计算机网络原理：第四次作业&802.3协议实验

刘泓尊 2018011446 计84

4.2

纯ALOHA协议最大吞吐量0.184，所以可用带宽0.184 x 56Kb/s = 10.304 Kb/s. 每个站的带宽1000 / 100 = 10b/s = 0.01 Kb/s

所以N = 10.304 / 0.01 = 1030.4

所以最多可以有1030个站，N的最大值为**1030**

4.13

以太网使用曼彻斯特编码，则发送每一位需要有2个信号周期。经典以太网的数据率为10Mbps，所以波特率是数据量的2倍。为**20MBaud**.

4.14

0表示低电平，1表示高电平。则编码输出为:

01 01 01 10 10 10 01 10 01 10

即`01010110101001100110`

4.15

首先双方监听冲突，两站距离最远为1km，单向传播时延1km / 200m/us = 5us. 所以发送方至少需要2x5us = 10us才能确定自己抓住了信道。

发送方传输数据需要256b / 10Mbps + 1km / 200m/us = 30.6us

接收方抓住信道时间：10us

接收方传输数据需要32 / 10Mbps + 1km / 200m/us = 8.2us

所以总时间为 10 + 30.6 + 10 + 8.2 = 58.8 us

有效数据224bit

所以有效数据率为 224b / 58.8us = **3.81 Mbps**

4.18

快速以太网只需要将电缆的最大长度降低到经典以太网的1/10, 就可以及时检测冲突，照搬10Mbps的经典以太网。

4.25

一帧全部正确的概率为(1 - 10^{-7})^(64x8) = 0.9999488

出错率为1 - 0.9999488 = 0.0000512

每秒帧数11Mbps / 512bit = 21484.375个

所以平均每秒有0.0000512 x 21484.375 = 1.1帧

所以平均每秒约1帧被损坏

4.27

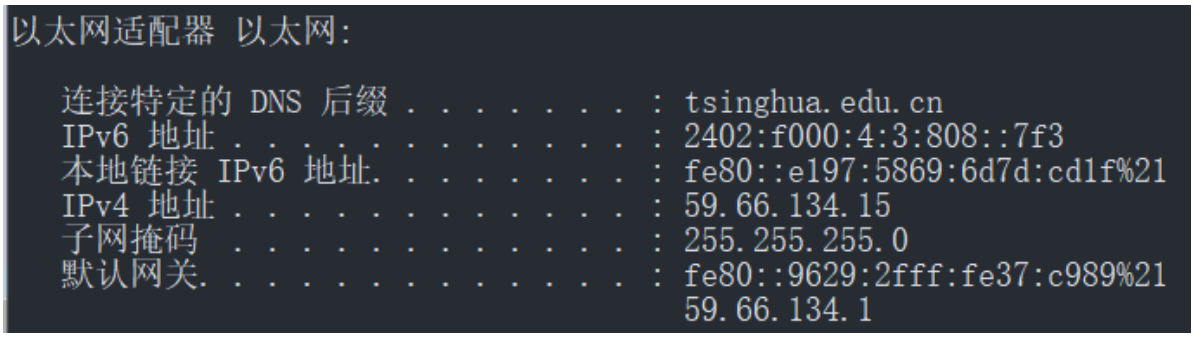
一个原因是要求实时通信质量的场合，发现错误之后必须纠正。

另一个原因是在通信质量很低的信道上，错误率很高，如果仅仅使用检错，那么所有错误帧都需要重传，甚至再次损坏。在这时使用纠错码可以降低重传率。

IEEE 802.3协议实验

实验内容

将我的实验主机连接到以太网，用`ipconfig`命令查看主机ip为`59.66.134.15`



一段时间内捕捉到的**目的地为实验主机**的数据帧如下：

No.	Time	Source	Destination	Protocol	Length	Info
55	3.502170	43.243.235.142	59.66.134.15	TCP	60	443 → 62520 [ACK] Seq=1 Ack=518 Win=62848 Len=0
56	3.503389	43.243.235.142	59.66.134.15	TLSv1.2	1514	Server Hello
57	3.504052	43.243.235.142	59.66.134.15	TCP	1514	443 → 62520 [ACK] Seq=1461 Ack=518 Win=62848 Len=1460 [
63	3.557962	58.87.81.176	59.66.134.15	TCP	66	443 → 62521 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=
66	3.562085	58.87.81.176	59.66.134.15	TCP	60	443 → 62521 [ACK] Seq=1 Ack=560 Win=30720 Len=0
67	3.562794	58.87.81.176	59.66.134.15	TLSv1.2	210	Server Hello, Change Cipher Spec, Encrypted Handshake M
70	3.566792	58.87.81.176	59.66.134.15	TCP	60	443 → 62521 [ACK] Seq=157 Ack=1095 Win=31744 Len=0
71	3.575456	58.87.81.176	59.66.134.15	TCP	60	443 → 62518 [ACK] Seq=3816 Ack=1089 Win=31744 Len=0
72	3.579210	58.87.81.176	59.66.134.15	TCP	60	443 → 62519 [ACK] Seq=3816 Ack=1254 Win=31744 Len=0
73	3.591516	182.254.79.23	59.66.134.15	UDP	356	8000 → 49335 Len=314
76	3.631628	58.87.81.176	59.66.134.15	TCP	1478	443 → 62518 [ACK] Seq=3816 Ack=1089 Win=31744 Len=1424
77	3.632323	58.87.81.176	59.66.134.15	TCP	1478	443 → 62518 [ACK] Seq=5240 Ack=1089 Win=31744 Len=1424
78	3.632323	58.87.81.176	59.66.134.15	TCP	1478	443 → 62518 [ACK] Seq=6664 Ack=1089 Win=31744 Len=1424
79	3.632323	58.87.81.176	59.66.134.15	TCP	1478	443 → 62518 [ACK] Seq=8088 Ack=1089 Win=31744 Len=1424
80	3.632323	58.87.81.176	59.66.134.15	TLSv1.2	333	Application Data
84	3.637574	58.87.81.176	59.66.134.15	TCP	60	443 → 62518 [ACK] Seq=9791 Ack=1567 Win=32768 Len=0

1.依次查看捕获的各数据帧，看看目的地为实验主机的数据帧中长度最小的是多大；查看这种帧的各个域，看看先导域是否包含在记录的数据中；记录的数据是从哪个字段开始，至哪个字段结束？这是否验证了IEEE 802.3 标准中规定的最小帧长为64 字节?

从上图中可以看到，目的地为实验主机的数据帧中长度最小为60Byte.

查看其中一个60Byte的数据帧，各个域如下：

```
> Frame 66: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interface \De
v Ethernet II, Src: NewH3CTe_37:c9:89 (94:29:2f:37:c9:89), Dst: LCFCHeFe_be:d7:ff (8c:
  > Destination: LCFCHeFe_be:d7:ff (8c:16:45:be:d7:ff)
  > Source: NewH3CTe_37:c9:89 (94:29:2f:37:c9:89)
    Type: IPv4 (0x0800)
    Padding: 000000000000
> Internet Protocol Version 4, Src: 58.87.81.176, Dst: 59.66.134.15
> Transmission Control Protocol, Src Port: 443, Dst Port: 62521, Seq: 1, Ack: 560, Len
```

```
0000  8c 16 45 be d7 ff 94 29  2f 37 c9 89 08 00 45 00   ··E····) /7····E·
0010  00 28 23 d2 40 00 34 06  d5 a5 3a 57 51 b0 3b 42   ·(#·@·4·  ··:WQ·;B
0020  86 0f 01 bb f4 39 47 09  c4 b0 67 88 b6 76 50 10   ·····9G·  ··g··vP·
0030  00 3c 42 92 00 00 00 00  00 00 00 00              ·<B·····  ····
```

从上图可以看到捕捉到的帧第一个域就是`destination`.所以先导域不再数据中。

记录的数据从`destination`开始，到`padding`结束。

和IEEE 802.3相比少了4字节的校验和，所以实际上最小帧长还是64字节。这验证了IEEE 802.3 标准中规定的最小帧长为64 字节。

2.查找捕获的帧中长度最长的帧。可以多访问一些网页以捕获更多的帧，看看这些帧的长度最大是多大？为什么?

从上面的图中可以看到，最大帧长为1514字节。

各个域的解析如下：

```
> Frame 57: 1514 bytes on wire (12112 bits), 1514 bytes captured (12112 bits) on in
v Ethernet II, Src: NewH3CTe_37:c9:89 (94:29:2f:37:c9:89), Dst: LCFCHeFe_be:d7:ff (
  > Destination: LCFCHeFe_be:d7:ff (8c:16:45:be:d7:ff)
  > Source: NewH3CTe_37:c9:89 (94:29:2f:37:c9:89)
    Type: IPv4 (0x0800)
v Internet Protocol Version 4, Src: 43.243.235.142, Dst: 59.66.134.15
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
  > Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 1500
    Identification: 0xc9c9 (51657)
```

这1514字节包括了6字节目的地址 + 6字节源地址 + 2字节类型 + 1500字节数据。没有包含4字节校验和。

这验证了最大数据长度为1500字节。

3.找到捕捉的数据帧中由实验主机发出的ARP 请求（request）帧，辨认其目的地址域和源地址域，如图2-2 所示。看看它的目的MAC 地址是多少？再ipconfig –all 命令查看实验主机的MAC 地址，看看是否和该帧中的源地址一致?

捕捉到ARP请求帧如下：

```
> Frame 4770: 42 bytes on wire (336 bits), 42 bytes captured (336 bits) on interface \Devic
v Ethernet II, Src: LCFCHeFe_be:d7:ff (8c:16:45:be:d7:ff), Dst: Broadcast (ff:ff:ff:ff:ff:f
  > Destination: Broadcast (ff:ff:ff:ff:ff:ff)
  > Source: LCFCHeFe_be:d7:ff (8c:16:45:be:d7:ff)
    Type: ARP (0x0806)
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: LCFCHeFe_be:d7:ff (8c:16:45:be:d7:ff)
    Sender IP address: 59.66.134.15
    Target MAC address: 00:00:00_00:00:00 (00:00:00:00:00:00)
    Target IP address: 59.66.134.1
```

可以看到其

目的MAC地址域为`ff:ff:ff:ff:ff:ff`

源MAC地址域为`8c:16:45:be:d7:ff`

使用`ipconfig -all`命令查看本机MAC地址:

```
以太网适配器 以太网:

   连接特定的 DNS 后缀 . . . . . . . : tsinghua.edu.cn
   描述. . . . . . . . . . . . . . . : Realtek PCIe GBE Family Controller
   物理地址. . . . . . . . . . . . . : 8C-16-45-BE-D7-FF
   DHCP 已启用 . . . . . . . . . . . : 是
   自动配置已启用. . . . . . . . . . : 是
   IPv6 地址 . . . . . . . . . . . . : 2402:f000:4:3:808::7f3(首选)
   获得租约的时间  . . . . . . . . . : 2020年12月8日 15:37:07
   租约过期的时间  . . . . . . . . . : 2020年12月9日 15:37:07
   本地链接 IPv6 地址. . . . . . . . : fe80::e197:5869:6d7d:cd1f%21(首选)
   IPv4 地址 . . . . . . . . . . . . : 59.66.134.15(首选)
   子网掩码 . . . . . . . . . . . . : 255.255.255.0
   获得租约的时间  . . . . . . . . . : 2020年12月8日 15:37:08
   租约过期的时间  . . . . . . . . . : 2020年12月8日 19:37:08
   默认网关. . . . . . . . . . . . . : fe80::9629:2fff:fe37:c989%21
                                       59.66.134.1
   DHCP 服务器 . . . . . . . . . . . : 166.111.8.18
   DHCPv6 IAID . . . . . . . . . . . : 59512389
   DHCPv6 客户端 DUID  . . . . . . . : 00-01-00-01-22-DB-49-91-8C-16-45-BE-D7-FF
   DNS 服务器  . . . . . . . . . . . : 2402:f000:1:801::8:28
                                       2402:f000:1:801::8:29
                                       166.111.8.28
                                       166.111.8.29
```

可以看到 实验主机的MAC地址 和 该帧中的源地址 一致。

4.对比一下封装ARP 分组的帧和其他帧（封装IP 分组的帧），看看它们的类型字段分别是多少?

从上面几张截图中都可以看到:

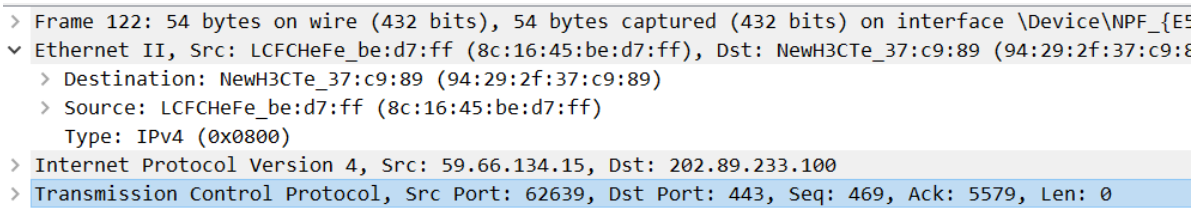
封装ARP分组的帧：`0x0806`

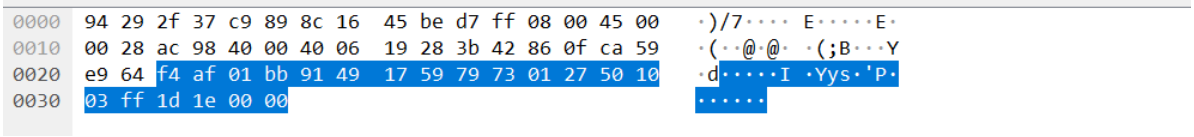
封装IP分组的帧：`0x0800`

2.1.4 思考题

1.在验证最小帧长的时候，选择的数据帧是目的地为实验主机的数据帧。如果选择由实验主机发出的数据帧则会发现，帧长度可能会比60 字节还小，例如图2-2 中的帧就只有42 字节。试分析这种帧的各个域，并解释这一现象。

下图是一个由实验主机发出的数据帧，长度只有**54字节**:





从对各个域的解析可以看到，该数据帧到了数据字段就结束了（依然包括目的地址、源地址、类型、数据），但**没有`padding`部分**。再加上省略掉的校验和，长度也小于64字节。

没有填充部分的**原因**是：当数据字段的长度小于46字节时，MAC子层就会在数据字段的后面填充以满足数据帧长不小于64字节。由于填充数据是由MAC子层负责，也就是设备驱动程序。不同的抓包程序和设备驱动程序所处的优先层次可能不同，抓包程序的优先级可能比设备驱动程序更高，也就是说，我们的抓包程序可能在设备驱动程序还没有填充不到64字节的帧的时候，抓包程序已经捕获了数据。

2.上网查找资料，看看除了IP 和ARP 之外，还有那些IEEE 802.3 协议支持的网络层分组类型，编码分别是什么?

查阅资料得到了一些常见的EtherType 值：

IP(v4): 0x0800

ARP: 0x0806

DRARP/RARP: 0x8035

IPX(因特网包交换): 0x8137

IPv6: 0x86dd

PPP: 0x880b

PPPoE: (发现阶段)0x8863, (会话阶段)0x8864

802.1Q tag: 0x8100

MPLS(组播)：0x8848

MPLS(单播)：0x8847

对于更多的网络层类型字段，请参阅https://en.wikipedia.org/wiki/EtherType