

华东师范大学数据科学与工程学院实验报告

课程名称：计算机网络与编程

年级：2020 级

上机实践成绩：

指导教师：张召

姓名：张熙翔

学号：10205501427

上机实践名称：UDP 和 TCP 协议分析

上机实践日期：2022/5/23

一、实验目的

了解 UDP 协议的工作原理

了解 TCP 协议的工作原理

- 学习 TCP 建立连接三次握手的过程
- 学习 TCP 断开连接四次挥手的過程

二、实验任务

使用 Wireshark 快速了解 UDP 协议

使用 Wireshark 快速了解 TCP 协议

三、使用环境

Wireshark

四、实验过程

Task1: 从跟踪中选择一个 UDP 数据包。从此数据包中，识别并确定 UDP 首部字段，请为这些字段命名并将实验结果附在实验报告中。

UDP 首部有 8 个字节，由源端口、目的端口、长度以及校验和 4 个字段构成，每个字段都是两个字节。

The image shows a Wireshark packet capture window. The top pane displays a list of packets, with packet 7201 selected. The middle pane shows the details of the selected packet, including Ethernet II, Internet Protocol Version 4, and User Datagram Protocol (UDP). The bottom pane shows the raw data of the packet. Red arrows point to specific fields in the UDP section with Chinese annotations:

- Source Port: 58470 → 源端口：源端口号，需要对方回信时选用，不需要时全部置0。
- Destination Port: 9405 → 目的端口：目的端口号。
- Length: 100 → 长度：UDP的数据报文的长度（包括首部和数据）其最小值为8（只有首部），这里为100
- Checksum: 0x437b [unverified] → 校验和：检测UDP数据报在传输中是否有错，有错则丢弃。
- [Checksum Status: Unverified]
- [Stream index: 29]
- [Timestamps]
- UDP payload (92 bytes)
- Data (92 bytes) → 数据部分长度，等于Length100-首部的长度8

Task2: UDP 首部中的长度字段的值指的是什么，以及为什么需要这样设计？使用捕获的 UDP 数据包进行验证，请将实验结果附在实验报告中。

UDP 的数据报文的长度包括首部长度和数据长度，其最小值为 8，即只有首部。这样设计是为了防止路由错误。

Source Port: 58470
Destination Port: 9405
Length: 100
Checksum: 0x437b [unverified]
[Checksum Status: Unverified]
[Stream index: 29]
> [Timestamps]
UDP payload (92 bytes)
> Data (92 bytes)

92=Length100-header8

0X64=Length:100

0000	74 ea cb 1e 4e 50 40 5b d8 c3 48 c3 08 00 45 00	t...NP@[...H...E.
0010	00 78 52 f6 00 00 80 11 23 d5 c0 a8 7c 11 78 83	.xR.....#... .x.
0020	0e 6d e4 66 24 bd 00 64 43 7b 00 79 83 4b 06 00	.m.f\$.d C{.y.K..
0030	22 00 3a 00 00 00 00 3c 00 00 19 01 00 00 00 00	".....<.....
0040	00 00 01 00 50 19 8e 21 34 65 0a 00 18 09 76 30P..! 4e....v0
0050	2e 30 2e 31 5f 31 30 18 20 61 66 33 31 63 37 61	.0.1_10. af31c7a
0060	30 38 66 39 65 34 36 65 32 62 33 38 62 66 33 30	08f9e46e 2b38bf30
0070	63 39 32 34 39 62 36 63 30 18 00 16 06 18 00 18	c9249b6c 0.....
0080	00 18 00 18 00 00

Task3: UDP 有效负载中可包含的最大字节数是多少？请将实验结果附在实验报告中。

有效负载是被传输数据中的一部分，而这部分才是数据传输的最基本的目的，和有效负载一同被传送的数据还有：数据头或称作元数据，有时候也被称为开销数据，这些数据用来辅助数据传输。

因为 UDP 首部中，长度字段只有 2 个字节即 16 位。所以能表示的最大数为 $2^{16}-1=65535$ 。又因为 UDP 首部占据了 8 字节，所以有效负载中可包含的最大字节数是 $65535-8=65527$ 字节。

Task4: 观察发送 UDP 数据包后接收响应的 UDP 数据包，这是对发送的 UDP 数据包的回复，请描述两个数据包中端口号之间的关系。(提示：对于响应 UDP 目的地应该为发送 UDP 包的地址)。请将实验结果附在实验报告中。

对于这组发送的 UDP 数据包和响应的 UDP 数据包。

No.	Time	Source	Destination	Protocol	Length	Info
23725	54.436635	192.168.124.17	120.131.14.109	UDP	134	58470 → 9405 Len=92
23726	54.455639	120.131.14.109	192.168.124.17	UDP	138	9405 → 58470 Len=96

编号为 23725 和 23726。No. 23725 包的源地址是：192.168.124.17 即我本地的电脑地址，目的地址是 120.131.14.109，源端口是 58470，目的端口是 9405。No. 23726 包的源地址是：120.131.14.109，而目的地之是 192.168.124.17，源端口是 9405，目的端口 58470。

发送 UDP 数据包的源端口号等于响应的 UDP 数据包的端口号。

响应 UDP 数据包的源端口号等于发送的 UDP 数据包的端口号。

User Datagram Protocol, Src Port: 58470, Dst Port: 9405	User Datagram Protocol, Src Port: 9405, Dst Port: 58470
Source Port: 58470	Source Port: 9405
Destination Port: 9405	Destination Port: 58470
Length: 100	Length: 104
Checksum: 0xd37a [unverified]	Checksum: 0xf14b [unverified]
[Checksum Status: Unverified]	[Checksum Status: Unverified]
[Stream index: 18]	[Stream index: 18]
> [Timestamps]	> [Timestamps]
UDP payload (92 bytes)	UDP payload (96 bytes)
Data (92 bytes)	Data (96 bytes)

Task5: 利用 Wireshark 抓取一个 TCP 数据包，查看其具体数据结构和实际的数据 (要求根据报文结构正确标识每个部分)，请将实验结果附在实验报告中。



图 3-29 TCP 报文段结构

```

20647 21.883118 192.168.124.17 39.136.30.84 TCP 54 63421 → 443 [ACK] Seq=518 Ack=2825 Win=131072 Len=0
20648 21.883167 39.136.30.84 192.168.124.17 TCP 1326 443 → 63421 [PSH, ACK] Seq=2825 Ack=518 Win=30720 Len=1272 [TCP segment of a reassembled PDU]
20649 21.883160 39.136.30.84 192.168.124.17 TLSv1.2 1466 Application Data [TCP segment of a reassembled PDU]

```

Frame 20647: 54 bytes on wire (432 bits), 54 bytes captured (432 bits) on interface \Device\NPF_{B602E7CE-D131-4B76-A846-5C19A2D7C768}, id 0
 Ethernet II, Src: Chongqin_c3:48:c3 (40:5b:d8:c3:48:c3), Dst: NewH3CTe_1e:4e:50 (74:ea:cb:1e:4e:50)
 Internet Protocol Version 4, Src: 192.168.124.17, Dst: 39.136.30.84
 Transmission Control Protocol, Src Port: 63421, Dst Port: 443, Seq: 518, Ack: 2825, Len: 0

Source Port: 63421 → **源端口号 f7 bd**
 Destination Port: 443 → **目的端口号 01 bb**
 [Stream index: 34]
 [Conversation completeness: Incomplete, DATA (15)]
 [TCP Segment Len: 0]
 Sequence Number: 518 (relative sequence number) → **序号 26 4d af 88**
 Sequence Number (raw): 642625416
 [Next Sequence Number: 518 (relative sequence number)]
 Acknowledgment Number: 2825 (relative ack number) → **确认号 af bf 9b 60**
 Acknowledgment number (raw): 2948569952
 0101 = Header Length: 20 bytes (5) → **首部长度 50**
 Flags: 0x010 (ACK)
 000. = Reserved: Not set → **保留未用**
 ...0 = Nonce: Not set
0... = Congestion Window Reduced (CWR): Not set
0... = ECN-Echo: Not set
0... = Urgent: Not set
0... = Acknowledgment: Set → **Flag中的ACK已经被标记10 其余Flag均未被标记**
0... = Push: Not set
0... = Reset: Not set
0... = Syn: Not set
0... = Fin: Not set
 [TCP Flags:A....]
 Window: 512 → **接收窗口大小 02 00**
 [Calculated window size: 131072]
 [Window size scaling factor: 256]
 Checksum: 0x10d0 [unverified] → **因特网校验和 10 d0**
 [Checksum Status: Unverified]
 Urgent Pointer: 0 → **紧急数据指针 00 00**
 [Timestamps]
 [SEQ/ACK analysis]

→ **选项 数据**

0000	74 ea cb 1e 4e 50 40 5b d8 c3 48 c3 08 00 45 00	t...NP@[..H...E.
0010	00 28 10 f8 40 00 80 06 67 42 c0 a8 7c 11 27 88	.(...@... gB... .'.
0020	1e 54 f7 bd 01 bb 26 4d af 88 af bf 9b 60 50 10	.T....&M`P.
0030	02 00 10 d0 00 00

Task6: 根据 TCP 三次握手的交互图以及 TCP 报文段结构图逐步分析三次握手过程, 请将实验结果附在实验报告中。

20614	21.864746	192.168.124.17	39.136.30.84	TCP	66 63421 → 443 [SYN] Seq=0 Win=64240
20623	21.872235	39.136.30.84	192.168.124.17	TCP	66 443 → 63421 [SYN, ACK] Seq=0 Ack=
20625	21.872312	192.168.124.17	39.136.30.84	TCP	54 63421 → 443 [ACK] Seq=1 Ack=1 Win
20627	21.872451	192.168.124.17	39.136.30.84	TLSv1.3	571 Client Hello

第一次握手, 是从本地客户端发向服务器的一个数据报。在这个数据报中, Seq=0, 说明一开始是从序号为 0 的包发送的。我们看到这里 SYN、这一位已经被设为 1 了, 因为还没收到服务器的确认信息, 因此这里 ACK 为设为 0。

```

Flags: 0x002 (SYN)
000. .... = Reserved: Not set
...0 .... = Nonce: Not set
.... 0... = Congestion Window Reduced (CWR): Not set
.... .0.. = ECN-Echo: Not set
.... ..0. = Urgent: Not set
.... ...0 = Acknowledgment: Not set
.... .... 0... = Push: Not set
.... .... .0.. = Reset: Not set
> .... .... ..1. = Syn: Set
.... .... ...0 = Fin: Not set
[TCP Flags: .....S.]
    
```

第二次握手, 这是服务器给本地电脑发送的报文, Seq=0, ACK = 1, 因为 TCP 是全双工通信的, 因此发送来的第一个报文段也是从 seq=0 开始的。但是这个报文段中包含了对我发的确认信息, 因此这里 ACK 被设置为 1, 这说明 1 以前的包我全部都收到了, 请本地发送 1 以后的包。

```

Flags: 0x012 (SYN, ACK)
000. .... = Reserved: Not set
...0 .... = Nonce: Not set
.... 0... = Congestion Window Reduced (CWR): Not set
.... .0.. = ECN-Echo: Not set
.... ..0. = Urgent: Not set
.... ...1 = Acknowledgment: Set
.... .... 0... = Push: Not set
.... .... .0.. = Reset: Not set
> .... .... ..1. = Syn: Set
.... .... ...0 = Fin: Not set
[TCP Flags: .....A..S.]
    
```

第三次握手是本地发送给服务器的，此时，Seq=1，说明这是本地发送的第二个包了（第一个包 Seq=0）；ACK = 1 说明已经收到了来自服务端的 1 以前的所有包。

✧ Flags: 0x010 (ACK)

```

000. .... = Reserved: Not set
...0 .... = Nonce: Not set
.... 0... = Congestion Window Reduced (CWR): Not set
.... .0.. = ECN-Echo: Not set
.... ..0. = Urgent: Not set
.... ...1 = Acknowledgment: Set
.... .... 0... = Push: Not set
.... .... .0.. = Reset: Not set
.... .... ..0. = Syn: Not set
.... .... ...0 = Fin: Not set
[TCP Flags: .....A.....]

```

本地和服务端的三次握手结束后，可以开始相互传递信息。

Task7: 根据 TCP 四次挥手的交互图以及 TCP 报文段结构图逐步分析四次挥手过程，请将实验结果附在实验报告中。

1845	11.184104	192.168.124.17	39.156.66.18	TCP	54 [64130 → 80 [FIN, ACK] Seq=1 Ack=1 Win=131072 Len=0
1846	11.204202	39.156.66.18	192.168.124.17	TCP	54 80 → 64130 [ACK] Seq=1 Ack=2 Win=78464 Len=0
1847	11.204502	39.156.66.18	192.168.124.17	TCP	54 80 → 64130 [FIN, ACK] Seq=1 Ack=2 Win=78464 Len=0
1848	11.204589	192.168.124.17	39.156.66.18	TCP	54 64130 → 80 [ACK] Seq=2 Ack=2 Win=131072 Len=0

第一次握手，主动关闭方发送 FIN=1, ACK=1, SEQ=1, ACK_SEQ=1, 状态 FIN_WAIT。

```

Sequence Number: 1      (relative sequence number)
Sequence Number (raw): 2864143095
[Next Sequence Number: 2      (relative sequence number)]
Acknowledgment Number: 1      (relative ack number)
Acknowledgment number (raw): 469747574
0101 .... = Header Length: 20 bytes (5)

```

✧ Flags: 0x011 (FIN, ACK)

```

000. .... = Reserved: Not set
...0 .... = Nonce: Not set
.... 0... = Congestion Window Reduced (CWR): Not set
.... .0.. = ECN-Echo: Not set
.... ..0. = Urgent: Not set
.... ...1 = Acknowledgment: Set
.... .... 0... = Push: Not set
.... .... .0.. = Reset: Not set
.... .... ..0. = Syn: Not set
> .... .... ...1 = Fin: Set
> [TCP Flags: .....A...F]

```

Window: 512

第二次握手，被动关闭方收到 FIN，回复 ACK=1, ACK_SEQ=2, 状态 CLOSE_WAIT。

Sequence Number: 1 (relative sequence number)

Sequence Number (raw): 469747574

[Next Sequence Number: 1 (relative sequence number)]

Acknowledgment Number: 2 (relative ack number)

Acknowledgment number (raw): 2864143096

0101 = Header Length: 20 bytes (5)

✓ Flags: 0x010 (ACK)

000. = Reserved: Not set

...0 = Nonce: Not set

.... 0... = Congestion Window Reduced (CWR): Not set

.... .0.. = ECN-Echo: Not set

.... ..0. = Urgent: Not set

.... ...1 = Acknowledgment: Set

.... 0... = Push: Not set

....0.. = Reset: Not set

....0. = Syn: Not set

....0 = Fin: Not set

[TCP Flags:A.....]

Window: 2452

第三次握手：close，被动关闭方发送 FIN=1, ACK=1, SEQ=1, ACK_SEQ=2, 状态：LAST_ACK

[TCP Segment Len: 0]

Sequence Number: 1 (relative sequence number)

Sequence Number (raw): 469747574

[Next Sequence Number: 2 (relative sequence number)]

Acknowledgment Number: 2 (relative ack number)

Acknowledgment number (raw): 2864143096

0101 = Header Length: 20 bytes (5)

✓ Flags: 0x011 (FIN, ACK)

000. = Reserved: Not set

...0 = Nonce: Not set

.... 0... = Congestion Window Reduced (CWR): Not set

.... .0.. = ECN-Echo: Not set

.... ..0. = Urgent: Not set

.... ...1 = Acknowledgment: Set

.... 0... = Push: Not set

....0.. = Reset: Not set

....0. = Syn: Not set

>1 = Fin: Set

第四次握手，主动关闭方收到 FIN，回复 ACK=1, SEQ=2, ACK_SEQ=2, 状态：TIME_WAIT。

```

Sequence Number: 2      (relative sequence number)
Sequence Number (raw): 2864143096
[Next Sequence Number: 2      (relative sequence number)]
Acknowledgment Number: 2      (relative ack number)
Acknowledgment number (raw): 469747575
0101 ..... = Header Length: 20 bytes (5)
v Flags: 0x010 (ACK)
  000. .... .... = Reserved: Not set
  ...0 .... .... = Nonce: Not set
  .... 0... .... = Congestion Window Reduced (CWR): Not set
  .... .0.. .... = ECN-Echo: Not set
  .... ..0. .... = Urgent: Not set
  .... ...1 .... = Acknowledgment: Set
  .... .... 0... = Push: Not set
  .... .... .0.. = Reset: Not set
  .... .... ..0. = Syn: Not set
  .... .... ...0 = Fin: Not set
  [TCP Flags: .....A.....]
Window: 512

```

五、总结

通过本次实验，使用 Wireshark 了解到了 UDP 协议和 TCP 协议，明确了 UDP 协议和 TCP 协议的工作原理，更直观地认识到了 TCP 建立连接三次握手和断开连接四次握手的过程，对 TCP 和 UDP 协议有了更深刻的理解。