第6章　ICMP：Internet控制报文协议

6.1　引言

ICMP经常被认为是IP层的一个组成部分。它传递差错报文以及其他需要注意的信息。ICMP报文通常被IP层或更高层协议（TCP或UDP）使用。一些ICMP报文把差错报文返回给用户进程。

ICMP报文是在IP数据报内部被传输的，如图6-1所示。

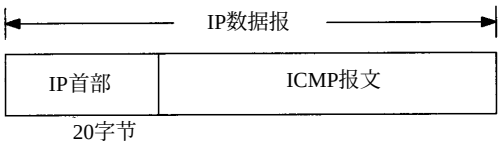

图6-1　ICMP封装在IP数据报内部

ICMP 的正式规范参见 RFC 792 [Posterl 1981b]。

ICMP报文的格式如图6-2所示。所有报文的前4个字节都是一样的，但是剩下的其他字节则互不相同。下面我们将逐个介绍各种报文格式。

类型字段可以有15个不同的值，以描述特定类型的ICMP报文。某些ICMP报文还使用代码字段的值来进一步描述不同的条件。

检验和字段覆盖整个ICMP报文。使用的算法与我们在3.2节中介绍的IP首部检验和算法相同。ICMP的检验和是必需的。

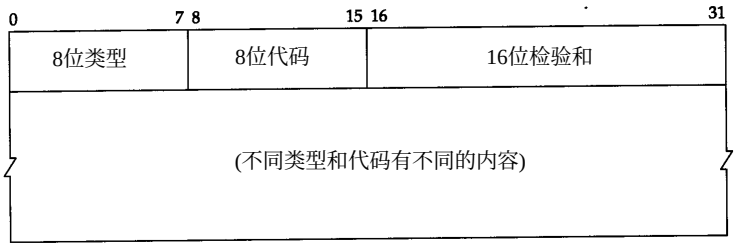

图6-2　ICMP报文

在本章中，我们将一般地讨论ICMP报文，并对其中一部分作详细介绍：地址掩码请求和应答、时间戳请求和应答以及不可达端口。我们将详细介绍第 27章Ping程序所使用的回应请求和应答报文和第9章处理IP路由的ICMP报文。

6.2　ICMP报文的类型

各种类型的ICMP报文如图6-3所示，不同类型由报文中的类型字段和代码字段来共同决定。

图中的最后两列表明ICMP报文是一份查询报文还是一份差错报文。因为对 ICMP差错报文有时需要作特殊处理，因此我们需要对它们进行区分。例如，在对 ICMP差错报文进行响应时，永远不会生成另一份ICMP差错报文（如果没有这个限制规则，可能会遇到一个差错产生另一个差错的情况，而差错再产生差错，这样会无休止地循环下去）。

当发送一份ICMP差错报文时，报文始终包含IP的首部和产生ICMP差错报文的IP数据报的前8个字节。这样，接收ICMP差错报文的模块就会把它与某个特定的协议（根据 IP数据报首

类 型	代 码	描 述	查 询	差 错
0	0	回显应答(Ping应答，第7章)	•	
3		目的不可达：		
	0	网络不可达（9.3节）		•
	1	主机不可达（9.3节）		•
	2	协议不可达		•
	3	端口不可达（6.5节）		•
	4	需要进行分片但设置了不分片比特（11.6节）		•
	5	源站选路失败（8.5节）		•
	6	目的网络不认识		•
	7	目的主机不认识		•
	8	源主机被隔离（作废不用）		•
	9	目的网络被强制禁止		•
	10	目的主机被强制禁止		•
	11	由于服务类型TOS，网络不可达（9.3节）		•
	12	由于服务类型TOS，主机不可达（9.3节）		•
	13	由于过滤，通信被强制禁止		•
	14	主机越权		•
	15	优先权中止生效		•
4	0	源端被关闭（基本流控制，11.11节）		•
5		重定向（9.5节）：		•
	0	对网络重定向		•
	1	对主机重定向		•
	2	对服务类型和网络重定向		•
	3	对服务类型和主机重定向		•
8	0	请求回显（Ping请求，第7章）	•	
9	0	路由器通告（9.6节）	•	
10	0	路由器请求（9.6节）	•	
11		超时：		
	0	传输期间生存时间为0（Traceroute, 第8章）		•
	1	在数据报组装期间生存时间为0（11.5节）		•
12		参数问题：		
	0	坏的IP首部（包括各种差错）		•
	1	缺少必需的选项		•
13	0	时间戳请求（6.4节）	•	
14	0	时间戳应答（6.4节）	•	
15	0	信息请求（作废不用）	•	
16	0	信息应答（作废不用）	•	
17	0	地址掩码请求（6.3节）	•	
18	0	地址掩码应答（6.3节）	•	

图6-3 ICMP报文类型

部中的协议字段来判断）和用户进程（根据包含在IP数据报前8个字节中的TCP或UDP报文首部中的TCP或UDP端口号来判断）联系起来。6.5节将举例来说明一点。

下面各种情况都不会导致产生ICMP差错报文：

1) ICMP差错报文（但是，ICMP查询报文可能会产生ICMP差错报文）。

2) 目的地址是广播地址（见图3-9）或多播地址（D类地址，见图1-5）的IP数据报。

3) 作为链路层广播的数据报。

4) 不是IP分片的第一片（将在11.5节介绍分片）。

5) 源地址不是单个主机的数据报。这就是说，源地址不能为零地址、环回地址、广播地址或多播地址。

这些规则是为了防止过去允许ICMP差错报文对广播分组响应所带来的广播风暴。

6.3 ICMP地址掩码请求与应答

ICMP地址掩码请求用于无盘系统在引导过程中获取自己的子网掩码（ 3.5节）。系统广播它的ICMP请求报文（这一过程与无盘系统在引导过程中用 RARP获取IP地址是类似的）。无盘系统获取子网掩码的另一个方法是 BOOTP协议，我们将在第 16章中介绍。 ICMP地址掩码请求和应答报文的格式如图 6-4所示。

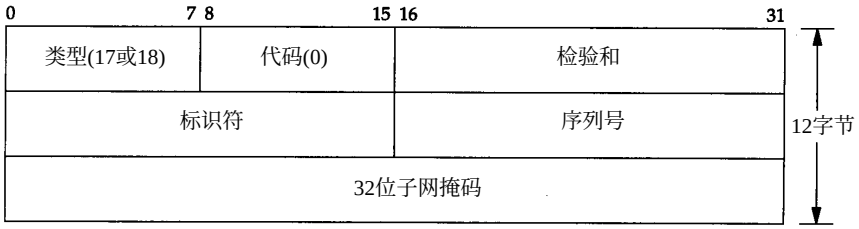

图6-4 ICMP地址掩码请求和应答报文

ICMP报文中的标识符和序列号字段由发送端任意选择设定，这些值在应答中将被返回。这样，发送端就可以把应答与请求进行匹配。

我们可以写一个简单的程序（取名为 icmpaddrmask），它发送一份ICMP地址掩码请求报文，然后打印出所有的应答。由于一般是把请求报文发往广播地址，因此这里我们也这样做。目的地址（140.252.13.63）是子网140.252.13.32的广播地址（见图3-12）。

```
sun % icmpaddrmask 140.252.13.63
received mask = ffffffe0, from 140.252.13.34    来自本机
received mask = ffffffe0, from 140.252.13.35    来自bsdi
received mask = ffff0000, from 140.252.13.33    来自svr4
```

在输出中我们首先注意到的是，从 svr4返回的子网掩码是错的。显然，尽管 svr4接口已经设置了正确的子网掩码，但是 SVR4还是返回了一个普通的 B类地址掩码，就好像子网并不存在一样。

```
svr4 %ifconfig emd0
emd0: flags=23<UP,BROADCAST,NOTRAILERS>
        inet 140.252.13.34 netmask ffffffe0 broadcast 140.252.13.63
```

SVR4处理ICMP地址掩码请求过程存在差错。

我们用tcpdump命令来查看主机bsdi上的情况，输出如图 6-5所示。我们用 - e选项来查看硬件地址。

```
1  0.0                  8:0:20:3:f6:42 ff:ff:ff:ff:ff:ff ip 60:
                        sun > 140.252.13.63: icmp: address mask request
2  0.00 (0.00)          0:0:c0:6f:2d:40 ff:ff:ff:ff:ff:ff ip 46:
                        bsdi > sun: icmp: address mask is 0xffffffe0
3  0.01 (0.01)          0:0:c0:c2:9b:26 8:0:20:3:f6:42 ip 60:
                        svr4 > sun: icmp: address mask is 0xffff0000
```

图6-5 发到广播地址的ICMP地址掩码请求

注意，尽管在线路上什么也看不见，但是发送主机 sun也能接收到ICMP应答（带有上面“来自本机”的输出行）。这是广播的一般特性：发送主机也能通过某种内部环回机制收到一份广播报文拷贝。由于术语“广播”的定义是指局域网上的所有主机，因此它必须包括发送

主机在内（参见图2-4，当以太网驱动程序识别出目的地址是广播地址后，它就把分组送到网络上，同时传一份拷贝到环回接口）。

接下来，bsdi广播应答，而svr4却只把应答传给请求主机。通常，应答地址必须是单播地址，除非请求端的源IP地址是0.0.0.0。本例不属于这种情况，因此，把应答发送到广播地址是BSD/386的一个内部差错。

> RFC规定，除非系统是地址掩码的授权代理，否则它不能发送地址掩码应答（为了成为授权代理，它必须进行特殊配置，以发送这些应答。参见附录E）。但是，正如我们从本例中看到的那样，大多数主机在收到请求时都发送一个应答，甚至有一些主机还发送差错的应答。

最后一点可以通过下面的例子来说明。我们向本机IP地址和环回地址分别发送地址掩码请求：

```
sun % icmpaddrmask sun
received mask= ff000000, from 140.252.13.33
sun % icmpaddrmask localhost
received mask= ff000000, from 127.0.0.1
```

上述两种情况下返回的地址掩码对应的都是环回地址，即A类地址127.0.0.1。还有，我们从图2-4可以看到，发送给本机IP地址的数据报（140.252.12.33）实际上是送到环回接口。ICMP地址掩码应答必须是收到请求接口的子网掩码（这是因为多接口主机每个接口有不同的子网掩码），因此两种情况下地址掩码请求都来自于环回接口。

6.4 ICMP时间戳请求与应答

ICMP时间戳请求允许系统向另一个系统查询当前的时间。返回的建议值是自午夜开始计算的毫秒数，协调的统一时间（Coordinated Universal Time, UTC）（早期的参考手册认为UTC是格林尼治时间）。这种ICMP报文的好处是它提供了毫秒级的分辨率，而利用其他方法从别的主机获取的时间（如某些Unix系统提供的rdate命令）只能提供秒级的分辨率。由于返回的时间是从午夜开始计算的，因此调用者必须通过其他方法获知当时的日期，这是它的一个缺陷。

ICMP时间戳请求和应答报文格式如图6-6所示。

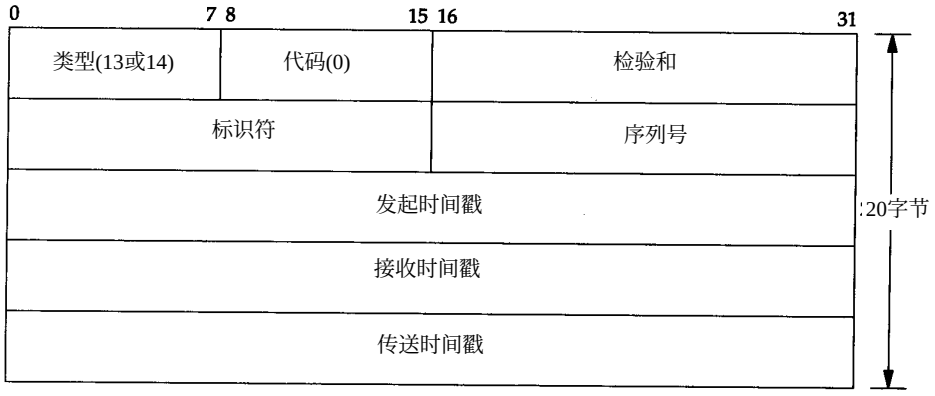

图6-6 ICMP时间戳请求和应答报文

请求端填写发起时间戳，然后发送报文。应答系统收到请求报文时填写接收时间戳，在发送应答时填写发送时间戳。但是，实际上，大多数的实现把后面两个字段都设成相同的值（提供三个字段的原因是可以让发送方分别计算发送请求的时间和发送应答的时间）。

6.4.1 举例

我们可以写一个简单程序（取名为 icmptime），给某个主机发送ICMP时间戳请求，并打印出返回的应答。它在我们的小互联网上运行结果如下：

```
sun % icmptime bsdi
orig = 83573336, recv = 83573330, xmit = 83573330, rtt = 2 ms
difference = -6 ms

sun % icmptime bsdi
orig = 83577987, recv = 83577980, xmit = 83577980, rtt = 2 ms
difference = -7 ms
```

程序打印出ICMP报文中的三个时间戳：发起时间戳（orig）、接收时间戳（recv）以及发送时间戳（xmit）。正如我们在这个例子以及下面的例子中所看到的那样，所有的主机把接收时间戳和发送时间戳都设成相同的值。

我们还能计算出往返时间（rtt），它的值是收到应答时的时间值减去发送请求时的时间值。difference的值是接收时间戳值减去发起时间戳值。这些值之间的关系如图6-7所示。

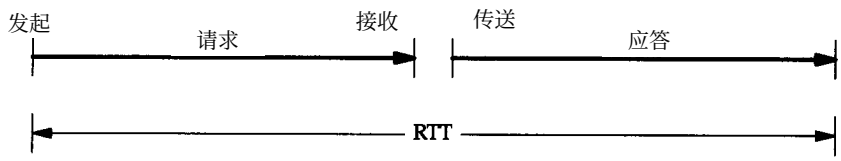

图6-7 icmptime 程序输出的值之间的关系

如果我们相信RTT的值，并且相信RTT的一半用于请求报文的传输，另一半用于应答报文的传输，那么为了使本机时钟与查询主机的时钟一致，本机时钟需要进行调整，调整值是difference减去RTT的一半。在前面的例子中，bsdi的时钟比sun的时钟要慢7 ms和8 ms。

由于时间戳的值是自午夜开始计算的毫秒数，即 UTC，因此它们的值始终小于86 400 000 (24×60×60×1000)。这些例子都是在下午4:00以前运行的，并且在一个比 UTC慢7个小时的时区，因此它们的值比82 800 000（2300小时）要大是有道理的。

如果对主机bsdi重复运行该程序数次，我们发现接收时间戳和发送时间戳的最后一位数总是0。这是因为该版本的软件（0.9.4版）只能提供10ms的时间分辨率（说明参见附录B）。

如果对主机svr4运行该程序两次，我们发现SVR4时间戳的最后三位数始终为0：

```
sun % icmptime svr4
orig = 83588210, recv = 83588000, xmit = 83588000, rtt = 4 ms
difference = -210 ms

sun % icmptime svr4
orig = 83591547, recv = 83591000, xmit = 83591000, rtt = 4 ms
difference = -547 ms
```

由于某种原因，SVR4在ICMP时间戳中不提供毫秒级的分辨率。这样，对秒以下的时间差调整将不起任何作用。

如果我们对子网 140.252.1上的其他主机运行该程序，结果表明其中一台主机的时钟与

sun相差3.7秒，而另一个主机时钟相差近75秒：

```
sun % icmptime gemini
orig = 83601883, recv = 83598140, xmit = 83598140, rtt = 247 ms
difference = -3743 ms

sun % icmptime aix
orig = 83606768, recv = 83532183, xmit = 83532183, rtt = 253 ms
difference = -74585 ms
```

另一个令人感兴趣的例子是路由器gateway（一个Cisco路由器）。它表明，当系统返回一个非标准时间戳值时（不是自午夜开始计算的毫秒数，UTC），它就用32 bit时间戳中的高位来表示。我们的程序证明了一点，在尖括号中打印出了接收和发送的时间戳值（在关闭高位之后）。另外，不能计算发起时间戳和接收时间戳之间的时间差，因为它们的单位不一致。

```
sun % icmptime gateway
orig = 83620811, recv = <4871036>, xmit = <4871036>, rtt = 220 ms

sun % icmptime gateway
orig = 83641007, recv = <4891232>, xmit = <4891232>, rtt = 213 ms
```

如果我们在这台主机上运行该程序数次，会发现时间戳值显然具有毫秒级的分辨率，而且是从某个起始点开始计算的毫秒数，但是起始点并不是午夜UTC（例如，可能是从路由器引导时开始计数的毫秒数）。

作为最后一个例子，我们来比较sun主机和另一个已知是准确的系统时钟——一个NTP stratum 1服务器（下面我们会更多地讨论NTP，网络时间协议）。

```
sun % icmptime clock.llnl.gov
orig = 83662791, recv = 83662919, xmit = 83662919, rtt = 359 ms
difference = 128 ms

sun % icmptime clock.llnl.gov
orig = 83670425, recv = 83670559, xmit = 83670559, rtt = 345 ms
difference = 134 ms
```

如果我们把difference的值减去RTT的一半，结果表明sun主机上的时钟要快38.5～51.5 ms。

6.4.2 另一种方法

还可以用另一种方法来获得时间和日期。

1) 在1.12节中描述了日期服务程序和时间服务程序。前者是以人们可读的格式返回当前的时间和日期，是一行ASCII字符。可以用telnet命令来验证这个服务：

```
sun % telnet bsdi daytime
Trying 140.252.13.35 ...
Connected to bsdi.
Escape character is '^]'.                前三行是Telnet客户的输出
Wed Feb  3 16:38:33 1993                 这是日期时间服务器的输出
Connection closed by foreign host.       这也是Telnet客户的输出
```

另一方面，时间服务程序返回的是一个32bit的二制进数值，表示自UTC，1900年1月1日午夜起算的秒数。这个程序是以秒为单位提供的日期和时间（前面我们提过的rdate命令使用的是TCP时间服务程序）。

2) 严格的计时器使用网络时间协议（NTP），该协议在RFC 1305中给出了描述[Mills 1992]。这个协议采用先进的技术来保证LAN或WAN上的一组系统的时钟误差在毫秒级以内。对计算机精确时间感兴趣的读者应该阅读这份RFC文档。

3) 开放软件基金会（OSF）的分布式计算环境（DCE）定义了分布式时间服务（DTS），

它也提供计算机之间的时钟同步。文献 [Rosenberg, Kenney and Fisher 1992]提供了该服务的其他细节描述。

4) 伯克利大学的Unix系统提供守护程序 `timed`(8)，来同步局域网上的系统时钟。不像NTP和DTS，`timed`不在广域网范围内工作。

6.5 ICMP端口不可达差错

最后两小节我们来讨论ICMP查询报文 —— 地址掩码和时间戳查询及应答。现在来分析一种ICMP差错报文，即端口不可达报文，它是 ICMP目的不可到达报文中的一种，以此来看一看ICMP差错报文中所附加的信息。使用UDP（见第11章）来查看它。

UDP的规则之一是，如果收到一份UDP数据报而目的端口与某个正在使用的进程不相符，那么UDP返回一个ICMP不可达报文。可以用TFTP来强制生成一个端口不可达报文（ TFTP将在第15章描述）。

对于TFTP服务器来说， UDP的公共端口号是 69。但是大多数的 TFTP客户程序允许用`connect`命令来指定一个不同的端口号。这里，我们就用它来指定 8888端口：

```
bsdi % tftp
tftp> connect svr4 8888                指定主机名和端口号
tftp> get temp.foo                     试图得到一个文件
Transfer timed out.                    大约25秒后
tftp> quit
```

`connect`命令首先指定要连接的主机名及其端口号，接着用 get命令来取文件。敲入 get命令后，一份UDP数据报就发送到主机svr4上的8888端口。tcpdump命令引起的报文交换结果如图6-8所示。

```
 1   0.0                  arp who-has svr4 tell bsdi
 2   0.002050 (0.0020)    arp reply svr4 is-at 0:0:c0:c2:9b:26

 3   0.002723 (0.0007)    bsdi.2924 > svr4.8888: udp 20
 4   0.006399 (0.0037)    svr4 > bsdi: icmp: svr4 udp port 8888 unreachable

 5   5.000776 (4.9944)    bsdi.2924 > svr4.8888: udp 20
 6   5.004304 (0.0035)    svr4 > bsdi: icmp: svr4 udp port 8888 unreachable

 7  10.000887 (4.9966)    bsdi.2924 > svr4.8888: udp 20
 8  10.004416 (0.0035)    svr4 > bsdi: icmp: svr4 udp port 8888 unreachable

 9  15.001014 (4.9966)    bsdi.2924 > svr4.8888: udp 20
10  15.004574 (0.0036)    svr4 > bsdi: icmp: svr4 udp port 8888 unreachable

11  20.001177 (4.9966)    bsdi.2924 > svr4.8888: udp 20
12  20.004759 (0.0036)    svr4 > bsdi: icmp: svr4 udp port 8888 unreachable
```

图6-8 由TFTP产生的ICMP端口不可达差错

在UDP数据报送到svr4之前，要先发送一份ARP请求来确定它的硬件地址（第1行）。接着返回ARP应答（第2行），然后才发送UDP数据报（第3行）（在`tcpdump`的输出中保留ARP请求和应答是为了提醒我们，这些报文交换可能在第一个 IP数据报从一个主机发送到另一个主机之前是必需的。在本书以后的章节中，如果这些报文与讨论的题目不相关，那么我们将省略它们）。

一个ICMP端口不可达差错是立刻返回的（第 4行）。但是，TFTP客户程序看上去似乎忽略了这个ICMP报文，而在5秒钟之后又发送了另一份 UDP数据报（第5行）。在客户程序放弃

之前重发了三次。

注意，ICMP报文是在主机之间交换的，而不用目的端口号，而每个 20字节的UDP数据报则是从一个特定端口（2924）发送到另一个特定端口（8888）。

跟在每个UDP后面的数字20指的是UDP数据报中的数据长度。在这个例子中， 20字节包括TFTP的2个字节的操作代码， 9个字节以空字符结束的文件名 `temp.foo`，以及9个字节以空字符结束的字符串`netascii`（TFTP报文的详细格式参见图15-1）。

如果用-e选项运行同样的例子，我们可以看到每个返回的 ICMP端口不可达报文的完整长度。这里的长度为70字节，各字段分配如图6-9所示。

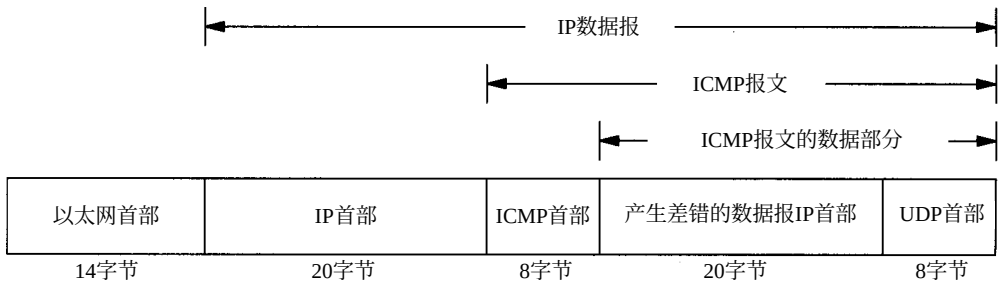

图6-9 “UDP端口不可达”例子中返回的ICMP报文

ICMP的一个规则是， ICMP差错报文（参见图6-3的最后一列）必须包括生成该差错报文的数据报IP首部（包含任何选项），还必须至少包括跟在该 IP首部后面的前8个字节。在我们的例子中，跟在IP首部后面的前8个字节包含UDP的首部（见图11-2）。

一个重要的事实是包含在 UDP首部中的内容是源端口号和目的端口号。就是由于目的端口号（8888）才导致产生了ICMP端口不可达的差错报文。接收 ICMP的系统可以根据源端口号（2924）来把差错报文与某个特定的用户进程相关联（在本例中是 TFTP客户程序）。

导致差错的数据报中的 IP首部要被送回的原因是因为 IP首部中包含了协议字段，使得ICMP可以知道如何解释后面的 8个字节（在本例中是 UDP首部）。如果我们来查看 TCP首部（图17-2）,可以发现源端口和目的端口被包含在 TCP首部的前8个字节中。

ICMP不可达报文的一般格式如图6-10所示。

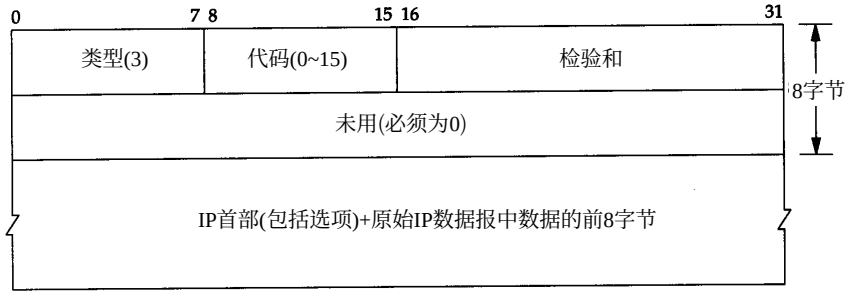

图6-10 ICMP不可达报文

在图6-3中，我们注意到有16种不同类型的ICMP不可达报文，代码分别从0到15。ICMP端口不可达差错代码是3。另外，尽管图6-10指出了在ICMP报文中的第二个32 bit字必须为0，但是当代码为4时（“需要分片但设置了不分片比特”），路径MTU发现机制（2.9节）却允许路由器把外

出接口的MTU填在这个32 bit字的低16 bit中。我们在11.6节中给出了一个这种差错的例子。

尽管ICMP规则允许系统返回多于8个字节的产生差错的IP数据报中的数据，但是大多数从伯克利派生出来的系统只返回 8个字节。 Solaris 2.2的ip_icmp_return_data_bytes选项默认条件下返回前64个字节（E.4节）。

tcpdump时间系列

在本书的后面章节中，我们还要以时间系列的格式给出tcpdump命令的输出，如图6-11所示。

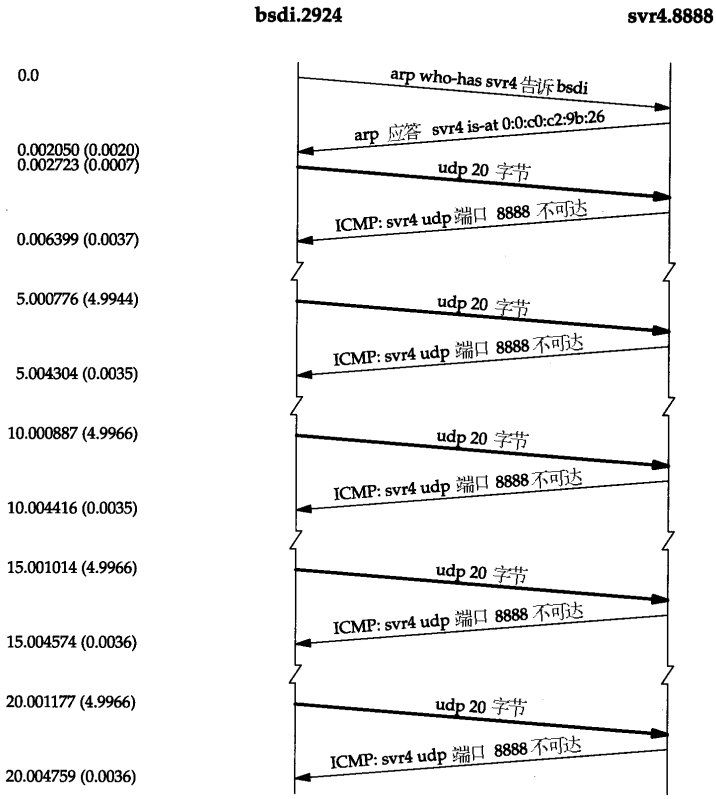

图6-11 发送到无效端口的TFTP请求的时间系列

时间随着向下而递增，在图左边的时间标记与 tcpdump命令的输出是相同的（见图 6-8）。位于图顶部的标记是通信双方的主机名和端口号。需要指出的是，随着页面向下的y坐标轴与真正的时间值不是成比例的。当出现一个有意义的时间段时，在本例中是每 5 秒之间的重发，我们就在时间系列的两侧作上标记。当UDP或TCP数据正在被传送时，我们用粗线的行来表示。

当ICMP报文返回时，为什么 TFTP客户程序还要继续重发请求呢？这是由于网络编程中的一个因素，即BSD系统不把从插口(socket)接收到的ICMP报文中的UDP数据通知用户进程，除非该进程已经发送了一个 connect命令给该插口。标准的 BSD TFTP客户程序并不发送connect命令，因此它永远也不会收到ICMP差错报文的通知。

这里需要注意的另一点是 TFTP客户程序所采用的不太好的超时重传算法。它只是假定 5秒是足够的，因此每隔5秒就重传一次，总共需要 25秒钟的时间。在后面我们将看到 TCP有一个较好的超时重发算法。

TFTP客户程序所采用的超时重传算法已被RFC所禁用。不过，在作者所在子网上的三个系统以及Solaris 2.2仍然在使用它。AIX 3.2.2采用一种指数退避方法来设置超时值，分别在0、5、15和35秒时重发报文，这正是所推荐的方法。我们将在第21章更详细地讨论超时问题。

最后需要指出的是，ICMP报文是在发送UDP数据报3.5 ms后返回的，这与第7章我们所看到的Ping应答的往返时间差不多。

6.6 ICMP报文的4.4BSD处理

由于ICMP覆盖的范围很广，从致命差错到信息差错，因此即使在一个给定的系统实现中，对每个ICMP报文的处理都是不相同的。图6-12的内容与图6-3相同，它显示的是4.4BSD系统对每个可能的ICMP报文的处理方法。

类 型	代 码	描 述	处 理 方 法
0	0	回显应答	用户进程
3		目的不可达：	
	0	网络不可达	“无路由到达主机”
	1	主机不可达	“无路由到达主机”
	2	协议不可达	“连接被拒绝”
	3	端口不可达	“连接被拒绝”
	4	需要进行分片但设置了不分片比特DF	“报文太长”
	5	源站选路失败	“无路由到达主机”
	6	目的网络不认识	“无路由到达主机”
	7	目的主机不认识	“无路由到达主机”
	8	源主机被隔离（作废不用）	“无路由到达主机”
	9	目的网络被强制禁止	“无路由到达主机”
	10	目的主机被强制禁止	“无路由到达主机”
	11	由于服务类型TOS，网络不可达	“无路由到达主机”
	12	由于服务类型TOS，主机不可达	“无路由到达主机”
	13	由于过滤，通信被强制禁止	（忽略）
	14	主机越权	（忽略）
	15	优先权中止生效	（忽略）
4	0	源站被抑制(quench)	TCP由内核处理，UDP则忽略
5		重定向	
	0	对网络重定向	内核更新路由表
	1	对主机重定向	内核更新路由表
	2	对服务类型和网络重定向	内核更新路由表
	3	对服务类型和主机重定向	内核更新路由表
8	0	回显请求	
9	0	路由器通告	用户进程
10	0	路由器请求	用户进程
11		超时：	
	0	传输期间生存时间为0	用户进程
	1	在数据报组装期间生存时间为0	用户进程
12		参数问题：	
	0	坏的IP首部（包括各种差错）	“协议不可用”
	1	缺少必需的选项	“协议不可用”
13	0	时间戳请求	内核产生应答
14	0	时间戳应答	用户进程
15	0	信息请求（作废不用）	（忽略）
16	0	信息应答（作废不用）	用户进程
17	0	地址掩码请求	内核产生应答
18	0	地址掩码应答	用户进程

图6-12 4.4BSD系统对ICMP报文的处理

如果最后一列标明是“内核”，那么ICMP就由内核来处理。如果最后一列指明是“用户进程”，那么报文就被传送到所有在内核中登记的用户进程，以读取收到的ICMP报文。如果不存在任何这样的用户进程，那么报文就悄悄地被丢弃（这些用户进程还会收到所有其他类型的ICMP报文的拷贝，虽然它们应该由内核来处理，当然用户进程只有在内核处理以后才能收到这些报文）。有一些报文完全被忽略。最后，如果最后一列标明的是引号内的一串字符，那么它就是对应的Unix差错。其中一些差错，如TCP对发送端关闭的处理等，我们将在以后的章节中对它们进行讨论。

6.7 小结

本章对每个系统都必须包括的Internet控制报文协议进行了讨论。图6-3列出了所有的ICMP报文类型，其中大多数都将在以后的章节中加以讨论。

我们详细讨论了ICMP地址掩码请求和应答以及时间戳请求和应答。这些是典型的请求—应答报文。二者在ICMP报文中都有标识符和序列号。发送端应用程序在标识字段内存入一个唯一的数值，以区别于其他进程的应答。序列号字段使得客户程序可以在应答和请求之间进行匹配。

我们还讨论了ICMP端口不可达差错，一种常见的ICMP差错。对返回的ICMP差错信息进行了分析：导致差错的IP数据报的首部及后续8个字节。这个信息对于ICMP差错的接收方来说是必要的，可以更多地了解导致差错的原因。这是因为TCP和UDP都在它们的首部前8个字节中存入源端口号和目的端口号。

最后，我们第一次给出了按时间先后的`tcpdump`输出，这种表示方式在本书后面的章节中会经常用到。

习题

6.1 在6.2节的末尾，我们列出了5种不发送ICMP差错报文的特殊条件。如果这些条件不满足而我们又在局域网上向一个似乎不存在的端口号发送一份广播UDP数据报，这时会发生什么样的情况？

6.2 阅读RFC [Braden 1989a]，注意生成一个ICMP端口不可达差错是否为“必须”，“应该”或者“可能”。这些信息所在的页码和章节是多少？

6.3 阅读RFC 1349 [Almquist 1992]，看看IP的服务类型字段（见图3-2）是如何被ICMP设置的？

6.4 如果你的系统提供`netstat`命令，请用它来查看接收和发送的ICMP报文类型。