[UDP 2](#_Toc471118214)

[数据结构 3](#_Toc471118215)

[UDP报文 3](#_Toc471118216)

[UDP端口号 4](#_Toc471118217)

[UDP检验和 5](#_Toc471118218)

[UDP伪首部 6](#_Toc471118219)

[发送端主机计算UDP检验和 6](#_Toc471118220)

[本主机的IP协议验证UDP检验和 6](#_Toc471118221)

[接收端主机验证UDP检验和 6](#_Toc471118222)

[使用TCPDUMP程序输出接收到的UDP检验和 7](#_Toc471118223)

[数据传输 8](#_Toc471118224)

[11.3.2一些统计结果 9](#_Toc471118225)

[一个简单的例子 10](#_Toc471118226)

[广播与多播仅用于UDP 11](#_Toc471118227)

[ICMP不可达差错（需要分片） 12](#_Toc471118228)

[用Traceroute确定路径MTU 14](#_Toc471118229)

[采用UDP的路径MTU发现 15](#_Toc471118230)

[UDP和ARP之间的交互作用 17](#_Toc471118231)

[最大UDP数据报长度 18](#_Toc471118232)

[ICMP源站抑制差错 19](#_Toc471118233)

[UDP服务器的设计 20](#_Toc471118234)

[11.12.1客户IP地址及端口号 21](#_Toc471118235)

[11.12.2目的IP地址 21](#_Toc471118236)

[11.12.3UDP输入队列 21](#_Toc471118237)

[11.12.4限制本地IP地址 23](#_Toc471118238)

[11.12.5限制远端IP地址 24](#_Toc471118239)

[11.12.6每个端口有多个接收者 25](#_Toc471118240)

[小结 26](#_Toc471118241)

UDP

而且UDP的首部长为8字节

运输层协议在生成报文首部时要存入一个应用程序的标识符。TCP和UDP都用一个16BIt的端口号来表示不同的应用程序。

前面已经指出过， TCP和UDP采用16bit的端口号来识别应用程序。那么这些端口号是如何选择的呢？

服务器一般都是通过知名端口号来识别的。

例如，对于每个TCP/ IP实现来说，

FTP服务器的TCP端口号都是21，

每个TELNEt服务器的TCP端口号都是23，

每个TFTP(简单文件传送协议)服务器的UDP端口号都是69。

任何TCP/ IP实现所提供的服务都用知名的1～1023之间的端口号。

这些知名端口号由Internet号分配机构（ InternetAssigneDNumberSAuthority, IANA）来管理。

到1992年为止，知名端口号介于1～255之间。256～1023之间的端口号通常都是由UNIx系统占用，以提供一些特定的UNIx服务—也就是说，提供一些只有UNIx系统才有的、而其他操作系统可能不提供的服务。现在IANA管理1～1023之间所有的端口号。

Internet扩展服务与UNIx特定服务之间的一个差别就是TELNEt和RLOGIn。它们二者都允许通过计算机网络登录到其他主机上。

TELNEt是采用端口号为23的TCP/ IP标准且几乎可以在所有操作系统上进行实现。相反，RLOGIn最开始时只是为UNIx系统设计的（尽管许多非Unix系统现在也提供该服务），因此在80年代初，它的有名端口号为513。

客户端通常对它所使用的端口号并不关心，只需保证该端口号在本机上是唯一的就可以了。

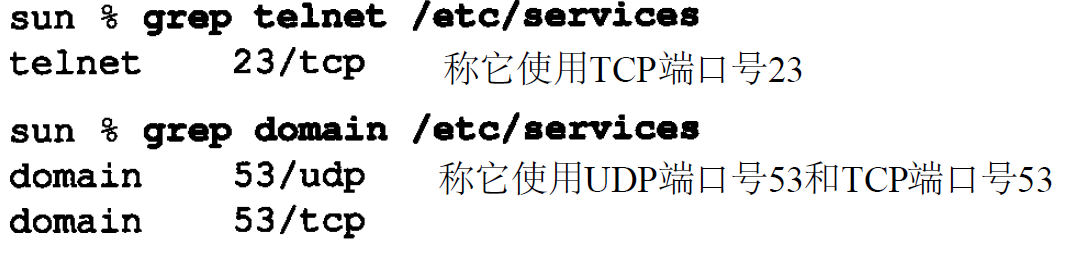
客户端口号又称作临时端口号（即存在时间很短暂）。这是因为它通常只是在用户运行该客户程序时才存在，而服务器则只要主机开着的，其服务就运行。

大多数TCP/ IP实现给临时端口分配1024～5000之间的端口号。大于5000的端口号是为其他服务器预留的（ Internet上并不常用的服务)。我们可以在后面看见许多这样的给临时端口分配端口号的例子。

SolariS2.2是一个很有名的例外。通常TCP和UDP的缺省临时端口号从32768开始。

在E.4节中，我们将详细描述系统管理员如何对配置选项进行修改以改变这些缺省项。

大多数UNIx系统的文件/ETC/ SERVICEs都包含了人们熟知的端口号。为了找到TELNEt服务器和域名系统的端口号，可以运行以下语句：



保留端口号

UNIx系统有保留端口号的概念。只有具有超级用户特权的进程才允许给它自己分配一个保留端口号。

这些端口号介于1～1023之间，一些应用程序（如有名的RLOGIn，26. 2节）将它作为客户与服务器之间身份认证的一部分。

数据结构

# UDP报文

|  |  |  |
| --- | --- | --- |
| 传输次序 | 发送UDP报文时先发送高位，再发送低位； | |
| 报文结构 | 高位部分  UDP首部 | UDP首部共有5个字段，长度位为12个字节 |
| 低位部分  UDP数据 | UDP数据是可选项，其字长也不确定，但要求是整数倍个字节； |

具体的报文结构见下表：从上到下的顺序是数据位的由高到低，也是传输的先后的顺序：

|  |  |  |
| --- | --- | --- |
| Fields | 位宽(bit) | 说明 |
| UDP长度 | 16 | 同下面的“UDP长度字段” |
| 源端口号 | 16 | 端口号表示发送进程和接收进程,在图1- 8中，我们画出了TCP和UDP用目的端口号来分用来自IP层的数据的过程。  由于IP层已经把IP数据报分配给TCP或UDP（根据IP首部中协议字段值），因此TCP端口号由TCP来查看，而UDP端口号由UDP来查看。  TCP端口号与UDP端口号是相互独立的。  尽管相互独立，如果TCP和UDP同时提供某种知名服务，两个协议通常选择相同的端口号。  这纯粹是为了使用方便，而不是协议本身的要求 |
| 目标端口号 | 16 |  |
| UDP长度 | 16 | 该UDP报文的字节长度，单位：字节  最小值 = 8（即发送一份0字节数据的UDP报文）；  这个UDP长度是有冗余的。IP数据报长度指的是数据报全长（图3- 1），因此UDP数据报长度是全长减去IP首部的长度（该值在首部长度字段中指定，如图3- 1所示）。 |
| UDP检验和 | 16 | UDP检验和覆盖 **UDP首部 、UDP数据**。  UDP报文的检验和是可选的，但一般都会选择使用；  设置UDP检验和的目的有2个  1.确认在封装过程中UDP报文是否正确地发送给本机IP协议  2.确认数据在发送端主机到接收端主机之间传输时，UDP报文是否发生改动 |
| 数据(可选项) | n\*8 |  |

# UDP检验和

UDP检验和覆盖 **UDP首部 、UDP数据**。

UDP报文的检验和是可选的，但一般都会选择使用；

设置UDP检验和的目的有2个：

是为了发现UDP报文在发送端主机到接收端主机之间发生的任何改动

## UDP伪首部

发送端主机计算UDP检验和时，会加入一个UDP伪首部到UDP报文前；

计算UDP检验和时，计算对象是：UDP伪首部、UDP首部、UDP数据(数据位由高到低)

注意这个UDP伪首部仅仅在计算UDP检验和时在会使用到，仅仅是作为计算UDP校验和的方法；

**伪首部是不占地址空间的，在实际传输中不存在这样的字段。**

|  |  |  |
| --- | --- | --- |
| Fields | 位宽(bit) | 说明 |
| 源IP地址 | 32 |  |
| 目标IP地址 | 32 |  |
| 填充位 | 8 | Value=0 |
| 协议 | 8 | Value=17 |

## 发送端主机计算UDP检验和

|  |  |
| --- | --- |
| 检验和字段初始化 | 把检验和字段的16bit设置为0 |
| 加入伪首部 | 加入UDP伪首部 |
| 填充 | 由于UDP报文可以为奇数个字节(不是16bit的整数倍)，所以先要进行填充，在UDP报文尾部填充0，知道整个UDP报文字长为16bit的整数倍； |
| 分割检验对象 | 将UDP报文分割成若干片段：从高位开始分割，每个片段16bit |
| 计算 | 将所有片段相加，得到的结果取反，然后放到检验和字段中； |

## 本主机的IP协议验证UDP检验和

## 接收端主机验证UDP检验和

|  |  |
| --- | --- |
| 获取UDP报文 | 接收端主机对收到的报文进行分用处理，得到UDP报文 |
| 分割 | 将UDP首部分割成若干片段：从高位开始分割，每个片段16bit； |
| 计算 | 将所有片段相加，得到的结果取反；接收方计算校验和时的首部与发送方计算校验和时的首部相比，多了一个发送方计算出来的校验和。因此，如果首部在传输过程中没有发生差错，那么接收方计算的结果应该为全1，即65535，因为接收方计算除校验和以外的部分得到值是校验和的反码。 |
| 错误处理 | 如果结果不是全1，那么就是检验和错误，该报文会被悄悄丢弃，不生成差错报文，由上层去发现丢失的数据报文并进行重传 |

## 使用TCPDUMP程序输出接收到的UDP检验和

很难知道某个特定系统是否打开了UDP检验和选项。应用程序通常不可能得到接收到的UDP首部中的检验和。为了得到这一点，作者在TCPDUMp程序中增加了一个选项，以打印出接收到的UDP检验和。如果打印出的值为0，说明发送端没有计算检验和。

测试网络上三个不同系统的输出如图11- 4所示（参见封面二）。运行我们自编的SOCk程序

（附录C），发送一份包含9个字节数据的UDP数据报给标准回显服务器。

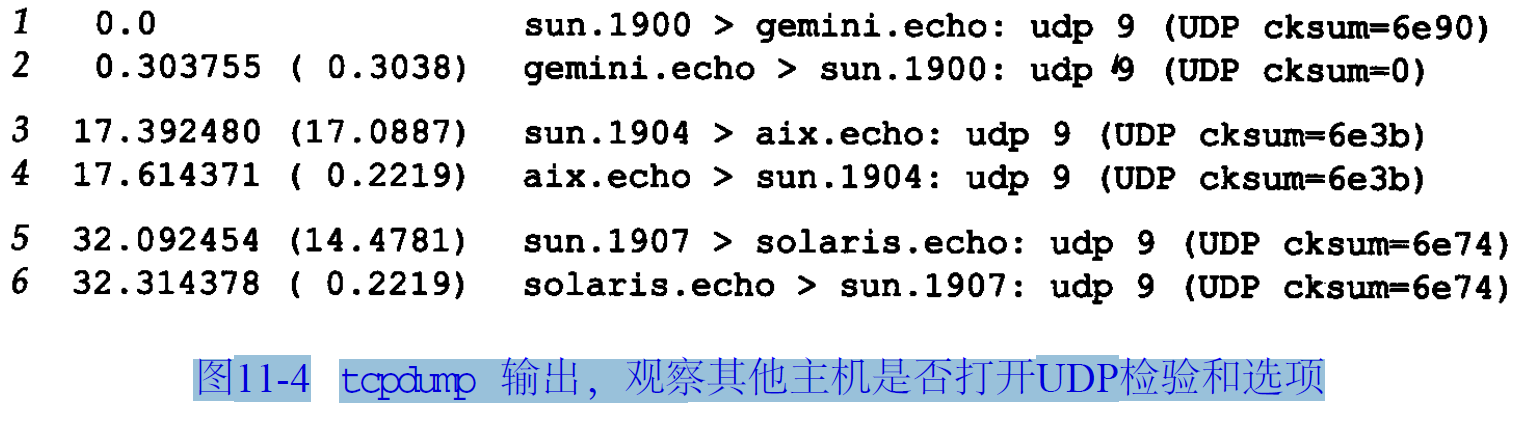


图11-4TCPDUMP输出，观察其他主机是否打开UDP检验和选项

从这里可以看出，三个系统中有两个打开了UDP检验和选项。

还要注意的是，在这个简单例子中，送出的数据报与收到的数据报具有相同的检验和值

（第3和第4行，第5和第6行）。从图11-3可以看出，两个IP地址进行了交换，正如两个端口号一样。

伪首部和UDP首部中的其他字段都是相同的，就像数据回显一样。这再次表明UDP检验和（事

实上，TCP/IP协议簇中所有的检验和）是简单的16bit和。它们检测不出交换两个16bit的差错。

作者在14. 2节中在8个域名服务器中各进行了一次DNS查询。DNS主要使用UDP，

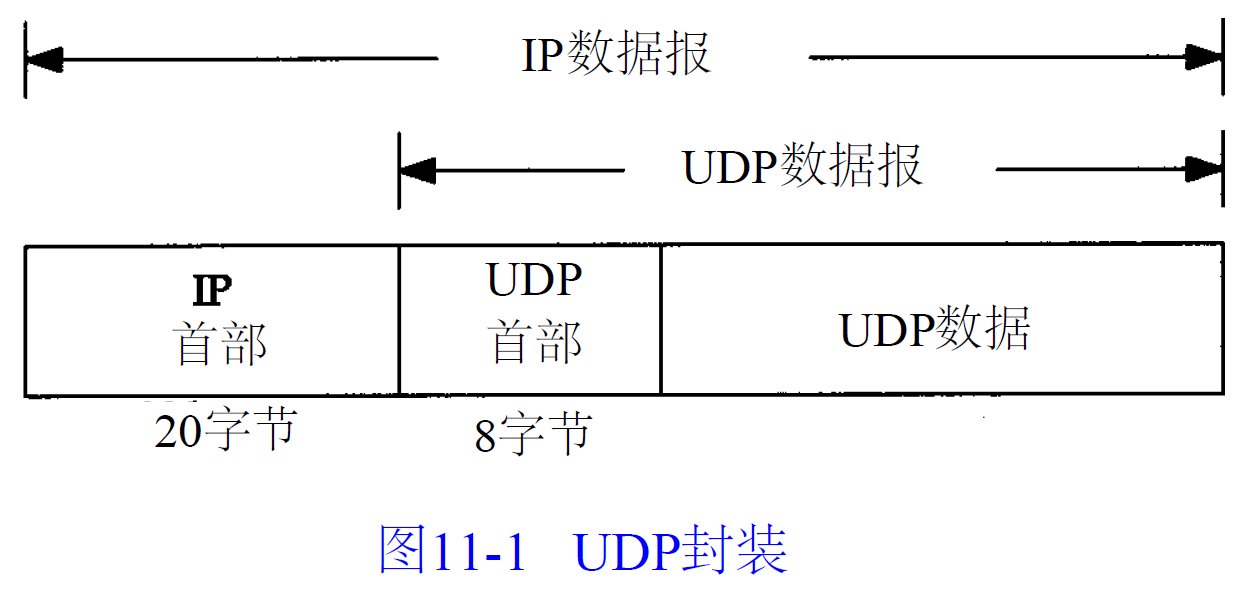
结果只有两台服务器打开了UDP检验和选项。

数据传输

UDP是一个简单的面向数据报的运输层协议：进程的每个输出操作都正好产生一个UDP数据报，并组装成一份待发送的IP数据报。

这与面向流字符的协议不同，如TCP，应用程序产生的全体数据与真正发送的单个IP数据报可能没有什么联系。

UDP数据报封装成一份IP数据报的格式如图11- 1所示。



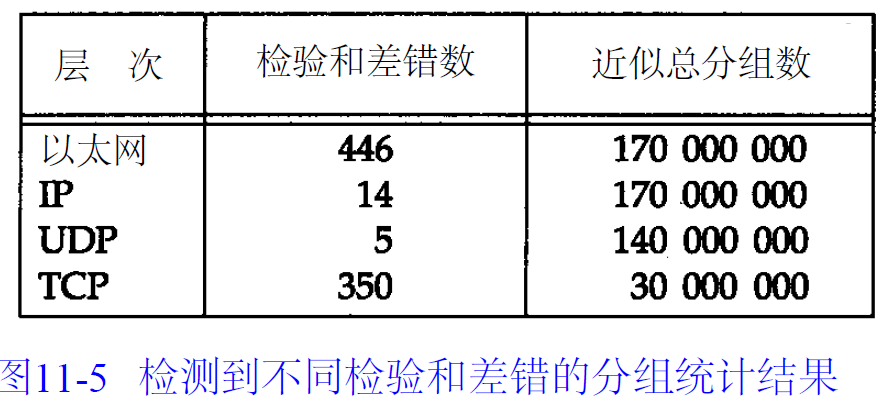
RFC768[PosteL1980] 是UDP的正式规范。UDP不提供可靠性：它把应用程序传给IP层的数据发送出去，但是并不保证它们能到达目的地。由于缺乏可靠性，我们似乎觉得要避免使用UDP而使用一种可靠协议如TCP。我们在第17章讨论完TCP后将再回到这个话题，看看什么样的应用程序可以使用UDP。

应用程序必须关心IP数据报的长度。如果它超过网络的MTU（2. 8节），那么就要对IP数据报进行分片。如果需要，源端到目的端之间的每个网络都要进行分片，并不只是发送端主机连接第一个网络才这样做（我们在2. 9节中已定义了路径MTU的概念）。在11. 5节中，我们将讨论IP分片机制。

## 11.3.2一些统计结果

文献[MoguL1992]提供了在一个繁忙的NFS服务器上所发生的不同检验和差错的统计结果，时间持续了40天。统计数字结果如图11- 5所

示。



最后一列是每一行的大概总数，因为以太网和IP层还使用其他的协议。

例如，不是所有的以太网数据帧都是IP数据报，至少以太网还要使用ARP协议。

不是所有的IP数据报都是UDP或TCP数据，因为ICMP也用IP传送数据。

注意， TCP发生检验和差错的比例与UDP相比要高得多。这很可能是因为在该系统中的TCP连接经常是“远程”连接（经过许多路由器和网桥等中间设备），而UDP一般为本地通信。

从最后一行可以看出，不要完全相信数据链路（如以太网，令牌环等）的CRC检验。应该始终打开端到端的检验和功能。而且，如果你的数据很有价值，也不要完全相信UDP或TCP的检验和，因为这些都只是简单的检验和，不能检测出所有可能发生的差错。

# 一个简单的例子

用我们自己编写的SOCk程序生成一些可以通过TCPDUMp观察的UDP数据报：

bsdI% socK-V-U-I-n4svr4discard

connecteDoN140.252.13.35.1108tO140.252.13.34.9

bsdI% socK-V-U-I-n4-w0svr4discard

connecteDoN140.252.13.35.1110tO140.252.13.34.9

第1次执行这个程序时，我们指定VERBOSe模式（- v）来观察EPHEMERAl端口号，指定UDP（- u）而不是默认的TCP，并且指定源模式（ - i）来发送数据，而不是读写标准的输入和输出。- N4选项指明输出4份数据报（默认条件下为1024），目的主机为SVR4。在1. 12节描述了丢

弃服务。每次写操作的输出长度取默认值1024。

第2次运行该程序时我们指定- W0，意思是写长度为0的数据报。两个命令的TCPDUMp输

出结果如图11- 6所示。

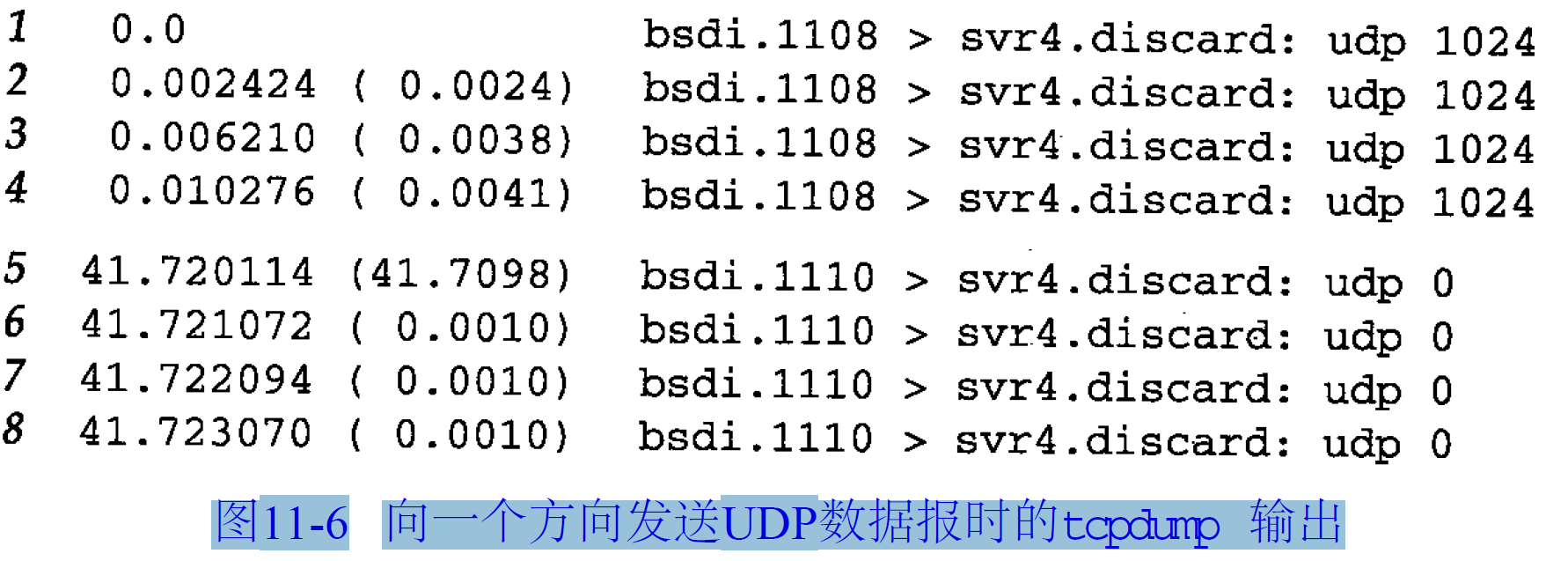


图11-6向一个方向发送UDP数据报时的TCPDUMP输出

输出显示有四份1024字节的数据报，接着有四份长度为0的数据报。每份数据报间隔几毫秒（输入第2个命令花了41秒的时间）。

在发送第1份数据报之前，发送端和接收端之间没有任何通信（在第17章，我们将看到TCP在发送数据的第1个字节之前必须与另一端建立连接）。

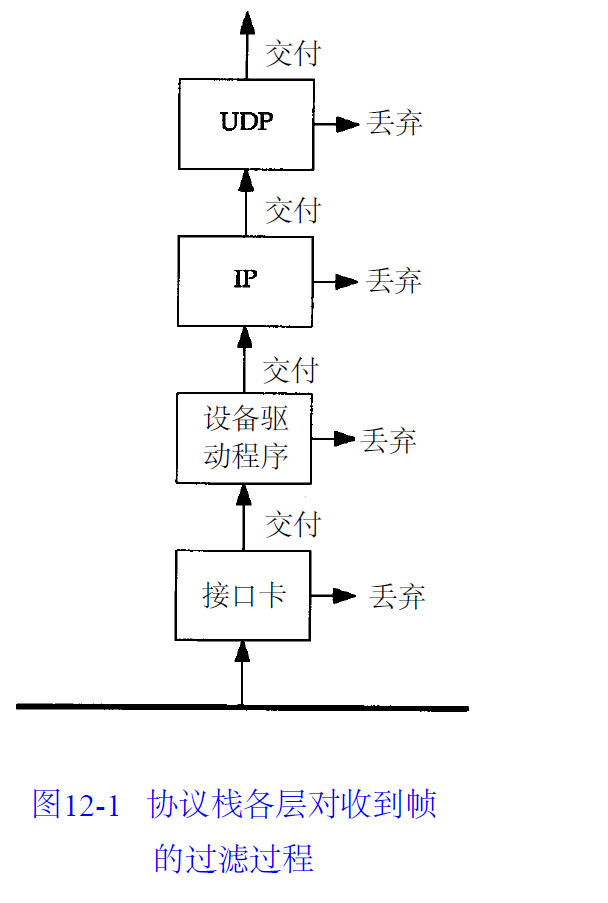
另外，当收到数据时，接收端没有任何确认。在这个例子中，发送端并不知道另一端是否已经收到这些数据报。

最后要指出的是，每次运行程序时，源端的UDP端口号都发生变化。第一次是1108，然后是110。在1. 9节我们已经提过，客户程序使用EPHEMERAl端口号一般在1024～5000之间，正如我们现在看到的这样。

# 广播与多播仅用于UDP

TCP是一个面向连接的协议，它意味着分别运行于两主机（由IP地址确定）内的两进程（由端口号确定）间存在一条连接。

考虑包含多个主机的共享信道网络如以太网。每个以太网帧包含源主机和目的主机的以太网地址（48BIt）。



首先，网卡查看由信道传送过来的帧，确定是否接收该帧，若接收后就将它传往设备驱动程序。

通常网卡仅接收那些目的地址为网卡物理地址或广播地址的帧。另外，多数接口均被设置为混合模式，这种模式能接收每个帧的一个复制。

作为一个例子， TCPDUMP使用这种模式。

目前，大多数的网卡经过配置都能接收目的地址为多播地址或某些子网多播地址的帧。

对于以太网，当地址中最高字节的最低位设置为1时表示该地址是一个多播地址，用十六进制可表示为01: 00: 00: 00: 00: 00（以太网广播地址fF: fF: fF: fF: fF: ff可看作是以太网多播地址的特例）。

如果网卡收到一个帧，这个帧将被传送给设备驱动程序（如果帧检验和错，网卡将丢弃该帧）。

设备驱动程序将进行另外的帧过滤。

首先，帧类型中必须指定要使用的协议（ IP、ARP等等）。

其次，进行多播过滤来检测该主机是否属于多播地址说明的多播组。

设备驱动程序随后将数据帧传送给下一层，比如，当帧类型指定为IP数据报时，就传往IP层。

IP根据IP地址中的源地址和目的地址进行更多的过滤检测。如果正常，就将数据报传送给下一层（如TCP或UDP）。

每次UDP收到由IP传送来的数据报，就根据目的端口号，有时还有源端口号进行数据报过滤。

如果当前没有进程使用该目的端口号，就丢弃该数据报并产生一个ICMP不可达报文（TCP根据它的端口号作相似的过滤）。

如果UDP数据报存在检验和错，将被丢弃。

使用广播的问题在于它增加了对广播数据不感兴趣主机的处理负荷。

拿一个使用UDP广播应用作为例子。如果网内有50个主机，但仅有20个参与该应用，每次这20个主机中的一个发送UDP广播数据时，其余30个主机不得不处理这些广播数据报。

一直到UDP层，收到的UDP广播数据报才会被丢弃。这30个主机丢弃UDP广播数据报是因为这些主机没有使用这个目的端口。

多播的出现减少了对应用不感兴趣主机的处理负荷。使用多播，主机可加入一个或多个多播组。这样，网卡将获悉该主机属于哪个多播组，然后仅接收主机所在多播组的那些多播帧。

广播是将数据报发送到网络中的所有主机（通常是本地相连的网络），而多播是将数据报发送到网络的一个主机组。

这两个概念的基本点在于当收到送往上一个协议栈的数据帧时采用不同类型的过滤。

每个协议层均可以因为不同的理由丢弃数据报。

# ICMP不可达差错（需要分片）

发生ICMP不可达差错的另一种情况是，当路由器收到一份需要分片的数据报，而在IP首部又设置了不分片（ DF）的标志比特。如果某个程序需要判断到达目的端的路途中最小MTU是多少—称作路径MTU发现机制（ 2. 9节），那么这个差错就可以被该程序使用。

这种情况下的ICMP不可达差错报文格式如图11- 9所示。这里的格式与图6- 10不同，因为在第2个32bit字中，16~31bit可以提供下一站的MTU，而不再是0。

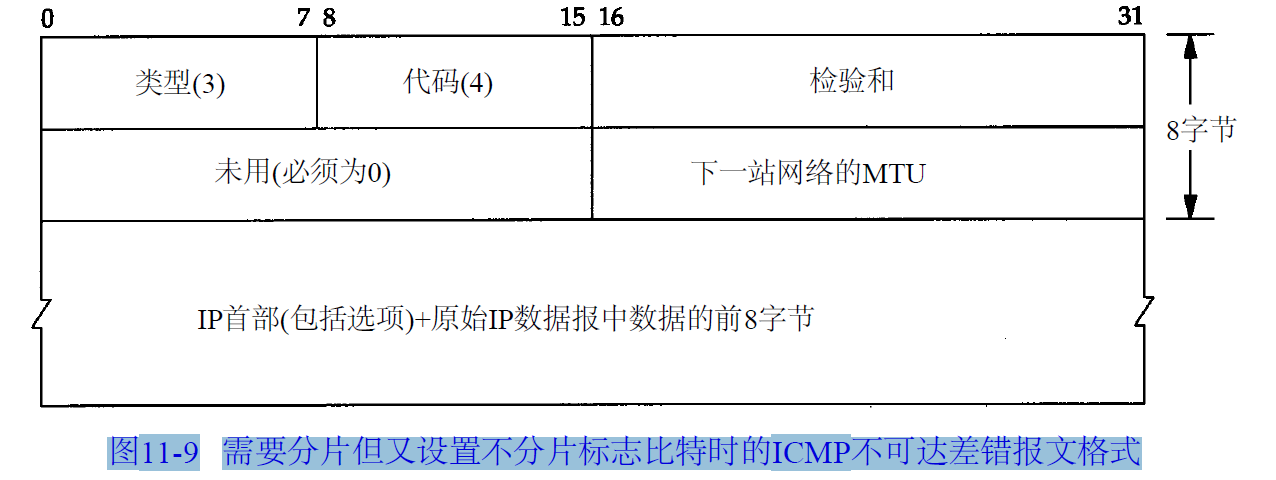


图11-9需要分片但又设置不分片标志比特时的ICMP不可达差错报文格式如果路由器没有提供这种新的ICMP差错报文格式，那么下一站的MTU就设为0。

新版的路由器需求RFC[AlmquisT1993]声明，在发生这种ICMP不可达差错时，路由器必须生成这种新格式的报文。

例子

关于分片作者曾经遇到过一个问题， ICMP差错试图判断从路由器NETb到主机SUn之间的拨号SLIP链路的MTU。我们知道从SUn到NETb的链路的MTU：当SLIP被安装到主机SUn时，这是SLIP配置过程中的一部分，加上在3. 9节中已经通过NETSTAt命令观察过。现在，我们想从另一个方向来判断它的MTU（在第25章，将讨论如何用SNMP来判断）。

在点到点的链路中，不要求两个方向的MTU为相同值。

所采用的技术是在主机SOLARIs上运行PINg程序到主机BSDi，增加数据分组长度，直

到看见进入的分组被分片为止。如图11- 10所示。

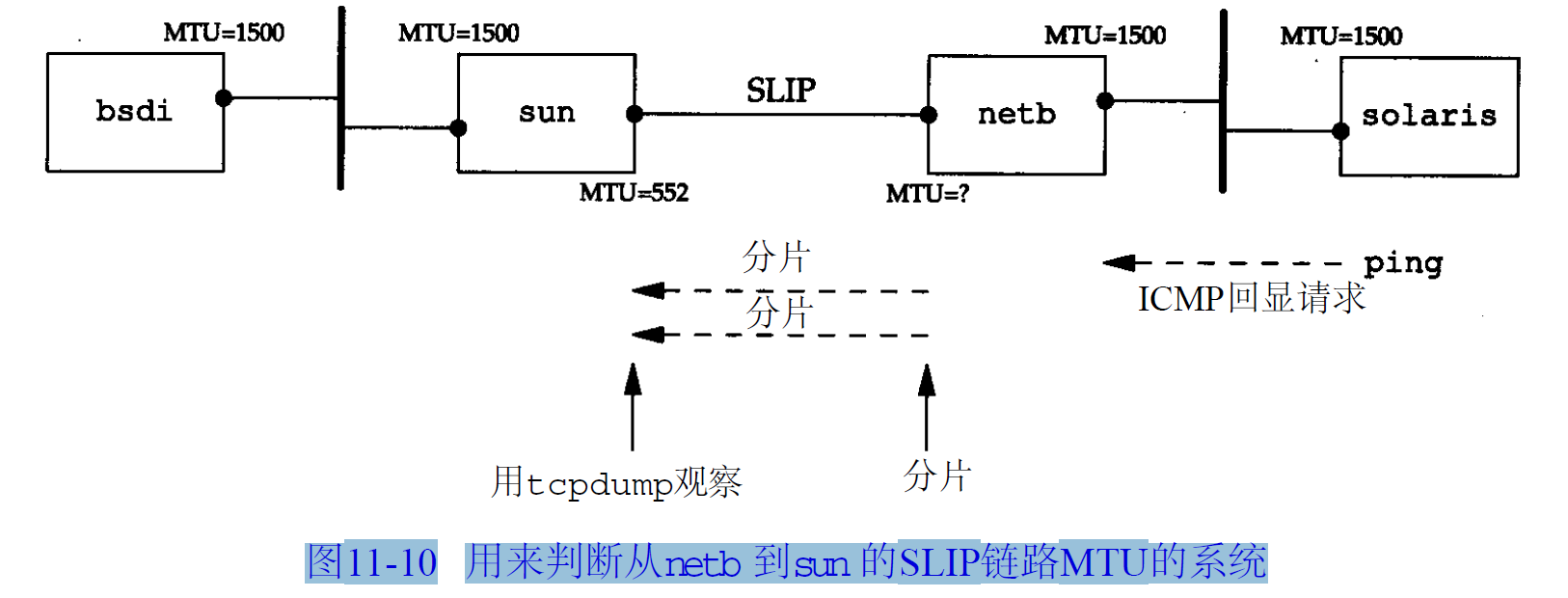


图11-10用来判断从NETB到SUN的SLIP链路MTU的系统在主机SUn上运行TCPDUMp，观察SLIP链路，看什么时候发生分片。开始没有观察到分片，一切都很正常直到PINg分组的数据长度从500增加到600字节。可以看到接收到的回显请求（仍然没有分片），但不见回显应答。为了跟踪下去，也在主机BSDi上运行TCPDUMp，观察它接收和发送的报文。输出如图

11- 11所示。

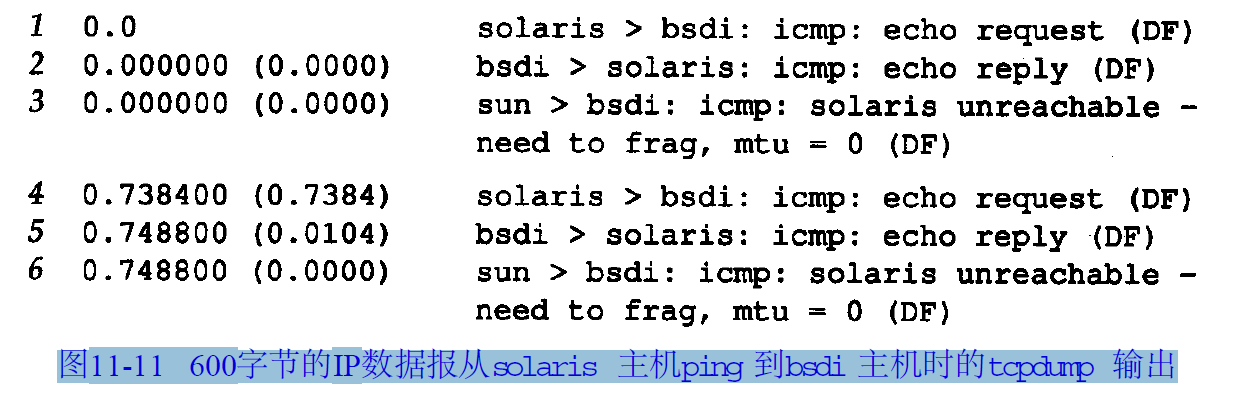


图11-11600字节的IP数据报从SOLARIS主机PING到BSDI主机时的TCPDUMP输出

首先，每行中的标记（DF）说明在IP首部中设置了不分片比特。

这意味着SolariS2.2一般把不分片比特置1，作为实现路径MTU发现机制的一部分。

第1行显示的是回显请求通过路由器NETb到达SUn主机，没有进行分片，并设置了DF比特，因此我们知道还没有达到NETb的SLIPMTU。

接下来，在第2行注意到DF标志被复制到回显应答报文中。这就带来了问题。回显应答与回显请求报文长度相同（超过600字节），但是SUn外出的SLIP接口MTU为552。因此回显应答需要进行分片，但是DF标志比特又被设置了。这样， SUn就产生一个ICMP不可达差错报文返回给BSDi（报文在BSDi处被丢弃）。

这就是我们在主机SOLARIs上没有看到任何回显应答的原因。这些应答永远不能通过SUn。分组的路径如图11- 12所示。

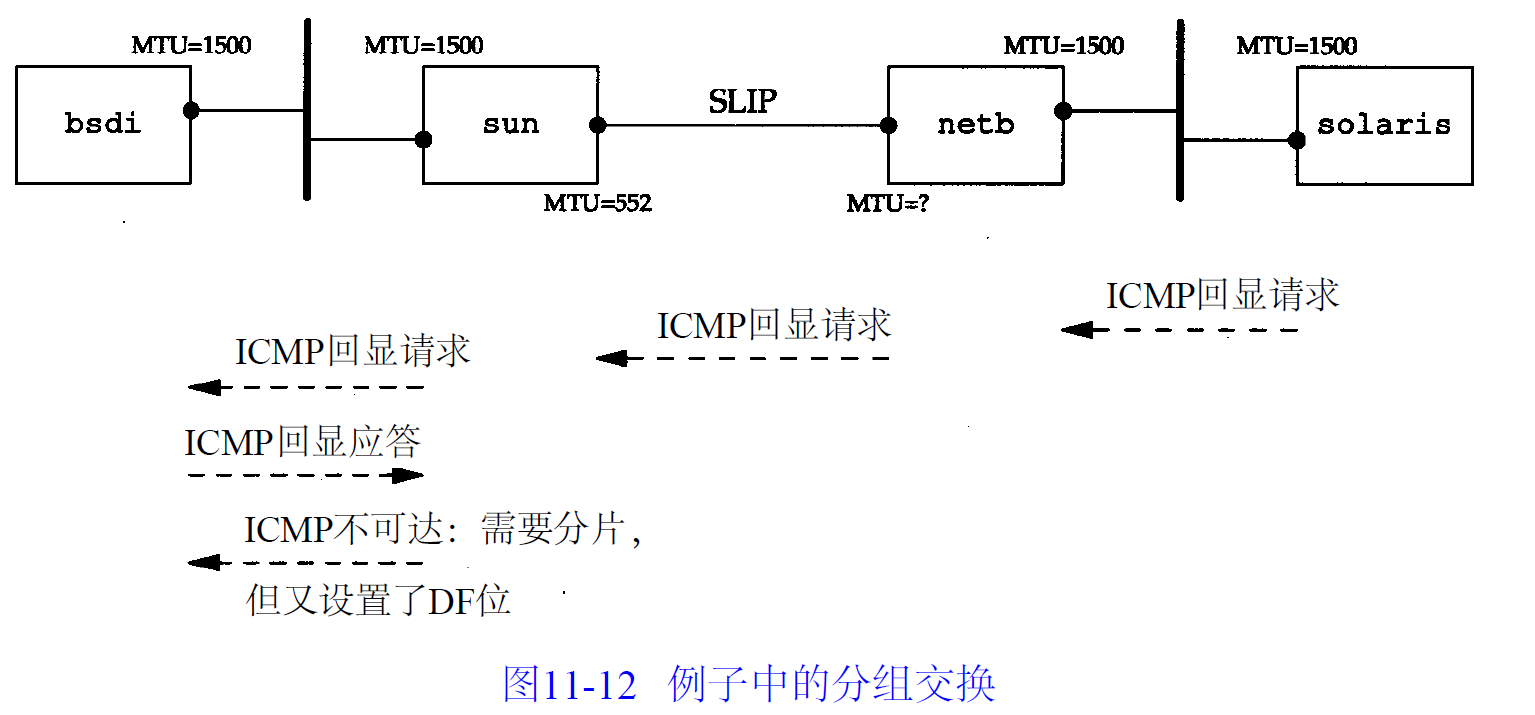


图11-12例子中的分组交换

最后，在图11- 11中的第3行和第6行中，MTU= 0表示主机SUn没有在ICMP不可达报文中返回出口MTU值，如图11- 9所示（在25. 9节中，将重新回到这个问题，用SNMP判断NETb上的SLIP接口MTU值为1500）。

# 用Traceroute确定路径MTU

尽管大多数的系统不支持路径MTU发现功能，但可以很容易地修改TRACEROUTe程序

（第8章），用它来确定路径MTU。要做的是发送分组，并设置“不分片”标志比特。发送的

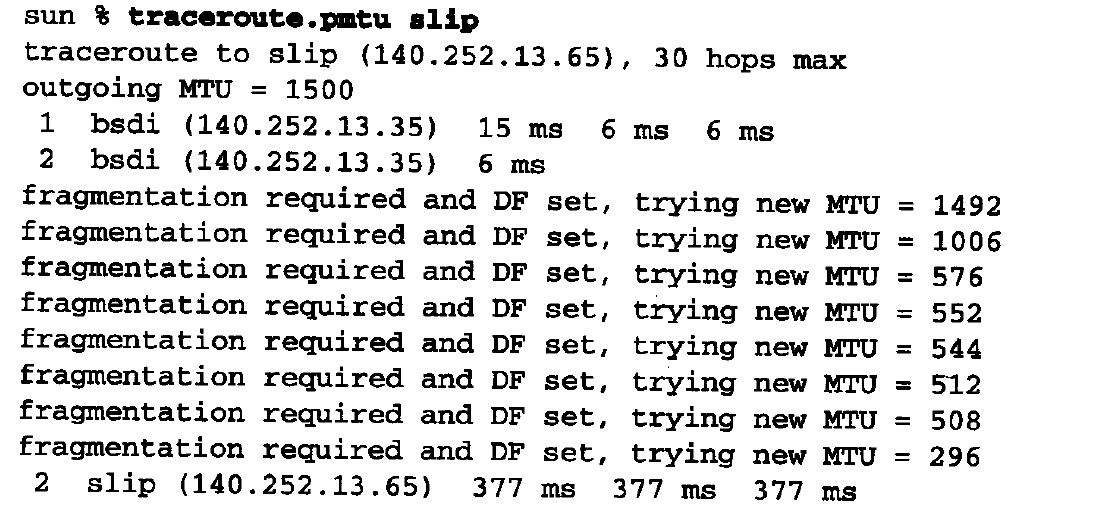
第一个分组的长度正好与出口MTU相等，每次收到ICMP“不能分片”差错时（在上一节讨论的）就减小分组的长度。如果路由器发送的ICMP差错报文是新格式，包含出口的MTU，那么

就用该MTU值来发送，否则就用下一个最小的MTU值来发送。正如RFC1191[MoguLand

DeerinG1990]声明的那样， MTU值的个数是有限的，因此在我们的程序中有一些由近似值构

成的表，取下一个最小MTU值来发送。

首先，我们尝试判断从主机SUn到主机SLIp的路径MTU，知道SLIP链路的MTU为296。



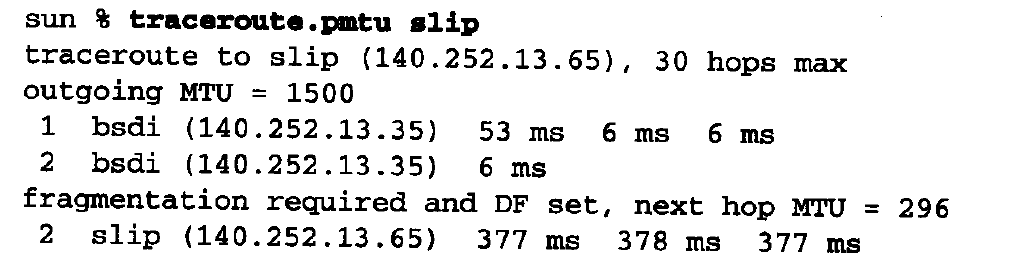
在这个例子中，路由器BSDi没有在ICMP差错报文中返回出口MTU，因此我们选择另一个

MTU近似值。TTL为2的第1行输出打印的主机名为BSDi，但这是因为它是返回ICMP差错报

文的路由器。TTL为2的最后一行正是我们所要找的。

在BSDi上修改ICMP代码使它返回出口MTU值并不困难，如果那样做并再次运行该程序，

得到如下输出结果：



这时，在找到正确的MTU值之前，我们不用逐个尝试8个不同的MTU值——路由器返回

了正确的MTU值。

全球互联网

作为一个实验，我们多次运行修改以后的TRACEROUTe程序，目的端为世界各地的主机。

可以到达15个国家（包括南极洲），使用了多个跨大西洋和跨太平洋的链路。但是，在这样做

之前，作者所在子网与路由器NETb之间的拨号SLIP链路MTU（见图11- 12）增加到1500，与

以太网相同。

在18次运行当中，只有其中2次发现的路径MTU小于1500。其中一个跨大西洋的链路

MTU值为572（其近似值甚至在RFC1191中也没有被列出），而路由器返回的是新格式的

ICMP差错报文。另外一条链路，在日本的两个路由器之间，不能处理1500字节的数据帧，并

且路由器没有返回新格式的ICMP差错报文。把MTU值设成1006则可以正常工作。

从这个实验可以得出结论，现在许多但不是所有的广域网都可以处理大于512字节的分组。

利用路径MTU发现机制，应用程序就可以充分利用更大的MTU来发送报文。

# 采用UDP的路径MTU发现

下面对使用UDP的应用程序与路径MTU发现机制之间的交互作用进行研究。看一看如果

应用程序写了一个对于一些中间链路来说太长的数据报时会发生什么情况。

例子

由于我们所使用的支持路径MTU发现机制的唯一系统就是SolariS2.x，因此，将采用它作

为源站发送一份650字节数据报经SLIp。由于SLIp主机位于MTU为296的SLIP链路后，因此，

任何长于268字节（296－20－8）且“不分片”比特置为1的UDP数据都会使BSDi路由器产生

ICMP“不能分片”差错报文。图11- 13给出了拓扑结构和MTU。

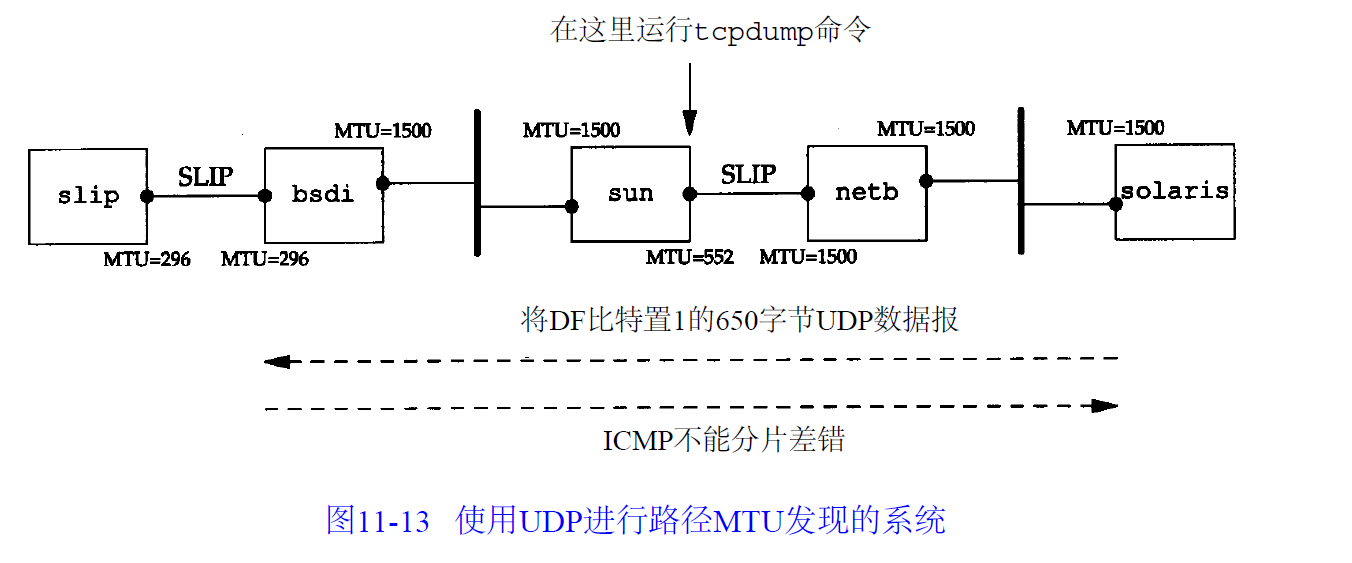


图11-13使用UDP进行路径MTU发现的系统

可以用下面的命令行来产生650字节UDP数据报，每两个UDP数据报之间的间隔是5秒：

solariS% socK-U-I-n10-w650-p5sliPdiscard

图11- 14是TCPDUMp的输出结果。在运行这个例子时，将BSDi设置成在ICMP“不能分片”

差错中，不返回下一跳MTU信息。

在发送的第一个数据报中将DF比特置1（第1行），其结果是从BSDi路由器发回我们可以

猜测的结果（第2行）。令人不解的是，发送一个DF比特置1的数据报（第3行），其结果是同

样的ICMP差错（第4行）。我们预计这个数据报在发送时应该将DF比特置0。

第5行结果显示， IP已经知道了发往该目的地址的数据报不能将DF比特置1，因此， IP进

而将数据报在源站主机上进行分片。这与前面的例子中， IP发送经过UDP的数据报，允许具

有较小MTU的路由器（在本例中是BSDi）对它进行分片的情况不一样。由于ICMP“不能分

片”报文并没有指出下一跳的MTU，因此，看来IP猜测MTU为576就行了。第一次分片（第5

行）包含544字节的UDP数据、8字节UDP首部以及20字节IP首部，因此，总IP数据报长度是

572字节。第2次分片（第6行）包含剩余的106字节UDP数据和20字节IP首部。

不幸的是，第7行的下一个数据报将其DF比特置1，因此BSDi将它丢弃并返回ICMP差错。

这时发生了IP定时器超时，通知IP查看是不是因为路径MTU增大了而将DF比特再一次置1。

我们可以从第19行和20行看出这个结果。将第7行与19行进行比较，可以看出IP每过30秒就将

DF比特置1，以查看路径MTU是否增大了。

这个30秒的定时器值看来太短。RFC1191建议其值取10分钟。可以通过修改

IPIREPATHMTUINTERVAl（E. 4节）参数来改变该值。同时，SolariS2.2无法对单个UDP应用或所有UDP应用关闭该路径MTU发现。只能通过修改IPPATHMTUDISCOVERy

参数，在系统一级开放或关闭它。正如在这个例子里所能看到的那样，如果允许路径MTU

发现，那么当UDP应用程序写入可能被分片数据报时，该数据报将被丢弃。

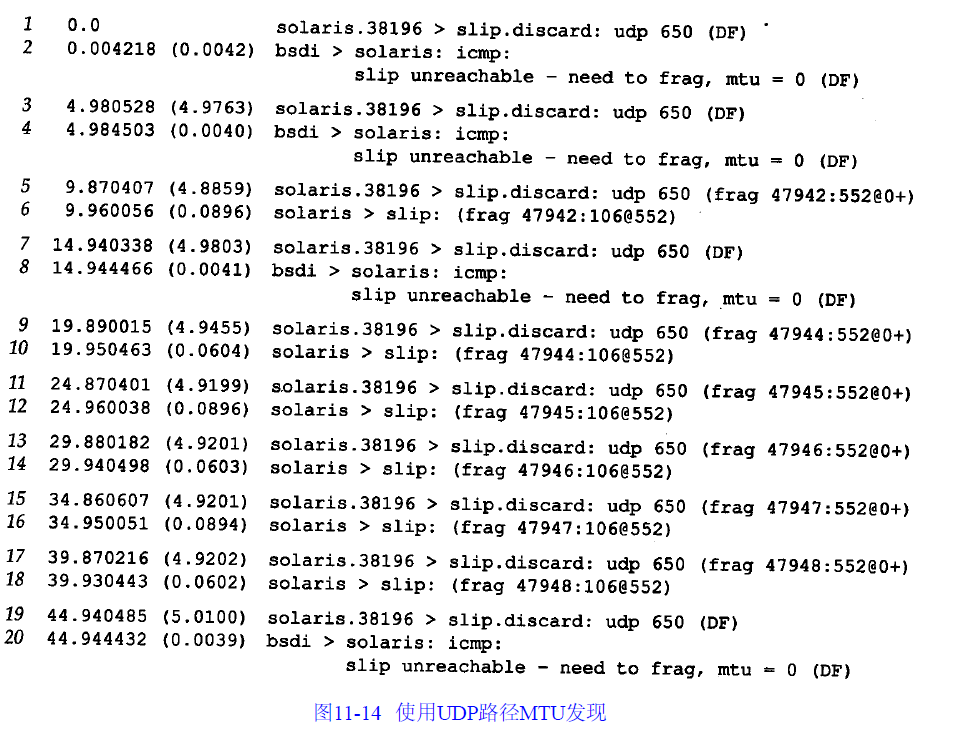


图11-14使用UDP路径MTU发现

SOLARIs的IP层所假设的最大数据报长度（ 576字节）是不正确的。在图11- 13中，我们

看到，实际的MTU值是296字节。这意味着经SOLARIs分片的数据报还将被BSDi分片。图

11- 15给出了在目的主机（ SLIp）上所收集到的TCPDUMp对于第一个到达数据报的输出结果

（图11- 14的第5行和第6行）。

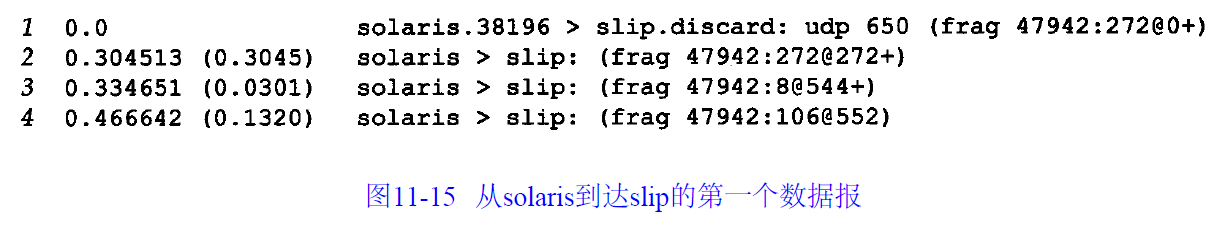


图11-15从solaris到达slip的第一个数据报

在本例中，SOLARIs不应该对外出数据报分片，它应该将DF比特置0，让具有最小MTU

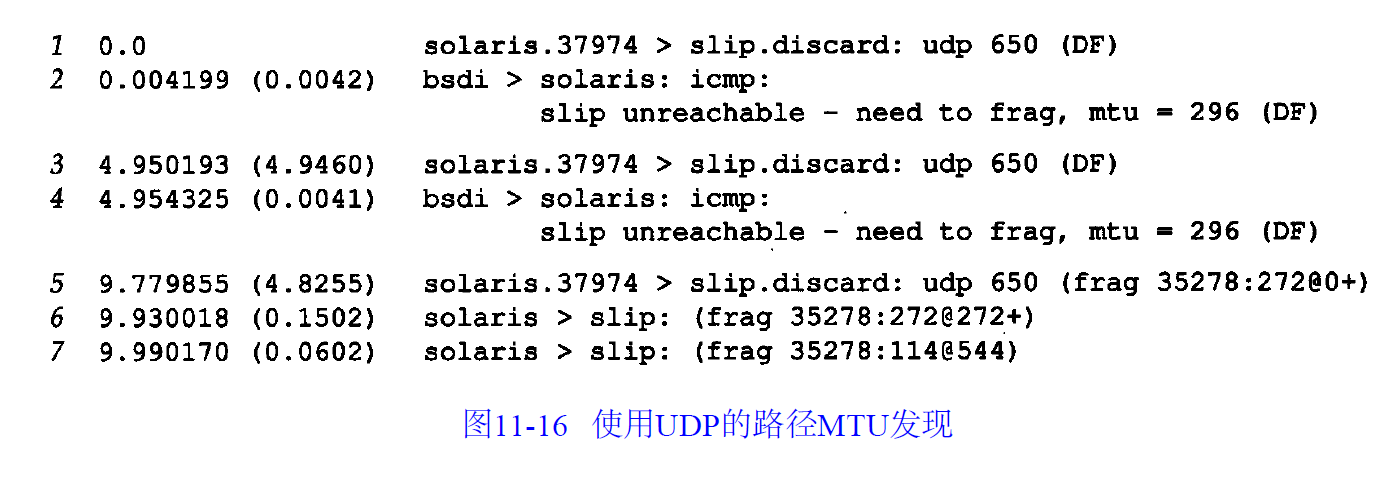
的路由器来完成分片工作。

现在我们运行同一个例子，只是对路由器BSDi进行修改使其在ICMP“不能分片”差错

中返回下一跳MTU。图11- 16给出了TCPDUMp输出结果的前6行。

与图11- 14一样，前两个数据报同样是将DF比特置1后发送出去的。但是在知道了下一跳

MTU后，只产生了3个数据报片，而图11- 15中的BSDi路由器则产生了4个数据报片。



# UDP和ARP之间的交互作用

使用UDP，可以看到UDP与ARP典型实现之间的有趣的（而常常未被人提及）交互作用。

我们用SOCk程序来产生一个包含8192字节数据的UDP数据报。预测这将会在以太网上产生6个数据报片（见习题11. 3）。同时也确保在运行该程序前， ARP缓存是清空的，这样，在发送第一个数据报片前必须交换ARP请求和应答。

bsdI% arP-A验证ARP高速缓存是空的

bsdI% socK-U-I-nL-w8192svr4discard

预计在发送第一个数据报片前会先发送一个ARP请求。IP还会产生5个数据报片，这样就提出了我们必须用TCPDUMp来回答的两个问题：在接收到ARP回答前，其余数据报片是否已经做好了发送准备？如果是这样，那么在ARP等待应答时，它会如何处理发往给定目的的多个报文？图11- 17给出了TCPDUMp的输出结果。

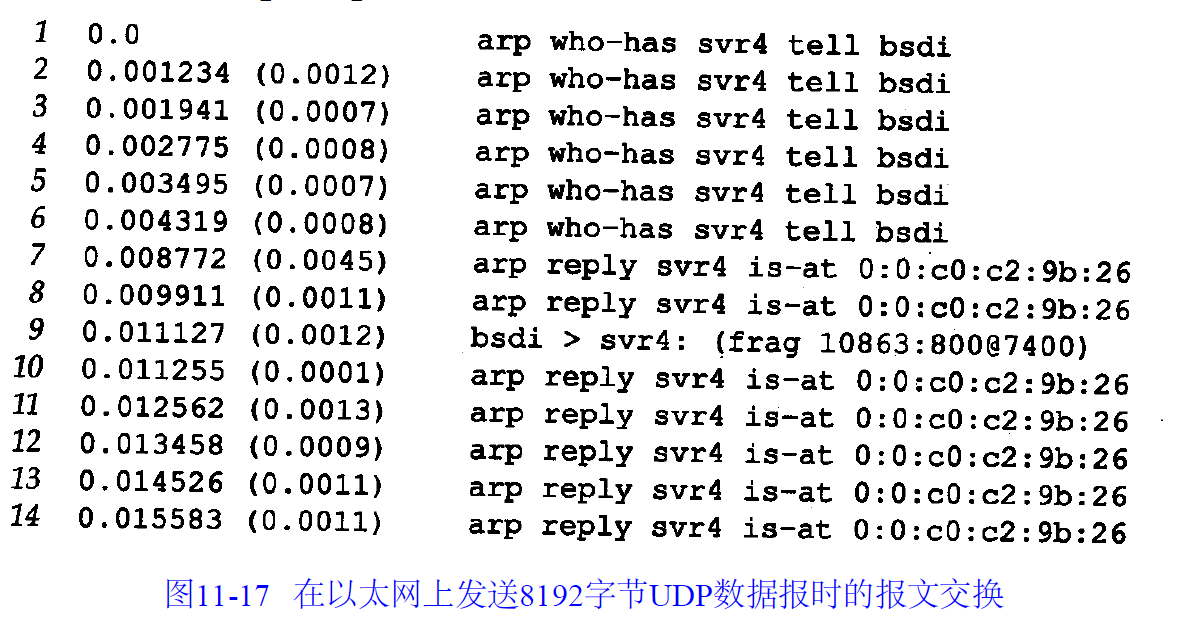


图11-17在以太网上发送8192字节UDP数据报时的报文交换在这个输出结果中有一些令人吃惊的结果。

首先，在第一个ARP应答返回以前，总共产生了6个ARP请求。我们认为其原因是IP很快地产生了6个数据报片，而每个数据报片都引发了一个ARP请求。

第二，在接收到第一个ARP应答时（第7行），只发送最后一个数据报片（第9行）！看来似乎将前5个数据报片全都丢弃了。实际上，这是ARP的正常操作。在大多数的实现中，在等待一个ARP应答时，只将最后一个报文发送给特定目的主机。

HosTRequirementSRFC要求实现中必须防止这种类型的ARP洪泛（ARPflooding，即以高速率重复发送到同一个IP地址的ARP请求）。建议最高速率是每秒一次。而这里却在4.3ms内发出了6个ARP请求。

HosTRequirementSRFC规定，ARP应该保留至少一个报文，而这个报文必须是最后一个报文。这正是我们在这里所看到的结果。

另一个无法解释的不正常的现象是， SVR4发回7个，而不是6个ARP应答。

最后要指出的是，在最后一个ARP应答返回后，继续运行TCPDUMp程序5分钟，以看看SVR4是否会返回ICMP“组装超时”差错。并没有发送ICMP差错（我们在图8- 2中给出了该消息的格式。CODe字段为1表示在重新组装数据报时发生了超时）。

在第一个数据报片出现时， IP层必须启动一个定时器。这里“第一个”表示给定数据报的第一个到达数据报片，而不是第一个数据报片（数据报片偏移为0）。正常的定时器值为30或60秒。如果定时器超时而该数据报的所有数据报片未能全部到达，那么将这些数据报片丢弃。如果不这么做，那些永远不会到达的数据报片（正如我们在本例中所看到的那样）迟早

会引起接收端缓存满。

这里我们没看到ICMP消息的原因有两个。

首先，大多数从BERKELEy派生的实现从不产生该差错！这些实现会设置定时器，也会在定时器溢出时将数据报片丢弃，但是不生成ICMP差错。

第二，并未接收到包含UDP首部的偏移量为0的第一个数据报片（这是被ARP所丢弃的5个报文的第1个）。除非接收到第一个数据报片，否则并不要求任何实现产生ICMP差错。其原因是因为没有运输层首部，ICMP差错的接收者无法区分出是哪个进程所发送的数据报被丢弃。

这里假设上层（TCP或使用UDP的应用程序）最终会超时并重传。在本节中，我们使用IP数据报片来查看UDP与ARP之间的交互作用。如果发送端迅速发送多个UDP数据报，也可以看到这个交互过程。我们选择采用分片的方法，是因为IP可以生成报文的速度，比一个用户进程生成多个数据报的速度更快。尽管本例看来不太可能，但它确实经常发生。NFS发送的UDP数据报长度超过8192字节。在以太网上，这些数据报以我们所指出的方式进行分片，如果适当的ARP缓存入口发生超时，那么就可以看到这里所显示的现象。NFS将超时并重传，但是由于ARP的有限队列，第一个IP数据报仍可能被丢弃。

# 最大UDP数据报长度

理论上，IP数据报的最大长度是65535字节，这是由IP首部（图3- 1）16比特总长度字段所限制的。

去除20字节的IP首部和8个字节的UDP首部， UDP数据报中用户数据的最长长度为65507字节。

但是，大多数实现所提供的长度比这个最大值小。

我们将遇到两个限制因素。

第一，应用程序可能会受到其程序接口的限制。

sockeTAPI提供了一个可供应用程序调用的函数，以设置接收和发送缓存的长度。对于UDPsocket，这个长度与应用程序可以读写的最大UDP数据报的长度直接相关。现在的大部分系统都默认提供了可读写大于8192字节的UDP数据报（使用这个默认值是因为8192是NFS读写用户数据数的默认值）。

第二个限制来自于TCP/ IP的内核实现。

可能存在一些实现特性（或差错），使IP数据报长度小于65535字节。

作者使用SOCk程序对不同UDP数据报长度进行了试验。在SunOS4.1.3下使用环回接口的最大IP数据报长度是32767字节。比它大的值都会发生差错。但是从BSD/ 386到SunOS4.1.3的情况下，SUn所能接收到最大IP数据报长度为32786字节（即32758字节用户数据）。在SolariS2.2下使用环回接口，最大可收发IP数据报长度为65535字节。从SolariS2.2到AIX3.2.2，发送的最大IP数据报长度可以是65535字节。很显然，这个限制与源端和目的端的实现有关。

我们在3. 2节中提过，要求主机必须能够接收最短为576字节的IP数据报。

在许多UDP应用程序的设计中，其应用程序数据被限制成512字节或更小，因此比这个限制值小。例如，我们在10. 4节中看到，路径信息协议总是发送每份数据报小于512字节的数据。我们还会在其他UDP应用程序如DNS（第14章）、TFTP（第15章）、BOOTP（第16章）以及SNMP（第25章）中遇到这个限制。

数据报截断由于IP能够发送或接收特定长度的数据报并不意味着接收应用程序可以读取该长度的数据。因此， UDP编程接口允许应用程序指定每次返回的最大字节数。如果接收到的数据报长度大于应用程序所能处理的长度，那么会发生什么情况呢？

不幸的是，该问题的答案取决于编程接口和实现。

典型的BERKELEy版sockeTAPI对数据报进行截断，并丢弃任何多余的数据。应用程序何时能够知道，则与版本有关（4.3BSDReno及其后的版本可以通知应用程序数据报被截断）。

SVR4下的sockeTAPI(包括SolariS2.x) 并不截断数据报。超出部分数据在后面的读取中返回。它也不通知应用程序从单个UDP数据报中多次进行读取操作。

TLIAPI不丢弃数据。相反，它返回一个标志表明可以获得更多的数据，而应用程序后面的读操作将返回数据报的其余部分。

在讨论TCP时，我们发现它为应用程序提供连续的字节流，而没有任何信息边界。TCP以应用程序读操作时所要求的长度来传送数据，因此，在这个接口下，不会发生数据丢失。

# ICMP源站抑制差错

我们同样也可以使用UDP产生ICMP“源站抑制(sourcEquench)”差错。当一个系统（路

由器或主机）接收数据报的速度比其处理速度快时，可能产生这个差错。注意限定词“可能”。

即使一个系统已经没有缓存并丢弃数据报，也不要求它一定要发送源站抑制报文。

图11- 18给出了ICMP源站抑制差错报文的格式。有一个很好的方案可以在我们的测试网

络里产生该差错报文。可以从BSDi通过必须经过拨号SLIP链路的以太网，将数据报发送给路

由器SUn。由于SLIP链路的速度大约只有以太网的千分之一，因此，我们很容易就可以使其缓

存用完。下面的命令行从主机BSDi通过路由器SUn发送100个1024字节长数据报给SOLARIs。

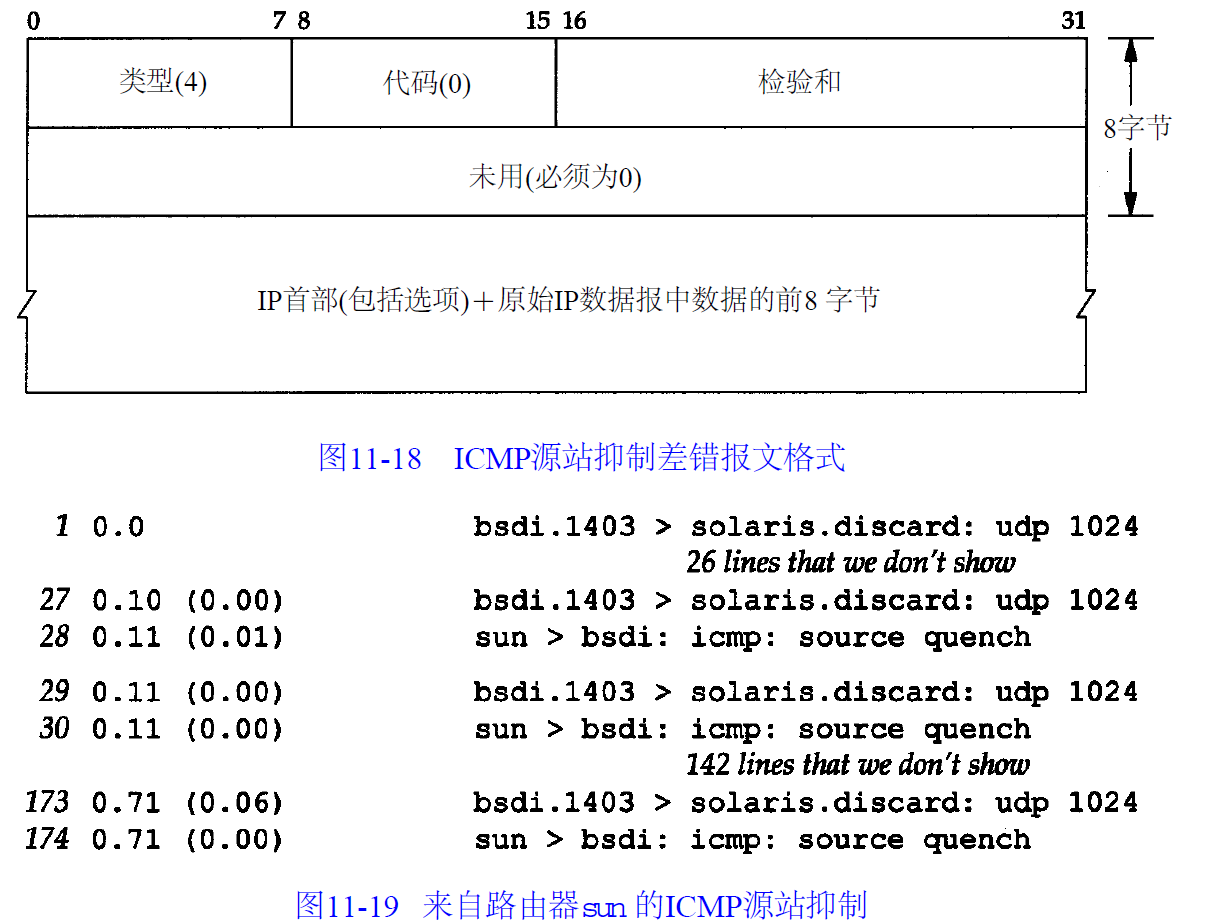
我们将数据报发送给标准的丢弃服务，这样，这些数据报将被忽略：

bsdI% socK-U-I-w1024-n100solariSdiscard

图11- 19给出了与此命令行相对应的TCPDUMp输出结果

在这个输出结果中，删除了很多行，这只是一个模型。接收前26个数据报时未发生差错；我们只给出了第一个数据报的结果。然而，从第27个数据报开始，每发送一份数据报，

就会接收到一份源站抑制差错报文。总共有26+（74×2）= 174行输出结果。



从2. 10节的并行线吞吐率计算结果可以知道，以9600b/s速率传送1024字节数据报只需要

1秒时间（由于从SUn到NETb的SLIP链路的MTU为552字节，因此在我们的例子中， 20+ 8+

1024字节数据报将进行分片，因此，其时间会稍长一些）。但是我们可以从图11- 19的时间中

看出， SUn路由器在不到1秒时间内就处理完所有的100个数据报，而这时，第一份数据报还

未通过SLIP链路。因此我们用完其缓存就不足不奇了。

尽管RFC1009[BradeNanDPosteL1987] 要求路由器在没有缓存时产生源站抑制差

错报文，但是新的RouteRRequirementSRFC[AlmquisT1993] 对此作了修改，提出路由

器不应该产生源站抑制差错报文。由于源站抑制要消耗网络带宽，且对于拥塞来说是

一种无效而不公平的调整，因此现在人们对于源站抑制差错的态度是不支持的。

在本例中，还需要指出的是， SOCk程序要么没有接收到源站抑制差错报文，要么接收到

却将它们忽略了。结果是如果采用UDP协议，那么BSD实现通常忽略其接收到的源站抑制报

文（正如我们在21. 10节所讨论的那样， TCP接受源站抑制差错报文，并将放慢在该连接上的

数据传输速度）。其部分原因在于，在接收到源站抑制差错报文时，导致源站抑制的进程可能

已经中止了。实际上，如果使用UniX的TIMe程序来测定SOCk程序所运行的时间，其结果是它

只运行了大约0. 5秒时间。但是从图11- 19中可以看到，在发送第一份数据报过后0. 71秒才接收

到一些源站抑制，而此时该进程已经中止。其原因是我们的程序写入了100个数据报然后中止

了。但是所有的100个数据报都已发送出去—有一些数据报在输出队列中。

这个例子重申了UDP是一个非可靠的协议，它说明了端到端的流量控制。尽管SOCk程序

成功地将100个数据报写入其网络，但只有26个数据报真正发送到了目的端。其他74个数据报可能被中间路由器丢弃。除非在应用程序中建立一些应答机制，否则发送端并不知道接收端

是否收到了这些数据。

# UDP服务器的设计

使用UDP的一些蕴含对于设计和实现服务器会产生影响。通常，客户端的设计和实现比

服务器端的要容易一些，这就是我们为什么要讨论服务器的设计，而不是讨论客户端的设计

的原因。典型的服务器与操作系统进行交互作用，而且大多数需要同时处理多个客户。

通常一个客户启动后直接与单个服务器通信，然后就结束了。而对于服务器来说，它启

动后处于休眠状态，等待客户请求的到来。对于UDP来说，当客户数据报到达时，服务器苏

醒过来，数据报中可能包含来自客户的某种形式的请求消息。

在这里我们所感兴趣的并不是客户和服务器的编程方面（ [StevenS1990]对这些方面的细

节进行了讨论），而是UDP那些影响使用该协议的服务器的设计和实现方面的协议特性（我们

在18. 11节中对TCP服务器的设计进行了描述）。尽管我们所描述的一些特性取决于所使用

UDP的实现，但对于大多数实现来说，这些特性是公共的。

## 11.12.1客户IP地址及端口号

来自客户的是UDP数据报。IP首部包含源端和目的端IP地址， UDP首部包含了源端和目

的端的UDP端口号。当一个应用程序接收到UDP数据报时，操作系统必须告诉它是谁发送了

这份消息，即源IP地址和端口号。

这个特性允许一个交互UDP服务器对多个客户进行处理。给每个发送请求的客户发回应

答。

## 11.12.2目的IP地址

一些应用程序需要知道数据报是发送给谁的，即目的IP地址。例如， HosTRequirementsRFC规定，TFTP服务器必须忽略接收到的发往广播地址的数据报（我们分别在第12章和第15章对广播和TFTP进行描述）。

这要求操作系统从接收到的UDP数据报中将目的IP地址交给应用程序。不幸的是，并非所有的实现都提供这个功能。

sockeTAPI以IPRECVDSTADDRsocket选项提供了这个功能。对于本文中使用的系统，只有BSD/ 386、4. 4BSD和AIX3.2.2支持该选项。SVR4、SunOS4.x和SolariS2.x都不支持该选项。

## 11.12.3UDP输入队列

我们在1. 8节中说过，大多数UDP服务器是交互服务器。这意味着，单个服务器进程对单个UDP端口上（服务器上的名知端口）的所有客户请求进行处理。

通常程序所使用的每个UDP端口都与一个有限大小的输入队列相联系。这意味着，来自不同客户的差不多同时到达的请求将由UDP自动排队。接收到的UDP数据报以其接收顺序交给应用程序（在应用程序要求交送下一个数据报时）。然而，排队溢出造成内核中的UDP模块丢弃数据报的可能性是存在的。可以进行以下试验。我们在作为UDP服务器的BSDi主机上运行SOCk程序：

bsdI% socK-S-U-V-E-R256-P306666

froM140.252.13.33, tO140.252.13.63: 1111111111从 SUN发 送到广播地址

froM140.252.13.34, tO140.252.13.35: 4444444444从4s4v4R4发 送到单播地址

我们指明以下标志： - s表示作为服务器运行， - u表示UDP，- v表示打印客户的IP地址，- E表示打印目的IP地址（该系统支持这个功能）。另外，我们将这个端口的UDP接收缓存设置为256字节（- R），其每次应用程序读取的大小也是这个数（ - r）。标志- P30表示创建UDP端口后，先暂停30秒后再读取第一个数据报。这样，我们就有时间在另两台主机上启动客户程序，发送一些数据报，以查看接收队列是如何工作的。

服务器一开始工作，处于其30秒的暂停时间内，我们就在SUn主机上启动一个客户，并发送三个数据报：

suN% socK-U-V140.252.13.636666到以太网广播地址

connecteDoN140.252.13.33.1252tO140.252.13.63.6666

111111111111字节的数据（新行）

22222222210字节的数据（新行）

3333333333312字节的数据（新行）

目的地址是广播地址（ 140. 252. 13. 63）。我们同时也在主机SVR4上启动第2个客户，并发送另外三个数据报：

svr4% socK-U-VbsdI6666

connecteDoN0.0.0.0.1042tO140.252.13.35.6666

444444444444414字节的数据（新行）

55555555555555516字节的数据（新行）

666666669字节的数据（新行）

首先，我们早些时候在BSDi上所看到的结果表明，应用程序只接收到2个数据报：来自SUn的第一个全1报文，和来自SVR4的第一个全4报文。其他4个数据报看来全被丢弃。图11- 20给出的TCPDUMp输出结果表明，所有6个数据报都发送给了目的主机。两个客户的数据报以交替顺序键入：第一个来自SUn，然后是来自SVR4的，以此类推。同时也可以看出，全部6个数据报大约在12秒内发送完毕，也就是在服务器休眠的30秒内完成的。

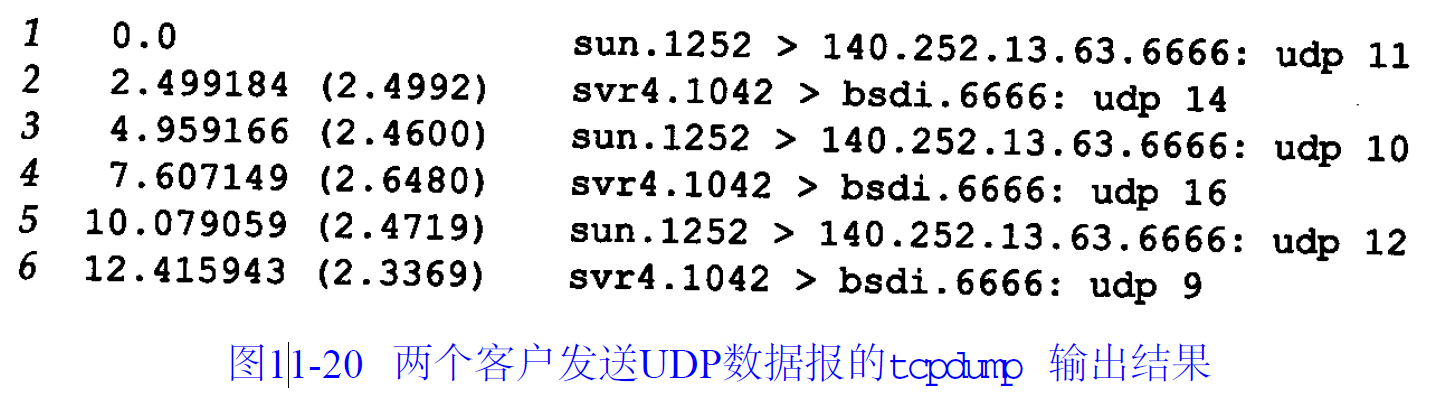


图11-20两个客户发送UDP数据报的TCPDUMP输出结果

我们还可以看到，服务器的- E选项使其可以知道每个数据报的目的IP地址。如果需要，

它可以选择如何处理其接收到的第一个数据报，这个数据报的地址是广播地址。

我们可以从本例中看到以下几个要点。首先，应用程序并不知道其输入队列何时溢出。

只是由UDP对超出数据报进行丢弃处理。同时，从TCPDUMp输出结果，我们看到，没有发回

任何信息告诉客户其数据报被丢弃。这里不存在像ICMP源站抑制这样发回发送端的消息。最

后，看来UDP输出队列是FIFO（先进先出）的，而我们在11. 9节中所看到的ARP输入却是

LIFO（后进先出）的。

## 11.12.4限制本地IP地址

大多数UDP服务器在创建UDP端点时都使其本地IP地址具有通配符( WILDCARD)的特点。这

就表明进入的UDP数据报如果其目的地为服务器端口，那么在任何本地接口均可接收到它。

例如，我们以端口号777启动一个UDP服务器：

suN% socK-U-S7777

然后，用NETSTAt命令观察端点的状态：

suN% netstaT-A-N-Finet

ActivEInterneTconnectionS(includinGservers)

ProtORecv-QSend-QLocaLAddresSForeigNAddresS(state)

udP00\*.7777\*.\*

这里，我们删除了许多行，只保留了其中感兴趣的东西。- a选项表示报告所有网络端点

的状态。- n选项表示以点数格式打印IP地址而不用DNS把地址转换成名字，打印数字端口号

而不是服务名称。-Finet选项表示只报告TCP和UDP端点。

本地地址以\* . 7777格式打印，星号表示任何本地IP地址。

当服务器创建端点时，它可以把其中一个主机本地IP地址包括广播地址指定为端点的本

地IP地址。只有当目的IP地址与指定的地址相匹配时，进入的UDP数据报才能被送到这个端

点。用我们的SOCk程序，如果在端口号之前指定一个IP地址，那么该IP地址就成为该端点的

本地IP地址。例如：

suN% socK-U-S140.252.1.297777

就限制服务器在SLIP接口( 140. 252. 1. 29)处接收数据报。NETSTAt输出结果显示如下：

ProtORecv-QSend-QLocaLAddresSForeigNAddresS(state)

udP00140.252.1.29.7777\*.\*

如果我们试图在以太网上的主机BSDi以地址140. 252. 13. 35向该服务器发送一份数据报，

那么将返回一个ICMP端口不可达差错。服务器永远看不到这份数据报。这种情形如图11- 21

所示。

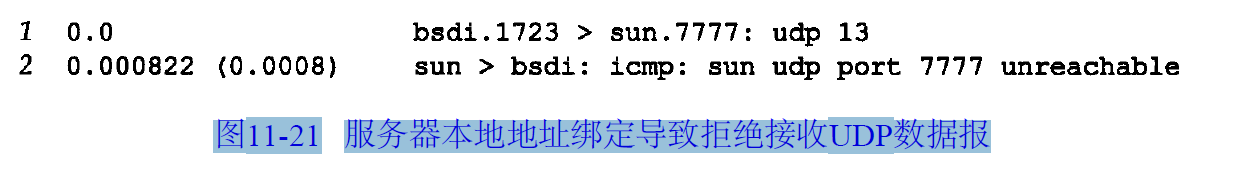


图11-21服务器本地地址绑定导致拒绝接收UDP数据报

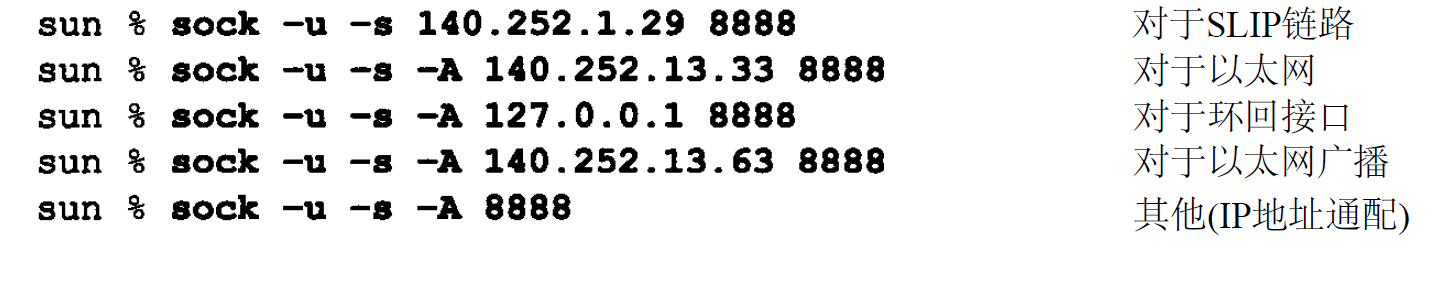
有可能在相同的端口上启动不同的服务器，每个服务器具有不同的本地IP地址。但是，

一般必须告诉系统应用程序重用相同的端口号没有问题。

使用socketSAPI时，必须指定SOREUSEADDRSOCKEt选项。在SOCk程序中是通过-A

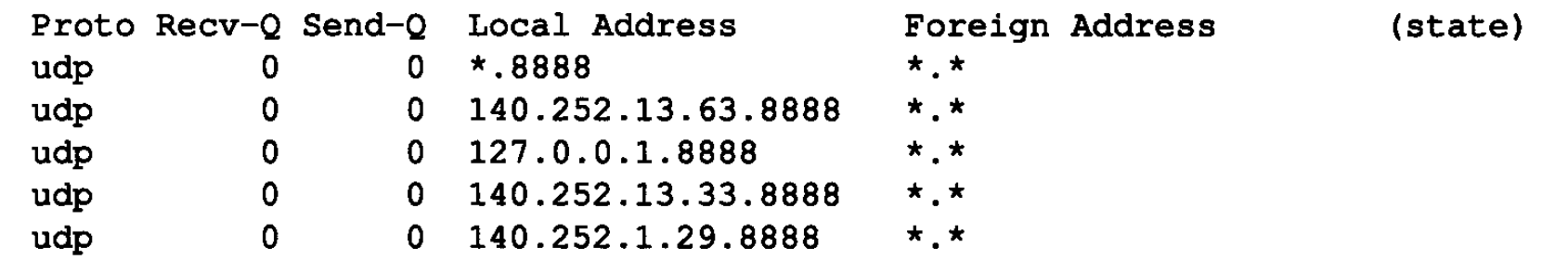
选项来完成的。

在主机SUn上，可以在同一个端口号（ 8888）上启动5个不同的服务器：



除了第一个以外，其他的服务器都必须以- A选项启动，告诉系统可以重用同一个端口号。

5个服务器的NETSTAt输出结果如下所示：



在这种情况下，到达服务器的数据报中，只有带星号的本地IP地址，其目的地址为

140. 252. 1. 255，因为其他4个服务器占用了其他所有可能的IP地址。

如果存在一个含星号的IP地址，那么就隐含了一种优先级关系。如果为端点指定了特定

IP地址，那么在匹配目的地址时始终优先匹配该IP地址。只有在匹配不成功时才使用含星号

的端点。

## 11.12.5限制远端IP地址

在前面所有的NETSTAt输出结果中，远端IP地址和远端端口号都显示为\* . \*，其意思是该端

点将接受来自任何IP地址和任何端口号的UDP数据报。大多数系统允许UDP端点对远端地址

进行限制。

这说明端点将只能接收特定IP地址和端口号的UDP数据报。SOCk程序用- f选项来指定远

端IP地址和端口号：

suN% socK-U-S-F140.252.13.35.44445555

这样就设置了远端IP地址140. 252. 13. 35（即主机BSDi）和远端端口号4444。服务器的有名端

口号为5555。如果运行NETSTAt命令，我们发现本地IP地址也被设置了，尽管我们没有指定。

ProtORecv-QSend-QLocaLAddresSForeigNAddresS(state)

udP00140.252.13.33.5555140.252.13.35.4444

这是在伯克利派生系统中指定远端IP地址和端口号带来的副作用：如果在指定远端地址

时没有选择本地地址，那么将自动选择本地地址。它的值就成为选择到达远端IP地址路由时

将选择的接口IP地址。事实上，在这个例子中， SUN在以太网上的IP地址与远端地址

140. 252. 13. 33相连。

图11- 22总结了UDP服务器本身可以创建的三类地址绑定。



图11-22为UDP服务器指定本地和远端IP地址及端口号

在所有情况下，LPORt指的是服务器有名端口号， LOCALIP必须是本地接口的IP地址。表中这

三行的排序是UDP模块在判断用哪个端点接收数据报时所采用的顺序。最为确定的地址（第

一行）首先被匹配，最不确定的地址（最后一行IP地址带有两个星号）最后进行匹配。

## 11.12.6每个端口有多个接收者

尽管在RFC中没有指明，但大多数的系统在某一时刻只允许一个程序端点与某个本地IP地址及UDP端口号相关联。当目的地为该IP地址及端口号的UDP数据报到达主机时，就复制

一份传给该端点。端点的IP地址可以含星号，正如我们前面讨论的那样。

例如，在SunOS4.1.3中，我们启动一个端口号为9999的服务器，本地IP地址含有星号：

suN% socK-U-S9999

接着，如果启动另一个具有相同本地地址和端口号的服务器，那么它将不运行，尽管我

们指定了- A选项：

suN% socK-U-S9999我们预计它会失败

can'TbinDlocaLaddress: AddresSalreadYiNuse

suN% socK-U-S-A9999因此，这次尝试- A参数

can'TbinDlocaLaddress: AddresSalreadYiNuse

在一个支持多播的系统上（第12章），这种情况将发生变化。多个端点可以使用同一个IP

地址和UDP端口号，尽管应用程序通常必须告诉API是可行的（如，用- A标志来指明

SOREUSEADDRSOCKEt选项）。

4. 4BSD支持多播传送，需要应用程序设置一个不同的SOCKEt选项（SOREUSEPORT）

以允许多个端点共享同一个端口。另外，每个端点必须指定这个选项，包括使用该端口

的第一个端点。

当UDP数据报到达的目的IP地址为广播地址或多播地址，而且在目的IP地址和端口号处

有多个端点时，就向每个端点传送一份数据报的复制（端点的本地IP地址可以含有星号，它

可匹配任何目的IP地址）。但是，如果UDP数据报到达的是一个单播地址，那么只向其中一个

端点传送一份数据报的复制。选择哪个端点传送数据取决于各个不同的系统实现。

# 小结

UDP是一个简单协议。它的正式规范是RFC768[PosteL1980]，只包含三页内容。它向用

户进程提供的服务位于IP层之上，包括端口号和可选的检验和。我们用UDP来检查检验和，

并观察分片是如何进行的。

接着，我们讨论了ICMP不可达差错，它是新的路径MTU发现功能中的一部分（ 2. 9节）。

用TRACEROUTe和UDP来观察路径MTU发现过程。还查看了UDP和ARP之间的接口，大多数的

ARP实现在等待ARP应答时只保留最近传送给目的端的数据报。

当系统接收IP数据报的速率超过这些数据报被处理的速率时，系统可能发送ICMP源站抑

制差错报文。使用UDP时很容易产生这样的ICMP差错。