

华东师范大学软件工程学院实验报告

实验课程:	计算机网络	年级:	2018级	实验成绩:	
实验名称:	Protocol Layer	姓名:	陈俊潼		
实验编号:	1	学号:	10185101210	实验日期:	2019.11.12
指导教师:	刘献忠	组号:	-	实验时间:	2019.11.12

一、实验目的

1. 学会通过Wireshark获取各协议层的数据包；
2. 掌握协议层数据包结构；
3. 分析协议开销；

二、实验内容与实验步骤

Step One: Capture a trace

使用Wireshark和wget捕捉数据包。

Step Two: Inspect a trace

掌握如何在Wireshark中查看一个包。

Step Three: Packet Structure

辨析并绘制一个数据包的结构，辨识出以太网协议、IP协议、TCP协议、HTTP协议的部分。

Step Four: Protocol Overhead

根据实验内容，分析除了HTTP以外的额外协议开销。

Step Three: Demultiplexing Keys

分析解复用键，找出以太网头部中哪一部分是解复用键并且告知它的下一个高层指的是IP、IP头部中哪一部分是解复用键并且告知它的下一个高层指的是TCP，并找出对应的值。

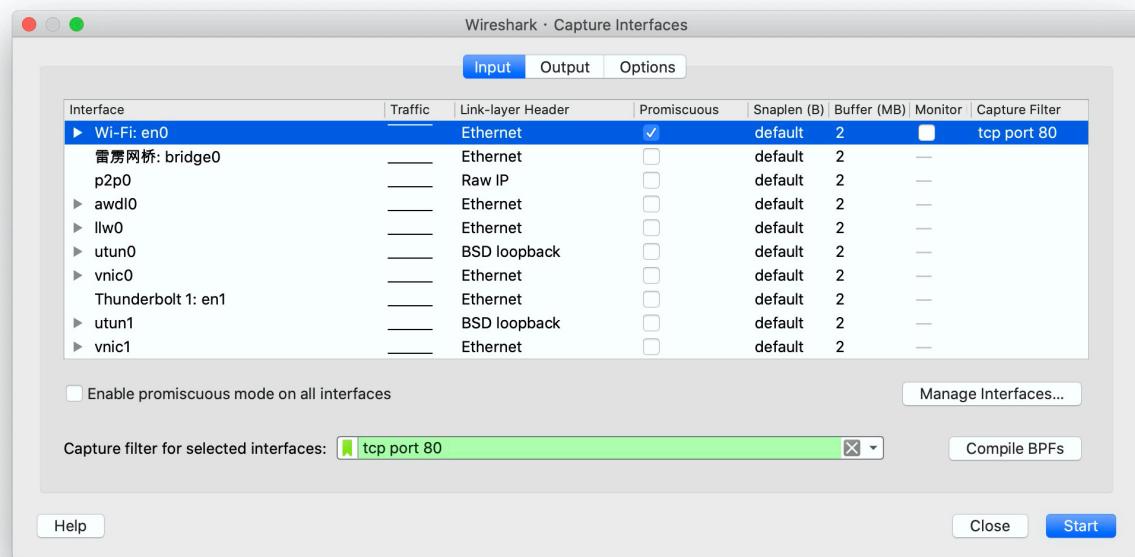
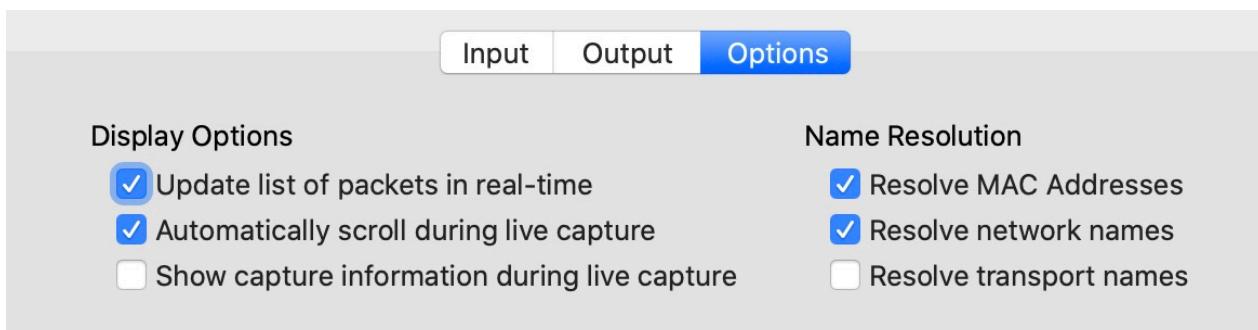
三、实验环境

- macOS 10.5.1
- Wireshark Version 3.0.6
- 电信宽带网络

四、实验过程与分析

Step One: Capture a trace

1. 在Options选项卡中启用Resolve network names，并勾选Wi-Fi的Promiscuous，在设置中勾选Resolve network names，使用Wireshark准备抓包：



2. 在终端中输入命令 `wget https://www.baidu.com` 即可抓取来自 `www.baidu.com` 的数据，Wireshark中会追踪这一过程中的数据包：

```

Last login: Tue Nov 19 00:03:53 on ttys000
billchen@Bills-MacBook-Pro-2018 ~ ➔ wger http://www.baidu.com
zsh: command not found: wger
x billchen@Bills-MacBook-Pro-2018 ~ ➔ wget http://www.baidu.com
--2019-11-19 13:09:54--  http://www.baidu.com/
正在解析主机 www.baidu.com (www.baidu.com)... 182.61.200.7, 182.61.200.6
正在连接 www.baidu.com (www.baidu.com)|182.61.200.7|:80... 已连接。
已发出 HTTP 请求, 正在等待回应... 200 OK
长度: 2381 (2.3K) [text/html]
正在保存至: "index.html"

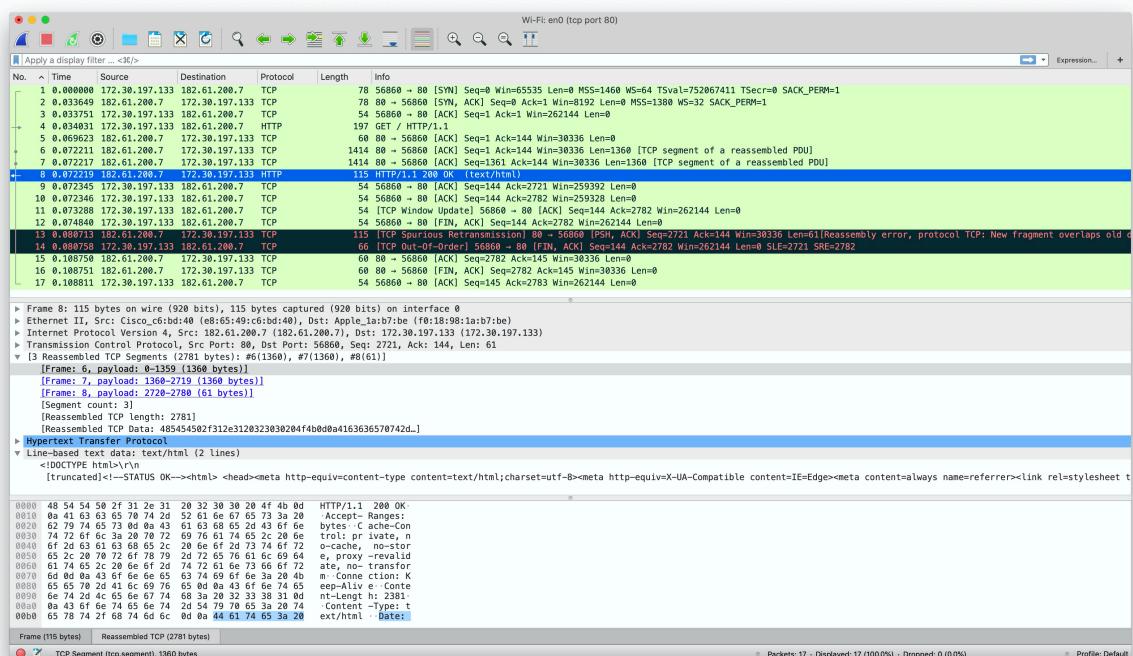
index.html          100%[=====] 2.33K --.-KB/s 用时 0s

2019-11-19 13:09:55 (64.9 MB/s) - 已保存 “index.html” [2381/2381]

billchen@Bills-MacBook-Pro-2018 ~ ➔

```

3. 接下来在Wireshark中确定抓取到数据之后，点击停止按钮，停止抓包。查看wget过程中包的数据。



Step 2: Inspect a Trace:

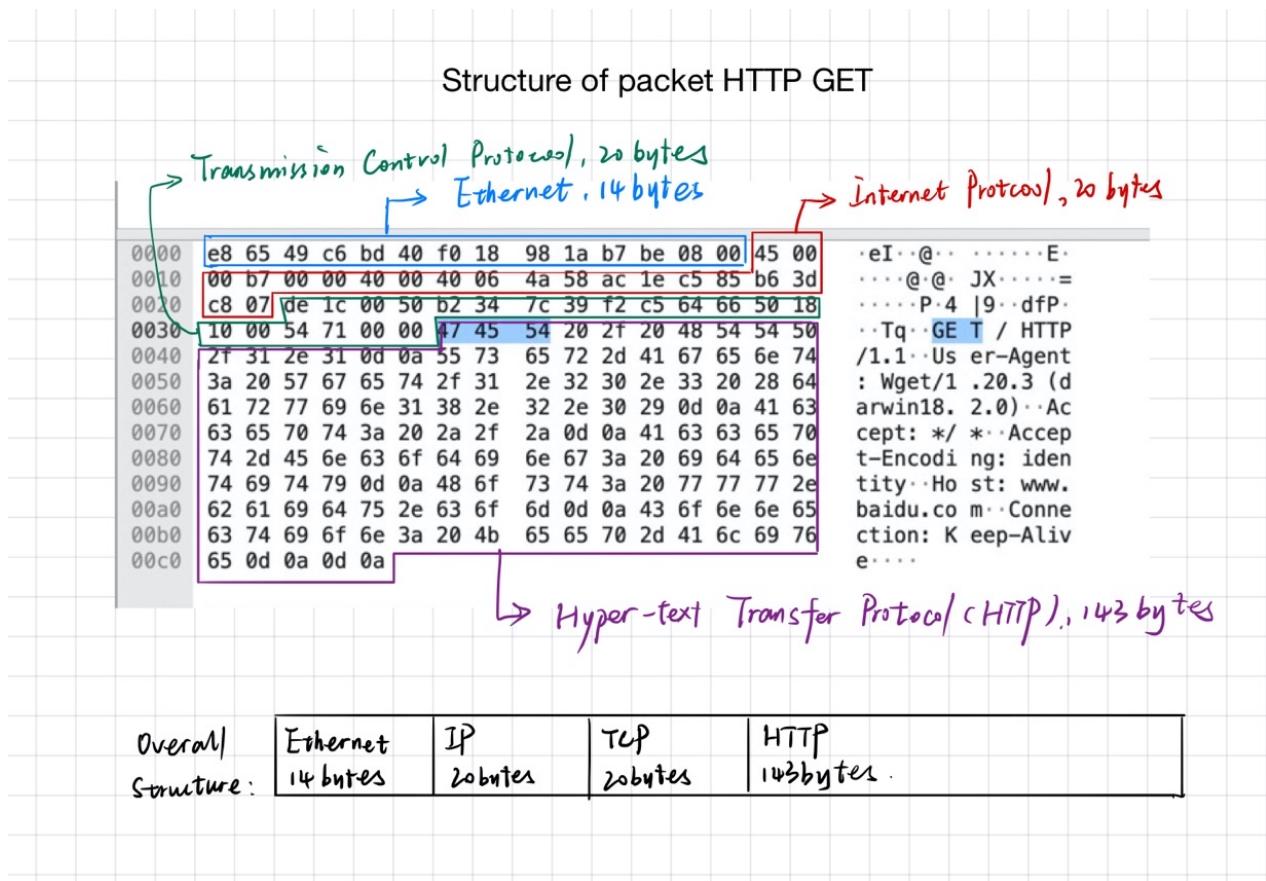
在Wireshark中点几集HTTP GET帧，可以在软件下方看到这一帧的详细信息：

```
▼ Frame 4: 197 bytes on wire (1576 bits), 197 bytes captured (1576 bits) on interface 0
  ▶ Interface id: 0 (en0)
    Encapsulation type: Ethernet (1)
    Arrival Time: Nov 19, 2019 13:28:03.581783000 CST
      [Time shift for this packet: 0.000000000 seconds]
    Epoch Time: 1574141283.581783000 seconds
      [Time delta from previous captured frame: 0.000280000 seconds]
      [Time delta from previous displayed frame: 0.000280000 seconds]
      [Time since reference or first frame: 0.034031000 seconds]
    Frame Number: 4
    Frame Length: 197 bytes (1576 bits)
    Capture Length: 197 bytes (1576 bits)
      [Frame is marked: False]

0000  e8 65 49 c6 bd 40 f0 18  98 1a b7 be 08 00 45 00  ·EI··@.. . . . . E·
0010  00 b7 00 00 40 00 40 06  4a 58 ac 1e c5 85 b6 3d  ···@·@· JX····=··
0020  c8 07 de 1c 00 50 b2 34  7c 39 f2 c5 64 66 50 18  ···P·4 |9·dfP·
0030  10 00 54 71 00 00 47 45  54 20 2f 20 48 54 54 50  ··Tq·GE T / HTTP
0040  2f 31 2e 31 0d 0a 55 73  65 72 2d 41 67 65 6e 74  /1.1··Us er-Agent
0050  3a 20 57 67 65 74 2f 31  2e 32 30 2e 33 20 28 64  : Wget/1 .20.3 (d
0060  61 72 77 69 6e 31 38 2e  32 2e 30 29 0d 0a 41 63  arwin18. 2.0)··Ac
0070  63 65 70 74 3a 20 2a 2f  2a 0d 0a 41 63 63 65 70  cept: */ *··Accep
0080  74 2d 45 6e 63 6f 64 69  6e 67 3a 20 69 64 65 6e  t-Encodi ng: iden
0090  74 69 74 79 0d 0a 48 6f  73 74 3a 20 77 77 77 2e  tity··Ho st: www.
00a0  62 61 69 64 75 2e 63 6f  6d 0d 0a 43 6f 6e 6e 65  baidu.co m··Conne
00b0  63 74 69 6f 6e 3a 20 4b  65 65 70 2d 41 6c 69 76  ction: K eep-Aliv
00c0  65 0d 0a 0d 0a  ···e....
```

Step 3: Packet Structure

分析HTTP GET帧的结构并绘制总体结构如下：



Step 4: Protocol Overhead

将接收到的http信息视为有效信息，http协议之下的TCP、IP、Ethernet数据为协议开销。

	4	0.034031	172.30.197.133	182.61.200.7	HTTP
→	5	0.069623	182.61.200.7	172.30.197.133	TCP
	6	0.072211	182.61.200.7	172.30.197.133	TCP
	7	0.072217	182.61.200.7	172.30.197.133	TCP
←	8	0.072219	182.61.200.7	172.30.197.133	HTTP

Wireshark统计的第8帧下载帧中的的有效http字节数为： 2781 bytes.

该帧是由3个TCP段数据合并而来的，加上协议开销为： $4 * (\text{Ethernet Overhead} + \text{IP Overhead} + \text{TCP Overhead})$

$$= 4 * (14+20+20) = 216 \text{ bytes.}$$

发现第5帧SYC ACK帧后有6字节的填充：

Type:	IPv4 (0x0800)
Padding:	000000000000
► Internet Protocol Version 4, Src: 182.61.200.7 (182.6	
0000	f0 18 98 1a b7 be e8 65 49 c6 bd 40 08 00 45 00
0010	00 28 c0 6a 40 00 2b 06 9f 7c b6 3d c8 07 ac 1e
0020	c5 85 00 50 de 1c f2 c5 64 66 b2 34 7c c8 50 10
0030	03 b4 57 a1 00 00 00 00 00 00 00 00 00 00 00 00

所以总的协议开销为 $216 \text{ bytes} + \text{padding} = 222 \text{ bytes}$.

所以协议开销占比为 $222 / (222 + 2781) = 7.39\%$.

Step 5: Demultiplexing Keys

- 观察第五帧的Ethernet部分：

....	0	= 16 bit: individual address (unicast)
Type:	IPv4 (0x0800)							
Padding:	000000000000							
► Internet Protocol Version 4, Src: 182.61.200.7 (182.61.200.7), Dst: 172.30.197.133 (1								
0000	f0 18 98 1a b7 be e8 65 49 c6 bd 40 08 00 45 00						e I·@·E·
0010	00 28 c0 6a 40 00 2b 06 9f 7c b6 3d c8 07 ac 1e							·(·j@·+· · ·=····
0020	c5 85 00 50 de 1c f2 c5 64 66 b2 34 7c c8 50 10							··P··· df·4 ·P·

发现第13字节和第14字节表示了下一层协议。在这里，0x0800表示为IPv4协议。

观察第五帧IP层的部分：

```
Time to live: 43
Protocol: TCP (6)
Header checksum: 0x9f7c [validation disabled]
[Header checksum status: Unverified]
Source: 182.61.200.7 (182.61.200.7)
0000 f0 18 98 1a b7 be e8 65 49 c6 bd 40 08 00 45 00  .....e I..@..E.
0010 00 28 c0 6a 40 00 2b 06 9f 7c b6 3d c8 07 ac 1e  .( .j@..+.. | ..=.....
0020 c5 85 00 50 de 1c f2 c5 64 66 b2 34 7c c8 50 10  ..P..... df 4| P..
0030 03 b4 57 a1 00 00 00 00 00 00 00 00 00 00 00 00  ..W..... ....
```

即IP层的第10个字节表示了下一层协议。这里0x06表示TCP协议。

五、实验结果总结

思考题目：

- Look at a short TCP packet that carries no higher-layer data. To what entity is this packet destined? After all, if it carries no higher-layer data then it does not seem very useful to a higher layer protocol such as HTTP!
 - 第一帧数据可以作为一个控制帧，有利于传输双方控制它们之间的传输状态，如果直接传输数据的话容易导致数据丢失。
 - In a classic layered model, one message from a higher layer has a header appended by the lower layer and becomes one new message. But this is not always the case. Above, we saw a trace in which the web response (one HTTP message comprised of an HTTP header and an HTTP payload) was converted into multiple lower layer messages (being multiple TCP packets). Imagine that you have drawn the packet structure (as in step 2) for the first and last TCP packet carrying the web response. How will the drawings differ?
 - In the classic layered model described above, lower layers append headers to the messages passed down from higher layers. How will this model change if a lower layer adds encryption?
 - In the classic layered model described above, lower layers append headers to the messages passed down from higher layers. How will this model change if a lower layer adds compression?

通过本次实验，我掌握了Wireshark的基本使用方法和数据包的查看方法，也了解了一个数据包中所包含的以太网协议、Internet Protocol协议、Transmission Control Protocol协议头的结构。同时，通过分析协议的开销，我认识到了OSI模型不能过多层的原因之一就是每增加一层就会增加一些额外的开销，降低带宽的使用效率，但这样却可以使协议更严谨，传输更稳定。

在下一次实验中，我将会尝试先关闭其他正在使用网络的程序，避免在抓包的时候被干扰，影响对数据的分析。

六、附录

无