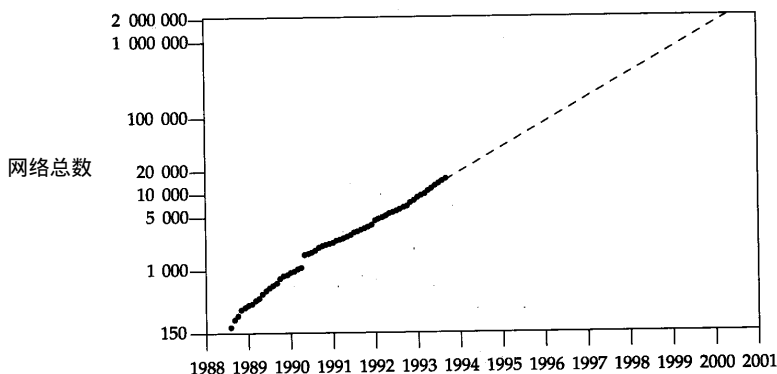


## 附录D 部分习题的解答

### 第1章

- 1.1 答案是： $2^7 - 2(126) + 2^{14} - 2(16\,382) + 2^{21} - 2(2\,097\,150) = 2\,113\,658$ 。每一部分都减去2是因为全0或全1网络ID是非法的。
- 1.2 图D-1显示了直到1993年8月的有关数据。



图D-1 宣布加入NSFNET的网络数

如果网络数继续呈指数增长的话，虚线估计了2000年可能达到的最大的网络数。

- 1.3 “自由地接收，保守地发送。”

### 第3章

- 3.1 不，任何网络ID为127的A类地址都是可行的，尽管大多数系统使用了127.0.0.1。
- 3.2 kpn0有5个接口：3个点对点链路和2个以太网接口。R10有4个以太网接口。gateway有3个接口：2个点对点链路和1个以太网接口。最后，netb有1个以太网接口和2个点对点链路。
- 3.3 没有区别：作为一个没有再区分子网的C类地址，它们都有一个255.255.255.0的子网掩码。
- 3.5 它是合法的，被称为非连续的子网掩码，因为其用于子网掩码的16位是不连续的。但是RFC建议反对使用非连续的子网掩码。
- 3.6 这是一个历史遗留问题。值是1024 + 512，但是打印的MTU值包含了所有需要的首部字节数。Solaris 2.2将回环接口的MTU设置为8232 (8192 + 40)，其中包含了8192字节的用户数据加上20字节的IP首部和20字节的TCP首部。
- 3.7 第一，数据报降低了路由器中对于连接状态的需求。第二，数据报提供了基本的构件，在它的上面可以构造不可靠的（UDP）和可靠的（TCP）的运输层。第三，数据报代表

了最小的网络层假定, 使得可以使用很大范围的数据链路层服务。

## 第4章

- 4.1 发出一条rsh命令与另一台主机建立一个TCP连接。这样做引起在两个主机之间交换IP数据报。为此, 在那台主机的ARP缓存中必须有我们这台主机的登记项。因此, 即使在执行rsh命令之前, ARP缓存是空的, 当rsh服务器执行arp命令时, 必须保证ARP缓存中登记有我们这台主机。
- 4.2 保证你的主机上的ARP缓存中没有登记以以太网上的某个叫作foo的主机。保证foo引导时发送一个免费ARP请求, 也许是在foo引导时, 在那台主机上运行tcpdump。然后关闭主机foo, 使用说明了temp选项的arp命令, 在你的系统的ARP缓存中为foo输入一个不正确的登记项。引导foo并在它启动好之后, 察看主机的ARP缓存, 看看不正确的登记项是不是已经被更正了。
- 4.3 阅读主机需求 (Host Requirement) RFC的2.3.2.2节和本书中的11.9节。
- 4.4 假设当服务器关闭时, 客户机保存了关于服务器的一个完整的ARP登记项。如果我们继续试图与 (已关闭的) 服务器联系, 过了20分钟以后, ARP将超时。最后, 当服务器以一个新的硬件地址重新启动。如果它没有发出一个免费ARP, 旧的、不再正确的ARP登记项仍然存在于客户机上。我们将无法和在新硬件地址上的服务器联系直到我们手工删除这个ARP登记项, 或者在20分钟内停止与服务器联系的尝试。

## 第5章

- 5.1 一个单独的帧类型并不是必需的, 因为图4-3中的op字段对于所有的四个操作 (ARP请求、ARP应答、RARP请求和RARP应答) 都有一个不同的值。但是实现一个RARP服务器, 独立于内核中的ARP服务器, 更容易处理不同的帧类型字段。
- 5.2 每个RARP服务器在发送一个响应之前可以延迟一个小的随机时间。  
作为一个优化, 可以指定一个RARP服务器为主服务器, 其他的为次服务器。主服务器发出响应不需要延迟, 而次服务器发出响应则需要一个随机的延迟。  
作为另一个优化, 也是指定一个主RARP服务器, 其他为次服务器。次服务器只在一个短时间段内发生的重复请求进行响应。这里假设出现重复请求的原因是由于主服务器停机了。

## 第6章

- 6.1 如果在局域网线上有一百个主机, 每个都可能在同一时刻发送一个ICMP端口不可达的报文。很多报文的传输都可能发生冲突 (如果使用的是以太网), 这将导致1秒或2秒的时间里网络不可用。
- 6.2 它是一个“should”。
- 6.3 如我们在图3-2所指出的, 发送一个ICMP差错总是将TOS置为0。发送一个ICMP查询请求可以将TOS置为任何值, 但是发送相应的应答必须将TOS置为相同的值。
- 6.4 netstat -s是查看每个协议统计数据的常用方法。在一台收到了4800万个IP数据报的SunOS 4.1.1主机 (gemini) 上, ICMP的统计为:

```
Output histogram:
  echo reply: 1757
  destination unreachable: 700
  time stamp reply: 1
Input histogram:
  echo reply: 211
  destination unreachable: 3071
  source quench: 249
  routing redirect: 2789
  echo: 1757
  #10: 21
  time exceeded: 56
  time stamp: 1
```

21个类型为10的报文是SunOS 4.1.1不支持的路由器的请求。

也可以使用SNMP（图25-26），有些系统，如Solaris 2.2，可以生成使用SNMP变量名的netstat-s的输出。

## 第7章

- 7.2 86字节除以960字节/秒，乘以2得到179.2 ms。当以这个速率运行ping时，打印的值为180 ms。
- 7.3  $(86 + 48)$  除以960字节/秒，乘以2得到279.2 ms。另外的48字节是因为56字节的数据部分的最后48字节必须忽略：0xc0是SLIP END字符。
- 7.4 CSLIP只压缩了TCP报文段的TCP首部和IP首部。它对ping使用的ICMP报文没有作用。
- 7.5 在一个SPARC工作站 ELC上，对回环地址的ping操作产生一个1.310 ms的RTT，而对一个主机的以太网地址的ping操作产生一个1.460 ms的RTT。这个差值是由于以太网驱动程序需要时间来判定这个数据报的目的地址是一个本地的主机。需要一个产生微秒级输出的ping来验证这一点。

## 第8章

- 8.1 如果一个输入数据报的TTL为0，做减一操作然后测试会将把TTL设置为255，并且让数据报继续传输。尽管一个路由器永远不会收到一个TTL为0的数据报，但这种情况确实会发生。
- 8.2 我们注意到traceroute在UDP数据报的数据部分存储了12个字节，其中包含了数据报发送的时间。然而，从图6-9可以看出ICMP只返回了出错的IP数据报的头8个字节，实际上这是8个字节的UDP首部。因此，ICMP的差错报文没有返回traceroute存储的时间值。traceroute保存了它发送分组的时间，当收到一个ICMP应答时，取出当时的时间，把两个值相减就可以得出RTT。
- 回忆一下第7章中，ping在输出的ICMP回显请求中存储了时间，这个值被服务器回显了回来。这样即使分组返回时失序，ping也能打印出正确的RTT。
- 8.3 第1行输出是正确的，并且标识了R1。下一个探测分组启动时将TTL置为2，并且这个值被R1减1。当R2收到这个分组时，把TTL从1减为0，但是错误地将它传递给了R3。R3看见进入的TTL是0就将超时的分组发送回来。这就意味着第2行输出（TTL为2）标识了R3，而不是R2。第3行输出正确地标识了R3。这个错误所表现出来的线索就是两个连续的输

出行标识了同一个路由器。

- 8.4 在这种情况下, TTL为1标识了R1, TTL为2标识了R2, TTL为3标识了R3, 但是当TTL为4时, UDP数据报到达了目的地, 其输入的TTL为1。ICMP端口不可达报文生成了, 但它的TTL是1 (错误地从进入的TTL复制而来)。这个ICMP报文到了R3, 在那儿TTL被减1, 报文被丢弃。没有生成一个ICMP超时报文, 因为被丢弃的数据报是一个ICMP的差错报文 (端口不可达)。类似的现象也出现在TTL为5的探测分组, 但这次输出的端口不可达报文以TTL为2开始 (进入的TTL), 这个报文被传给R2, 在那儿被丢弃。对应于TTL为6探测分组的端口不可达报文被传递给R1, 在那儿被丢弃。最后, 对应于TTL为7探测分组的端口不可达报文被送回了原地, 到达时它的TTL为1 (tracert认为一个TTL为0或1的到达ICMP报文是有问题的, 因此它在RTT后面打印了一个惊叹号)。总之, TTL为1、2和3的行正确地标识了R1、R2和R3, 接下来的三行每个都包含三个超时, 再接下来的TTL为7的行标识了目的地。

- 8.5 它表明这些路由器都将一个ICMP报文的输出TTL设置为255, 这是共同的。从netb输入的255的TTL值是我们想要的, 而从butch输入的253的TTL值表明在butch和netb之间可能有一个未觉察的路由器。否则, 在这个点上我们应该看到一个TTL值为254的输入报文。类似地, 我们希望看到一个值为252而不是249的、来自enss142.UT.westnet.net的报文。这表明这些未觉察的路由器没有正确地处理向外输出的UDP数据报, 但它们都对返回的ICMP报文正确进行了TTL减1操作。

我们必须在查看输入的TTL时非常小心, 因为有时候一个和我们想要的不同的值可能是由于返回的ICMP报文采用了一条与输出UDP数据报不同的路径。但是, 在这个例子中证实了我们所怀疑的——当使用松源路由选项时, 确实存在tracert没有发现的未觉察的路由器。

- 8.7 ping的客户把ICMP回显请求报文 (图7-1) 的标识符字段设置为它的进程ID。ICMP回显应答报文包含同样值的标识符字段。每个客户都要查看这个返回的标识符字段, 并且只处理那些它发送过的报文。

tracert客户将它的UDP源端口号设置为它的进程ID和32768的逻辑或。因为返回的ICMP报文总是包含产生错误的IP数据报的前8个字节 (图6-9), 这8个字节包括了完整的UDP首部, 所以这个源端口号在ICMP差错报文中被返回。

- 8.8 ping客户将ICMP回显请求报文的可选数据部分设置为分组发送的时间。这个可选的数据必须在ICMP回显应答中返回。这样使得即使分组返回时失序, 客户也能计算出精确的回环时间。

tracert客户不能这样操作, 因为在ICMP差错报文中返回的只是UDP首部 (图6-9), 没有UDP数据。因此, tracert必须记住它发送一个请求的时间, 等待应答, 然后计算两者的时间差。

这里显示了ping和tracert的另一个不同点: ping每秒发送一个分组, 而不管是否收到任何应答; tracert发送一个请求, 然后在发送下一个请求前等待一个应答或者一个超时。

- 8.9 因为默认情况下Solaris 2.2从32768开始使用临时的UDP端口, 所以目的主机上的目的端口已经被使用的机会更大。

## 第9章

- 9.1 当ICMP标准第1次发布时，RFC 792 [Postel 1981b]所述的划分子网技术还没有使用。另外，使用一个网络重定向而不是 $N$ 个主机重定向（对于目的网络中的所有 $N$ 个主机）也节省了路由表的空间。
- 9.2 这一项并不需要，但是如果把它删除了，所有到slip的IP数据报将被发送到默认的路由器（sun），后者又将把它们送到路由器bsd1。既然sun将数据报从与接收数据报相同的接口转发出去，它把一个ICMP重定向到svr4。这样在svr4中又创建了我们删除过的同样的路由表项，尽管这一次是因为重定向而创建的，而不是在引导时增加的。
- 9.3 当那个4.2BSD主机收到目的地址是140.1.255.255的数据报，发现它有一个通往该网络（140.1）的路由，因此就试图转发数据报。它发送一个ARP广播来寻找140.1.255.255。这个ARP请求没有收到任何应答，所以这个数据报最终被丢弃。如果在网线上有很多这样的4.2BSD主机，每一个都在差不多同一时刻发送ARP这个广播，将会暂时地阻塞网络。
- 9.4 这次，每一个ARP请求都收到一个应答，告诉每个4.2BSD主机向一个指定的硬件地址（以太网广播）发送数据报。如果网线上有 $k$ 个这样的4.2BSD主机，全部收到了它们自己的ARP应答，使得每一个生成了另一个广播。每个主机都收到了每一个目的地址为140.1.255.255的广播IP数据报，既然现在每个主机都有一个ARP缓存项，这个数据报又被转发给了广播地址。这个过程继续下去，就会产生一次以太网的熔毁（Ethernet meltdown）。[Manber 1990]描述了网络中另一种形式的链式反应。

## 第10章

- 10.1 路由表中有13条来自于kpno：除了140.252.101.0和140.252.104.0之外的所有gateway直接相连的其他网络。
- 10.2 丢失的数据报中通告的25条路由需要60秒才能得到更新。这不成问题，因为一般来说一条路由如果连续3分钟没有得到更新，RIP才会声明它失效。
- 10.3 RIP运行在UDP上，而UDP提供了UDP数据报中数据部分的一个可选的检验和（11.3节）。然而，OSPF运行在IP上，IP的检验和只覆盖了IP首部，所以OSPF必须增加它自己的检验和字段。
- 10.4 负载平衡增加了分组被失序交付的机会，并且很可能使得运输层计算的环回时间出错。
- 10.5 这叫作简单的分裂范围（split horizon）。
- 10.6 在图12-1中，我们显示了100个主机的每一个都通过设备驱动程序、IP层和UDP层来处理这个广播的UDP数据报。当它们发现UDP端口520没有被使用时，这个广播数据报才最终被丢弃。

## 第11章

- 11.1 因为使用IEEE 802封装时，存在8个额外的首部字节，所以1465个字节的用户数据是引起分片的最小长度。
- 11.3 对于IP来说有8200字节的数据需要发送，8192字节的用户数据和8个字节的UDP首部。



采用tcpdump记号, 第1个分片是1480@0+ (1480字节的数据, 偏移为0, 将“更多片”比特置1)。第2个是1480@1480+, 第3个是1480@2960+, 第4个是1480@4440+, 第5个是1480@5920+, 第6个是800@7400。 $1480 \times 5 + 800 = 8200$ , 正好是要发送的字节。

- 11.4 每个1480字节的数据报片被分成三小片: 两个528字节和一个424字节。小于532 (552-20) 的8的最大倍数是528。800字节的数据报片被分成两小片: 一个528字节和一个272字节。这样, 原来8192字节的数据报变成了SLIP链路上的17个帧。
- 11.5 不。问题是当应用程序超时重传时, 重传产生的IP数据报有一个新的标识字段。而重新装配只针对那些具有相同标识字段的分段。
- 11.6 IP首部中的标识字段 (47942) 是一样的。
- 11.7 第一, 从图11-4我们看到gemini没有使能输出UDP的检验和。如果输出UDP的检验和没有被使能, 这个主机上的操作系统 (SunOS 4.1.1) 就不会验证一个进入UDP的检验和。第二, 大多数的UDP通信量都是本地的, 而不是WAN的, 因此没有服从所有的WAN特征。
- 11.8 不严格的和严格的源站选路选项被复制到每一个数据报片中。时间戳选项和记录路由选项没有被复制到每一个数据报片中——它们只出现在第1个数据报片中。
- 11.9 不。在11.12节中, 我们看到很多实现可以根据目的IP地址、源IP地址和源端口号来过滤送往一个给定UDP端口号的输入数据报。

## 第12章

- 12.1 广播本身不会增加网络通信量, 但它增加了额外的主机处理时间。如果接收主机不正确地响应了诸如ICMP端口不可达之类的差错, 那么广播也可能导致额外的网络通信量。路由器一般不转发广播分组, 而网桥一般转发, 所以在一个桥接网络上的广播分组可能比在一个路由网络上走得更远。
- 12.2 每个主机都收到了所有广播分组的一个副本。接口层收到了帧, 把它传递给设备驱动程序。如果类型字段指的是其他协议, 设备驱动程序就会丢弃该帧。
- 12.3 首先执行netstat-r来看一下路由表, 结果显示了所有接口的名字。然后对每个接口执行ifconfig (3.8节): 标志指出了一个接口是否支持广播, 如果支持, 相应的广播地址也会被输出。
- 12.4 伯克利演变的实现不允许对一个广播数据报进行分片。当我们说明了1472字节的长度, 产生的IP数据报将是1500字节, 正好是以太网的MTU。不允许分片一个广播数据报是一个策略上的决定——没有技术上的原因 (并不是想要减少广播分组的数目)。
- 12.5 依赖于100个主机上不同的以太网接口卡的多播支持, 多播数据报可能被接口卡忽略, 或者被设备驱动程序丢弃。

## 第13章

- 13.1 生成随机数时要使用对于主机唯一的值。IP地址和链路层地址是每个主机都应该不一样的两个值。日期时间是一个不好的选择, 尤其是在所有的主机都运行了一个类似于NTP的协议来同步它们的时钟的情况下。
- 13.2 他们增加了一个包括一个序号和一个时间戳的应用协议首部。

## 第14章

- 14.1 一个解析器总是一个客户，但一个名字服务器既是一个客户又是一个服务器。
- 14.2 问题被返回，它占用了前 44 个字节。一个回答占用了剩下来的 31 个字节：2 个字节指向域名的指针（即，指向问题中域名的一个指针），10 字节固定长度的字段（类型、种类、TTL 和资源长度），19 字节的资源数据（一个域名）。注意到资源数据中的域名（`svr4.tuc.noao.edu.`）没有共享问题（`34.13.252.140.in-addr.arpa.`）中域名的后缀，所以不能使用一个指针。
- 14.3 将顺序颠倒意味着首先使用 DNS，如果使用 DNS 失败，然后才将参数翻转过来作为一个点分十进制数。这就是说每次说明一个点分十进制数，都要使用 DNS，涉及一个名字服务器。这是对资源的一种浪费。
- 14.4 RFC 1035 的 4.2.2 节说明了在实际的 DNS 报文之前的两个字节长度的字段。
- 14.5 当一个名字服务器启动时，它一般从一个磁盘文件中读出一个根服务器列表（可能已经过时了）。然后尝试和这些根服务器中的一个联系，请求根域的名字服务器记录（一个 NS 的查询类型）。这个请求返回了当前最新的根服务器列表。启动磁盘文件中根服务器项中至少需要一个是有有效的。
- 14.6 InterNIC 的注册服务每一周更新三次根服务器。
- 14.7 就像应用是不定的一样，解析器也是不定的。如果系统配置成使用多个名字服务器，而且解析器是无状态的，那么解析器就不能记住不同的名字服务器的往返时间。这样定时太短的解析器将会超时，引起不必要的重传。
- 14.8 对 A 记录的排序应该由解析器来执行，而不是名字服务器，因为解析器一般比服务器了解更多的客户的网络拓扑（更新版本的 BIND 提供了解析器对 A 记录排序的功能）。

## 第15章

- 15.1 送往广播地址的 TFTP 请求应该被忽略。正像 Host Requirements RFC 所描述的，对一个广播请求的响应可能产生一个非常严重的安全漏洞。但是，问题是并不是所有的实现和 API 都对接收一个 UDP 数据报的进程提供了该数据报的目的地址（11.12 节）。因为这个原因，很多 TFTP 服务器没有严格遵守这个限制。
- 15.2 不幸的是，RFC 没有提到这个块数目环绕问题。具体实现时应该能够传输最大为  $33\,553\,920$  ( $65535 \times 512$ ) 字节的文件。但是当文件的长度超过  $16\,776\,704$  ( $32767 \times 512$ ) 时，很多实现都会失败，因为它们将块数目错误地表示为一个有符号的 16 位整数，而不是一个无符号的整数。
- 15.3 这样简化了编写一个适合于只读内存的 TFTP 客户的工作，因为服务器是引导文件的发送者，所以服务器必须实现超时和重传机制。
- 15.4 利用它的停止等待协议，TFTP 可以在每一次客户与服务器的往返过程中最多传输 512 字节的数据。TFTP 的最大吞吐量就是 512 字节除以客户与服务器之间的往返时间。在以太网上，假设一个往返时间为 3 ms，那么最大的吞吐量就是大约 170 000 字节/秒。

## 第16章

- 16.1 一个路由器可以转发一个 RARP 请求到路由器连接的其他网络上的任何一台主机上。但

是发送应答就成问题了, 路由器还必须转发 RARP 应答。

BOOTP 没有这个应答问题, 因为应答的地址是路由器知道如何转发的一般 IP 地址。问题是 RARP 只使用了链路层地址, 路由器一般不知道在其他的、没有连接在路由器的网络上主机的链路层地址。

- 16.2 它可能使用了自己的硬件地址。该地址应该是唯一的, 在请求报文中设置, 在应答中返回。

## 第17章

- 17.1 除了 UDP 的检验和, 其他都是必需的。IP 检验和只覆盖了 IP 首部, 而其他字段都紧接着 IP 首部开始。
- 17.2 源 IP 地址、源端口号或者协议字段可能被破坏了。
- 17.3 很多 Internet 应用使用一个回车和换行来标记每个应用记录的结束。这是 NVT ASCII 采用的编码 (26.4 节)。另外一种技术是在每个记录之前加上一个记录的字节计数, DNS (习题 14.4) 和 Sun RPC (29.2 节) 采用了这种技术。
- 17.4 就像我们在 6.5 节所看到的, 一个 ICMP 差错报文必须至少返回引起差错的 IP 数据报中除了 IP 首部的前 8 个字节。当 TCP 收到一个 ICMP 差错报文时, 它需要检查两个端口号以决定差错对应于哪个连接。因此, 端口号必须包含在 TCP 首部的 8 个字节里。
- 17.5 TCP 首部的最后有一些选项, 但 UDP 首部中没有选项。

## 第18章

- 18.1 ISN 是一个 32 bit 的计数器, 它在系统引导大约 9.5 小时后从 4 294 912 000 环绕到 8704。再过大约 9.5 小时它将环绕到 17 408, 然后再过 9.5 小时是 26 112, 如此继续下去。因为系统引导时 ISN 从 1 开始, 并且因为最低次序的数字在 4、8、2、6 和 0 之间循环, 所以 ISN 应该总是一个奇数。
- 18.2 在第 1 种情况下, 我们使用了 sock 程序。默认情况下它把 Unix 的新行字符不作改变地进行传输——单个 ASCII 字符 012 (八进制)。在第 2 种情况下, 我们使用了 Telnet 客户, 它把 Unix 的新行字符转变为两个 ASCII 字符——一个回车符 (八进制 015) 跟着一个换行符 (八进制 012)。
- 18.3 在一个半关闭的连接上, 一个端点已经发送了一个 FIN, 正等待另一端的数据或者一个 FIN。一个半打开的连接是当一个端点崩溃了, 而另一端还不知道的情况。
- 18.4 一个连接只有经过了已建立状态才能进入 2MSL 等待状态。
- 18.5 首先, 日期服务器在将时间和日期写给客户之后对 TCP 连接做一个主动关闭。这可以通过 sock 程序打印的消息: “connection closed by peer.” 表现出来。连接的客户端经历了被动关闭的状态。这样就把一对插口置于服务器端的 TIME\_WAIT 状态, 而不是在客户端。
- 其次, 正如在 18.6 节所示, 大多数伯克利演变的实现都允许一个新的连接请求到达一个正处于 TIME\_WAIT 状态的一对插口, 这也就是这里所发生的情况。
- 18.6 因为在一个已经关闭的连接上到达了一个 FIN, 所以相应于这个 FIN 发送了一个复位。
- 18.7 拨号的一方做主动打开, 电话振铃的一方做被动打开。不允许同时打开, 但允许同时关



闭。

- 18.8 我们将只看到 ARP 请求，而不是 TCP SYN 报文段，但 ARP 请求将和图中具有相同的计时。
- 18.9 客户在主机 solaris 上，服务器在主机 bsdi 上。客户对服务器 SYN 的确认 (ACK) 和客户的第一个数据段结合在一起 (第 3 行)。这种处理非常符合 TCP 的规则，尽管大多数的实现都没有这么做。接着，客户在等待它的数据的确认之前发送了它的 FIN (第 4 行)。这样使得服务器可以在第 5 行同时确认客户数据和它的 FIN。
- 这次交互 (将一个报文段从客户发送到服务器) 需要 7 个报文段，而正常的连接建立和终止 (图 18-13)，以及一个数据段和它的确认，需要 9 个报文段。
- 18.10 首先，服务器对客户的 FIN 的确认一般不会被延迟 (我们在 19.3 节讨论延迟的确认)，而是在 FIN 到达后立即发送。应用进程需要一些时间来接收 EOF，告诉它的 TCP 关闭它这一端的连接。第二，服务器收到客户的 FIN 后，并不一定要关闭它这一端的连接。就像我们在 18.5 节中看到的，仍然可以发送数据。
- 18.11 如果一个产生 RST 的到达报文段有一个 ACK 字段，那么 RST 的序号就是到达的 ACK 字段。第 6 行中值为 1 的 ACK 是相对于第 2 行中的 26368001 的 ISN。
- 18.12 参见 [Crowcroft et al. 1992] 中对分层的评论。
- 18.13 发出了 5 个查询。假设有 3 个分组用于建立连接，1 个用于查询，1 个用于确认查询，1 个用于响应，1 个用于确认响应，4 个用于终止连接。这就是说每次查询需要 11 个分组，总共需要 55 个分组。使用 UDP 则可以减少到 10 个分组。
- 如果对查询的确认和响应结合在一起，每个查询需要的分组可以减少到 10 个 (19.3 节)。
- 18.14 限制是每秒 268 个连接：最大数目的 TCP 端口号 ( $65536 - 1024 = 64512$ ，忽略知名端口) 除以 TIME\_WAIT 状态的 2MSL。
- 18.15 重复的 FIN 会得到确认，2MSL 定时器重新开始。
- 18.16 在 TIME\_WAIT 状态中收到一个 RST 引起状态过早地终止。这就叫作 TIME\_WAIT 断开 (assassination)。RFC1337 [Braden 1992a] 仔细讨论了这个现象，并显示了潜在的问题。这个 RFC 提出的简单的修改就是在 TIME\_WAIT 状态时忽略 RST 段。
- 18.17 它是在具体实现不支持半关闭连接的时候。一旦应用进程引起发送一个 FIN，应用进程就不能再从这个连接读数据了。
- 18.18 不。输入的数据段通过源 IP 地址、源端口号、目的 IP 地址和目的端口号进行区分。在 18.11 节中我们看到对于输入的连接请求，一个 TCP 服务器一般可以通过目的 IP 地址来拒绝接收。

## 第19章

- 19.1 应用程序的两个写操作，跟着一个读操作，引起了迟延，因为 Nagle 算法很可能被激活。第一个报文段 (包含 8 个字节的数据) 被发送后，在发送后面 12 个字节的数据之前必须等待第一个报文段的确认。如果服务器实现了延迟的确认，在收到这个确认之前，可能会有一个达到 200 ms 的时延 (加上 RTT)。
- 19.2 假设 5 个字节的 CSLIP 首部 (IP 和 TCP) 和两个字节的的数据，这些段通过 SLIP 链路的 RTT 大约是 14.5 ms。我们要加上通过以太网的 RTT (一般是 5~10 ms)，加上在 sun 和 bsdi

上的选路时间。因此, 观察到的时间看起来是正确的。

- 19.3 在图19-6中, 第6和第9报文段之间的时间是533ms。在图19-8中, 第8和第12报文段之间的时间是272ms (我们测量了F2键的时间, 而不是F1键, 因为F1键的第1个回显在第2个图中丢掉了)。

## 第20章

- 20.1 字节号0是SYN, 字节号8193是FIN。SYN和FIN在序号空间里各占了一个字节。
- 20.2 应用程序的第1个写操作引起置位 PUSH标志发送第1个报文段。因为 BSD/386总是使用慢启动, 它在发送更多数据之前等待第1个确认。在这段时间里, 下面三个写操作发生了, 发送TCP把要发送的数据缓存了起来。因为在缓存中有更多的数据要发送, 下面的三个报文段没有包含 PUSH标志。最后, 慢启动跟上了应用程序的写操作, 每个应用程序的写操作引起了发送一个报文段, 并且因为那个报文段是缓存中最后的一个, 所以设置 PUSH标志。
- 20.3 为容量求解带宽延迟方程式, 第一种情况是 1920字节, 卫星的情况是 2062字节。看起来 TCP只声明了一个 2048字节的窗口。
- 一个大于 16000字节的窗口应该能够使卫星链路饱和。
- 20.4 不, 因为 TCP超时之后可能重新对数据进行分组, 就像我们将会在 21.11节中看到的。
- 20.5 作为应用进程读数据的结果, 第 15报文段是 TCP模块自动发送的窗口更新, 它引起窗口的打开。这类似于图中的第 9报文段。但是, 第 16报文段是应用进程关闭它这一端连接的结果。
- 20.6 这可能引起发送者以比网络实际能够处理的更快的速率向网络发送分组。这叫作确认压缩(ACK compression)或确认粉碎(ACK smashing) [Mogul 1993, 15.8.13节]。这个引用显示了在Internet上发生的ACK压缩, 尽管它很少会导致拥塞。

## 第21章

- 21.1 下一个超时设定是48秒:  $0 + 4 \times 12$ 。因子4是指数退避的下一个乘数。
- 21.2 看起来SVR4在计算RTO时仍然使用了因子2D, 而不是4D。
- 21.3 TFTP使用的停止等待协议被限制为每个往返过程传送 512字节的数据。  $32768/512 \times 1.5$  是96秒。
- 21.4 显示了4个报文段, 编号为1、2、3和4。假设接收的顺序是1、3、2和4, 接收者产生的确认将是ACK 1 (一个正常的确认), ACK 1 (当收到了报文段3, 失序后一个重复的确认), 当收到了报文段2后的ACK 3 (对报文段2和报文段3的确认), 然后是ACK 4。这儿产生了一个重复的确认。如果接收的顺序是1、3、4和2, 将会产生两个重复的确认。
- 21.5 不, 因为斜率仍然是向上和向右, 不是向下。
- 21.6 参见图E-1。
- 21.7 在图21-2中, 报文段包含256个字节的数据, 在slip和bsdi之间的9600 b/s的CSLIP链路传输大约需要250 ms。假定数据段没有在bsdi和vangogh之间的某个地方排队, 它们分别需要大约250 ms到达vangogh。因为这个时间超过了延迟确认定时器的200 ms时间, 当下一个延迟确认定时器超时, 每个报文段得到确认。

## 第22章

- 22.1 主机bsdi上的确认很可能都要被延迟，因为没有理由立即发送它们。这就是为什么相对次数有一个0.170和0.370的小数部分。看起来bsdi上200 ms的定时器在sun上同样的定时器之后18 ms才开始计时。
- 22.2 FIN标志，和SYN标志一样，在序号空间占据了1个字节。通知的窗口看起来小了一个字节，因为TCP允许FIN标志在序号空间占用1个字节的空间。

## 第23章

- 23.1 通常激活keepalive选项比显式地编写应用程序探测报文更容易；keepalive探测报文比应用程序探测报文占用更少的网络带宽（因为keepalive探测报文和应答不包含任何数据）；如果连接不是空闲的，就不会发送探测报文。
- 23.2 keepalive选项可能会由于一个临时性的网络中断而引起一个非常好的连接断开；发送探测报文的间隔（2小时）一般不可以根据应用程序进行配置。

## 第24章

- 24.1 它意味着发送TCP支持窗口扩缩选项，但这个连接并不需要扩缩它的窗口。另一端（接收这个SYN的）可以说明一个窗口扩缩因子（可以是0或非0）。
- 24.2 64 000：接收缓存大小（128 000）向右移1位。55 000：接收缓存（220 000）向右移2位。
- 24.3 不。问题是确认没有可靠地交付（除非它们被数据捎带在一起发送），因此，一个确认上的扩缩改变可能会丢失。
- 24.4  $2^{32} \times 8 / 120$  等于286 Mb/s，2.86倍于FDDI数据率。
- 24.5 每个TCP将不得不记住从每个主机的任何一个连接上收到的上一个时间戳。阅读 RFC 1323的附录B.2以了解进一步的细节。
- 24.6 应用程序必须在和另一端建立连接之前设置接收缓存的大小，因为窗口扩缩选项在初始的SYN段中发送。
- 24.7 如果接收者每次确认第2个数据报文段，吞吐量是1 118 881字节/秒。若使用一个62个报文段的窗口，每31个报文段确认一次，则数值是1 158 675。
- 24.8 使用这个选项，确认中回显的时间戳总是来自于引起确认的报文段。对哪个重传的报文段进行确认没有疑问，但仍然需要Karn算法的另一部分，即处理重传的指数退避。
- 24.9 接收TCP对数据进行排队，但只有完成了三次握手后，数据才能传递给应用程序：当接收TCP进入了ESTABLISH状态。
- 24.10 交换了5个报文段：
- 1) 客户到服务器：SYN，数据（请求）和FIN。服务器必须像上个习题所述的一样对数据进行排队。
  - 2) 服务器到客户：SYN和对客户SYN的确认。
  - 3) 客户到服务器：对服务器的SYN的确认和客户的FIN（再次）。这样使得服务器进入已建立状态，并且来自报文段1的排队数据被传递给服务器应用程序。
  - 4) 服务器到客户：客户FIN的确认（它也确认了客户的数据）、数据（服务器的应答）

和服务器的FIN。这里假定了SPT足够短以允许这个延迟的确认。当客户TCP收到这个报文段, 就将应答传递给客户端的应用程序, 但是整个时间是RTT加上SPT的两倍。

5) 客户到服务器: 对服务器FIN的确认。

24.11 每秒16 128次交互 (64 512除以4)。

24.12 使用T/TCP的交互时间不可能比两个主机之间交换一个UDP数据报所需的时间短。因为T/TCP涉及了UDP没有做的状态处理, 所以T/TCP总是要花更多的时间。

## 第25章

25.1 如果一个系统运行了一个管理进程和一个代理进程, 它们很可能是不同的进程。管理进程在UDP端口162监听trap告警, 代理进程在UDP端口161等待请求。如果trap告警和SNMP请求使用了同样的端口, 将很难区分管理进程和代理进程。

25.2 参考25.7节中的“表访问 (Table Access)”部分。

## 第26章

26.1 我们预期从服务器来的第2、4和9报文段将被延迟。第2和第4报文段之间的时间差是190.7 ms, 第2和第9报文段之间的时间差是400.7 ms。

从客户到服务器的所有确认看起来都被延迟: 第6、11、13、15、17和19报文段。从第6报文段开始的最后5个时间差是400.0、600.0、800.0、1000.0和2.600 ms。

26.2 如果连接的一个端点处于TCP的紧急模式, 每次收到一个报文段, 就会发送一个报文段。这个报文段没有告诉接收者任何新的东西 (例如, 它不是对新数据的确认), 它也不包含数据, 它只是重申进入了紧急模式。

26.3 只有512个这样的保留端口 (512~1023), 限制了一个主机只能有512个远程登录的 (Rlogin) 客户。在实际生活中, 这个限制一般小于512个, 因为在这个范围中的一些端口号被不同的服务器, 如远程登录服务器, 用作了知名端口。

TCP的限制是一对插口定义的一个连接 (4元组) 必须是唯一的。因为Rlogin服务器总是使用了同样的知名端口 (513), 一台主机上多个Rlogin客户只有在它们和不同的服务器主机相连接时才可以使用相同的保留端口。然而, Rlogin客户没有采用重用保留端口的技术。如果使用了这种技术, 理论限制就是在同一时刻最多有512个Rlogin客户和同一个的服务器主机相连。

## 第27章

27.1 当一对插口处于2MSL等待另一端状态时, 理论上不能建立连接。然而, 实际上, 在18.6节中我们看到大多数伯克利演变的实现确实为一个处于TIME\_WAIT状态的连接接受了一个新的SYN。

27.2 这些行不是以3个数字作为应答代码开始的服务器应答的一部分, 因此它们不可能来自于服务器。

## 第28章

28.1 一个域文字 (domain literal) 是在一对方括号里的点分十进制IP地址。例如: mail

rstevens@[140.252.1.54]。

- 28.2 6个来回：HELO命令、MAIL、RCPT、DATA报文主体和QUIT。
- 28.3 这是合法的，称为流水线技术 (pipelining) [Rose 1993, 4.4.4节]。不幸的是，有一些脑子坏了 (brain-damaged) 的SMTP接收者实现，每处理完一条命令就要清除它们的输入缓存，使得这种技术不可用。如果使用了这种技术，客户自然不能丢弃报文直到所有的应答都已检查过，确信报文已被服务器接受了。
- 28.4 考虑习题28.2的前5个网络上的往返。每个都是一个小命令 (很可能是只有一个报文段)，对网络几乎没有负载。如果所有5个都没有重传地送到了服务器，当报文主体发送时，拥塞窗口可能是6个报文段。如果报文主体很大，客户可能一次发送前6个报文段，造成网络可能来不及处理。
- 28.5 更新版本的BIND使用同样的值来正移MX记录，作为平衡负载的一种方式。

## 第29章

- 29.1 不，因为tcpdump不能从其他UDP数据报中区别RPC请求或应答。它只有在源端口号或目的端口号为2049时，才将UDP数据报的内容理解为NFS分组。随机的RPC请求和应答可以使用两个端点上的一个临时的端口号。
- 29.2 回忆一下1.9节中，一个进程必须有超级用户权限才能给自己分配一个小于1024的端口号 (一个知名端口)。尽管这对于系统提供的服务器没问题，如Telnet服务器、FTP服务器、和端口映射器，但我们不想给所有的RPC服务器提供这个权限。
- 29.3 这个例子中的两个概念是客户忽略那些服务器应答，如果这些应答不具有客户正在等待的XID；UDP对收到的数据报进行排队 (队列长度有一个上限)，直到应用进程读取了数据报。另外，XID不会在超时和重传时改变，它只在调用另一个服务器过程时改变。
- 客户stub执行的事件为：时刻0：发送请求1；时刻4：超时并重传请求1；时刻5：接收服务器应答1，将应答返回给应用程序；时刻5：发送请求2；时刻9：超时并重传请求2；时刻10：收到服务器的应答1，但因为我们在等待应答2，所以忽略它；时刻11：收到服务器的应答2，将应答返回给应用程序。
- 服务器的事件如下：时刻0：收到请求1，启动操作；时刻5：发送应答1；时刻5：收到请求1 (来自于客户在时刻4的重传)，启动操作；时刻10：发送应答1；时刻10：收到请求2 (来自于客户在时刻5的重传)，启动操作；时刻11：发送应答2；时刻11：收到请求2 (来自于客户在时刻9的重传)，启动操作；时刻12：发送应答2。这个最后的服务器应答仅仅被客户的UDP排队，用于客户的下一次接收操作。当客户读了这个应答时，XID将是错误的，客户将忽略它。
- 29.4 改变服务器的以太网卡就改变了它的物理地址。即使我们注意到在4.7节中SVR4没有在引导时发送一个免费ARP，它仍然必须在能够应答sun的NFS请求之前，向sun发送一个请求sun的物理地址的ARP请求。因为sun已经有了svr4的一个ARP登记项，它从这个ARP请求中根据发送者的 (新) 硬件地址更新这个登记项。
- 29.5 客户块I/O守护进程的第2个 (在偏移73728处读的) 与第1个失去同步大约0.74秒。即在行131~145，第2个守护进程在第1个之后超时0.74秒。看来服务器没有看到在167行的请求，但它看到了168行的请求。第2个块I/O守护进程只会在168行之后0.74秒才会重传。



同时, 第1个块I/O守护进程继续发送请求。

- 29.6 如果使用的是TCP, 包含着服务器应答的TCP报文段在网络中丢失了, 当服务器的TCP模块没有从客户的TCP模块收到一个确认时, 它将超时, 并重传应答。最终, 这个报文段将到达客户的TCP。这里不同的是两个TCP模块完成超时和重传, 而不是NFS客户和服务器的(当使用UDP时, NFS客户代码完成超时和重传)。因此, NFS客户并不知道应答丢失了, 需要被重传。
- 29.7 NFS服务器在重新启动之后获得一个不同的端口号是可能的。这将使客户变得很复杂, 因为它需要知道服务器崩溃了, 并且在服务器重新启动之后与服务器主机的端口映射器联系以找到NFS服务器的新的端口号。
- 这种情况, 即服务器主机崩溃然后重新启动时, 一个服务器的RPC应用程序获得一个新的临时性端口, 可能发生在任何一个没有使用知名端口的RPC应用程序上。
- 29.8 不。NFS客户可以为不同的服务器主机使用相同的本地的保留端口号。TCP要求由本地IP地址、本地端口、远端IP地址和远端端口组成的4元组是唯一的, 对于每个服务器主机来说, 远端的IP地址是不同的。

## 第30章

- 30.1 键入`whois "net 88"`。A类网络号64~95是保留的。
- 30.2 键入`whois whitehouse-do`可以使用`host`命令或者`nslookup`查询DNS。
- 30.3 不, `xscope`可以与服务器运行在不同的主机上。如果主机不同, `xscope`也可以使用TCP端口6000作为它的呼入连接。