# UM-SJTU Joint Institute

Computer Networks

(Ve489)

---

## Mini-Project 1 Report

Name: Sun Yiwen ID: 517370910213
Date: June 21 2020

# 1 Step 1: Capture a Trace

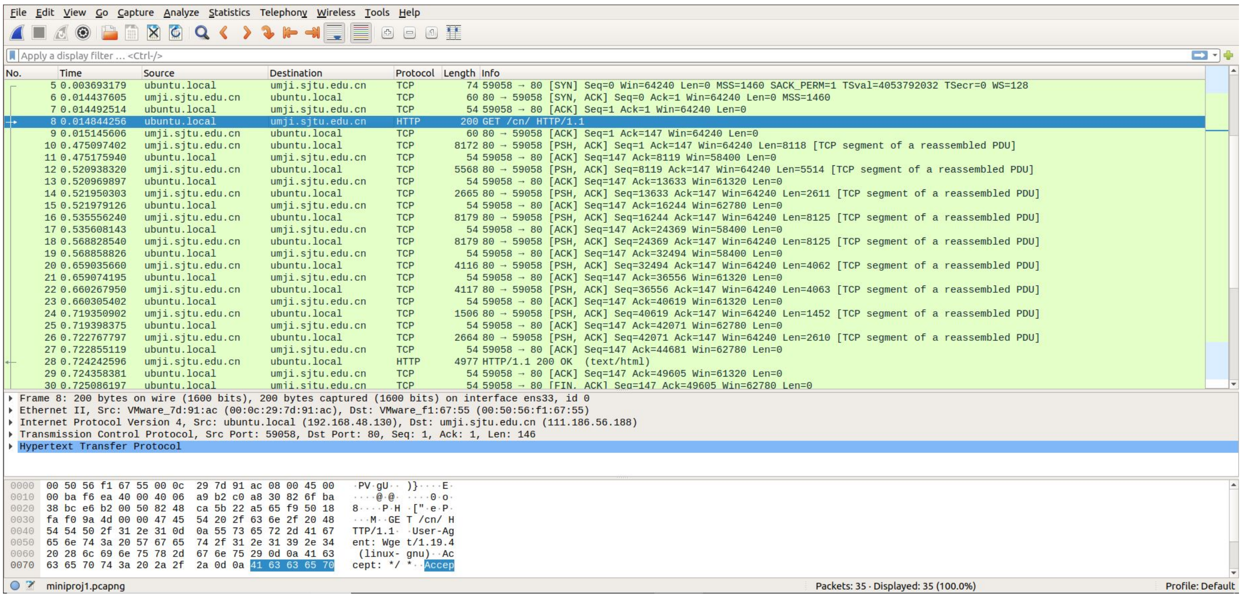


Figure 1: My screenshot of a packet trace of wget traffic.

A screenshot of a packet trace of “wget http://umji.sjtu.edu.cn” traffic is shown in Figure 1. These packets are all colored green in Wireshark, indicating that the fetch is successful. The packet selected in the figure is a packet whose Protocol column is “HTTP” and Info column is a GET request. It is the packet that carries the web (HTTP) request sent from my computer to the server.

# 2 Step 2: Inspect the Trace

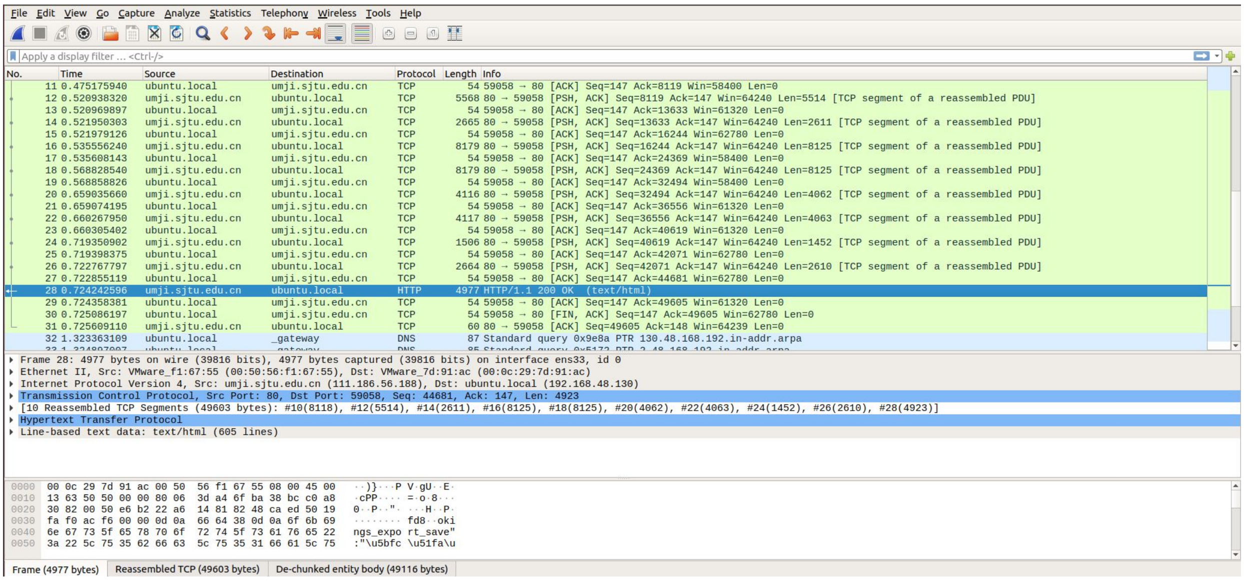


Figure 2: My screenshot of a HTTP “200 OK” response.

A screenshot of a HTTP “200 OK” response is shown in Figure 2. This packet is the response from the server to my computer. The “200 OK” in the Info field indicates that the web fetch is successful.

# 3 Step 3: Packet Structure

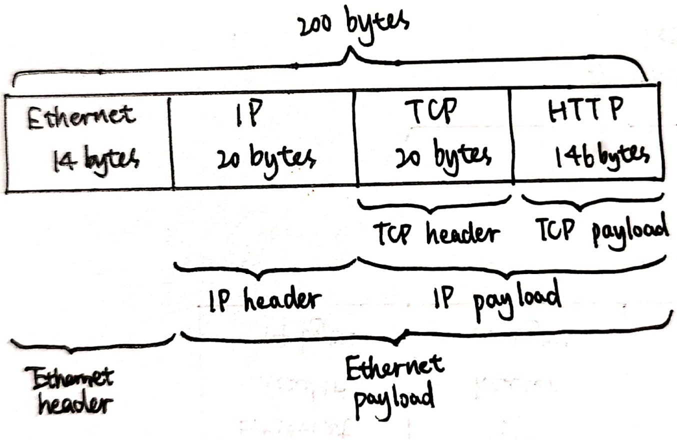
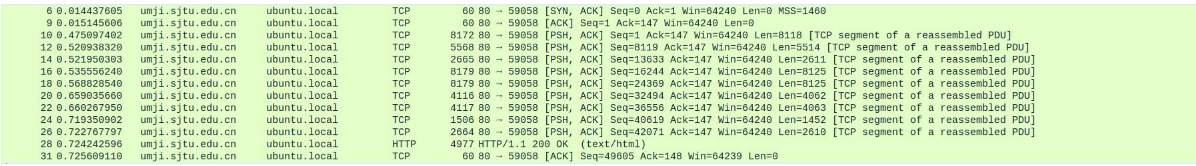



Figure 3: The packet structure of an HTTP GET packet.

# 4 Step 4: Protocol Overhead



| | | | | | | |
|---|---|---|---|---|---|---|
| 6 | 0.014437605 | umji.sjtu.edu.cn | ubuntu.local | TCP | 60 | 80 → 59058 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460 |
| 9 | 0.015145606 | umji.sjtu.edu.cn | ubuntu.local | TCP | 60 | 80 → 59058 [ACK] Seq=1 Ack=147 Win=64240 Len=0 |
| 10 | 0.475097402 | umji.sjtu.edu.cn | ubuntu.local | TCP | 8172 | 80 → 59058 [PSH, ACK] Seq=1 Ack=147 Win=64240 Len=8118 [TCP segment of a reassembled PDU] |
| 12 | 0.520938320 | umji.sjtu.edu.cn | ubuntu.local | TCP | 5568 | 80 → 59058 [PSH, ACK] Seq=8119 Ack=147 Win=64240 Len=5514 [TCP segment of a reassembled PDU] |
| 14 | 0.521950303 | umji.sjtu.edu.cn | ubuntu.local | TCP | 2665 | 80 → 59058 [PSH, ACK] Seq=13633 Ack=147 Win=64240 Len=2611 [TCP segment of a reassembled PDU] |
| 16 | 0.535556240 | umji.sjtu.edu.cn | ubuntu.local | TCP | 8179 | 80 → 59058 [PSH, ACK] Seq=16244 Ack=147 Win=64240 Len=8125 [TCP segment of a reassembled PDU] |
| 18 | 0.568828540 | umji.sjtu.edu.cn | ubuntu.local | TCP | 8179 | 80 → 59058 [PSH, ACK] Seq=24369 Ack=147 Win=64240 Len=8125 [TCP segment of a reassembled PDU] |
| 20 | 0.659035660 | umji.sjtu.edu.cn | ubuntu.local | TCP | 4116 | 80 → 59058 [PSH, ACK] Seq=32494 Ack=147 Win=64240 Len=4062 [TCP segment of a reassembled PDU] |
| 22 | 0.660267950 | umji.sjtu.edu.cn | ubuntu.local | TCP | 4117 | 80 → 59058 [PSH, ACK] Seq=36556 Ack=147 Win=64240 Len=4063 [TCP segment of a reassembled PDU] |
| 24 | 0.719350902 | umji.sjtu.edu.cn | ubuntu.local | TCP | 1506 | 80 → 59058 [PSH, ACK] Seq=40619 Ack=147 Win=64240 Len=1452 [TCP segment of a reassembled PDU] |
| 26 | 0.722767797 | umji.sjtu.edu.cn | ubuntu.local | TCP | 2664 | 80 → 59058 [PSH, ACK] Seq=42071 Ack=147 Win=64240 Len=2610 [TCP segment of a reassembled PDU] |
| 28 | 0.724242596 | umji.sjtu.edu.cn | ubuntu.local | HTTP | 4977 | HTTP/1.1 200 OK (text/html) |
| 31 | 0.725609110 | umji.sjtu.edu.cn | ubuntu.local | TCP | 60 | 80 → 59058 [ACK] Seq=49605 Ack=148 Win=64239 Len=0 |

Figure 4: A screenshot of the packets in the download direction.

As shown in Figure 4, the download packets start with a short TCP packet described as a SYN ACK, which denotes the beginning of a connection. Then, there are 10 longer packets, of which the last one is an HTTP packet. Lastly, there is a short TCP packet indicating the end of the connection.

To calculate download protocol overheads, we focus on the 10 longer packets in the middle. For each packet, the overheads exist in the form of Ethernet / IP / TCP headers, which is 14+20+20=54 bytes. So there are totally $54 \times 10 = 540$ bytes of download protocol overheads. The overhead rate can be calculated as:

$$\begin{aligned} Overhead\ Rate &= \frac{54 \times 10}{8172 + 5568 + 2665 + 8179 + 8179 + 4116 + 4117 + 1506 + 2664 + 4977} \\ &= \frac{540}{50143} \\ &= 1.077\% \end{aligned}$$

which, in my opinion, is not very significant and is fairly acceptable.

# 5 Step 5: Demultiplexing Keys

1. The demultiplexing key that tells Ethernet the next higher layer is IP is the Type field. The value 0x0800 is used in this field to indicate “IP”.

2. The demultiplexing key that tells IP the next higher layer is TCP is the Protocol field. The value 0x06 is used in this field to indicate “TCP”.

# 6 Explore on my own

1. The packet is destined to the TCP protocol entity of the receiving computer instead of entities that have HTTP. It is only exchanged between TCP entities in a peer-to-peer style to maintain their connection. Therefore, it does not need to carry higher-layer data.

2. The drawing of the first TCP packet will have an HTTP header inside while the drawing of the last TCP packet will only have HTTP payload data but no HTTP header. Moreover, if the HTTP header is very long, then the first drawing will only have HTTP header but no HTTP payload data.

3. If a lower layer adds encryption, it will rewrite its payload and append its header. Correspondingly, the payload of higher protocols will be inaccessible when the packet is transmitted. It will only be accessible after the receiving end decrypts the message.

4. If a lower layer adds compression, it will rewrite its payload and append its header. The new payload data will be shorter than the original one. Correspondingly, the payload of higher protocols will be inaccessible when the packet is transmitted. It will only be accessible after the receiving end decompresses the message.