r\n
    Content-Type: image/jpeg\r\n

```

该时间和请求报文中的
时间一致

响应报文返回 304

c) 服务器对较晚的请求响应状态码是 304，并未返回了文件内容，原因是：客户端在找到目的主机 URL 的本地缓存后，会构造请求报文确认该缓存是否未更新，若未更新，则无需向客户端发送响应的数据，而客户端直接从本地缓存读取文件即可（注：服务器端若已更新缓存文件时，会返回 200，并携带数据）。

```

Hypertext Transfer Protocol
  HTTP/1.1 304 Not Modified\r\n
    [Expert Info (Chat/Sequence): HTTP/1.1 304 Not Modified\r\n]
    [HTTP/1.1 304 Not Modified\r\n]
    [Severity level: Chat]
    [Group: Sequence]
    Response Version: HTTP/1.1
    Status Code: 304
    [Status Code Description: Not Modified]
    Response Phrase: Not Modified
    Date: Mon, 04 Nov 2019 07:28:55 GMT\r\n
    Server: Apache/2.4.18 (Ubuntu)\r\n
    ETag: "19b92-58621a4e57844"\r\n
    X-Content-Type-Options: nosniff\r\n
    Last-Modified: Tue, 09 Apr 2019 23:52:22 GMT\r\n
    Content-Type: image/jpeg\r\n
    X-Varnish: 1016483864 1016932815\r\n
    Age: 12\r\n
    Via: 1.1 varnish-v4\r\n
    X-Varnish-Cache: HIT\r\n
    Connection: keep-alive\r\n
    \r\n
    [HTTP response 6/6]
    [Time since request: 0.003030000 seconds]

```

头部行结束后无数据携带

响应报文并未携带请求数据

2. TCP 分析

(1) 准备工作

下载爱丽丝漫游仙境，并上传文件，抓包分析

Congratulations!

You've now transferred a copy of alice.txt ffrom your computer to gaia.cs.umass.edu. You should now stop Wireshark packet capture. It's time to start analyzing the captured Wireshark packets!

上传成功

No.	Time	Source	Destination	Protocol	Length	Info
20	3.421189	172.20.20.19	128.119.245.12	TCP	66	59834 → 80 [...]
21	3.686636	128.119.245.12	172.20.20.19	TCP	66	80 → 59834 [...]
22	3.686718	172.20.20.19	128.119.245.12	TCP	54	59834 → 80 [...]
23	3.687086	172.20.20.19	128.119.245.12	TCP	666	59834 → 80 [...]
24	3.687165	172.20.20.19	128.119.245.12	TCP	1514	59834 → 80 [...]
25	3.687166	172.20.20.19	128.119.245.12	TCP	1514	59834 → 80 [...]

抓取到的 TCP 报文

(2)问题 1

向服务器传送文件的客户端主机 IP 为 172.20.20.19，TCP 端口号为 59834，服务器的 IP 为 128.119.245.12，发送和接受 TCP 报文的端口号为 80；

Internet Protocol Version 4, Src: 172.20.20.19, Dst: 128.119.245.12
Transmission Control Protocol, Src Port: 59834, Dst Port: 80, Seq: 0, Len: 0
Source Port: 59834
Destination Port: 80
[Stream index: 0]
[TCP Segment Len: 0]
Sequence number: 0 (relative sequence number)
[Next sequence number: 0 (relative sequence number)]
Acknowledgment number: 0
1000 = Header Length: 32 bytes (8)

源 IP 地址和端口号

> Internet Protocol Version 4, Src: 128.119.245.12, Dst: 172.20.20.19
Transmission Control Protocol, Src Port: 80, Dst Port: 59834, Seq: 0, Ack: 1, Len: 0
Source Port: 80
Destination Port: 59834

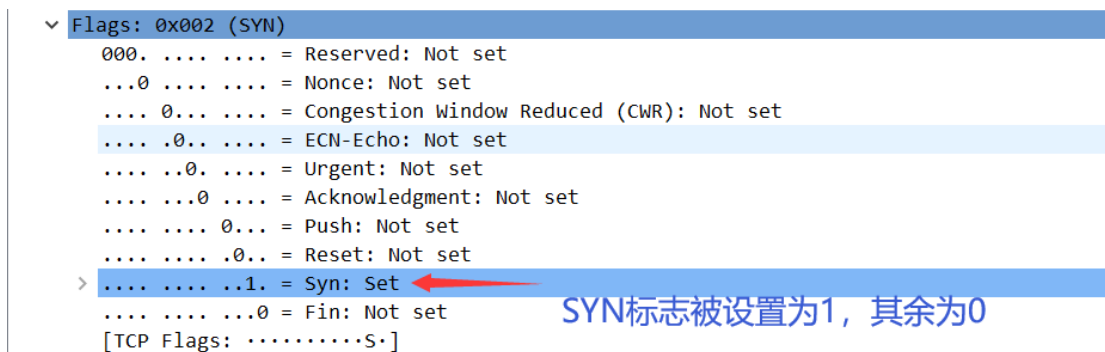
服务器端 IP 和端口号

(3)问题 2

建立连接的 SYN 报文段序号为 906 705 657 (10 进制)，可以通过 SYN 标志位是否为 1 来判断该报文段是否为 SYN 报文段；

[TCP Segment Len: 0]
Sequence number: 0 (relative sequence number)
相对序列号为0，实际值可以通过字节数据看到
0000 58 69 6c a5 e2 d3 80 2b f9 ad ac c7 08 00 45 00 Xil...+E
0010 00 34 b1 fa 40 00 80 06 13 1e ac 14 14 13 80 77 .4..@... ..w
0020 f5 0c e9 ba 00 50 36 0b 3a f9 00 00 00 80 02P6.
0030 ff ff de 55 00 00 02 04 05 b4 01 03 03 08 01 01 ...U... ..
0040 04 02

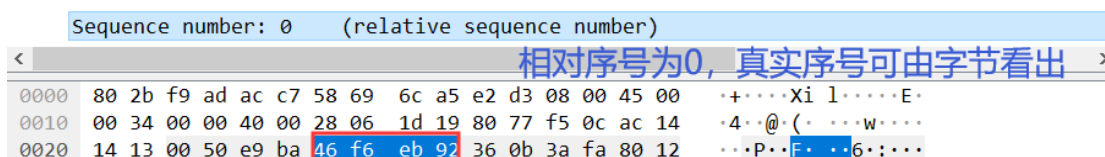
SYN 报文段序号可以通过字节数据看到



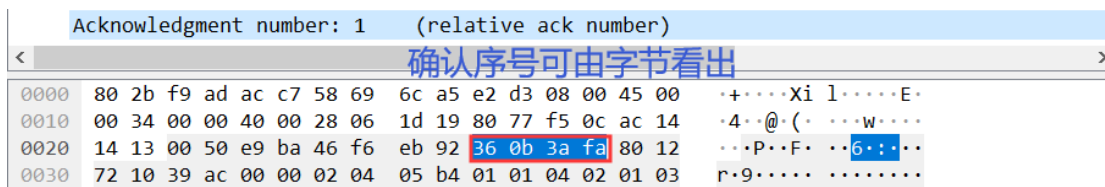
标志此报文段为 SYN 报文段

(4)问题 3

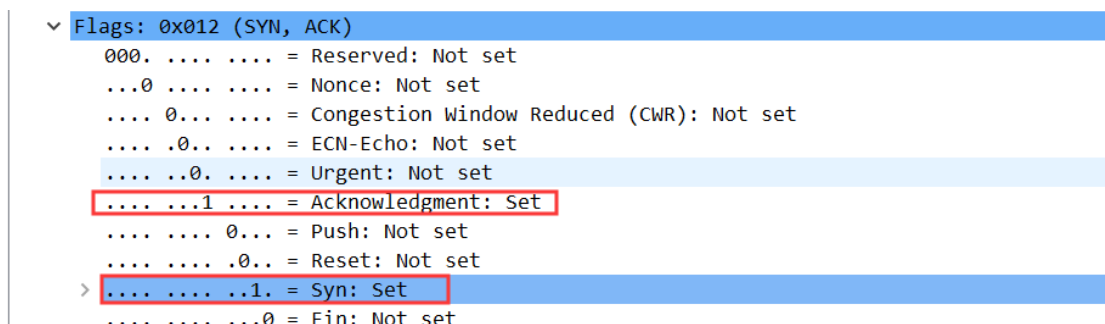
服务器端发送的 SYNACK 报文段序号为 **0x46 f6 eb 92**, Acknowledgement 字段值为 **0x36 0b 3a fa**(刚好为 SYN 报文段序号的值+1),可以通过 SYN 和 ACK 标志位都为 1 标识该报文段;



此报文段序号为 0x46 f6 eb 92



确认序号为 0x36 0b 3a fa



设置 SYN=1&ACK=1

(5)问题 4

可以分析出三次握手, 因为这三三次报文段长度很小, 且相应的标志位进行了设置 SYN、ACK;

Length	Info
66	59834 → 80 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=256 SACK_PERM=1
66	80 → 59834 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1460 SACK_PERM=1 WS=128
54	59834 → 80 [ACK] Seq=1 Ack=1 Win=262144 Len=0

三次握手，标志位的设置

(6)问题 5

包含 POST 命令的 TCP 报文段序号为 0x36 0b 3a fa (刚好为 SYN 报文段序号的值+1);

[TCP Segment Len: 612]	
Sequence number: 1	(relative sequence number) 序号真实值可由字节数据看出
< 0020 f5 0c e9 ba 00 50 36 0b 3a fa 46 f6 eb 93 50 18P6...F...P...	
0030 04 00 79 2d 00 00 50 4f 53 54 20 2f 77 69 72 65 ...y...PO ST/wire	

(7)问题 6

以 POST 命令的第一个 TCP 报文段为首 (即原连接的第四个报文段), 则第六个报文段的序号绝对值为 0x36 0b 54 2e (相对值为 6453), 该报文段于 TCP 三次握手之后 (作为第 9 个 TCP 报文段发送), 四次挥手之前发送的, 该报文段对应的 ACK 确认序号对应的报文是在第三次握手的时候接收的;

No.	Source	Destination	Protocol	Length
20	172.20.20.19	128.119.245.12	TCP	66
21	128.119.245.12	172.20.20.19	TCP	66
22	172.20.20.19	128.119.245.12	TCP	54
23	172.20.20.19	128.119.245.12	TCP	666
24	172.20.20.19	128.119.245.12	TCP	1514
25	172.20.20.19	128.119.245.12	TCP	1514
26	172.20.20.19	128.119.245.12	TCP	1514
27	172.20.20.19	128.119.245.12	TCP	1514
28	172.20.20.19	128.119.245.12	TCP	1514

6 个报文段

[TCP Segment Len: 1460]	
Sequence number: 6453	(relative sequence number) 相对序号
[Next sequence number: 7913 (relative sequence number)]	
< 0020 f5 0c e9 ba 00 50 36 0b 54 2e 46 f6 eb 93 50 10P6...T...F...P...	
0030 04 00 ee 2f 00 00 68 65 61 72 20 69 74 0d 0a 73 .../...he ar it...s	

该报文段相对序号为 6453, 实际为 0x36 0b 54 2e

Acknowledgment number: 1		(relative ack number)
<		>
0020	f5 0c e9 ba 00 50 36 0b 54 2e	46 f6 eb 93 50 10P6...T...F...P...

该报文段对应的 ACK 序号所标识的报文是在第三次握手时接收

Sequence number: 1 (relative sequence number)
[Next sequence number: 1 (relative sequence number)]
Acknowledgment number: 7913 (relative ack number)

第6个报文已收到

```

0000  80 2b f9 ad ac c7 3c f5 cc 86 b7 40 08 00 45 00  +.....<...@...E.
0010  00 28 2b c5 40 00 28 06 f1 5f 80 77 f5 0c ac 14  +(+@.(...w....
0020  14 13 00 50 e9 ba 46 f6 eb 93 36 0b 59 e2 50 10  ...P...F...6.Y.P.

```

第6个报文成功被接受

(8)问题 7

前6个TCP报文段的长度各是：666、1514、1514、1514、1514、1514；

23	3.687086	172.20.20.19	128.119.245.12	TCP	666	59834 → 80
24	3.687165	172.20.20.19	128.119.245.12	TCP	1514	59834 → 80
25	3.687166	172.20.20.19	128.119.245.12	TCP	1514	59834 → 80
26	3.687166	172.20.20.19	128.119.245.12	TCP	1514	59834 → 80
27	3.687167	172.20.20.19	128.119.245.12	TCP	1514	59834 → 80
28	3.687167	172.20.20.19	128.119.245.12	TCP	1514	59834 → 80

前6个报文段的长度

(9)问题 8

整个跟踪过程中，接收端公示最小可用缓存空间是 29200，在前面绝大多数接收方发送的ACK报文中的可用缓存空间是在递增的，只有在最后几个ACK确认报文中的可用缓存空间出现了小幅度波动，最终可用缓存大小达到如截图2所示。我们可以看到当限制了发送方的传输后，接收端的缓存空间开始不断增大直到达到305920。因此得到结论，接收端缓存足够。

Protocol	Length	Info
TCP	66	59834 → 80 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=256 SACK_PERM=1
TCP	66	80 → 59834 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1460 SACK_PERM=1 W
TCP	54	59834 → 80 [ACK] Seq=1 Ack=1 Win=262144 Len=0
TCP	666	59834 → 80 [PSH, ACK] Seq=1 Ack=1 Win=262144 Len=612 [TCP segment of a r
TCP	1514	59834 → 80 [ACK] Seq=613 Ack=1 Win=262144 Len=1460 [TCP segment of a rea
TCP	1514	59834 → 80 [ACK] Seq=2073 Ack=1 Win=262144 Len=1460 [TCP segment of a re

接收端公示的最小缓存空间为 29200

172.20.20.19	TCP	60	80 → 59834 [ACK] Seq=1 Ack=613 Win=30464 Len=0
172.20.20.19	TCP	60	80 → 59834 [ACK] Seq=1 Ack=2073 Win=33408 Len=0
172.20.20.19	TCP	60	80 → 59834 [ACK] Seq=1 Ack=3533 Win=36352 Len=0
172.20.20.19	TCP	60	80 → 59834 [ACK] Seq=1 Ack=4993 Win=39296 Len=0
172.20.20.19	TCP	60	80 → 59834 [ACK] Seq=1 Ack=6453 Win=42112 Len=0
172.20.20.19	TCP	60	80 → 59834 [ACK] Seq=1 Ack=7913 Win=45056 Len=0
172.20.20.19	TCP	60	80 → 59834 [ACK] Seq=1 Ack=10833 Win=50944 Len=0
172.20.20.19	TCP	60	80 → 59834 [ACK] Seq=1 Ack=9373 Win=48000 Len=0
172.20.20.19	TCP	60	80 → 59834 [ACK] Seq=1 Ack=12293 Win=53888 Len=0
172.20.20.19	TCP	60	80 → 59834 [ACK] Seq=1 Ack=13753 Win=56704 Len=0

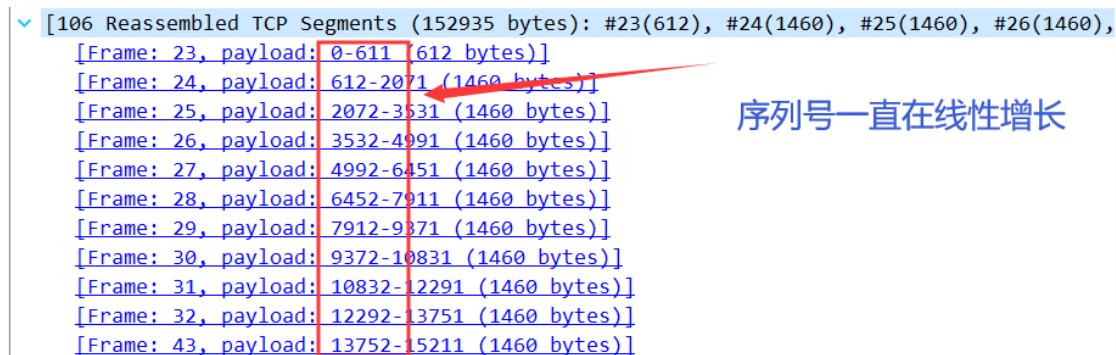
大部分递增，小部分波动

172.20.20.19	TCP	66	80 → 59834 [ACK] Seq=1 Ack=143693 Win=288384 Len=0
172.20.20.19	TCP	66	80 → 59834 [ACK] Seq=1 Ack=146613 Win=294144 Len=0
172.20.20.19	TCP	66	80 → 59834 [ACK] Seq=1 Ack=148073 Win=297088 Len=0
172.20.20.19	TCP	66	80 → 59834 [ACK] Seq=1 Ack=149533 Win=300032 Len=0
172.20.20.19	TCP	60	80 → 59834 [ACK] Seq=1 Ack=152936 Win=305920 Len=0

可用缓存空间最终达到 305920

(10) 问题 9

过程中无重传的报文段，判断依据是：我人工走查了客户端所有的唯一标识一个分组的序列号，发现序列号一直在增长，从未出现重复，因此可以断定无重传报文段。



```

106 Reassembled TCP Segments (152935 bytes): #23(612), #24(1460), #25(1460), #26(1460),
[Frame: 23, payload: 0-611 (612 bytes)]
[Frame: 24, payload: 612-2071 (1460 bytes)]
[Frame: 25, payload: 2072-3531 (1460 bytes)]
[Frame: 26, payload: 3532-4991 (1460 bytes)]
[Frame: 27, payload: 4992-6451 (1460 bytes)]
[Frame: 28, payload: 6452-7911 (1460 bytes)]
[Frame: 29, payload: 7912-9371 (1460 bytes)]
[Frame: 30, payload: 9372-10831 (1460 bytes)]
[Frame: 31, payload: 10832-12291 (1460 bytes)]
[Frame: 32, payload: 12292-13751 (1460 bytes)]
[Frame: 43, payload: 13752-15211 (1460 bytes)]
  
```

序列号一直在线性增长

人工走查序列号

(11) 问题 10

TCP 连接的 throughput 是 109730.88 字节每秒，计算过程如下：

152935B/1.393728s=109730.88Bps(字节每秒)



```

106 Reassembled TCP Segments (152935 bytes): #23(612), #24(1460), #25(1460), #26(1460),
[Frame: 23, payload: 0-611 (612 bytes)]
[Frame: 24, payload: 612-2071 (1460 bytes)]
[Frame: 25, payload: 2072-3531 (1460 bytes)]
[Frame: 26, payload: 3532-4991 (1460 bytes)]
[Frame: 27, payload: 4992-6451 (1460 bytes)]
[Frame: 28, payload: 6452-7911 (1460 bytes)]
  
```

TCP 总大小为 152935B

No.	Time	Source	Destination	Protocol	Length	Info
20	3.421189	172.20.20.19	128.119.245.12	TCP	66	59834 → 80
21	3.686636	128.119.245.12	172.20.20.19	TCP	66	80 → 59834
22	3.686718	172.20.20.19	128.119.245.12	TCP	54	59834 → 80
23	3.687086	172.20.20.19	128.119.245.12	TCP	666	59834 → 80

连接建立开始时间

194	4.814798	128.119.245.12	172.20.20.19	TCP	66	80 → 59834
195	4.814799	128.119.245.12	172.20.20.19	TCP	60	80 → 59834
196	4.814799	128.119.245.12	172.20.20.19	HTTP	831	HTTP/1.1 20
197	4.814917	172.20.20.19	128.119.245.12	TCP	54	59834 → 80

结束时间

3. IP 分析(选择知乎官网www.zhihu.com 用于 Ipv4 分析)

(1) 分析主机发出的第一个 ICMP echo Request 请求

a) 主机 IP 为 172.20.20.19, IP 数据包头中上层协议字段的值为 1 表示 ICMP 协议, IP 头字节数为 20B, IP 数据包的净载为 36B, 确定方式为: IP 分组总长度-IP 首部长度;

```
Source: 172.20.20.19
Destination: 58.205.214.144
```

主机 IP

```
Protocol: ICMP (1)
Header checksum: 0x2310 [validation disabled]
```

上层协议字段值为 1, 标识 ICMP

```
0100 .... = Version: 4
.... 0101 = Header Length: 20 bytes (5)
```

首部长度为 20B

```
Total Length: 56
```

IP 总长度为 56B

b) 该 IP 数据包未分片, 由于标志位全 0, 表示允许分片但是未分片;

```
Flags: 0x0000
0... .. = Reserved bit: Not set
.0.. .. = Don't fragment: Not set
..0. .. = More fragments: Not set
...0 0000 0000 0000 = Fragment offset: 0
```

该数据报未分片

(2) 分析源主机发出的一系列报文

a) 主机发出的 ICMP 报文中 IP 数据报一些字段总在发生改变: 标识 ID、生存时间、首部校验和、数据域, 除了以上四个数据外其余的数据必须保持常量, 原因是: 标识 ID 唯一, 所以每个数据报有所区别, 随之首部校验和也不断改变; TTL 在不断变大 (因为是 ICMP 的 ping 探测), 而且数据域中封装有 ICMP 的报文, 因为 ICMP 的头部信息也在变化, 所以 IP 数据报的数据域也随之变化;

b) 我看到的 IP 数据报的 Identification 字段值的形式是: 每一个报文一个唯一的 16b 的数值, 且在线性递增, 不断+1;


```
Identification: 0xddce (56782)
```

```
Identification: 0xddcf (56783)
```

(3) 分析第一跳返回的 ICMP 消息

- a) Identification 字段值为 0x27cb，TTL 字段值为 64，第一跳返回给我的主机的 ICMP 消息中的这些值中 TTL 保持不变，ID 一直在变化：原因是 因为第一跳路由器设置 TTL 字段为 RFC 指定的值，即 64，所以保持不变（而且发送给源主机时未经过路由器改变 TTL 值）；而 ID 的值标识一个 IP 报文，是唯一的，因此发生改变；

```
Identification: 0x27cb (10187)
```

第一跳返回的报文中的 ID

```
Time to live: 64
```

第一跳返回报文中的 TTL

```
Identification: 0x853b (34107)
```

第二次 traceroute 得到的标识

```
Time to live: 64
```

第二次 traceroute 得到的 TTL

- b) 当改为 2000 字节后，我的主机发送的第一个 ICMP 请求消息被分解为 2 个数据报，第一个分片中标志位的 MF 被置为 1，说明后面还有分片，该分片的数据域长度为 1480B，IP 总长度为 1500B

```
IPv4 1514 Fragmented IP protocol (proto=ICMP 1, off=0, ID=0bc8) [Reassembled in #47]
ICMP 534 Echo (ping) request id=0x0001, seq=13149/23859, ttl=1 (no response found!)
```

分为了两个数据报

```
..1. .... = More fragments: Set
```

标志位 MF 被置为 1

```
.... 0101 = Header Length: 20 bytes (5)
```

```
> Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
```

```
Total Length: 1500
```

总长度为 1500B（含 IP 首部），数据长度为 1480B

- c) 改为 3500B 后，分为了 3 片，头部中标志位 MF 变化、片偏移发生变化；第一个和第二个分片标志位 MF 为 1 表示后面还有分片，第一个分片的片偏移为 0，第二个为 185，第三个为 370；

```
1514 Fragmented IP protocol (proto=ICMP 1, off=0, ID=1ebb) [Reassembled in #52]
1514 Fragmented IP protocol (proto=ICMP 1, off=180, ID=1ebb) [Reassembled in #52]
554 Echo (ping) request id=0x0001, seq=17998/20038, ttl=1 (no response found!)
```


分了三片

```

v Flags: 0x20b9, More fragments
  0... .. = Reserved bit: Not set
  .0... .. = Don't fragment: Not set
  ..1... .. = More fragments: Set
  ...0 0000 1011 1001 = Fragment offset: 185
    
```

片偏移和标志位

4. Ethernet 抓包分析

(1) 访问哈工大主页，开始抓包分析：www.hit.edu.cn

(2) 你的主机 IP 是什么，目的主机 IP 是什么？

回答：我的主机 IP 为 172.20.29.25，目的主机 IP 为 61.167.60.60；

```

> Internet Protocol Version 4, Src: 172.20.29.25, Dst: 61.167.60.70
> Transmission Control Protocol, Src Port: 54731, Dst Port: 80, Seq: 0, Len: 0
    
```

源和目的主机

(3) 你的主机发送的第一跳 HTTP 请求报文的帧结构什么，封装了上层的哪些数据？

回答：以我主机发送的第一条请求 HTTP 报文为例分析，以太网帧结构封装了上层的 IP 数据，IP 封装了上层的 TCP 数据报，TCP 数据包封装上层的 HTTP 数据，如下表

以太网帧首部	IP 首部	TCP 首部	HTTP 请求报文	CRC
--------	-------	--------	-----------	-----

数据结构

(4) 详解回答以太网帧结构？

回答：以太网帧结构如下

目的MAC、源MAC地址(各6B):若网卡的MAC地址与收到的帧的目的MAC地址匹配，或者帧的目的MAC地址为广播地址(FF-FF-FF-FF-FF-FF)，则网卡接收该帧，并将其封装的网络层分组交给相应的网络层协议；否则，网卡丢弃(不接收)该帧；

类型 Type2B:指示帧中封装的是哪种高层协议的分组（如，IP 数据报、Novell IPX 数据报、AppleTalk 数据报等）；

数据(Data)(46-1500B):指上层协议载荷；

CRC(4B):循环冗余校验码，丢弃差错帧



以太网帧结构

```

v Ethernet II, Src: HonHaiPr_ad:ac:c7 (80:2b:f9:ad:ac:c7), Dst: RuijieNe_a5:e2:d3 (58:69:6c:a5:e2:d3)
  > Destination: RuijieNe_a5:e2:d3 (58:69:6c:a5:e2:d3)
  > Source: HonHaiPr_ad:ac:c7 (80:2b:f9:ad:ac:c7)
  Type: IPv4 (0x0800)

```

抓包的以太网帧结构

(5) 我主机的 MAC 地址为 80:2b:f9:ad:ac:c7，目的主机 MAC 地址为 58:69:6c:a5:e2:d3；类型是 IPv4。

(6) 发送报文的数据域长度范围是 46B-1500B，数据域最小值计算过程如下，而以太网帧 MTU 为 1500B，因为数据域最长为 1500B。

$R=10\text{Mbps}$, $\text{RTT}_{\text{max}}=512\mu\text{s}$, $L_{\text{min}} / R = \text{RTT}_{\text{max}}$

$L_{\text{min}}=512\text{bits}=64\text{B}$, $\text{Datamin}=L_{\text{min}}-18=46\text{B}$

```

> Ethernet II, Src: HonHaiPr_ad:ac:c7 (80:2b:f9:ad:ac:c7), Dst: RuijieNe_a5:e2:d3 (58:69:6c:a5:e2:d3)
> Internet Protocol Version 4, Src: 172.20.29.25, Dst: 61.167.60.70
> Transmission Control Protocol, Src Port: 54731, Dst Port: 80, Seq: 1, Ack: 1, Len: 533
> Hypertext Transfer Protocol

```

总结构截图

5. ARP 抓包分析

我的命令行输入为 ping 172.20.19.1（已清除本地 ARP 缓存，且 ping 的目的 IP 和我在在同一个子网内）；

抓的两个包截图如下

```

v Address Resolution Protocol (request)
  Hardware type: Ethernet (1)
  Protocol type: IPv4 (0x0800)
  Hardware size: 6
  Protocol size: 4
  Opcode: request (1)
  Sender MAC address: HonHaiPr_ad:ac:c7 (80:2b:f9:ad:ac:c7)
  Sender IP address: 172.20.29.25
  Target MAC address: 00:00:00_00:00:00 (00:00:00:00:00:00)
  Target IP address: 172.20.29.1

```

查询 ARP

```

v Address Resolution Protocol (reply)
  Hardware type: Ethernet (1)
  Protocol type: IPv4 (0x0800)
  Hardware size: 6
  Protocol size: 4
  Opcode: reply (2)
  Sender MAC address: RuijieNe_a5:e2:d3 (58:69:6c:a5:e2:d3)
  Sender IP address: 172.20.29.1
  Target MAC address: HonHaiPr_ad:ac:c7 (80:2b:f9:ad:ac:c7)
  Target IP address: 172.20.29.25

```

响应ARP

响应 ARP

(1) 利用 MS-DOS 命令: arp 或 c:\windows\system32\arp 查看主机上 ARP 缓存的内容。说明 ARP 缓存中每一列的含义是什么？

回答：查看主机的 ARP 缓存的内容：命令行中输入 **arp -a**；第一列指的是 ARP 协议的缓存的 **IP 地址**，第二列是对应的 **MAC 地址**，第三列是**类型**，即静态类型还是动态类型；

```
C:\Users\86185>arp -a
接口: 172.20.20.19 --- 0x10
Internet 地址      物理地址      类型
172.20.0.1         58-69-6c-a5-e2-d3 动态
172.20.127.255     ff-ff-ff-ff-ff-ff 静态
224.0.0.22         01-00-5e-00-00-16 静态
224.0.0.251        01-00-5e-00-00-fb 静态
224.0.0.252        01-00-5e-00-00-fc 静态
239.255.255.250    01-00-5e-7f-ff-fa 静态
255.255.255.255    ff-ff-ff-ff-ff-ff 静态
```

本机 ARP 缓存

(2) ARP 数据包的格式是怎样的？由几部分构成，各个部分所占的字节数是多少？

回答：首先以管理员方式运行命令行提示符，输入 **arp -d** 删除本机 ARP 缓存，如下图；接着格式如下，**9 部分**组成，分别是：**硬件类型 2B、协议类型 2B、硬件地址长度 1B、协议地址长度 1B、OP 2B、源 MAC 地址 6B、源 IP 地址 4B、目的 MAC 地址 6B、目的 IP 地址 4B**；

```
C:\WINDOWS\system32>arp -d
C:\WINDOWS\system32>arp -a
接口: 172.20.29.25 --- 0x10
Internet 地址      物理地址      类型
224.0.0.22         01-00-5e-00-00-16 静态
```

命令行输入与结果



ARP 数据包格式

(3) 如何判断一个 ARP 数据是请求包还是应答包？

回答：通过 **OP**，当 OP 值为 1 时是请求包，当 OP 值为 2 时是应答包；如图：

```
Opcode: request (1)
```

请求 ARP

```
Opcode: reply (2)
```

响应 ARP

(4) 为什么 ARP 查询要在广播帧中传送，而 ARP 响应要在一个有着明确目的局域网地址的帧中传送？

回答：因为查询 ARP 不知道目的 IP 对应的 MAC 地址，因此需要广播查询，即设置目的 MAC 地址为 ff:ff:ff:ff:ff:ff；而相应 ARP 由于在接收到的查询 ARP 中找到了源 MAC 地址，因此响应可以直接有一个明确的目的地址。

6. UDP 抓包分析

(1) 消息是基于 UDP 的还是 TCP 的？

回答：消息是基于 UDP 的，如截图

9	172.20.29.25	111.161.107.159	UDP	225	4000 → 8000	Len=183
10	111.161.107.159	172.20.29.25	UDP	73	8000 → 4000	Len=31
11	172.20.29.25	111.161.107.159	UDP	225	4000 → 8000	Len=183
12	111.161.107.159	172.20.29.25	UDP	73	8000 → 4000	Len=31

UDP 报文

(2) 你的主机 ip 地址是什么？目的主机 ip 地址是什么？

回答：我的主机 IP 是 172.20.29.25，目的主机 IP 是 111.161.107.159；如图

Internet Protocol Version 4, Src: 172.20.29.25, Dst: 111.161.107.159

发送的 UDP 报文

(3) 你的主机发送 QQ 消息的端口号和 QQ 服务器的端口号分别是多少？

回答：我主机发送消息的端口号为 4000，服务器端口号为 8000；如图

User Datagram Protocol, Src Port: 4000, Dst Port: 8000

Source Port: 4000
Destination Port: 8000

端口号

(4) 数据报的格式是什么样的？都包含哪些字段，分别占多少字节？

回答：格式如图，字段：源端口号 2B，目的端口号 2B，UDP 段长度 2B，校验和 2B；

User Datagram Protocol, Src Port: 4000, Dst Port: 8000																	
<																	
0000	58	69	6c	a5	e2	d3	80	2b	f9	ad	ac	c7	08	00	45	00	Xil.....+.....E.
0010	00	d3	22	41	00	00	80	11	73	6b	ac	14	1d	19	6f	a1	.. "A.... sk....o.
0020	6b	9f	0f	a0	1f	40	00	bf	af	34	02	38	41	00	cd	13	k...@...4.8A...

抓包分析格式

字段	源端口号	目的端口号	UDP 长度	UDP 校验和
长度	2B	2B	2B	2B

数据报格式

(5) 为什么你发送一个 ICQ 数据包后，服务器又返回给你的主机一个 ICQ 数据包？这与 UDP 的不可靠数据传输有什么联系？对比前面的 TCP 协议分析，你能看出 UDP 是无连接的吗？

服务器返回 ICQ 是为了作为确认。因为 UDP 是不可靠的无连接的传输服务，客户端无法确认服务器已经收到了数据报，所以需要服务器返回 ICQ 报文，

可以看出 UDP 是无连接的。因为 TCP 报文需要三次握手建立连接，而且需要 TCP 报文段首部中的标志位，但是 UDP 首部无标志位，UDP 也无序列号。通过抓包分析 UDP 的数据结构可以判断 UDP 是无连接的。

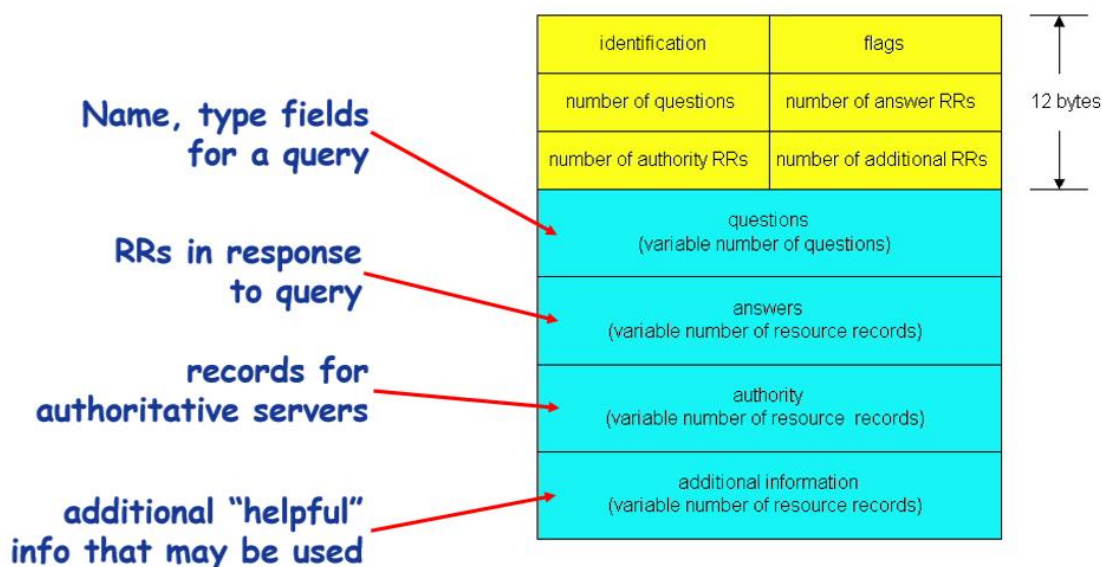
7. DNS 协议解析

(1) 我的主机 IP 是 172.20.29.25，目的主机 IP 是 202.118.224.101，如截图

Source	Destination	Protocol	Length	Info
172.20.29.25	202.118.224.101	DNS	73	Standard query 0xcf6c A
202.118.224.101	172.20.29.25	DNS	132	Standard query response

DNS 查询和响应

(2) DNS 查询与响应消息的格式如图，包括消息头部中的 ID、flags 等和消息体。



DNS 消息格式

(3) DNS 的下层协议是 UDP 协议，是不可靠无连接的传输服务；

```
> User Datagram Protocol, Src Port: 64137, Dst Port: 53
v Domain Name System (query)
  Transaction ID: 0xcf6c
  > Flags: 0x0100 Standard query
    Questions: 1
    Answer RRs: 0
```

基于 UDP 协议的 DNS 协议

(4) DNS 使用 Transaction ID 来标识一次查询和响应报文，其占据 2B，观察到的一次请求和对应的响应发现 ID 是一致的。

▼ Domain Name System (query)
Transaction ID: 0xcf6c

请求 ID

▼ Domain Name System (response)
Transaction ID: 0xcf6c

响应 ID

(5) 请求体内容: Name 表示请求域名、Type 表示请求类型、Class 一般为 IN

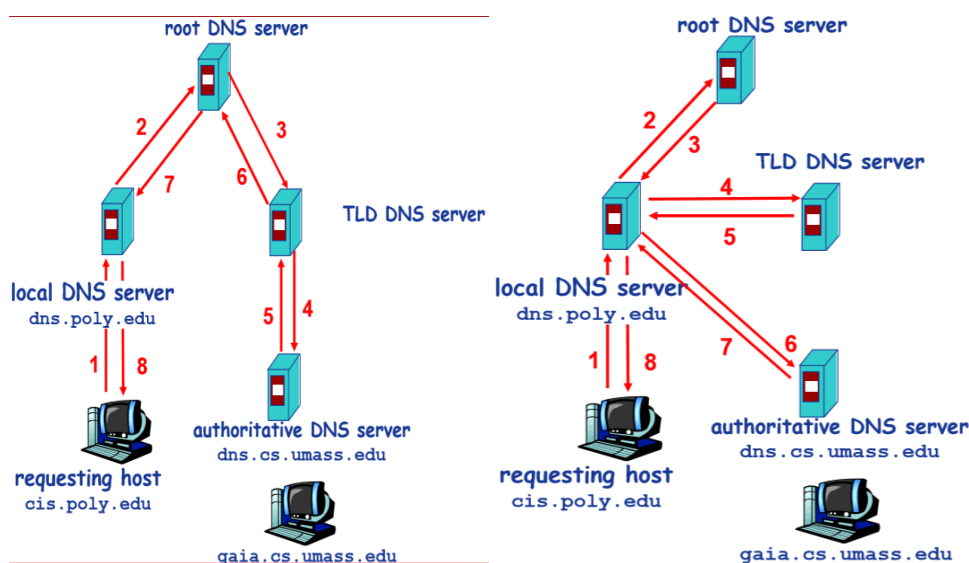
▼ Queries
▼ www.baidu.com: type A, class IN
Name: www.baidu.com
[Name Length: 13]
[Label Count: 3]
Type: A (Host Address) (1)
Class: IN (0x0001)

(6) DNS 记录的一般形式如下表

Name	TTL	Class	Type	Value
主机域名	60	IN	A	IP 地址
域(edu.cn)	86400	IN	NS	该权威域名解析服务器的主机域名
真实域名别名	60	IN	CNAME	真实域名
收件地址后缀	86400	IN	MX	与 Name 相对应的邮件服务器

DNS 资源记录

(7) DNS 查询的形式有递归查询和迭代查询。



左为迭代查询，右为递归查询

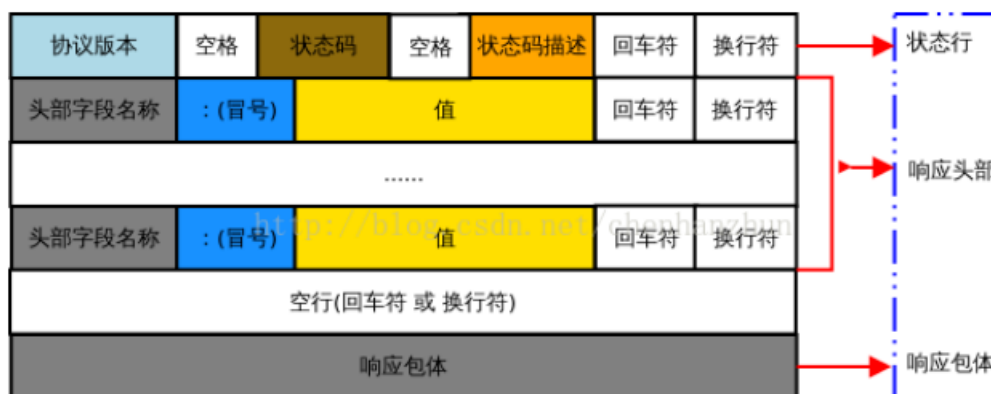
六、 协议总结

6.1 HTTP 协议

1. HTTP 协议支持客户/服务器模式；
2. 简单快速：客户向服务器请求服务时，只需传送请求方法和路径；请求方法常用 GET、HEAD、POST 等，每种方法规定了客户与服务器联系的不同类型；HTTP 协议简单，服务器程序规模小，通信速度较快；
3. 灵活性：HTTP 允许传输任意类型数据对象；正在传输数据类型由 Content-Type 标记；
4. 无连接：无连接是指每次连接只处理一个请求；服务器处理完客户请求，并收到客户应答后，即断开连接，节省传输时间；
5. 无状态：指协议对于事务处理无记忆能力；应答快，但传输数据量较大。



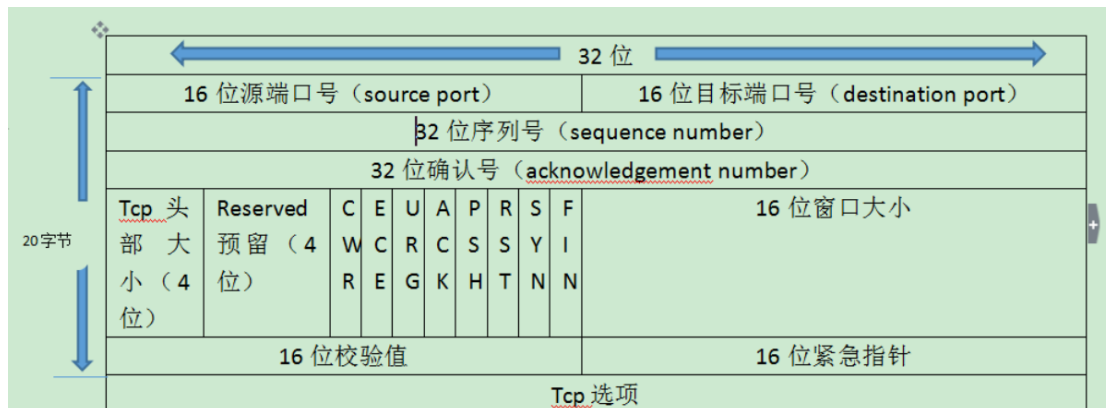
HTTP 请求报文



HTTP 响应报文

6.2 TCP 协议

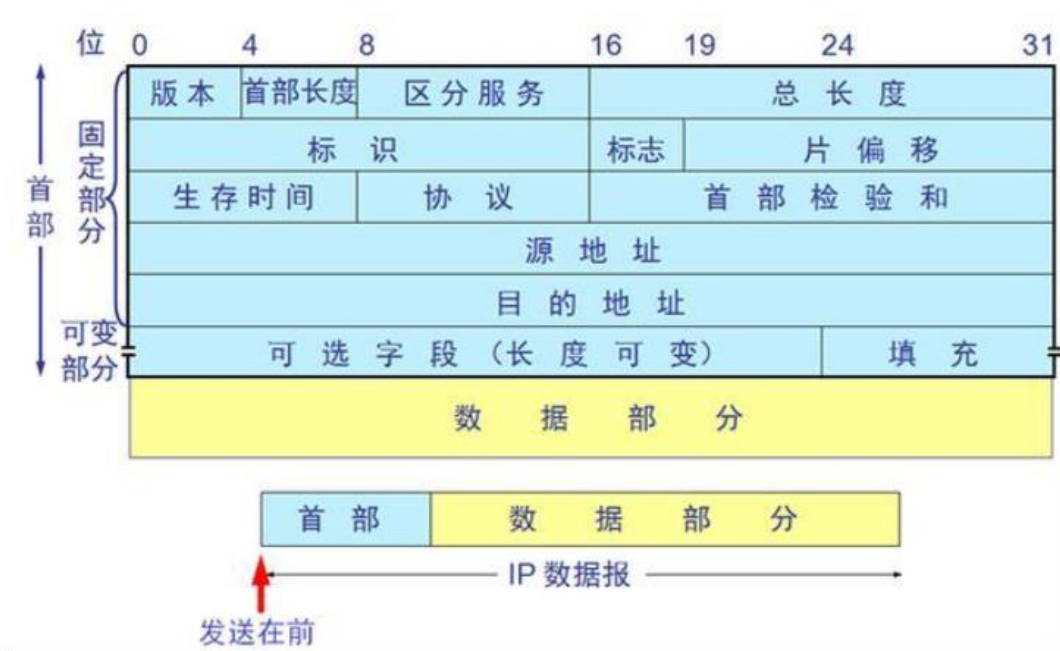
1. 提供面向连接的、可靠的、基于流的数据传输服务，传输单位是报文段；
2. 使用超时重发、数据确认等方式确保数据被正确发送至目的地。



TCP 首部

6.3 IP 协议

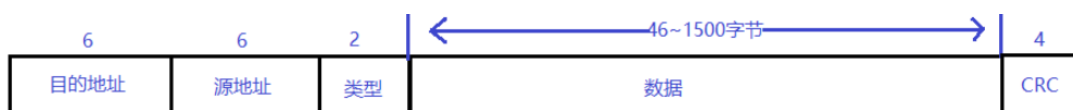
1. 任务: 负责对数据包进行路由选择和存储转发；
2. 负责为分组交换网上的不同主机提供通信服务。在发送数据时，网络层把运输层产生的报文段和用户数据报封装成分组 (IP 数据报) 或包进行传送；
3. IP 协议: 逐跳发送模式；根据数据包的目的地 IP 地址决定数据如何发送；如果数据包不能直接发送至目的地，IP 协议负责寻找一个合适的下一跳路由器，并将数据包交付给该路由器转发；
4. ICMP 协议: 因特网控制报文协议，用于检测网络连接；



IP 协议

6.4 Ethernet 协议

1. 两个相邻节点之间传送数据时，数据链路层将网络层交下来的 IP 数据报组装成帧，在两个相邻的链路上上传送帧（frame）。每一帧包括数据和必要的控制信息；
2. 网卡接口的网络驱动程序，处理数据在物理媒介上的传输；不同的物理网络具有电气特性，网络驱动程序隐藏实现细节，为上层协议提供一致接口



以太网帧结构

6.5 ARP 协议

地址解析协议 ARP（Address Resolution Protocol），负责完成逻辑地址向物理地址的动态映射，将 32 位逻辑地址（IP 地址）转换为 48 位的物理地址（MAC 地址）。

硬件类型		协议类型
硬件地址长度	协议长度	操作 (请求 1)
发送方硬件地址 (前 32 位)		
发送方硬件地址 (后 16 位)		发送方 IP 地址 (前 16 位)
发送方 IP 地址 (后 16 位)		目标硬件地址 (前 16 位)
目标硬件地址 (后 32 位)		
目标 IP 地址 (32 位)		

ARP 协议

6.6 UDP 协议

1. 为了在给定的主机上能识别多个目的地址，同时允许多个应用程序在同一台主机上工作并能独立地进行数据包的发送和接收，设计用户数据报协议 UDP。
2. UDP 使用底层的互联网协议来传送报文，同 IP 一样提供不可靠的无连接数据包传输服务。它不提供报文到达确认、排序、及流量控制等功能。



UDP 报文

6.7 DNS 协议

1. DNS 是一种可以将域名和 IP 地址相互映射的以层次结构分布的数据库系统。
2. DNS 系统采用递归查询请求的方式来响应用户的查询，为互联网的运行提供关键性的基础服务。
3. 目前绝大多数的防火墙和网络都会开放 DNS 服务，DNS 数据包不会被拦截，因此可以基于 DNS 协议建立隐蔽信道，从而顺利穿过防火墙，在客户端和服务器之间传输数据。

0	15	16	31				
Transaction ID (会话标识)		Flags (标志)					
Questions (问题数)		Answer RRs (回答 资源记录数)					
Authority RRs (授权 资源记录数)		Additional RRs (附加 资源记录数)					
Header							
				Queries (查询问题区域)			
				Answers (回答区域)			
				Authoritative nameservers (授权区域)			
Additional records (附加 区域)							

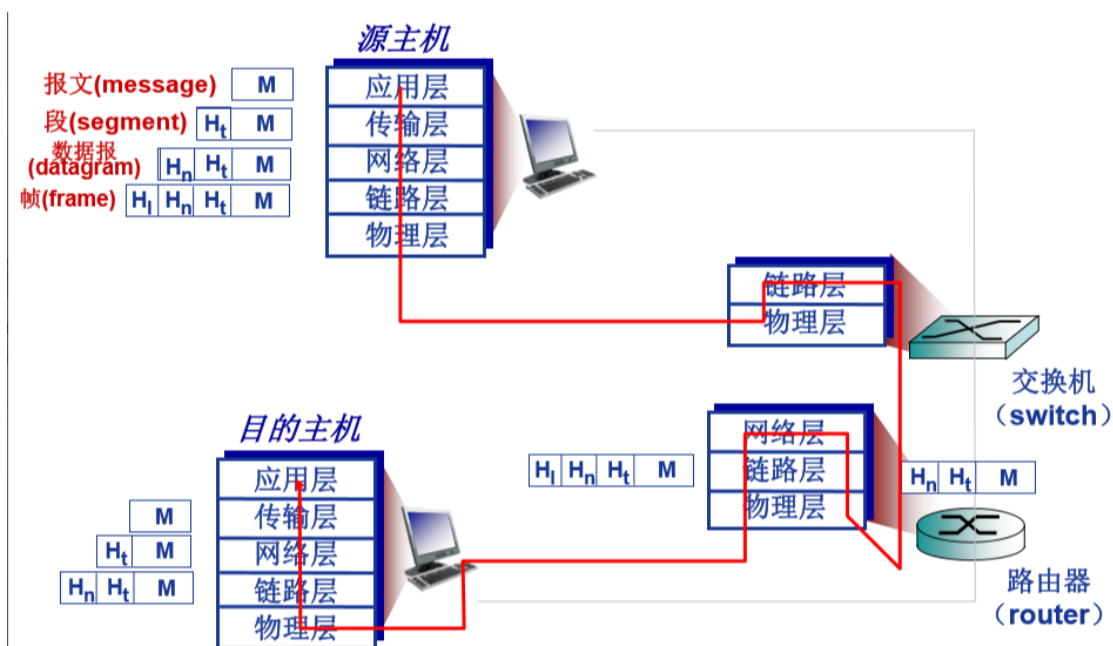
DNS协议报文格式

DNS 报文格式

七、 实验心得

1. 知识收获

- (1) 对互联网的 5 层协议有了一个系统的认识；
- (2) 应用层：支持各种网络应用，比如 FTP, SMTP, HTTP；
- (3) 传输层：进程-进程的数据传输如 TCP, UDP；
- (4) 网络层：源主机到目的主机的数据分组路由与转发如 IP、路由协议等
- (5) 链路层：相邻网络元素（主机、交换 机、路由器等）的数据传输 以
以太网（Ethernet）、802.11 (WiFi)、 PPP；



协议结构