计算机网络原理：第五次作业&IP-ICMP-ARP协议实验

刘泓尊 2018011446 计84

5.2

不是。虚电路网络建立预先确定的路由连接的时候需要有将**建立连接数据包**从任意源端路由到任意接收方的能力。

5.3

在连接建立的时候可能要协商**窗口的大小、最大分组尺寸和超时值。**

5.9

4800 = 15 * 16 * 20

所以选择15个簇、16个区、每个区20个路由；（或者20个簇、16个区、每个区15个路由，或其他等效形式）的时候，路由表尺寸最少。

这时路由表尺寸为15 + 16 + 20 = **51**

5.10

家乡代理通过响应ARP请求来欺骗路由器，使得路由器认为家乡代理是移动主机。

具体来说，当路由器得到一个发送给移动主机的IP包时，它广播一个ARP请求，请求具有该IP的MAC地址。当移动主机不存在时，家乡代理响应ARP。所以路由器将移动用户的IP地址和家乡代理的802.3 MAC地址关联。

5.22

是。只需把分组封装在属于所经过的子网的数据报的载荷段中，并进行发送。

5.28

掩码20位。全1为广播地址，全0为网络号。

所以主机数量2^{32 - 20}-2 = 4094个

5.34

可以。

在安装了NAT之后，与单个连接相关的所有数据包都必须通过同一个路由器进出公司，因为这是存储映射的地方。如果每个路由器都有自己的IP地址，并且属于某个给定连接的所有流量都可以发送到同一个路由器，那么映射就可以正确地完成，可以使用NAT。

5.40

16字节总地址数为2^128 = 3.4 x 10^38

持续 3.4 x 10^38 / 10^6 / 10^12 = 3.4 x 10^20 s = 10^13 year

大约**10^13年**可以全部分配。

5.42

从概念性上来讲，不需要改变。在技术性上讲，因为被请求的IP地址变大了，所以需要**更大的域**来存储IPv6地址。

IP-ICMP-ARP协议实验

注：IP实验在校园网环境下完成，ICMP和ARP实验在宿舍局域网下完成。

2.3

实验内容

1．观察通常的IPv4 分组

1.Version 字段的值是不是4？IHL 字段的值一般是多少？结合IPv4 分组头部格式可以看出IHL 的单位是什么？有没有更大IHL 值的IPv4 分组？如果有的话找到这种分组数据部分的开始。

下图是一个IPv4分组的数据包：

```
> Frame 123: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface \Device\NPF_{E5722
v Ethernet II, Src: NewH3CTe_37:c9:89 (94:29:2f:37:c9:89), Dst: LCFCHeFe_be:d7:ff (8c:16:45: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: 182.61.200.6, 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: 60
    Identification: 0x5dd2 (24018)
  > Flags: 0x00
    Fragment Offset: 0
    Time to Live: 53
    Protocol: ICMP (1)
```

从图中可以看到，`Version` 字段的值是4。`IHL` 字段的值一般是5，`IHL` 的单位是字，即4 字节。所以该数据包IPv4头部长度为20Byte.

IHL的最小值是5，最大十进制值是15。由于IPv4首部可能包含数目不定的选项，这个字段也用来确定数据的偏移量。

有更大的IHL值，为15：这时表示含 `Options` **字段**， 因为 `Options` 字段最大长度是 40 Bytes ，因此IPV4包的 Header 长度 最大也就60Bytes,所以只要带有 `Options` 字段，那么 `IHL` 字段的值 一般都大于5。

2.观察Type of service 字段。找几个分组验证一下 `Total length` ， `Header checksum` 以及 `Source address` 和 `Destination address` 字段是否正确。

`Type of service` 字段基本上都是 `0x00` . 该字段指出分组的重要程度，大多数情况下不使用该字段。如下图。

```
v Internet Protocol Version 4, Src: 59.66.134.15, Dst: 182.254.79.23
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
  v Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
      0000 00.. = Differentiated Services Codepoint: Default (0)
      .... ..00 = Explicit Congestion Notification: Not ECN-Capable Transport (0)
    Total Length: 262
    Identification: 0x92c4 (37572)
```

找了若干分组，发现Total length，Header checksum 以及Source address 和Destination address 字段都正确。

3.观察`tracert`程序发送的一系列分组中TTL字段，它们有何特点?

`tracert`程序第一次请求的帧为:

```
Internet Protocol Version 4, Src: 59.66.134.15, Dst: 182.61.200.7
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
  > Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 92
    Identification: 0x8c56 (35926)
  > Flags: 0x00
    Fragment Offset: 0
  > Time to Live: 1
    Protocol: ICMP (1)
    Header Checksum: 0xedb4 [validation disabled]
    [Header checksum status: Unverified]
    Source Address: 59.66.134.15
```

第二次请求的帧为:

```
Internet Protocol Version 4, Src: 59.66.134.15, Dst: 182.61.200.7
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
  > Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 92
    Identification: 0x8c59 (35929)
  > Flags: 0x00
    Fragment Offset: 0
  > Time to Live: 2
```

结合后面几帧，总结规律得出，**TTL从1 开始递增。**

4.观察PC2、PC3 及R1 的ping 应答分组中的TTL 值（由于在同一局域网内，TTL 的值就等于分组发送时填入的值）。Windows 和Linux 操作系统中初始的TTL 值有何不同？利用这一点大体判断一下那些用ping 测试过的互联网上站点运行的是哪一类操作系统，以及分组到达PC1之前经过了多少个路由器。

Windows 操作系统中的Ping应答分组数据帧:

```
    Time to Live: 128
    Protocol: ICMP (1)
    Header Checksum: 0xd6ad [validation disabled]
    [Header checksum status: Unverified]
    Source Address: 192.168.3.26
    Destination Address: 192.168.3.4
Internet Control Message Protocol
    Type: 0 (Echo (ping) reply)
```

Linux 操作系统中的Ping应答分组数据帧:

```
Time to Live: 64
Protocol: ICMP (1)
Header Checksum: 0x82d5 [validation disabled]
[Header checksum status: Unverified]
Source Address: 192.168.3.50
Destination Address: 192.168.3.4
Internet Control Message Protocol
Type: 0 (Echo (ping) reply)
```

Windows 操作系统中初始的TTL 值多为**128**，而Linux 的多为64, Unix多为255。我的ubuntu 20.04操作系统的初始TTL为**64**.

`ping www.baidu.com`得到的应答帧为：

```
Time to Live: 52
Protocol: ICMP (1)
Header Checksum: 0x6e71 [validation disabled]
[Header checksum status: Unverified]
Source Address: 39.156.66.18
Destination Address: 192.168.3.4
Internet Control Message Protocol
Type: 0 (Echo (ping) reply)
Code: 0
```

初步判断该应答主机为**Linux操作系统**，经过了64 - 52 = 12跳路由器（不包括它本身）。下面是`tracert`的运行结果，验证了我们的计算：

```
C:\Users\lenovo>tracert www.baidu.com

通过最多 30 个跃点跟踪
到 www.a.shifen.com [39.156.66.18] 的路由:

  1     1 ms     2 ms     1 ms  192.168.3.1
  2     5 ms     4 ms    10 ms  100.102.192.1
  3     8 ms     8 ms     7 ms  211.136.66.29
  4     *        *        *     请求超时。
  5     *        *        *     请求超时。
  6     *        *        *     请求超时。
  7     9 ms     9 ms     8 ms  39.156.27.5
  8    10 ms     9 ms    10 ms  39.156.67.29
  9     *        *        *     请求超时。
 10     *        *        *     请求超时。
 11     *        *        *     请求超时。
 12     *        *        *     请求超时。
 13    12 ms    10 ms     8 ms  39.156.66.18
```

5.不同分组的`Identification`值是否相同?

一般是不同的。这个字段主要被用来唯一地标识一个报文的所有分片，因为分片不一定按序到达，所以在重组时需要知道分片所属的报文。每产生一个数据报，计数器加1，并赋值给此字段。如果相同说明在同一个分组。

6. `ping -f` 命令发送的分组 `DF` 值是否设1了?

运行`ping www.baidu.com -f`命令，可以看到`DF`已经被置为1。

```
Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    0000 00.. = Differentiated Services Codepoint: Default (0)
    .... ..00 = Explicit Congestion Notification: Not ECN-Capable Transport (0)
  Total Length: 60
  Identification: 0xeb1f (60191)
Flags: 0x40, Don't fragment
    0... .... = Reserved bit: Not set
    .1.. .... = Don't fragment: Set
    ..0. .... = More fragments: Not set
  Fragment Offset: 0
  Time to Live: 64
```

7.TCP、UDP、ICMP 对应的 `Protocol` 值分别是什么?

TCP: 0x06

UDP: 0x11

ICMP: 0x01

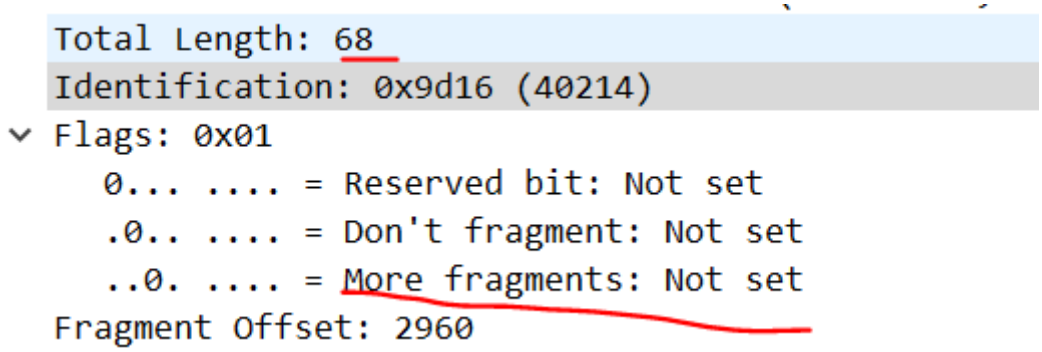
2．观察IPv4 分段与重组

1.可以看到应答分组的返回时间比起通常ping应答要长一些，并且有可能刚开始几ping请求会得不到应答，为什么?

可以看到分成了**3个IPv4分组**，每个分组的长度信息如下：

```
Total Length: 1500
Identification: 0x9d16 (40214)
Flags: 0x20, More fragments
    0... .... = Reserved bit: Not set
    .0.. .... = Don't fragment: Not set
    ..1. .... = More fragments: Set
  Fragment Offset: 0
```

```
Total Length: 1500
Identification: 0x9d16 (40214)
Flags: 0x20, More fragments
    0... .... = Reserved bit: Not set
    .0.. .... = Don't fragment: Not set
    ..1. .... = More fragments: Set
  Fragment Offset: 1480
```



应答分组的返回时间比起通常ping应答要长一些，并且有可能刚开始几ping请求会得不到应答，是IPv4分段造成的结果。

2.图中三个分段的`Identification`值是多少？是否相等以表明它们是同一个IPv4 分组分段得到的?

`Identification`为`0x9d16`.

三者相等，所以是同一个IPv4分组得到的。

3.前两个分段a、b 的`Flags`值多少？观察对应的16 进制原始数据看看这表示`MF`值为多少？是否还有更多分段（More Fragment）？

均为`0x02`.

对应的`MF`值为1.

说明还有更多分段。

4.第三个分组c 的`Flags`值为多少？表示了什么意思?

第三个分组的flag为`0x00`. 表明这已经是最后一个分段。

5.图中`Fragment offset`的值依次为多少，以确保在乱序到达时也能正确重组出原来的分组?

依次为**0， 1480， 2960**。以确保在乱序到达时也能正确重组出原来的分组

6.三个分段的总的数据长度为1500+1500+68-3*20=3008，比ping 命令中的参数3000多了8，为什么?

因为最后一个分段包括了8字节的ICMP头部。

7.细心的读者可能已经发现ping 分组的载荷是一些简单字母的重复（从字母a 到字母w，然后重复），利用这一点可以观察IPv4 分段重组后恢复成原来的数据的过程。第一个分段中最后一个字母、第二个分段中第一个字母、第二个分段中最后一个字母、第三个分段中第一个字母各是什么?

第一个分段最后一个字母：w

第二个分段中第一个字母：a

第二个分段中最后一个字母：h

第三个分段中第一个字母：i

3．观察IP 选项的使用

运行`ping -r 8 192.168.3.50`，(该IP为同一局域网下的主机)

Ping请求分组：

```
Options: (36 bytes), Record Route
  IP Option - Record Route (35 bytes)
    Type: 7
    Length: 35
    Pointer: 4
    Empty Route: 0.0.0.0 <- (next)
    Empty Route: 0.0.0.0
    Empty Route: 0.0.0.0
    Empty Route: 0.0.0.0
    Empty Route: 0.0.0.0
    Empty Route: 0.0.0.0
    Empty Route: 0.0.0.0
    Empty Route: 0.0.0.0
  IP Option - End of Options List (EOL)
Internet Control Message Protocol
```

```
000  48 a4 72 30 e2 56 30 24  32 74 6e 13 08 00 4e 00   H
010  00 60 9d 1b 00 00 40 01  41 d8 c0 a8 03 04 c0 a8
020  03 32 07 23 04 00 00 00  00 00 00 00 00 00 00 00
030  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
040  00 00 00 00 00 00 08 00  4c 7f 00 01 00 dc 61 62
```

Ping应答分组：

```
Source Address: 192.168.3.50
Destination Address: 192.168.3.4
Options: (36 bytes), Record Route
  IP Option - Record Route (35 bytes)
    Type: 7
    Length: 35
    Pointer: 12
    Recorded Route: 192.168.3.50
    Recorded Route: 192.168.3.50
    Empty Route: 0.0.0.0 <- (next)
    Empty Route: 0.0.0.0
    Empty Route: 0.0.0.0
    Empty Route: 0.0.0.0
    Empty Route: 0.0.0.0
    Empty Route: 0.0.0.0
  IP Option - End of Options List (EOL)
```

1.可以看到在ping 请求分组a 中，IPv4 选项的code 值为多少？表示什么?

IPv4选项的值为`0x07`, 表示记录路径。

2.len 值为多少？表示可以记录多少条IPv4 地址（此时指针为0x04，而IPv4 地址记录都是0）？

len = 0x23 = 35.

表示可以记录8条IPv4地址。因为`0x04 - 1 + 8 x 4 = 35`.

3.ping 应答分组b 与a 的不同之处在于分组经过路径上的路由器出口地址被记录下来了，图中指针的值是多少？表示下一条记录是第几条记录?

指针的值为`12`. 表示下一条记录为第12 / 4 = 3条。

思考题

1.什么情况下IPv4 分组需要分段？在哪里分段？又是在哪里重新组装起来的?

以太网上数据帧的最大长度限制为`1500Byte`。某些广域网为`576Byte`. 所以，当IPv4包长度大于最大长度时，只好采用分段技术。

分段在发送方网络层进行分段，通常在源主机上进行（而不是网络路由设备）。但是在某些中间设备如防火墙和NAT路由设备上，可能需要查看完整的数据包内容，这时候也可能进行分段和重组。

重组是在接收方网络层实现，通常在目的主机上进行。其余情况同上。

2.阅读RFC791，看看IPv4 定义的选项（option）类型有哪些?

`Option Type`（8bit)通常包括1 bit的`Copied`域，2bit 的 `Option Class`域，和 5bit 的 `Option Number`域。

`Copied`指示options字段是否需要被拷贝到所有分段。

`Option Class`是一个选项分类指示。0 为control, 2为debugging and measurement, 1和3被保留。

`Option Number`域的类型如下：

Option Number(5bit)	Option Name	Description
0 / 0x00	EOOL	End of Option List
1 / 0x01	NOP	No Operation
2 / 0x02	SEC	Security (defunct)
7 / 0x07	RR	Record Route
10 / 0x0A	ZSU	Experimental Measurement
11 / 0x0B	MTUP	MTU Probe
12 / 0x0C	MTUR	MTU Reply
15 / 0x0F	ENCODE	ENCODE
25 / 0x19	QS	Quick-Start
30 / 0x1E	EXP	RFC3692-style Experiment
68 / 0x44	TS	Time Stamp
82 / 0x52	TR	Traceroute
94 / 0x5E	EXP	RFC3692-style Experiment
130 / 0x82	SEC	Security (RIPSO)
131 / 0x83	LSR	Loose Source Route
133 / 0x85	E-SEC	Extended Security (RIPSO)
134 / 0x86	CIPSO	Commercial IP Security Option
136 / 0x88	SID	Stream ID
137 / 0x89	SSR	Strict Source Route
142 / 0x8E	VISA	Experimental Access Control
144 / 0x90	IMITD	IMI Traffic Descriptor
145 / 0x91	EIP	Extended Internet Protocol
147 / 0x93	ADDEXT	Address Extension
148 / 0x94	RTRALT	Router Alert
149 / 0x95	SDB	Selective Directed Broadcast
151 / 0x97	DPS	Dynamic Packet State
152 / 0x98	UMP	Upstream Multicast Pkt.
158 / 0x9E	EXP	RFC3692-style Experiment
205 / 0xCD	FINN	Experimental Flow Control
222 / 0xDE	EXP	RFC3692-style Experiment

参见https://en.wikipedia.org/wiki/IPv4#Options

2.4

实验内容

1．观察ICMPv4 目标不可达消息

运行`tftp 192.168.3.50 get 1.txt`命令。

请求为：

```
Internet Protocol Version 4, Src: 192.168.3.4, Dst: 192.168.3.50
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
  > Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 45
    Identification: 0x9d1f (40223)
  > Flags: 0x00
    Fragment Offset: 0
    Time to Live: 64
    Protocol: UDP (17)
    Header Checksum: 0x561a [validation disabled]
    [Header checksum status: Unverified]
    Source Address: 192.168.3.4
    Destination Address: 192.168.3.50
> User Datagram Protocol, Src Port: 62650, Dst Port: 69
> Trivial File Transfer Protocol
```

收到如下应答：

```
Internet Control Message Protocol
    Type: 3 (Destination unreachable)
    Code: 3 (Port unreachable)
    Checksum: 0x84ae [correct]
    [Checksum Status: Good]
    Unused: 00000000
  Internet Protocol Version 4, Src: 192.168.3.4, Dst: 192.168.3.50
      0100 .... = Version: 4
      .... 0101 = Header Length: 20 bytes (5)
    > Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
      Total Length: 45
      Identification: 0x9d1f (40223)
    > Flags: 0x00
      Fragment Offset: 0
      Time to Live: 64
      Protocol: UDP (17)
      Header Checksum: 0x561a [validation disabled]
      [Header checksum status: Unverified]
      Source Address: 192.168.3.4
      Destination Address: 192.168.3.50
  > User Datagram Protocol, Src Port: 62650, Dst Port: 69
> Trivial File Transfer Protocol
```

1.观察捕捉到的 ICMPv4 目标不可达消息的各个字段。Type、Code、Checksum 字段是否和标准一致？unused 及后面的内容是否正确？

可以看到Type = 3, Code = 3, Checksum正确。和标准一致。

unused为0，和标准一致。

数据段为请求的IPv4数据包。和发送的请求一致。所以都是正确的。

2.如果Code 值为10，表示什么含义?

表示目的主机被**强制禁止**。

2．观察ICMPv4 超时消息

执行如下命令：

```
C:\Users\lenovo>ping www.baidu.com -i 3

正在 Ping www.a.shifen.com [39.156.66.18] 具有 32 字节的数据:
来自 211.136.66.29 的回复: TTL 传输中过期。
来自 211.136.66.29 的回复: TTL 传输中过期。
来自 211.136.66.29 的回复: TTL 传输中过期。
来自 211.136.66.29 的回复: TTL 传输中过期。

39.156.66.18 的 Ping 统计信息:
    数据包: 已发送 = 4，已接收 = 4，丢失 = 0 (0% 丢失)，
```

1.命令行下显示的信息为：

`Reply from 211.136.66.29: TTL expired in transit.`

表示什么含义?

表示路径上的路由器`211.136.66.29`传回了TTL超时信息。从2.3.1节可以看到`211.136.66.29`正是路径上的第三个路由器IP。

2.利用Wireshark 捕捉分组可以看到`211.136.66.29`向PC1 发回了ICMPv4 超时消息，如图所示。其中ICMPv4 部分的各个字段是否与标准一致?

```
    Source Address: 211.136.66.29
    Destination Address: 192.168.3.4
Internet Control Message Protocol
    Type: 11 (Time-to-live exceeded)
    Code: 0 (Time to live exceeded in transit)
    Checksum: 0x9fa3 [correct]
    [Checksum Status: Good]
    Unused: 00000000
  Internet Protocol Version 4, Src: 192.168.3.4, Dst: 39.156.66.18
```

可以看到Type = 11，Code = 0， unused = 0, checksum 正确。

所以ICMPv4部分的各个字段和标准一致。

3.观察ICMPv4 回显请求及应答消息

执行`ping www.baidu.com`命令，得到如下请求和应答报文信息：

```
Internet Control Message Protocol
    Type: 8 (Echo (ping) request)
    Code: 0
    Checksum: 0x4c74 [correct]
    [Checksum Status: Good]
    Identifier (BE): 1 (0x0001)
    Identifier (LE): 256 (0x0100)
    Sequence Number (BE): 231 (0x00e7)
    Sequence Number (LE): 59136 (0xe700)
    [Response frame: 25644]
  Data (32 bytes)
      Data: 6162636465666768696a6b6c6d6e6f707172737475767761626364656667686 9
      [Length: 32]
```

```
Internet Control Message Protocol
    Type: 0 (Echo (ping) reply)
    Code: 0
    Checksum: 0x5474 [correct]
    [Checksum Status: Good]
    Identifier (BE): 1 (0x0001)
    Identifier (LE): 256 (0x0100)
    Sequence Number (BE): 231 (0x00e7)
    Sequence Number (LE): 59136 (0xe700)
    [Request frame: 25643]
    [Response time: 11.755 ms]
  Data (32 bytes)
      Data: 6162636465666768696a6b6c6d6e6f707172737475767761626364656667686 9
      [Length: 32]
```

1.利用Wireshark 捕捉分组，观察ICMPv4 回显请求及应答消息，其中ICMPv4 部分的各个字段是否与标准一致?

可以看到：请求包Type = 8, code = 0, checksum正确；应答包Type = 0, code = 0, checksum正确。所以ICMPv4 部分的各个字段是否与标准一致。

2.一对请求与应答的标识符、序号以及数据是否相等?

从上面两幅图可以看出，一对请求和应答的**标识符、序号和数据**都**相等**。

思考题

1.为什么有些类型ICMPv4消息（例如目标不可达消息）中有一unused（未使用）字段，而另一些（例如回显消息）则没有？注意它们的长度，分析这样设计可能是出于什么考虑。

有的ICMPv4 Header使用了unused字段是一种填充，它保证了所有的ICMPv4 Header长度均为8字节。这样就不需要再Header部分存储ICMP包头长度，节省了空间和节点计算开销。同时，因为格式规整，硬件可以并行地解析头部的多个字段，提高了处理的效率。（变长头部和不规整的格式都会使得硬件处理复杂化）。

2.上网查找资料，看看ICMPv4 消息的隐患，以及黑客是如何利用它发起攻击的，由此思考为什么很多系统不发送ICMPv4 消息。

ICMPv4包可以实现多种网络攻击，比如ICMP超时攻击、ICMP头部攻击、ICMP重定向攻击等等。

ICMPv4超时攻击：发送TTL=n的包，当它到达防火墙时刚好TTL=0，所以发送大量此类包就可以构成DDoS攻击。

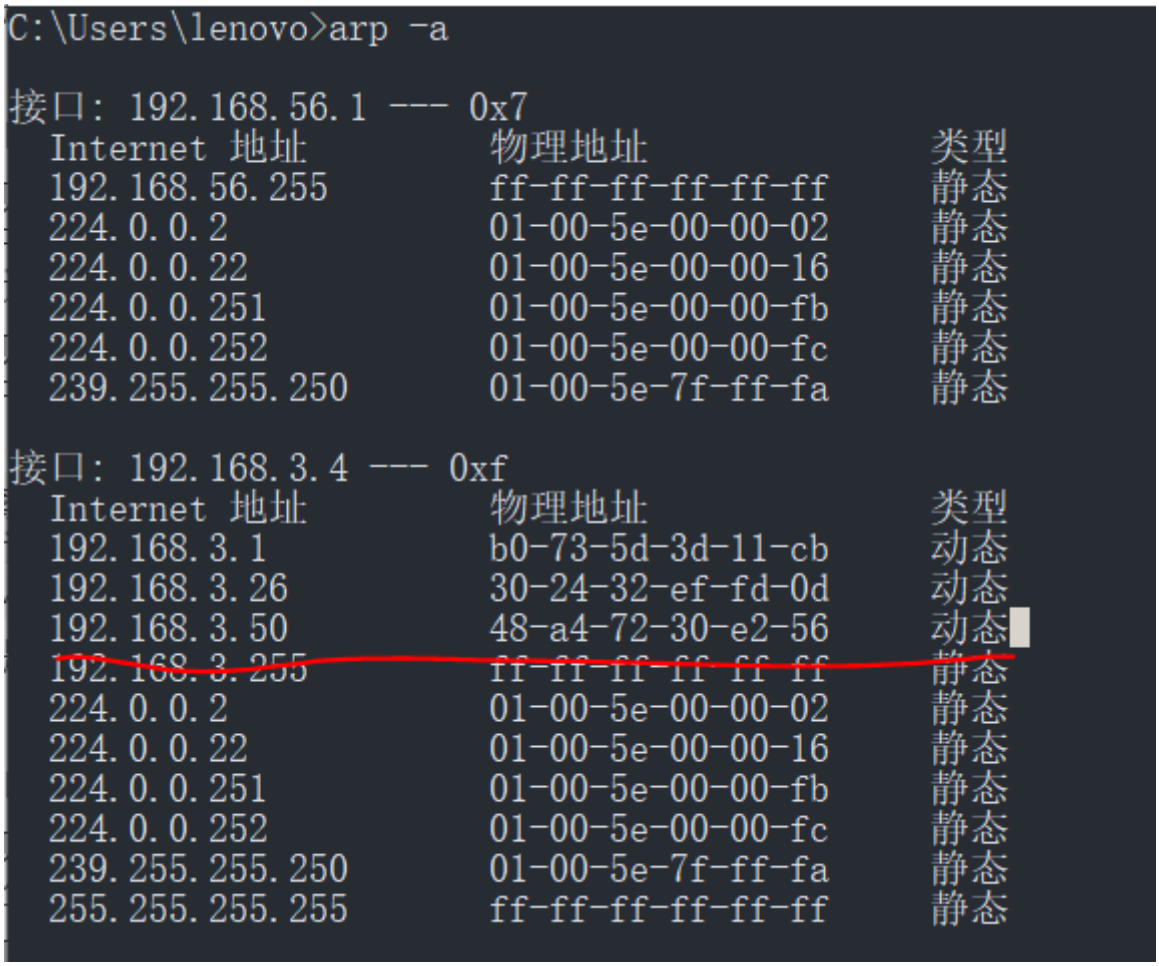
ICMPv4重定向攻击：攻击者通过发送ICMP重定向报文可以在目标机路由表中添加一条到达特定主机的路由信息，使得受害者发到特定主机的数据包被放到攻击者主机，攻击者可以进行欺骗和监听。核心是欺骗路由表。解决办法是通过防火墙屏蔽所有的ICMP重定向报文。

基于上述ICMPv4的安全隐患，很多系统不发送ICMPv4消息，以免目标机将自己识别为攻击者，并且还能保证自己的安全。

2.5

实验内容

`ping 192.168.3.50`之后，执行`arp -a`命令得到如下结果：

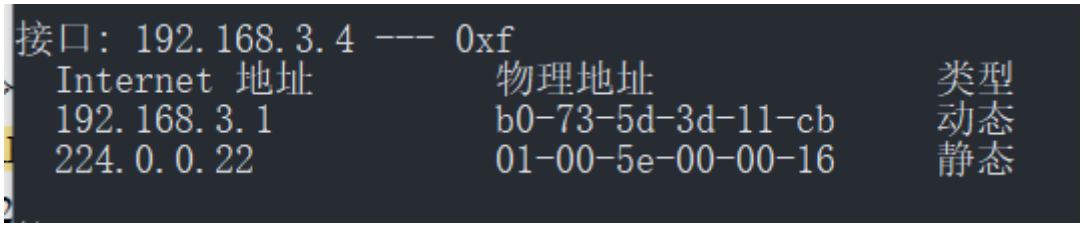


1．观察ARP 缓存生存时间

1.如果在生存期内没有用到该缓存，会有什么情况发生?

等待2min之后，再次执行`arp -a`命令，可以看到该表项已经被删除。这表明ARP动态缓存的生存期为**2min**.

下图展示了2min之后的arp表项，可以看到该ip对应项**已经被删除。**



2.如果在生存期内该缓存被使用了（比ping 192.168.3.50).会有什么情况发生?

如果2min之内使用了该表项，那么自使用之时起生存期再延长2min.

被删除了的ARP动态缓存需要通过广播新的ARP请求-应答来获得。下面是对应的包结构：

```
> Destination: Broadcast (ff:ff:ff:ff:ff:ff)
> Source: IntelCor_74:6e:13 (30:24:32:74:6e:13)
  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: IntelCor_74:6e:13 (30:24:32:74:6e:13)
  Sender IP address: 192.168.3.4
  Target MAC address: 00:00:00_00:00:00 (00:00:00:00:00:00)
  Target IP address: 192.168.3.50
```

2．观察ARP 过程

在命令行下用`arp -d`命令删除PC1 上的所有ARP 表项，然后用`ping 192.168.3.50`命令来触发ARP过程。根据自己观察到的现象解释ARP 过程。

抓到的包依次为：

```
IntelCor_74:6e:13   Broadcast          ARP      42 Who has 192.168.3.50? Tell 192.168.3.4
IntelCor_30:e2:56   IntelCor_74:6e:13  ARP      42 192.168.3.50 is at 48:a4:72:30:e2:56
192.168.3.4         192.168.3.50       ICMP     74 Echo (ping) request  id=0x0001, seq=243/62
192.168.3.50        192.168.3.4        ICMP     74 Echo (ping) reply    id=0x0001, seq=243/62
192.168.3.4         192.168.3.50       ICMP     74 Echo (ping) request  id=0x0001, seq=244/62
192.168.3.50        192.168.3.4        ICMP     74 Echo (ping) reply    id=0x0001, seq=244/62
```

运行`ping`命令时，PC1首先检查ARP缓存，发现没有`192.168.3.50`对应的MAC地址，所以发送广播ARP请求分组。PC2返回ARP应答，告知PC1自己的MAC地址。之后进行第一次ICMP请求和应答。后面3次不会触发ARP请求分组的发送，直接通过本地ARP缓存得到。

3．观察ARP 分组格式

1.观察封装ARP请求/应答分组的以太网帧的内容，看看有何异同?

ARP请求分组的以太网帧内容：

```
v Ethernet II, Src: LCFCHeFe_be:d7:ff (8c:16:45:be:d7:ff), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
  > 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
```

ARP应答分组的以太网帧内容：

```
Ethernet II, Src: NewH3CTe_37:c9:89 (94:29:2f:37:c9:89), Dst: LCFCHeFe_be:d7:ff (8c:16:45: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: ARP (0x0806)
    Padding: 000000000000000000000000000000000000
Address Resolution Protocol (reply)
    Hardware type: Ethernet (1)
    Protocol type: IPv4 (0x0800)
    Hardware size: 6
    Protocol size: 4
    Opcode: reply (2)
    Sender MAC address: NewH3CTe_37:c9:89 (94:29:2f:37:c9:89)
    Sender IP address: 59.66.134.1
    Target MAC address: LCFCHeFe_be:d7:ff (8c:16:45:be:d7:ff)
    Target IP address: 59.66.134.15
```

异同：ARP请求分组的以太网帧内容只显示了**以太网头部**，源MAC地址是本机MAC地址，目标地址的广播地址，分组类型为`0x0806`.

ARP应答分组的以太网帧内容包含了**以太网头部和尾部。**目的地址是单播地址`30:24:32:74:6e:13`. 以太网尾部被**填充**为16字节全0.

2.观察ARP请求/应答分组内容，看看有何异同?

相同之处：硬件类型为以太网`0x0001`, 协议类型为IP(`0x0800`), 硬件地址长度6，协议地址长度为4.

不同之处：请求分组操作类型为请求(`0x0001`)，源MAC地址为`30:24:32:74:6e:13`，目的MAC地址位置，填充为全0. 目的IP地址为`192.168.3.50`.

应答分组的操作类型为应答(`0x0002`), 源MAC地址为`48::a4:72:30:e2:56`, 目的MAC地址为`30:24:32:74:6e:13`。目的IP地址为`192.168.3.4`。

4．观察无偿ARP

下图是我抓到的无偿ARP request和ARP reply, 需要注意的是，我的版本的wireshark将无偿ARP request显示为ARP annoncement, 但实际上包的含义正是**无偿ARP请求包。**

```
Address Resolution Protocol (ARP Announcement)
    Hardware type: Ethernet (1)
    Protocol type: IPv4 (0x0800)
    Hardware size: 6
    Protocol size: 4
    Opcode: request (1)
    [Is gratuitous: True]
    [Is announcement: True]
    Sender MAC address: NewH3CTe_29:f2:01 (74:ea:c8:29:f2:01)
    Sender IP address: 183.173.112.1
    Target MAC address: 00:00:00_00:00:00 (00:00:00:00:00:00)
    Target IP address: 183.173.112.1
```

```
0000  ff ff ff ff ff ff 74 ea  c8 29 f2 01 08 06 00 01   ·····t· ·)·····
0010  08 00 06 04 00 01 74 ea  c8 29 f2 01 b7 ad 70 01   ······t· ·)····p·
0020  00 00 00 00 00 00 b7 ad  70 01 00 00 00 00 00 00   ········ p·······
0030  00 00 00 00 00 00 00 00  00 00 00 00               ········ ····
```

```
Address Resolution Protocol (reply/gratuitous ARP)
    Hardware type: Ethernet (1)
    Protocol type: IPv4 (0x0800)
    Hardware size: 6
    Protocol size: 4
    Opcode: reply (2)
    [Is gratuitous: True]
    Sender MAC address: NewH3CTe_29:f2:01 (74:ea:c8:29:f2:01)
    Sender IP address: 183.173.112.1
    Target MAC address: Guangdon_a8:c1:ff (44:66:fc:a8:c1:ff)
    Target IP address: 183.173.112.1
```

```
0000  ff ff ff ff ff ff 74 ea  c8 29 f2 01 08 06 00 01   ......t· ·)......
0010  08 00 06 04 00 02 74 ea  c8 29 f2 01 b7 ad 70 01   ......t· ·)....p·
0020  44 66 fc a8 c1 ff b7 ad  70 01 00 00 00 00 00 00   Df...... p·......
0030  00 00 00 00 00 00 00 00  00 00 00 00              ........ ....
```

1.观察无偿ARP 分组的个数，试分析其原因?

利用wireshark得到的无偿ARP请求分组如下：

```
    45 0.119690    NewH3CTe_29:f2:01   Broadcast   ARP   60 ARP Announcement for 183.173.144.1
101715 315.603659  NewH3CTe_29:f2:01   Broadcast   ARP   60 ARP Announcement for 183.173.144.1
216016 742.423297  NewH3CTe_29:f2:01   Broadcast   ARP   60 ARP Announcement for 183.173.144.1
```

初始化某个IP地址时发送无偿ARP的数量为**3**，这是系统默认值。可以通过Windows注册表来改变。

过程为：B 发送广播请求分组，A 回复给 B 一个应答分组，A 再发送广播请求分组。

2.比较ARP request 和 ARP reply的不同之处。

PC2首先发送一个无偿ARP请求，企图初始化自己的IP地址为192.168.1.46. PC1受到请求之后回复给PC2对应的应答。然后PC2广播一个无偿ARP请求帧，目的是使网络上的其他计算机获得正确的ARP缓存。重复2次就可以确定是否有重复的IP地址。

从上面两图中可以知道，无偿ARP应答分组的源和目的IP地址相同，但是源和目的MAC地址不同。所以源节点和目的节点就可以知道同一网段内有冲突的IP地址。

其他不同还有：

请求分组的 Opcode 是 request (1)，应答分组的 Opcode 是 reply (2)；

请求分组是广播，应答分组是单播。

思考题

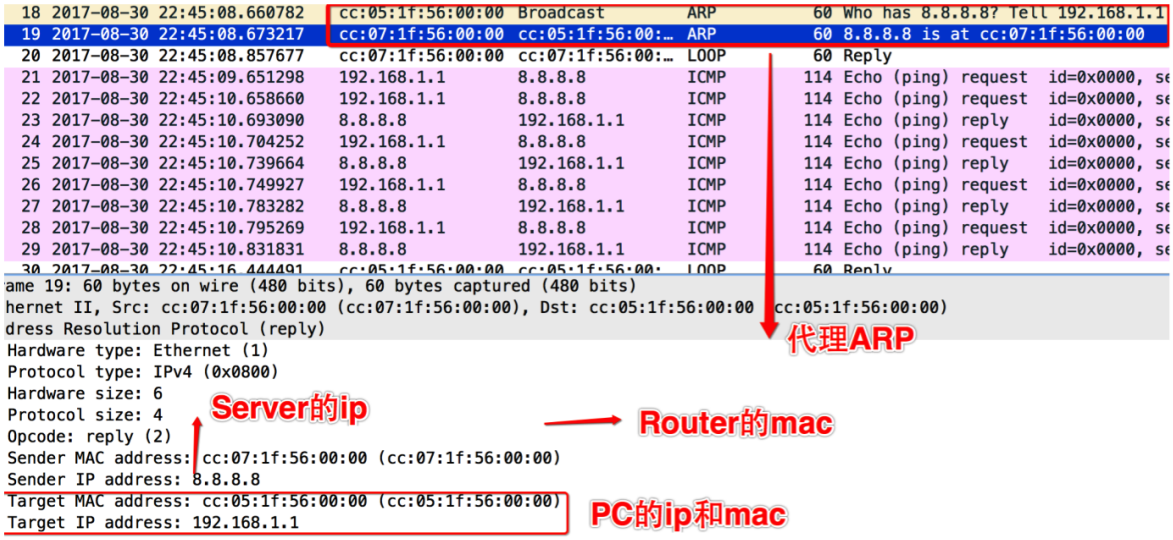
1.试用Wireshark观察ARP代理过程。

电脑没有网关时，ARP直接询问目标IP对应的MAC地址（跨网段），采用代理ARP；电脑有网关时，ARP只需询问网关IP对应的MAC地址（同网段），采用正常ARP；无论是正常ARP还是代理ARP，电脑最终都拿到同一个目标MAC地址：网关MAC。

当ARP请求目标跨网段时，网关设备收到此ARP请求，会用自己的MAC地址返回给请求者，这便是代理ARP。必须满足两个条件：①网关已经开启代理ARP功能；②网关有目标的路由信息。

从上面的讨论可以知道，**实现ARP代理需要本机关闭Default-Gateway**.

之后执行`ping 8.8.8.8`,通过wireshark观察过程：

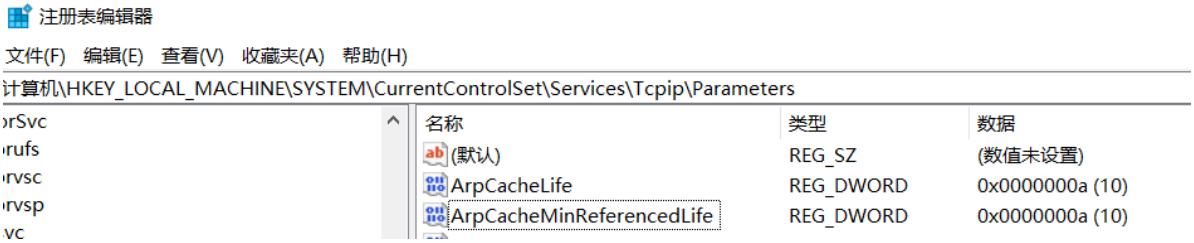


详细的信息已经在图中标注。可以看到，由于PC1没有设置默认网关，所以直接采用代理ARP方式询问。

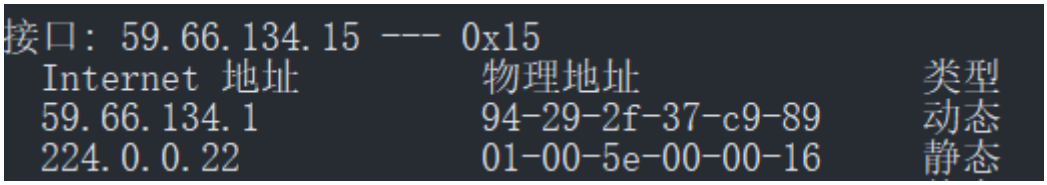
运行ping命令时，PC1首先检查ARP缓存，发现没有8.8.8.8对应的MAC地址，所以发送广播ARP请求分组。直接询问跨网段目的8.8.8.8的IP地址。由于第一跳路由器Router默认开启了代理ARP功能，所以直接用自己的MAC地址(`cc:07:1f:56:00:00`)回应了。随后PC1更新ARP缓存。之后进行第一次ICMP请求和应答。后面3次不会触发ARP请求分组的发送，直接通过本地ARP缓存得到。这里第一跳路由器充当了“代理”的角色，使得PC1可以访问到其他网段的资源。

2.查阅相关文献，尝试在注册表中更改动ARP缓存的生存时间，并观察更改后的动态ARP缓存生存时间。

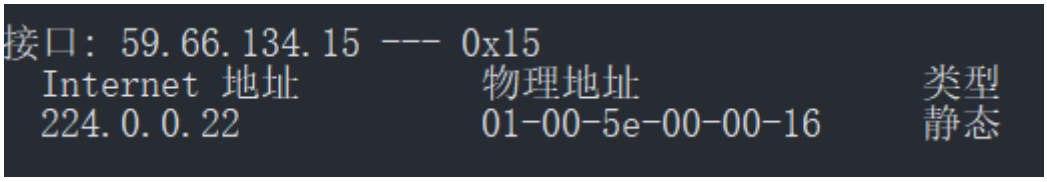
修改注册表项如下，设定缓存失效时间为**10s**：



重启计算机，执行`arp -d`之后执行`ping www.baidu.com`，再执行`arp -a`，可以看到此时arp表中有默认网关`59.66.134.1`的MAC地址。

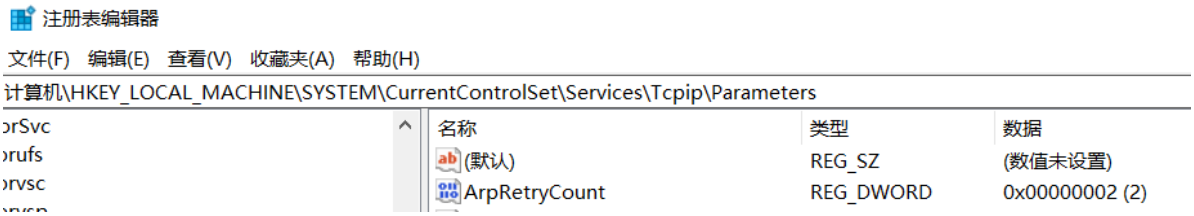


等待10s之后，再次查ARP缓存，发现该项已经消失，说明ARP缓存的生存时间修改成功。

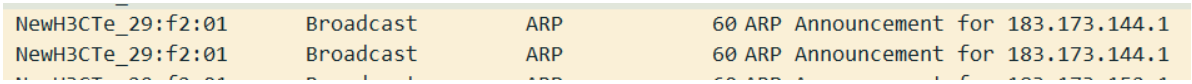


3.查阅相关文献，尝试在注册表中更改无偿ARP发送的数量值，并观察更改后的无偿ARP过程。

注册表中对`ArpRetryCount`的设置控制了无偿ARP的发送数量。如果发送了`ArpRetryCount`个无偿ARP后，都没有收到ARP回应，IP就假定此IP地址在此网络段中是唯一的。修改该数量为2：



更改之后在另一台主机上捕捉到的无偿ARP过程如下：



可以看到PC1修改注册表后，**广播的无偿ARP个数变为了2**. 设置成功。