

目录

第 1 章	引言与概述.....	1
1.1	网际互连的动机.....	1
1.2	TCP/IP 因特网.....	1
1.3	互联网的服务.....	2
1.3.1	应用层的因特网服务.....	2
1.3.2	网络层的互联网服务.....	3
1.4	因特网的历史和范围.....	3
1.5	因特网体系结构研究委员会.....	5
1.6	IAB 的重组.....	5
1.7	因特网 RFC.....	5
1.8	未来的发展和技术.....	6
1.9	本书的组织.....	6
1.10	小结.....	7
1.11	深入研究.....	7
1.12	习题.....	7
第 2 章	底层网络技术的回顾.....	8
2.1	引言.....	8
2.2	网络通信的两种途径.....	8
2.3	广域网和局域网.....	9
2.3.1	网络硬件地址.....	9
2.4	以太网技术.....	10
2.4.1	以太网容量.....	10
2.4.2	快速和吉比特以太网.....	11
2.4.3	10/100/1000 以太网.....	11
2.4.4	以太网上供电.....	11
2.4.5	以太网的性质.....	11
2.4.6	碰撞检测与恢复.....	12
2.4.7	无线网络和以太网.....	12
2.4.8	以太网硬件地址.....	13
2.4.9	以太网帧格式.....	13
2.4.10	用网桥扩展以太网.....	14
2.5	交换的以太网.....	15
2.6	异步传递方式.....	15
2.6.1	ATM 信元大小.....	15
2.6.2	面向连接的连网.....	15
2.6.3	广域点到点网络.....	16
2.6.4	拨号 IP	17
2.7	小结.....	17
2.8	深入研究.....	17
2.9	习题.....	17
第 3 章	网际互连的概念和结构模型.....	19
3.1	引言.....	19
3.2	应用层互连.....	19

3.3	网络层互连.....	19
3.4	互联网的性质.....	20
3.5	互联网结构.....	20
3.6	通过 IP 路由器互连	21
3.7	从用户的角度看.....	21
3.8	所有网络是相同的.....	22
3.9	未解答的问题.....	23
3.10	小结.....	23
3.11	深入研究.....	23
3.12	习题.....	23
第 4 章	分类的因特网地址.....	25
4.1	引言.....	25
4.2	通用标识符.....	25
4.3	最初的分类编址方案.....	25
4.4	用地址指明网络连接.....	26
4.5	网络地址和定向广播地址.....	26
4.6	受限广播.....	27
4.7	全零地址.....	27
4.8	子网和无类型扩展.....	28
4.9	IP 多播地址	28
4.10	互联网编址中的缺陷.....	28
4.11	点分十进制记法.....	29
4.12	环回地址.....	29
4.13	特殊地址约定小结.....	30
4.14	因特网编址管理机构.....	30
4.15	保留的地址前缀.....	31
4.16	示例.....	31
4.17	网络字节顺序.....	32
4.18	小结.....	33
4.19	深入研究.....	33
4.20	习题.....	33
第 5 章	因特网地址到物理地址的映射 (ARP)	34
5.1	引言.....	34
5.2	地址解析问题.....	34
5.3	两种类型的物理地址.....	34
5.4	通过直接映射进行解析.....	34
5.5	通过动态绑定进行解析.....	35
5.6	地址解析高速缓存.....	36
5.7	ARP 高速缓存超时	36
5.8	ARP 的改进.....	37
5.9	ARP 与其他协议之间的关系.....	37
5.10	ARP 的实现.....	37
5.11	ARP 的封装与标识.....	38
5.12	ARP 协议格式.....	39

5.13	ARP 缓存自动重新确认	40
5.14	逆地址解析协议.....	40
5.15	小结.....	40
5.16	深入研究.....	41
5.17	习题.....	41
第 6 章	网际协议：无连接数据报交付（IPv4）	42
6.1	引言.....	42
6.2	虚拟网络.....	42
6.3	互联网体系结构和基本原理.....	42
6.4	概念化服务的组织结构.....	42
6.5	无连接交付系统.....	43
6.6	网际协议的目的.....	43
6.7	IPv4 数据报	43
6.7.1	数据报格式.....	44
6.7.2	数据报的服务类型和区分服务	44
6.7.3	数据报的封装.....	45
6.7.4	数据报的大小、网络 MTU 及分片	46
6.7.5	数据报片的重装.....	47
6.7.6	分片控制.....	48
6.7.7	生存时间.....	48
6.7.8	其他数据报首部字段.....	49
6.8	Internet 数据报选项	49
6.8.1	记录路由选项.....	50
6.8.2	源路由选项.....	51
6.8.3	时间戳选项.....	52
6.8.4	分片时对选项的处理.....	52
6.9	小结.....	53
6.10	深入研究.....	53
6.11	习题.....	53
第 7 章	网际协议：转发 IP 数据报	55
7.1	引言.....	55
7.2	互联网中的转发.....	55
7.3	直接和间接交付	56
7.3.1	单个物理网络上的数据报交付	56
7.3.2	间接交付.....	57
7.4	表驱动 IP 转发	57
7.5	下一跳转发.....	58
7.6	默认路由.....	59
7.7	特定主机的路由.....	59
7.8	IP 转发算法	59
7.9	利用 IP 地址转发	60
7.10	处理传入的数据报.....	61
7.11	建立路由表.....	62
7.12	小结.....	62

7.13	深入研究.....	62
7.14	习题.....	62
第 8 章	网际协议：差错与控制报文（ICMP）	64
8.1	引言.....	64
8.2	网际控制报文协议.....	64
8.3	差错报告与差错纠正的对比.....	65
8.4	ICMP 报文交付	65
8.5	ICMP 报文格式	66
8.6	测试目的站的可达性和状态.....	66
8.7	回送请求和回答报文的格式.....	67
8.8	目的站不可达的报告.....	67
8.9	拥塞和数据报流量控制.....	68
8.10	源站抑制的格式.....	69
8.11	来自路由器的改变路由请求.....	69
8.12	检测循环或过长的路由.....	71
8.13	报告其他问题.....	71
8.14	时钟同步和传输时间估计.....	72
8.15	不再需要的较早 ICMP 报文	72
8.16	小结.....	73
8.17	深入研究.....	73
8.18	习题.....	73
第 9 章	无分类和子网地址扩展（CIDR）	75
9.1	引言.....	75
9.2	相关情况的回顾.....	75
9.3	使网络数量最少.....	75
9.4	代理 ARP.....	76
9.5	子网编址.....	77
9.6	子网地址分配的灵活性.....	78
9.7	变长子网.....	79
9.8	带掩码的子网的实现.....	80
9.9	子网掩码的表示.....	80
9.10	子网存在时的转发.....	81
9.11	子网转发算法.....	81
9.12	统一的转发算法.....	82
9.13	子网掩码的维护.....	83
9.14	广播到子网.....	83
9.15	匿名的点对点网络.....	84
9.16	无分类编址和构造超网.....	85
9.17	CIDR 地址块和比特掩码	85
9.18	地址块和 CIDR 记法	86
9.19	无分类编址举例.....	86
9.20	无分类查找所用的数据结构和算法	87
9.20.1	按掩码长度进行搜索.....	87
9.20.2	二叉线索结构.....	87

9.21	最长匹配转发和混合路由类型	88
9.21.1	PATRICIA 和层压缩线索	89
9.22	为专用网络保留的 CIDR 块	89
9.23	小结	90
9.24	深入研究	90
9.25	习题	90
第 10 章	协议的分层	92
10.1	引言	92
10.2	对多个协议的需求	92
10.3	协议软件的概念分层	93
10.4	各层的功能	94
10.4.1	ISO 七层参考模型	94
10.5	X.25 及其与 ISO 模型的关系	95
10.5.1	TCP/IP 的五层参考模型	96
10.6	智能的位置	97
10.7	协议分层的原理	98
10.7.1	TCP/IP 互联网环境中的分层	98
10.8	存在网络子结构时的分层	99
10.9	TCP/IP 模型中的两条重要分界线	100
10.9.1	高级协议地址分界线	100
10.9.2	操作系统分界线	101
10.10	分层的缺点	101
10.11	复用和分用的基本思想	101
10.12	小结	102
10.13	深入研究	103
10.14	习题	103
第 11 章	用户数据报协议（UDP）	104
11.1	引言	104
11.2	识别最终目的地	104
11.3	用户数据报协议	104
11.4	UDP 报文的格式	105
11.5	UDP 伪首部	106
11.6	UDP 的封装与协议的分层	106
11.7	分层及 UDP 检验和的计算	107
11.8	UDP 的复用、分用和端口	108
11.9	保留和可用的 UDP 端口号	109
11.10	小结	110
11.11	深入研究	110
11.12	习题	110
第 12 章	可靠的流运输服务（TCP）	112
12.1	引言	112
12.2	对流交付的需求	112
12.3	可靠交付服务的特征	112
12.4	提供可靠性	113

12.5	滑动窗口的概念.....	114
12.6	传输控制协议.....	116
12.7	端口、连接与端点.....	116
12.8	被动打开和主动打开.....	118
12.9	报文段、流和序号.....	118
12.10	可变的窗口大小与流量控制.....	119
12.11	TCP 报文段的格式.....	119
12.12	带外数据.....	120
12.13	TCP 选项.....	121
12.13.1	最大报文段长度的选项.....	121
12.13.2	窗口扩大比例选项.....	122
12.13.3	时间戳选项.....	122
12.14	TCP 检验和的计算.....	122
12.15	确认、重传和超时.....	123
12.16	往返时间样本的精确测量.....	125
12.17	Karn 算法与计时器补偿.....	125
12.18	对高方差时延的响应.....	126
12.19	对拥塞的响应.....	128
12.20	快恢复和其他改进.....	129
12.21	显式反馈机制: SACK 和 ECN	130
12.21.1	选择确认.....	130
12.21.2	显式拥塞通知.....	130
12.22	拥塞、尾部丢弃和 TCP.....	131
12.23	随机早期检测.....	131
12.24	建立一个 TCP 连接.....	133
12.25	初始序号.....	133
12.26	关闭一个 TCP 连接.....	134
12.27	TCP 连接的复位	135
12.28	TCP 状态机	135
12.29	强迫数据交付.....	136
12.30	保留的 TCP 端口号	136
12.31	TCP 的性能	137
12.32	糊涂窗口综合症与小分组.....	137
12.33	避免糊涂窗口综合症.....	138
12.33.1	接收方对糊涂窗口的避免.....	138
12.33.2	推迟确认.....	139
12.33.3	发送方对糊涂窗口的避免	139
12.34	小结.....	140
12.35	深入研究.....	140
12.36	习题.....	141
第 13 章	路由选择: 核心网络、对等网络与算法.....	143
13.1	引言.....	143
13.2	路由表的产生.....	143
13.3	利用部分信息转发.....	144

13.4	最初的因特网体系结构与核心.....	145
13.5	从核心结构到对等主干网结构.....	146
13.6	自动路由传播.....	147
13.7	距离向量 (Bellman-Ford) 路由选择.....	147
13.8	可靠性和路由选择协议.....	148
13.9	链路状态 (SPF) 路由选择	148
13.10	小结.....	149
13.11	深入研究.....	150
13.12	习题.....	150
第 14 章	对等网络间的路由选择 (BGP)	151
14.1	引言.....	151
14.2	路由更新协议的范围.....	151
14.3	确定组大小的实际限度.....	151
14.4	一个基本思想：额外跳.....	152
14.5	自治系统的概念.....	153
14.6	外部网关协议和可达性.....	153
14.7	BGP 的特征.....	154
14.8	BGP 的功能和报文类型	155
14.9	BGP 报文首部.....	155
14.10	BGP OPEN 报文.....	156
14.11	BGP UPDATE 报文.....	157
14.12	压缩的掩码地址对.....	158
14.13	BGP 路径属性.....	158
14.14	BGP KEEPALIVE 报文.....	159
14.15	从接收方的角度来看信息.....	160
14.16	外部网关协议的关键约束.....	160
14.17	因特网的路由选择体系结构.....	161
14.18	BGP NOTIFICATION 报文	162
14.19	小结.....	163
14.20	深入研究.....	163
14.21	习题.....	164
第 15 章	自治系统内的路由选择 (RIP 和 OSPF)	165
15.1	引言.....	165
15.2	静态内部路由与动态内部路由的对比.....	165
15.3	路由信息协议.....	167
15.3.1	RIP 的发展史.....	167
15.3.2	RIP 操作.....	167
15.4	慢收敛问题.....	168
15.5	解决慢收敛问题.....	169
15.6	RIP1 报文格式.....	170
15.7	RIP2 地址约定.....	171
15.8	RIP 路由的解释和聚合.....	171
15.9	RIP2 扩展和报文格式.....	171
15.10	RIP 跳计数的缺点.....	172

15.11	时延度量 (HELLO 协议)	173
15.12	时延度量和振荡.....	173
15.13	将 RIP 和 HELLO 以及 BGP 组合起来.....	174
15.14	gated: 自治系统之间的通信.....	174
15.15	开放 SPF 协议	175
15.15.1	OSPF 的报文格式	176
15.15.2	OSPF 的 HELLO 报文格式	177
15.15.3	OSPF 的数据库描述报文格式	177
15.15.4	OSPF 的链路状态请求报文格式	178
15.15.5	OSPF 的链路状态更新报文格式	178
15.16	利用部分信息选择路由.....	179
15.17	小结.....	180
15.18	深入研究.....	180
15.19	习题.....	180
第 16 章	因特网多播.....	182
16.1	引言.....	182
16.2	硬件广播.....	182
16.3	多播的硬件起源.....	182
16.4	以太网多播.....	183
16.5	IP 多播	183
16.6	概念性组成部分	184
16.7	IP 多播地址	184
16.8	多播地址语义.....	185
16.9	把 IP 多播映射到以太网多播	185
16.10	主机和多播交付	186
16.11	多播作用域.....	186
16.12	扩展主机软件以处理多播.....	187
16.13	网际组管理协议.....	187
16.14	IGMP 的实现.....	188
16.15	群组成员状态的转换.....	189
16.16	IGMP 成员关系查询报文格式.....	189
16.17	IGMP 成员关系报告报文格式.....	191
16.18	多播转发和路由选择信息	192
16.18.1	对动态路由选择的需求	192
16.18.2	目的地址转发的不足之处	192
16.18.3	任意的发送方	192
16.19	基本的多播转发范例.....	193
16.20	TRPF 的后果	193
16.21	多播树	195
16.22	多播路由选择的实质	195
16.23	反向路径多播	196
16.24	多播路由选择协议	196
16.24.1	距离向量多播路由选择协议	197
16.24.2	核心基干树	197

16.24.3	协议无关多播.....	198
16.24.4	OSPF 的多播扩展	198
16.25	可靠多播和 ACK 内爆	198
16.26	小结.....	199
16.27	深入研究.....	200
16.28	习题.....	200
第 17 章	IP 交换和 MPLS.....	202
17.1	引言.....	202
17.2	交换技术.....	202
17.3	大规模网络、标记交换和路径.....	202
17.4	IP 的交换应用	203
17.5	IP 交换技术和 MPLS.....	204
17.6	分类、流和高层交换.....	204
17.7	MPLS 的分层应用	204
17.8	MPLS 封装	205
17.9	标记交换路由器.....	206
17.10	控制过程和标记分发.....	206
17.11	MPLS 和分片	207
17.12	网孔拓扑和流量工程.....	207
17.13	小结.....	208
17.14	深入研究.....	208
17.15	习题.....	208
第 18 章	移动 IP	209
18.1	引言.....	209
18.2	移动性、路由选择和编址.....	209
18.3	移动 IP 特性	209
18.4	移动 IP 操作概述	209
18.5	移动编址细节.....	210
18.6	外地代理的发现.....	210
18.7	代理登记.....	211
18.8	登记报文格式.....	211
18.9	与外地代理之间的通信.....	212
18.10	数据报的传输和接收.....	212
18.11	两次穿越问题.....	213
18.12	与归属网上的计算机之间的通信.....	214
18.13	小结.....	214
18.14	深入研究.....	214
18.15	习题.....	214
第 19 章	专用网络互连（NAT 和 VPN）	216
19.1	引言.....	216
19.2	专用的和混合的网络.....	216
19.3	VPN 编址技术和路由选择.....	217
19.4	扩展 VPN 技术应用到个人主机.....	218
19.5	使用专用地址的 VPN.....	218

19.6	网络地址转换。	219
19.7	NAT 转换表的创建.....	219
19.8	多地址 NAT	220
19.9	端口映射 NAT	220
19.10	NAT 和 ICMP 之间的交互	221
19.11	NAT 和应用之间的交互.....	221
19.12	数据报分片情况下的 NAT	222
19.13	概念性地址域.....	222
19.14	slirp 和 iptables 程序	222
19.15	小结.....	223
19.16	深入研究.....	223
19.17	习题.....	223
第 20 章	客户—服务器交互模型.....	225
20.1	引言.....	225
20.2	客户—服务器模型.....	225
20.3	一个简单的例子： UDP 回送服务器.....	225
20.4	时间和日期服务	226
20.4.1	日期和时间的表示.....	226
20.4.2	时间服务器的交互.....	227
20.5	服务器的复杂性.....	227
20.6	广播一个请求.....	228
20.7	客户—服务器模型的替代方案.....	228
20.8	小结.....	228
20.9	深入研究.....	229
20.10	习题.....	229
第 21 章	套接字接口.....	230
21.1	引言.....	230
21.2	UNIX I/O 范式与网络 I/O	230
21.3	在 UNIX 中添加网络 I/O.....	230
21.4	套接字的抽象化.....	231
21.5	创建一个套接字.....	231
21.6	套接字的继承与终止.....	231
21.7	指明本地地址.....	232
21.8	将套接字连接到目的地址.....	232
21.9	通过套接字发送数据.....	233
21.10	通过套接字接收数据.....	234
21.11	获取本地的和远程的套接字地址.....	235
21.12	获取和设置套接字选项.....	235
21.13	指明服务器的队列长度.....	236
21.14	服务器如何接受连接.....	236
21.15	处理多重服务的服务器.....	237
21.16	获取与设置主机名字.....	237
21.17	获取与设置内部主地域.....	237
21.18	套接字库的调用.....	238

21.19	网络字节顺序转换例程.....	238
21.20	IP 地址操作例程	239
21.21	访问域名系统.....	240
21.22	获取主机信息.....	241
21.23	获取网络的信息.....	241
21.24	获取协议信息.....	241
21.25	获取网络服务信息.....	242
21.26	客户举例.....	242
21.27	服务器举例.....	245
21.28	小结.....	248
21.29	深入研究.....	248
21.30	习题.....	248
第 22 章	启动引导与自动配置 (DHCP)	250
22.1	引言.....	250
22.2	启动引导的发展历程.....	250
22.3	用 IP 来确定 IP 地址.....	250
22.4	DHCP 的重传策略.....	251
22.5	DHCP 报文格式.....	251
22.6	动态配置的必要性.....	253
22.7	DHCP 的租用概念.....	253
22.8	多个地址和中继.....	254
22.9	地址获取状态.....	254
22.10	提前终止租期.....	255
22.11	租用续租状态.....	255
22.12	DHCP 选项与报文类型.....	256
22.13	选项过载.....	257
22.14	DHCP 与域名.....	257
22.15	小结.....	257
22.16	深入研究.....	257
22.17	习题.....	258
第 23 章	域名系统 (DNS)	259
23.1	引言.....	259
23.2	机器的名字.....	259
23.3	平面名字空间.....	259
23.4	分级的名字.....	260
23.5	名字的授权管理.....	260
23.6	子集管理机构.....	261
23.7	因特网的域名.....	261
23.8	顶级域.....	262
23.9	名字的语法和类型.....	263
23.10	从域名映射到地址.....	264
23.11	域名解析.....	265
23.12	高效率的转换.....	266
23.13	高速缓存：高效率的关键.....	266

23.14	域名系统的报文格式.....	267
23.15	压缩的名字格式.....	269
23.16	域名的缩写.....	269
23.17	反向映射.....	270
23.18	指针查询.....	270
23.19	对象类型与资源记录的内容.....	271
23.20	获得子域的管理权.....	271
23.21	动态 DNS 更新和通知.....	272
23.22	DNS 安全扩展.....	272
23.23	小结.....	273
23.24	深入研究.....	273
23.25	习题.....	273
第 24 章	远程登录和桌面（TELNET 和 SSH）.....	275
24.1	引言.....	275
24.2	远程交互式计算.....	275
24.3	TELNET 协议.....	275
24.4	适应异构性.....	276
24.5	传递用于控制远端的命令.....	277
24.6	强制服务器读取一个控制功能.....	279
24.7	TELNET 选项.....	279
24.8	TELNET 选项的协商.....	280
24.9	安全外壳.....	280
24.10	其他远程访问技术.....	281
24.11	小结.....	282
24.12	深入研究.....	282
24.13	习题.....	282
第 25 章	文件传送与存取（FTP, TFTP 和 NFS）.....	284
25.1	引言.....	284
25.2	远程文件存取、传送和存储网络.....	284
25.3	在线共享存取.....	284
25.4	通过文件传送的共享.....	285
25.5	主要的 TCP/IP 文件传送协议 <u>FTP</u>	285
25.6	FTP 的特点.....	285
25.7	FTP 进程模型.....	285
25.8	TCP 端口号和数据连接.....	286
25.9	从用户的角度看 <u>FTP</u>	287
25.10	匿名 <u>FTP</u>	287
25.11	安全文件传送（SSL-FTP, scp 和 sftp）.....	287
25.12	<u>TFTP</u>	288
25.13	NFS	289
25.14	NFS 的实现（RPC 和 XDR）.....	290
25.15	小结.....	290
25.16	深入研究.....	291
25.17	习题.....	291

第 26 章	电子邮件 (SMTP, POP, IMAP 和 MIME)	292
26.1	引言.....	292
26.2	电子邮件.....	292
26.3	邮箱名字和别名.....	293
26.4	别名扩展与邮件转发.....	293
26.5	电子邮件服务的 TCP/IP 标准.....	294
26.6	简单邮件传送协议.....	294
26.7	邮件取回和邮箱操纵协议.....	296
26.7.1	邮局协议.....	296
26.7.2	网际报文访问协议.....	297
26.8	非 ASCII 码数据的 MIME 扩充.....	297
26.9	MIME 多部分报文.....	298
26.10	小结.....	299
26.11	深入研究.....	299
26.12	习题.....	299
第 27 章	万维网 (HTTP)	301
27.1	引言.....	301
27.2	万维网的重要性.....	301
27.3	体系结构组成.....	301
27.4	统一资源定位符.....	301
27.5	示例文档.....	302
27.6	超文本传送协议.....	302
27.7	HTTP 的 GET 请求.....	303
27.8	错误消息.....	303
27.9	持久连接和长度.....	304
27.10	数据长度和程序输出.....	304
27.11	长度编码和首部.....	304
27.12	协商.....	305
27.13	条件请求.....	306
27.14	代理服务器和缓存.....	306
27.15	缓存.....	307
27.16	其他 HTTP 功能.....	307
27.17	HTTP、安全性和电子商务.....	307
27.18	小结.....	308
27.19	深入研究.....	308
27.20	习题.....	308
第 28 章	IP 上的话音和视频传输 (RTP, RSVP 和 QoS)	310
28.1	引言.....	310
28.2	数字化和编码技术.....	310
28.3	音频和视频的传输及再现.....	310
28.4	抖动和回放延迟.....	311
28.5	实时运输协议.....	311
28.6	数据流、混合和多播.....	312
28.7	RTP 封装.....	313

28.8	RTP 控制协议.....	313
28.9	RTCP 操作.....	313
28.10	IP 电话和信令	314
28.10.1	H.323 标准.....	315
28.10.2	会话发起协议.....	316
28.11	服务质量之争.....	316
28.12	QoS 和利用率以及容量.....	316
28.13	综合业务资源预留.....	317
28.14	综合服务执行.....	317
28.15	区分服务和每跳行为.....	318
28.16	通信量调度.....	318
28.17	通信量管制.....	319
28.18	小结.....	319
28.19	深入研究.....	320
28.20	习题.....	320
第 29 章	网络管理 (SNMP)	321
29.1	引言.....	321
29.2	管理协议的级别.....	321
29.3	体系结构模型.....	322
29.4	协议框架结构.....	322
29.4.1	一个标准网络管理协议.....	323
29.4.2	管理信息的标准.....	323
29.5	MIB 变量的例子.....	323
29.6	管理信息的结构.....	324
29.7	使用 ASN.1 的正式定义.....	324
29.8	MIB 对象名的结构和表示法	325
29.9	简单网络管理协议.....	328
29.9.1	用名字搜索表.....	329
29.10	SNMP 的报文格式.....	330
29.11	SNMP 报文编码举例.....	332
29.12	SNMPv3 的新特性.....	333
29.13	小结.....	333
29.14	深入研究.....	333
29.15	习题.....	333
第 30 章	互联网的安全性和防火墙设计 (IPsec 和 SSL)	335
30.1	引言.....	335
30.2	对资源的保护.....	335
30.3	信息策略.....	336
30.4	互联网安全.....	336
30.5	IP 安全 (IPsec)	336
30.6	IPsec 鉴别首部	337
30.7	安全关联.....	338
30.8	IPsec 封装安全有效载荷	338
30.9	鉴别和可变首部字段.....	339

30.10	IPsec 隧道	339
30.11	必要的安全算法.....	340
30.12	安全套接字.....	340
30.13	防火墙和互联网的访问.....	340
30.14	多重连接和最薄弱的链接.....	341
30.15	防火墙的实现和分组过滤器.....	341
30.16	安全性与分组过滤器规则.....	342
30.17	客户受限访问的后果.....	342
30.18	有状态防火墙.....	343
30.19	内容保护和代理.....	343
30.20	监视与日志.....	344
30.21	小结.....	344
30.22	深入研究.....	344
30.23	习题.....	345
第 31 章	下一代 IP (IPv6)	346
31.1	引言.....	346
31.2	为什么要改变.....	346
31.3	超越 IPv4	346
31.4	通向新版 IP 之路	346
31.5	下一代 IP 的名字	347
31.6	IPv6 的特点	347
31.7	IPv6 数据报的一般形式	347
31.8	IPv6 基本首部格式	348
31.9	IPv6 的扩展首部	349
31.10	IPv6 数据报的解析	349
31.11	IPv6 的分片和重装	350
31.12	端到端分片的后果	350
31.13	IPv6 源路由	351
31.14	IPv6 的选项	351
31.15	IPv6 地址空间的大小	352
31.16	IPv6 的冒号十六进制记法	353
31.17	三种基本 IPv6 地址类型	353
31.18	广播和多播的二重性	354
31.19	模拟广播——工程上的选择	354
31.20	建议的 IPv6 地址空间分配	354
31.21	嵌入的 IPv4 地址和过渡	355
31.22	未指明的地址和环回地址	356
31.23	单播地址的结构	356
31.24	接口标识符	356
31.25	本地地址	357
31.26	自动配置和重新编号	357
31.27	小结	358
31.28	深入研究	358
31.29	习题	359

第1章 引言与概述

1.1 网际互连的动机

因特网 (Internet) 通信目前已成为人们生活中不可缺少的一部分。**万维网** (World Wide Web) 可以提供诸如气象情况、农作物生产、股票价格和航班交通等非常多样化的信息。一些团体建立了电子邮件列表，以便分享他们共同感兴趣的信息。专业同行们通过电子方式交换业务往来的信件，亲友之间也通过这种方式互致个人的问候。

虽然因特网看起来像是一个统一的网络在运行着，但它并不是采用单一的连网技术构建的，因为没有哪种技术是万能的。实际上，在设计连网所用的硬件时，人们往往要考虑特定的使用场合和经费预算。有些人需要使用高速网络连接同一栋建筑物内的计算机，他们可以选择一些价格便宜的硬件。但是，由于这些廉价的硬件虽能在一栋建筑物内工作得很好，却不能在地理跨度很大的区域内工作，因此人们就必须选择其他的硬件来连接相隔数千英里^①远的机器。

在过去的 25 年里，已经创造了一项技术，使许多不同物理网络的互连成为可能，并使这些网络就像一个协调的整体在运行着。这种技术称为**网际互连** (internetworking)，它适用于多种不同的底层网络技术，提供了一种网络互连的方法，并定义了一组网络互操作所用的通信约定，从而成为因特网形成的基础。**互联网** (internet) 技术隐藏了网络硬件的细节，允许计算机之间的通信与底层物理网络连接无关。

本书描述的互联网技术是一个**开放系统互连** (open system interconnection) 的例子。之所以称为开放系统，是因为它不像某个厂家私有的通信系统，其**规范** (specification) 是可以公开得到的。因此，任何人都能自己构造互联网通信所需的软件。更重要的是，整个技术的设计能够让不同硬件体系结构的机器相互通信，能够使用几乎所有的分组交换网络硬件，能够适用于各种不同的应用程序，并可适用于多种计算机操作系统。

1.2 TCP/IP 因特网

美国政府机构许多年前就认识到了互联网技术的重要性和潜力，并提供了研究基金，使因特网全球化成为可能^②。本书论述的一些原理和观点就是美国**国防部远景规划局** (Defense Advanced Research Projects Agency，简称 DARPA)^③资助研究的成果。DARPA 技术包括一组说明计算机通信细节的网络标准，以及一组关于网络互连和通信量路由的约定。它的正式名称是**TCP/IP 网际协议族** (TCP/IP Internet Protocol Suite)，一般称为 TCP/IP (该名称来自于它的两个主要标准)，可用于任何一组相互连接的网络之间的通信。例如，TCP/IP 可用于一栋建筑内多个网络的互连，也可用于一个校园范围内多个网络的互连，还可用于多个校园网的网络互连。

尽管 TCP/IP 技术本身非常令人瞩目，但人们真正关注的是它所呈现的巨大生命力。TCP/IP 是构建全球因特网的基本技术，几乎在所有人类居住区，都有因特网将家庭、学校、公司和国家政府实验室中的用户（达 6.5 亿）连接起来。因特网的一个突出成就在于它显示了 TCP/IP 技术的生命

^① 1 英里=1.6093km——编者注。

^② 我们将按照一般的习惯，当特指全球因特网时，用大写的 I，即 Internet，当提到使用 TCP/IP 的专用互联网时，用小写的 i，即 internet。

^③ DARPA 又称为远景规划局 (Advanced Research Projects Agency，简称 ARPA)。

力，并展示了如何包容多种多样的底层硬件技术。

1.3 互联网的服务

如果不了解 TCP/IP 提供的服务，就很难理解它所蕴含的技术细节。本节简要论述互联网服务，重点放在大多数用户访问的服务上面，后面的章节将再讨论计算机如何连接到一个 TCP/IP 互联网以及如何实现此功能。

我们对服务的讨论重点放在称为**协议** (protocol) 的标准上。诸如 TCP 和 IP 的协议提供了通信的语法和语义规则。协议详细说明了报文的格式，描述了当报文到达时计算机如何响应，指明计算机如何处理错误或其他异常情况。更为重要的是，通过使用协议，就能够不考虑厂家的网络硬件而讨论计算机通信问题。从某种意义上说，协议对于通信就像算法对于计算一样。通过使用算法，不必知道特定 CPU 指令集就能指定或理解计算过程；同样，通过使用通信协议，不依赖于特定厂家的网络硬件就能指定或理解数据通信。

隐藏通信的底层细节从以下几方面有助于提高生产效率。第一，程序员是与较高层协议抽象打交道的，他们不必学习或记住诸如硬件配置之类的细节问题，因此能快速编写新的程序；第二，因为用更高一级抽象编写的程序不受机器结构或网络硬件的限制，所以当重新配置机器或网络时也不必修改它们；第三，因为使用更高一级协议编写的程序是独立于底层硬件的，所以它们能为任意一对机器提供直接通信。程序员不必为每种计算机类型或每种网络类型建立特殊的应用程序软件版本。事实上，程序员编写的那些使用协议的软件是通用的，同样的代码可以在任意计算机上编译和运行。

我们将看到，因特网上所有可用的服务都是用一个协议给出的。下一节将描述指定应用层服务的协议和定义网络层服务的协议。后面的章节将更详细地讲解每个协议。

1.3.1 应用层的因特网服务

从用户的角度来看，因特网像是由一些应用程序集组成的，这些程序使用底层的网络来执行有用的通信任务。我们用术语**互操作性** (interoperability) 表示不同计算系统相互协作来解决计算问题的能力。因特网应用程序表现出很高程度的互操作性。大多数访问因特网的用户仅仅是在运行一些应用程序，而并不了解所访问的计算机的类型、TCP/IP 技术、底层互联网的基本结构，甚至不了解数据传输到目的站的路径；他们依靠应用程序和底层的网络软件来处理这些细节。只有编写网络应用程序的程序员才需要把 TCP/IP 互联网看成一个网络，并且需要了解网络的一些技术。

最为流行并且广泛传播的因特网应用服务包括：

- **万维网** (World Wide Web)。通过 Web，用户能够查看包含文本和图形的文档，并沿着超链接从一个文档转到另一个文档。从 1994 年到 1995 年，Web 已逐渐成长为因特网上最大的通信量来源，并继续成为一个主要的通信量来源。
- **电子邮件** (Electronic Mail 或 E-mail)。使用电子邮件，用户可以书写函件并将一个副本发送给其他个人或群组。邮件应用程序的另一部分则允许用户阅读其收到的函件。用户可以在一个电子邮件的邮件报文里包含由任意文件组成的“附件”。电子邮件发展得非常成功，以至于许多因特网用户大多依靠其进行通信联络。因特网电子邮件之所以流行的一个原因是，协议使得交付十分可靠。发送端服务器上的邮件系统与接收端计算机上的邮件系统会直接进行联系，协议还规定只有当接收者把邮件副本保存在永久存储中以后，发送者才能删除邮件。
- **文件传送** (File Transfer)。文件传送应用程序允许用户发送或接收数据文件的副本。文件传送是因特网上的一种最古老但又最常用的应用服务。
- **远程登录** (Remote Login) 和**远程桌面** (Remote Desktop)。远程登录和远程桌面服务允许

用户坐在一台与远端机器连接的计算机面前，并像使用本地机器那样来使用远端机器。这就是说，用户键盘上的每次击键都会发送到远端机器，并且远端计算机的屏幕输出会显示在用户的屏幕上（可以显示在一个窗口中，或者显示在整个屏幕上）。

后面几章会详细讨论这些应用程序以及其他应用程序。我们将仔细探讨应用程序如何使用下层的 TCP/IP 协议，以及这些应用程序协议的标准为何有助于确保它们得以广泛应用。

1.3.2 网络层的互联网服务

使用 TCP/IP 协议创建应用程序的程序员，与那些仅仅使用电子邮件之类的应用程序的用户相比。对互联网有着完全不同的观点。在网络层，互联网提供给所有应用程序使用的服务有两大类型。尽管目前理解细节并不重要，但在关于 TCP/IP 的概述中却不能忽略这些服务：

- **无连接分组交付服务** (Connectionless Packet Delivery Service)。我们将在本文中详细解释这种服务，它形成所有互联网服务的基础。无连接交付是大多数分组交换网络提供服务的一种抽象。简单地讲，指的是 TCP/IP 互联网根据报文携带的地址信息把短报文从一台机器传递到另一台机器上。因为无连接服务单独传递每个分组，所以不能保证可靠、有序地传递。由于无连接服务通常直接映射到底层的硬件上，所以它的效率非常高。更重要的是，无连接分组交换作为所有互联网服务的基础，可使 TCP/IP 协议适应多种网络硬件。
- **可靠的数据流运输服务** (Reliable Stream Transport Service)。大多数应用程序要求得到比分组交换更多的服务，因为它们要求通信软件能从传输错误、分组丢失或发送者与接收者之间路径上的交换机的故障中自动恢复过来。可靠的运输服务处理了这些问题。它允许一台计算机上的应用程序与另一台计算机上的应用程序之间建立一个“连接”，然后通过该连接发送大量数据，就好像这是一个永久的直接硬件连接。当然，在底层，通信协议把数据流分成一个个小报文来发送，一次一个，同时等待接收者发送确认接收的信息。

许多网络都提供了与上述服务类似的基本服务，所以人们可能奇怪 TCP/IP 与其他网络相比究竟有什么区别。最主要的区别特征是：

- **网络技术独立性** (Network Technology Independence)。尽管 TCP/IP 基于传统的分组交换技术，但它独立于任何品牌或类型的硬件；全球因特网包括许多种网络技术。TCP/IP 协议定义了数据传输的单元，称为**数据报** (datagram)，并指明了如何在一个特定的网络上传输数据报。
- **普遍互连** (Universal Interconnection)。TCP/IP 互联网允许连接到它的任何一对计算机之间彼此进行通信。每台计算机都被分配了一个互联网内普遍可识别的地址 (address)。每个数据报都带有源地址和目的地址。中间的交换设备使用目的地址来选择路由。
- **端到端确认** (End-to-End Acknowledgement)。TCP/IP 互联网协议提供的是最初的源站和最终目的站之间的确认，而不是路径上相继网点之间的确认，即使源主机和目的主机并没有连接到同一个物理网络上。
- **应用协议标准** (Application Protocol Standard)。除了基本的运输层服务（如可靠的数据流连接）以外，TCP/IP 协议也包含了用于许多常用应用程序的标准，这些程序包括电子邮件、文件传送和远程登录。因此，在设计使用 TCP/IP 的应用程序时，程序员常会发现已有一些现成的软件提供了他们所需的通信服务。

后面的章节将详细讨论给程序员提供的服务以及多种应用程序协议标准。

1.4 因特网的历史和范围

TCP/IP 技术如此激动人心的部分原因是，它被广泛采用以及全球因特网的规模和增长速度。DARPA 在 20 世纪 70 年代中期开始了互联网技术方向的研究，而它的体系结构和协议在 1977 年到

1979 年左右就开始呈现出当今的形式。在那时，DARPA 是分组交换网络研究的主要资助机构，主导提出了许多分组交换的思想，其中包括其著名的 ARPANET。ARPANET 使用传统的点到点租用线路互连，但 DARPA 也资助了在无线网络和卫星通信信道上分组交换的研究。事实上，网络硬件技术的多样化也促使 DARPA 开始网络互连的研究，并推动了网际互连的发展。

从 DARPA 那里可以得到研究资金吸引了许多研究组织的注意，特别是以前曾在 ARPANET 上使用过分组交换的研究人员。DARPA 组织研究人员召开会议来分享观点和讨论实验结果。这个组织有一个非正式名称是**互联网研究组织**（Internet Research Group）。到 1979 年，很多研究人员都致力于 TCP/IP 的研究，所以 DARPA 组建了一个非正式的委员会，以协调和指导新兴的因特网的协议及体系结构的设计。这个组织被称为**因特网控制和配置委员会**（Internet Control and Configuration Board，简称 ICCB），该组织定期会晤，直到 1983 年被重组为止。

全球因特网大约于 1980 年开始形成，当时 DARPA 开始把挂接到其研究网络上的机器进行改动，让它们使用新的 TCP/IP 协议。这使得已经存在的 ARPANET 迅速成为新的因特网主干，并且在许多早期 TCP/IP 的实验中使用。直到 1983 年 1 月，当时国防部长办公室要求所有连到远程网的计算机都使用 TCP/IP，这才彻底完成了到因特网技术的转变。同时，**国防通信局**（Defense Communication Agency，简称 DCA）把 ARPANET 拆分成两个独立的网络，一个用于进一步研究，另一个用于军用通信。用于研究的部分保持 ARPANET 的名字，用于军用的部分稍微大一些，变成**军用网**（military network），即 MILNET。

为鼓励大学的研究人员采纳和使用新的协议，DARPA 把实现价格压到最低限度。那时，许多大学的计算机科学系都使用加利福尼亚大学**伯克利软件发布中心**（Berkeley Software Distribution）的一个 UNIX 的版本，该软件通常被称为 Berkeley UNIX 或 BSD UNIX。DARPA 通过资助 BBN(Bolt Beranek and Newman) 股份有限公司用 UNIX 来实现 TCP/IP 协议，以及给加利福尼亚大学伯克利分校投资，将 TCP/IP 协议集成到它的软件发布版本中，DARPA 的触角得以延伸到超过 90% 的大学计算机科学系。新协议软件的发布时机非常有意义，当时恰逢许多系正要更换第二代或第三代计算机，并要用局域网把它们连接起来。这些部门需要一些提供诸如文件传送之类应用服务的通信协议。

除了一套实用工具程序外，Berkeley UNIX 还提供了一种新的操作系统抽象，称为**套接字**（socket），允许应用程序使用通信协议。在 UNIX 的一般 I/O 机制中，套接字接口除了可以选择 TCP/IP 协议以外，还可以选择其他几种类型的网络协议。套接字抽象的引入很有意义，它使程序员很容易使用 TCP/IP 协议。套接字接口目前已在大多数操作系统中使用，成为一个事实标准。

当**美国国家科学基金会**（National Science Foundation，简称 NSF）认识到网络通信将很快成为科学研究的关键环节后，它在扩展 TCP/IP 因特网，使其尽可能多地为科学家所用的过程中扮演了积极的角色。在 20 世纪 70 年代末，基金会筹建了**计算机科学网**（Computer Science NETwork，简称 CSNET）计划，目标是把所有计算机科学家连接起来。从 1985 年开始，它就规划围绕着它的 6 个超级计算机中心，建立接入网络。1986 年，该基金会继续扩展连网行动，资助建立了一个新的广域主干网，称为 NSFNET 主干网。NSF 还为一些区域性网络提供种子资金，每个网络都在特定区域连接到主要的科学研究所机构。

在最初的 7 年里，因特网已成长为连接几百个位于美国和欧洲的网络的互联网。它连接着大学、政府和公司研究实验室里的近两万台计算机。因特网的规模及其用途都在继续快速增长，其速度比预想的要快得多。到 1987 年末，每月的增长率据估计已达到 15%。到 2005 年，全球因特网已连接了 209 个国家的近 3 亿台计算机。

早期 TCP/IP 协议的采用和因特网的增长不只是局限于政府投资的项目。一些大的计算机公司都连接到了因特网上，许多其他大公司也是如此，包括一些石油公司、汽车工厂、电器商行、制药公司和电信公司等。中小公司在 20 世纪 90 年代也开始连接到因特网上。尽管许多公司不选择让其内部的**内联网**（intranet）成为全球因特网的一部分，但仍在其内部网中使用 TCP/IP 协议。

1.5 因特网体系结构研究委员会

由于 TCP/IP 网际协议族不是从某个特定厂家或被公认的专家组中产生的，所以人们自然会问，“是谁决定了技术方向？是谁来决定协议何时可以变成标准？”答案是称为**因特网体系结构研究委员会**（Internet Architecture Board，简称 IAB^①）的工作组。IAB 是 DARPA 于 1983 年重新组建因特网控制和配置委员会时形成的。IAB 为在 TCP/IP 协议下所进行的研究和开发提供了方向并进行协调，还对因特网的发展加以引导。它决定哪些协议是 TCP/IP 协议族的一部分并制定官方政策。

1.6 IAB 的重组

到 1989 年夏天，TCP/IP 技术和因特网的增长都已经超过了最初目标，它由最初的研究项目发展成为无数人日常工作所依赖的工具。研究人员再也不可能像早期那样，在一夜之间仅靠改变几个装置就可以推行一些新概念了。在很大程度上，几百个提供 TCP/IP 产品的商业公司决定了其产品是否可以互操作，而这又取决于这些公司何时决定把一些改动加入其软件中。在实验室中起草规范和测试新概念的研究人员，对新概念能否被立即接受和使用不再抱有期望。具有讽刺意味的是，设计和观察 TCP/IP 发展的研究人员发现，自己的研究成果所获得的商业成功已经超越他们，在起着主导作用。一句话，TCP/IP 变成了一个成功的产品技术，市场开始主宰它的演变。

为了反映 TCP/IP 和因特网的政治实体与商业实体的利益，IAB 在 1989 年夏天进行了重组。IAB 的研究人员被转移到 IAB 下属的一个工作组——**因特网研究部**（Internet Research Task Force，简称 IRTF）中，另外又组成一个新的 IAB 委员会，成员来自更广泛的社团代表。协议标准和其他技术方面的事情被交给一个名为**因特网工程部**（Internet Engineering Task Force，简称 IETF）的工作组。

IETF 在原先的 IAB 机构中就已经存在了，它的成功成为 IAB 重组的一个动机。大多数 IAB 的工作组只由几个研究某特定问题的人员组成，IETF 与这些组织不同，它在重组前规模就很大，当时已发展成拥有众多活跃成员的组织，这些成员在共同研究许多问题。重组之后，IETF 被分成 20 多个工作组（working group），每个组着重研究一个特定问题。

由于 IETF 过于庞大，一个主席已难于管理，它又被分成近 10 个区域，每个区域都有自己的管理员。IETF 主席和各区域管理员组成**因特网工程指导小组**（Internet Engineering Steering Group，简称 IESG），每个成员都负责协调 IETF 工作组的工作。现在，“IETF”这个名字指的是一个整体，包括主席、领域管理员和所有工作组成员。

1.7 因特网 RFC

前面已经谈及，TCP/IP 技术不是由哪个厂家、专业协会或标准团体拥有的。因此，协议、标准和政策的文档无法从某个厂家那里得到，而是被存放在联机的资料库中，可供人们免费使用。

有关因特网工作的文档、有关新协议或修订协议的建议以及 TCP/IP 协议标准，都出现在一系列技术报告中，这些报告被称为**因特网 RFC**（Internet Request For Comment）或 RFC。RFC 可长可短，内容可以涵盖很多概念或细节，也可以是标准，或仅仅是新协议的建议。本书全文贯穿了对 RFC 的引用。然而，RFC 和学术研究论文被引用的方式不同，因为 RFC 是编辑出来的。以前一直是 Jon Postel 一个人进行 RFC 的编辑工作。现在编辑 RFC 的任务交给了 IETF 的各区域管理员，IESG 总体上认可新的 RFC。

RFC 系列是以 RFC 编写的年代顺序进行编号的。给每个新的或修订过的 RFC 分配一个新编号，所以读者一定要拿到文档的最高编号的版本，使用 RFC 索引有助于识别正确的版本。而且，我们

^① IAB 以前曾代表因特网工作委员会（Internet Activities Board）。

还可以得到 RFC 文档的早期版本，即**因特网草案**（Internet draft）。

RFC 或因特网草案可以从下面的网站得到：

www.ietf.org

1.8 未来的发展和技术

TCP/IP 技术和因特网都还在继续发展，不断有新协议被提出来，有旧协议被修订。对下层技术的最主要需求并非来自增加的网络连接，而是来自增加的网络通信量。当新用户连接到因特网并出现新应用程序时，通信量模型发生了改变。例如，当用户开始使用 World Wide Web 之类的服务来浏览信息时，通信量急剧增加。后来，当文件共享被广为使用时，流量模型再次发生变化。当因特网被用于电话和视频服务时会发生更多变化。

图 1.1 总结了因特网的扩展，并说明了它增长的一个重要因素：多个工作组管理因特网的各个部分，使网络复杂性发生了变化。因特网技术的发展是在 DARPA 靠一个人控制因特网的各个方面的时候，所以许多子系统的设计依赖于集中式管理和控制。随着因特网的逐渐成长，责任和控制被划分到了多个组织。特别是，随着因特网变得全球化，需要的操作和管理延伸到多个国家。从 20 世纪 90 年代早期起，很多工作转向寻求办法来扩充原来的设计，以适应非集中式管理。

	number of networks	number of computers	number of users	number of managers
1980	10^1	10^2	10^2	10^0
1990	10^3	10^5	10^6	10^1
2000	10^5	10^7	10^8	10^2
2005	10^6	10^8	10^9	10^3

图 1.1 因特网的增长。除了规模增长带来通信量增加外，非集中式管理导致复杂性的产生

1.9 本书的组织

有关 TCP/IP 的资料分成三卷编写。本卷将详细介绍 TCP/IP 技术、使用它的应用程序以及全球因特网的体系结构。本卷还讨论了 TCP 和 IP 等协议的基础，说明它们如何应用于互联网。除了给出细节以外，书中还强调了网络协议下层的一般原理，解释为什么 TCP/IP 协议容易适应多种底层物理网络技术。第二卷更深入地讨论 TCP/IP 协议，并详细说明协议软件的内部组织。文中给出了一个可运转系统的代码，以说明各个协议如何协同工作。第三卷说明分布式应用程序如何使用 TCP/IP 进行通信，重点是客户—服务器模式，这是所有分布式编程的基础。文中讨论了程序和协议之间的接口^①，并展示客户程序和服务器程序是如何组织的。另外，第三卷还描述了远程过程的概念、中间件，说明程序员如何使用工具构建客户和服务器软件。

到目前为止，我们已经一般性地讨论了 TCP/IP 技术和因特网，总结了它们提供的服务以及发展历史。下一章简要总结了在因特网上广泛使用的网络硬件。其目的不是展现不同厂家硬件之间的细微差别，而是着重给出对互联网设计师而言最重要的各种技术的特征。后面章节将仔细研究每个协议和因特网，以达到 3 个目的：探讨一般概念并回顾因特网构造的模型；考察 TCP/IP 协议的细

^① 第三卷有三个版本：一个使用 Linux 套接字接口，一个使用微软定义的 Windows 套接字接口，还有一个使用运输层接口（Transport Layer Interface，简称 TLI）。

节；查看电子邮件和电子文件传送等高级服务所用的标准。第 3 章到第 12 章回顾了一些基本原理，并描述了在使用 TCP/IP 的机器上创建的网络协议软件。后面各章描述了跨接多台机器的服务，包括路由信息的传播、**名字解析**（name resolution）以及电子邮件之类的应用程序。

正文后有一个附录是本书所用术语和缩略语的字母序清单。由于初学者经常被新术语所困扰。术语又难以记忆，所以鼓励他们使用这个清单来查找，而不要在书中到处翻找。

1.10 小结

互联网由一组网络组成，这些网络互相连接，但又像一个整体协作运行。互联网的主要优点是提供了通用互连，同时又允许每个组织使用最适合自己需求的任何网络硬件。我们将一般性地考察互联网通信底层的原理，并专门详细讨论一个网际协议族。我们还要讨论在一个互联网中如何使用网际协议。书中所讲解的技术，根据它的两个主要协议而被称为 TCP/IP，它是由美国国防部远景规划局开发的。TCP/IP 为全球因特网提供了基础，因特网是一个庞大的互联网，它连接了世界上许多国家的个人、公司和政府部门以及其他组织。全球因特网还在迅速扩展着。

1.11 深入研究

Cerf 的 A History Of The ARPANET [1989] 和 History of the Internet Activities Board [RFC 1160] 给读者提供了有关 TCP/IP 和网际互连的早期研究论文，值得一读。Denning [Nov-Dec 1989] 对 ARPANET 的历史从不同角度进行了研究。Jennings et. al. [1986] 讨论了计算机连网对科学家的重要性。Denning [Sept-Oct 1989] 也指出了网际互连的重要性，并给出了世界范围的因特网可能出现的一种场面。美国联邦科学、工程和技术协调委员会（Federal Coordinating Committee for Science, Engineering and Technology，简称 FCCSET）建议连网应成为一个国家优先考虑的项目。

IETF (ietf.org) 出版其定期会议的备忘录；World Wide Web 联盟 (w3c.org) 发布了 Web 技术的协议和标准。最后，我们希望读者记住：TCP/IP 协议族和因特网在不断地发生变化，从 RFC 或者一些年度 ACM SIGCOMM 讨论会之类的会议上可以获得新的信息。

1.12 习题

1. 在使用 TCP/IP 的网点上找出一些应用程序。
2. 绘图示意本单位的 TCP/IP 技术和因特网访问的增长情况。每年有多少计算机、用户和网络连入因特网？
3. TCP/IP 产品每年创造数十亿美元的总收入。读一读商业出版物，查找提供此类产品的厂家清单。

第2章 底层网络技术的回顾

2.1 引言

因特网不是一种新的物理网络，理解这一点是非常重要的。实际上，它是一种将物理网络互连的方法以及一组使用网络的约定，这些约定允许联网的计算机进行交互。虽然网络硬件在整体的设计中只起着较小的作用，但是我们若想真正理解互联网技术，就要区分硬件本身提供的低级机制和TCP/IP 协议软件提供的更高级工具。我们还要理解下层分组交换技术提供的接口如何影响我们对高层抽象的选择，这一点也很重要。

本章介绍基本的分组交换概念和术语，然后回顾一些已在 TCP/IP 互联网上使用的底层网络硬件技术。后面的章节描述了这些网络如何互连，以及 TCP/IP 协议如何适应不同硬件的巨大差异。尽管此处列举的情况肯定不够全面，但也能清楚地说明运行 TCP/IP 的物理网络的多样性。读者可放心地略过许多技术细节，但应该试着掌握分组交换的思想，并设想用这样异构的硬件建立一个同构的通信系统。更重要的是，读者应仔细研究各种技术所使用的物理地址方案。后面的章节将详细介绍高层协议如何使用物理地址。

2.2 网络通信的两种途径

根据通信网络提供的是两台计算机之间的连接还是终端与计算机之间的连接，通信网络可以分成两种基本类型：面向连接的（有时称为电路交换）和无连接的（有时称为分组交换^①）。面向连接的网络的运行需要在两点之间形成一条专用连接（connection）或线路（circuit）。传统的电话系统使用面向连接的技术，即一个电话呼叫建立一条连接，从发起呼叫的电话机通过本地交换局、经过干线到一个远端的电话交换局，最后到达目的电话机^②。由于连接的存在，电话设备可对话筒输出反复进行采样，把采样进行数字编码，并通过连接把它们传输到接收方。由于连接确保提供了一个 64 kbps（每秒千比特）的数据通路，这个速率是发送数字化话音所必需的，因而可保证发送方的采样能被递送出去并在接收方复制出来。面向连接联网的好处在于它可以保证容量：一旦建立一条电路，其他网络活动都不会减少该条电路的容量。面向连接技术的缺点是开销大：电路的开销是固定的，与通信量的大小无关。例如，一个人要为一个电话呼叫交付固定的费用，即使是呼叫双方并没有交谈。

无连接类型的网络常用于连接计算机，它采取完全不同的处理方法。在一个无连接网络中，在网络中传送的数据被分成小片，这样的小片称为分组（packet），分组被多路复用到机器间一些高容量的连接上。一个分组，通常含有几千比特的数据，携带有标识信息，让网络硬件知道怎样把数据发送到指定目的地。例如，要在两台机器之间传输一个大文件，该文件必须被分成许多分组，在网络上一个一个地传输。网络硬件把分组递送到指定目的地，在那里，软件把它们重新组装成一个文件。分组交换的主要优点是：计算机之间的多路通信可以并行进行，同时机器间的若干连接被正在通信的每一对机器所共享。当然，它的缺点是，随着通信活动的增加，某一对计算机的通信量与网络容量相比所占的比例会更少。也就是说，一旦某个分组交换网络超载，那么，使用这个网络的计算机在能继续发送额外的分组之前，必须等待。

^① 也有可能是混合技术。

^② 基于 IP 的电话技术正在取代传统的电话技术。

尽管无连接的网络具有网络容量无法保证的潜在缺点，但无连接的网络还是非常流行。采用分组交换的动机是出于开销和性能方面。因为多台计算机可以共享网络带宽，所以需要的连接比较少，开销低。由于工程师们已能制造出高速网络硬件，容量通常不成问题。许多计算机互连都使用无连接的网络，所以本书后面若未加其他声明，**网络**（network）术语将特指无连接的网络。

2.3 广域网和局域网

地理范围跨度大（如美国大陆）的分组交换网与短距离跨度（如一个房间内）的网络在本质上是不同的。为有助于刻画两种网络在容量和使用意图上的区别，分组交换技术通常被分成两大类：广域网（WAN）和局域网（LAN）。这两类技术并没有正式的定义，但厂家可使用这些术语来帮助客户区别不同的技术。

广域网有时又称作**远程网**（long haul network），它能提供远距离的通信。大多数广域网技术并不限制通信的距离；广域网可以允许通信的两端相隔任意远。例如，一个广域网可以横跨一个大陆或横跨一个大洋把计算机连接起来。通常，广域网的工作速率比局域网的低，而且在连接之间有更大的时延。广域网的典型速度是 1.5 Mbps（每秒兆比特）～2.4 Gbps（每秒吉比特）。在广域网上的时延可从几毫秒到十分之几秒^①。

局域网技术提供最高速的计算机间连接，但却牺牲了远距离通信的能力。例如，一个典型的局域网只能覆盖一个小区域，诸如一幢建筑物或一个小校园，运行速率在 100 Mbps 到 10 Gbps 之间。由于局域网技术延伸的距离较短，所以它的时延比广域网的小。局域网上的时延可以少到十分之几毫秒，多到 10 毫秒。

我们已经陈述了速度和距离间通常存在的折中关系：提供较高速通信的技术在较短的距离上起作用。这两大类技术之间还有其他区别。在局域网技术中，每台计算机通常包含一个直接连接到网络的设备，该设备称为**网络接口卡**（Network Interface Card，简称 NIC）。网络本身不需要包括太多智能，它可以依赖联网计算机中的电子接口设备来产生和接收复杂的电信号。在广域网技术中，网络通常由许多称为**分组交换机**（packet switch）的复杂计算机组成，这些机器通过长距离的通信线路相互连接起来。网络的规模可以通过增加新的交换机和通信线路来扩展。把用户的计算机连到一个广域网上，意味着把它连到了其中一个分组交换机上。在广域网上一条路径沿线的每个分组交换机，在接收一个分组并把它转发到下一个交换机上时，都会产生时延。因此，广域网越大，在它上面传递信息所需的时间也就越长。

本书讨论的协议软件隐藏了不同网络在技术上的差异，并且使互连与底层的硬件无关。要搞清楚该协议软件中为什么选择这样的设计，就有必要理解它与网络硬件的关系。下一节给出已经在因特网上使用的网络技术的例子，并说明它们之间的一些区别。后面的章节阐述了 TCP/IP 软件如何隔离这些区别，并使通信系统与底层硬件技术无关。

2.3.1 网络硬件地址

每种网络硬件技术都定义了一种**寻址机制**（addressing mechanism），计算机使用这种机制指明每个分组的目的地。每台连接到网络的计算机都被分配了一个唯一的地址，地址通常是一个整数。在网络上传送的分组都包括一个**目的地址字段**（destination address field），其中有分组接收方的地址。在所有分组中，目的地址出现在同一位置上，这使得网络硬件容易检查目的地址。发送者必须知道接收者的地址，并且必须在发送分组之前把接收者的地址放到分组的目的地址字段中。

每种硬件技术都规定了如何给计算机分配地址。例如，硬件规定了地址中的位数以及分组中目的地址字段的位置。尽管有的技术使用兼容的寻址方案，但也有很多是不兼容的。本章包含几个硬

^① 一些广域网要通过发送信号给绕地球运转的卫星进行通信，从而造成这么长的时延。

件寻址方案的例子，后面的章节将解释 TCP/IP 如何适应多种不同的硬件寻址方案。

2.4 以太网技术

以太网 (Ethernet) 是一种流行的分组交换局域网技术，由 Xerox Palo Alto 研究中心 (PARC) 在 20 世纪 70 年代早期发明。Xerox 公司、Intel 公司和 DEC 公司于 1978 年把以太网标准化；IEEE 用标准号 802.3 发布了一个兼容版本的标准。以太网已成为一种最流行的局域网技术，现在几乎所有公司和个人的网络都采用以太网。以太网如此流行，因而出现了许多种变形的以太网技术。最初的布线方式是用同轴电缆连接所有的计算机。这种方式已被取代了。现在的方式称为双绞线以太网 (twisted pair Ethernet)，允许计算机使用普通的非屏蔽铜线接入以太网，这种线类似电话连线^①。使用双绞线（称为 5 类电缆线）的主要优点是：减少了开销，而且比同轴电缆更容易安装。

第一个双绞线以太网正式名称为 10Base-T，它工作在 10 Mbps 速率上，与最初的以太网非常类似。一组八根线（四对）用于把每台计算机连接到一个位于中心的以太网**集线器** (hub) 或**交换机** (switch)，如图 2.1 所示。八根线中只用了四根线：一对线用于从计算机到集线器的数据传送，另一对线用于从集线器到计算机的数据传送。

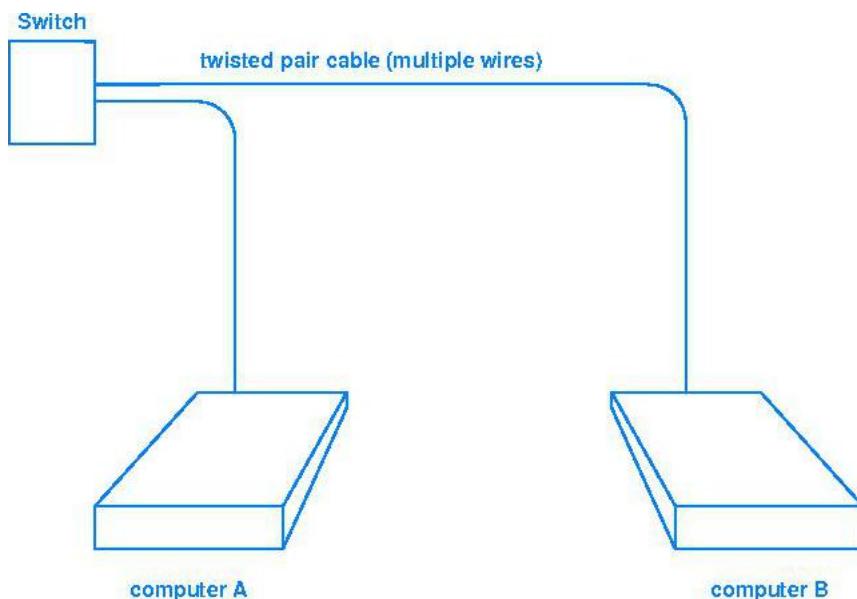


图 2.1 一个使用双绞线布线的以太网。每台计算机使用含多对双绞线的电缆连接到一个集线器

集线器是一个电子设备，它仿真了以太网电缆上的信号。在物理上，集线器是一个通常存放于布线柜中的小盒子；一个集线器和一台计算机之间的连接长度必须小于 100 m。集线器需要使用电源，并且允许被授权的人员通过网络监视和控制它的操作。对于一台计算机而言，在机器中与一个集线器相连的网络接口卡，就像是它的一个 I/O 设备，可用于收发数据。

2.4.1 以太网容量

尽管布线方式从最初的同轴电缆发展成双绞线，但仍有许多原来的布线设计保持未变。而且，初期的双绞线以太网设计与最初的以太网以同样的速率运行，数据能以 10 Mbps 的速率传输。尽管计算机能够以以太网的速度产生数据，但不应该把原始的网络速度当成是两台计算机可以交换数据的速度，而是应该把网络速度当成一个对全部通信量容量的度量。我们可以把网络看成连接多个城市的高速公路，把分组看成高速公路上的汽车。高带宽可能承载更高的流量负载，而低带宽则意味

^① 由于常规的电话线使用绞线的技术来避免干扰，从而产生了双绞线这个术语。

着高速公路不能承载那么多的交通流量。例如，一个 10 Mbps 的以太网可容纳几台产生重负载的计算机，或者容纳多台产生轻负载的计算机。

在 20 世纪 70 年代末期，在以太网被标准化以后，以 10 Mbps 速率工作的局域网有过多的容量可供许多计算机使用，因为当时可用的 CPU 速度和网络接口硬件阻碍了计算机快速收发数据。但是，到了 20 世纪 90 年代中期，CPU 速度以及网络的使用都有了惊人的增长。因此，即使对于中等规模的公司，以 10 Mbps 速率工作的以太网也没有足够的容量来充当公司的中心主干网，以太网已成为一个瓶颈。

2.4.2 快速和吉比特以太网

为了克服以太网的吞吐量限制，工程师们设计了一个速度更快的以太网版本。一个版本是被市场认可的**快速以太网**(Fast Ethernet)，它以 100 Mbps 的速率工作；而**吉比特以太网**(Gigabit Ethernet，简称 GigE)以 1000 Mbps 的速率工作。前者被正式命名为 100Base-T，后者被正式命名为 1000Base-T，这两种更快速的以太网布线使用的 5 类双绞线与 10Base-T 使用的一样；通过采用更多根线来传输数据，并通过改变信号机制而获得速度的增长。

除了铜线上的吉比特以太网，IEEE 还定义了光纤上的以太网标准。这种技术被命名为 1000Base-X，它把以太网分组转变成光脉冲，然后通过一根光纤进行传输。光纤的主要优点是：更高的容量和抗电子干扰。光纤的容量足以支持远高于 1 Gbps 的比特率。因此，一些工程师已开始开发 10~40 Gbps 的以太网技术。

为理解容量增加的意义，我们要理解两个事实。首先，虽然计算机速度已经变快了，但计算机系统还是很少能以 1 Gbps 的稳定速率传输数据。其次，新的以太网版本并没有修改标准的其他部分，特别是最大分组长度仍与 10Base-T 的相同。这两个事实意味着，较高速的以太网技术不是为了在一对计算机之间提供可能的最高吞吐率而最优化的，而是为允许更多的网点接入以及更多的通信总量而优化设计的。

2.4.3 10/100/1000 以太网

在快速以太网问世不久，厂家开始制造能够接受 10 Mbps 或 100 Mbps 连接的设备；现代以太网可以工作在 10 Mbps，100 Mbps 或 1000 Mbps。这种技术被称为 10/100/1000 以太网，它既可用于 NIC，也可用于交换机。更重要的是，一个 10/100/1000 设备在首次连网时将自动进行协商，以确定电缆的类型(直通或交叉)和连接另一端可支持的最大速度。实质上是硬件插入了额外的信号，连接的另一端可使用此信号来确定正确的配置。

自动协商和保持原来的分组格式，意味着计算机不需要重新配置软件和改动设备驱动程序，就可以从一个 10 Mbps 以太网交换机上转移到一个吉比特以太网交换机上。从协议的角度看，下层硬件和布线只改变了分组发送的速度；分组本身在不同的以太网技术上可以相互交换。

2.4.4 以太网上供电

除了其他创新外，工程师们还开发了一个称为**以太网上供电**(Power over Ethernet，简称 PoE)的技术，它在以太网所用的铜线电缆上传送少量的电源。要点在于这种技术的设计确保了电源的出现不会降低数据传输的质量。虽然 PoE 提供的电量不够计算机使用，但这一创新还是对于一些诸如 IP 电话之类的小设备特别重要，这就意味着这些小设备只需要一根电缆。

2.4.5 以太网的性质

以太网被设计为一种支持广播的共享总线技术，它使用**尽最大努力交付**(best-effort delivery)的语义，并且有分布式接入控制。由于所有网点连接到一个共享的单一通信信道，这种拓扑方式被

称为**共享总线** (shared bus)；由于所有网点能收到每次传输的数据，因此可把一个分组同时传输到所有网点，该拓扑方式又称为**广播技术** (broadcast technology)。后面还将讨论把分组直接从一个网点传输到另一个网点或所有网点的一个子集的方法。以太网之所以被认为采用的是尽最大努力交付的机制，是因为硬件没有向发送者提供任何信息来判断分组是否已被交付。例如，如果目的主机突然掉电，那么发送给它的分组将丢失，而发送者不会得到通知。稍后我们将看到 TCP/IP 协议如何适应尽最大努力交付的硬件。

以太网的接入控制是分布式的，这是因为以太网与其他一些网络技术不同，它没有任何中央权威机构来授权接入。以太网的接入方式称为**带有碰撞检测的载波监听多点接入** (Carrier Sense Multiple Access with Collision Detect，简称 CSMA/CD)。它是 CSMA 的，因为多台机器可同时接入一个以太网，并且各机通过探测是否存在载波来判断网络是否空闲。当某主机接口要发送一个分组时，先要进行监听，以判断是否有报文正在传输（即进行载波探测）。如果没有监听到传输，主机接口就开始发送。由于有最大分组长度的限制，每次发送都在一个限定的时间段内完成。另外，硬件必须留意在两次发送之间保持一个最小空闲时间，也就是说，没有哪一对正在通信的机器可以连续使用网络而不给其他机器接入网络的机会。

2.4.6 碰撞检测与恢复

当一个网点开始传输时，信号并未同时到达网络各处，而是大约以 70% 光速的速率沿铜线传播。这样就可能有两个网点同时探测到网络空闲，并同时开始发送。当这两个电信号相遇时，它们混杂在一起，每个信号都失去了意义。这种事件称为**碰撞** (collision)。

以太网以一种巧妙的方式处理碰撞。每个网点在发送时监听电缆，判断是否有外来信号干扰其传输。从技术上说，这种监视称为**碰撞检测** (Collision Detect，简称 CD)，使以太网成为一个 CSMA/CD 网。当检测到碰撞时，主机接口将会放弃传输，等待碰撞活动消退，然后再次试发送。必须注意，要避免所有网点都忙于尝试发送而导致每次发送都产生碰撞，使整个网络拥塞。为了避免这种局面，以太网使用一种**二进制指数退避策略** (binary exponential backoff policy)：发送方在第一次碰撞发生后延迟一个随机时间；如果第二次发送也产生碰撞，延迟的时长则为第一次时延的两倍；若第三次发送还碰撞，就延迟四倍时长，依次类推。采用指数退避的动机是：如果发生许多网点同时发送的情况，将发生严重的拥塞。在这种拥塞中，两个网点很可能选择非常接近的随机时间进行退避。这样，发生另一次碰撞的可能性就高。通过把延迟时间加倍，指数退避策略会很快把网点尝试重传的时间间隔显著拉开，使得以后发生碰撞的可能性变得非常小。

2.4.7 无线网络和以太网

IEEE 还推出了一系列无线网络的标准。无线网络与以太网是非常近似的。其中，最有名的标准在市场上称为 Wi-Fi，其 IEEE 标准号为 802.11b，很多便携式电脑都支持此标准。Wi-Fi 提供的带宽最高为 11 Mbps，但通常工作在 2.5~4 Mbps 之间；另有两个编号为 802.11a 和 802.11g^① 的标准最高可工作在 54 Mbps。这 3 个标准都可以有两种使用形式：一是作为接入技术，让单个基站[称为一个**接入点** (access point)]能够允许多个客户（例如使用便携式电脑的用户）接入；二是用在一个点对点配置中，把两个接入点连接起来。IEEE 还定义了一个更高速的技术，主要用于点对点互连。该技术在市场上称为 Wi-Max，其 IEEE 标准号为 802.16，一些需要连接两个地点的网络提供商或公司对该技术最感兴趣。最后，IEEE 已计划引入一个带宽最高为 540 Mbps 的无线标准 802.11n，以及一个安全无线网络的标准 802.11i。

^① 802.11g 可与 802.11b 进行互操作。

2.4.8 以太网硬件地址

以太网定义了一个 48 比特寻址方案，该方案已成为一个工业标准。每个网络接口卡都分配了一个唯一的 48 位数字，即一个**以太网地址**（Ethernet address）。为了分配地址，以太网硬件制造商需要购买以太网地址块^①，并在生产以太网接口硬件时按顺序进行分配。因此，没有两个硬件接口具有同样的以太网地址。

通常，以太网地址是以机器可读形式安装在主机接口硬件上的。因为以太网地址属于硬件设备，所以它们有时也称为**硬件地址**（hardware address）、**物理地址**（physical address）、**媒体接入**（media access，简称 MAC）地址或第 2 层（layer 2）地址。注意，以太网物理地址有如下重要性质：

以太网地址是与接口硬件相关联的，而不是与计算机相关联的；把硬件接口移到新机器上或替换失效的硬件接口都将改变机器的物理地址。

一旦知道了以太网地址可以改变后，就会明白为什么要求网络高层软件的设计能适应这种变化。

回想一下，当多台计算机连接到一个集线器时，每台计算机收到的是通过集线器传递的每个分组的副本——甚至包括那些发送到其他机器的分组。主机接口硬件把分组中的目的地址字段用来作为过滤器。接口忽略那些目的地址为其他机器的分组，只把目的地址是本地主机的分组传递给主机。尽管计算机的中央处理器可以完成这种检测，但由主机接口完成检测则能够防止以太网上的通信量降低所有计算机的处理速度。

一个 48 比特的以太网地址不仅仅指定一个目的计算机地址，还可以是以下 3 种形式之一：

- 一个网络接口的物理地址，即**单播地址**（unicast address）
- 网络**广播地址**（broadcast address）
- **多播地址**（multicast address）

按照惯例，广播地址（全 1）被预留下来用于同时将分组发送到所有网点。多播地址提供了一种有限广播形式，网络上所有计算机的某个子集同意监听一个给定的多播地址。这些参与监听的计算机子集称为**多播组**（multicast group）。为加入某个多播组，计算机必须指示它的主机接口接受该组的多播地址。多播的优点在于它限制广播的能力：分组只需发送一次就可到达多播组内的每台计算机，但未加入多播组的计算机则不接收发送给该多播组的分组。

为适应广播及多播寻址，以太网接口硬件必须识别多种地址。接口至少要接受两种分组：那些目的地址为接口物理地址的分组（即单播）和目的地址为网络广播地址的分组。有的接口经过配置可以接受多播地址，一个其他物理地址，或者所有分组。最后这种方式称为**混杂模式**（promiscuous mode），用于网络监控。当计算机启动时，操作系统初始化并配置接口，给它指定要接受的地址。然后，接口检测每个分组中的目的地址字段，只把配置允许的那些分组传递给计算机。

2.4.9 以太网帧格式

以太网可看成是机器之间的链路层连接。因此，我们说以太网传输的是**帧**^②（frame）。以太网帧的长度可变，但不能小于 64 八位组^③（octet）或大于 1518 八位组（首部、数据和 CRC）。在所有分组交换网络中，每个以太网帧都包含一个存放目的地址的字段。图 2.2 给出了以太网的帧格式，其中包含物理源地址和目的地址。

^① 电气与电子工程师协会 IEEE (Institute for Electrical and Electronic Engineers) 管理以太网地址空间并根据需要分配地址。

^② 术语“帧”（frame）来源于串行线路上的通信，其中，发送者在传输的数据前后加上特殊字符，使其成为一帧。

^③ 从技术上讲，术语“字节”（byte）是指一个硬件无关的字符大小；网络专业人员使用术语“八位组”（octet），因为在所有计算机上，它都是指 8 个比特。

Preamble	Destination Address	Source Address	Frame Type	Frame Data	CRC
8 octets	6 octets	6 octets	2 octets	46–1500 octets	4 octets

图 2.2 以太网帧（分组）的格式。帧的头部有一个前同步码，帧后面是分组间的空隙。字段没有按比例画出

除能识别源站和目的站以外，以太网上传输的每一帧还包含一个**前同步码**（preamble）、**类型字段**（type field）、**数据字段**（data field）和**循环冗余检验码**（Cyclic Redundancy Check，简称 CRC）。前同步码是 64 位的“0”、“1”交替序列，用来帮助接收帧的接口实现同步。32 位的 CRC 帮助接口检测传输错误，CRC 相当于一个函数，发送方为帧中的数据计算 CRC 的值，接收方重新计算 CRC 的值来验证分组是否接收无误。

帧类型字段包括一个 16 位整数，用于识别帧中承载数据的类型。从因特网的角度来看，帧类型字段是不可或缺的，因为它意味着以太网的帧是**自识别的**（self-identifying）。当一帧到达一台指定机器时，操作系统根据帧类型决定用哪个协议软件模块对它进行处理。自识别帧的主要优点是，它允许在单一计算机上同时使用多个协议，还允许在同一物理网络中混合使用多个协议而互不干扰。例如，可以让一台计算机上既有使用因特网协议的应用程序，又有使用本地实验协议的应用程序。操作系统根据到达帧的类型字段决定怎样处理它的内容。以后将看到 TCP/IP 协议用自识别以太网帧来区别于几个不同的协议。

2.4.10 用网桥扩展以太网

以太网桥（Ethernet bridge）是连接两个以太网并在其间传递帧的设备。网桥不复制噪声、错误或畸形帧，在网桥从一个网段接受一帧并将其转发到另一网段之前，必须接收一个完整有效的帧。此外，网桥和以太网络之间的每个连接都遵从以太网 CSMA/CD 原则，所以一个网段上的碰撞以及传播时延与其他网段隔离开来。因此，任意（几乎）多个以太网可通过网桥连接到一起。桥接被用在很多网络系统中。例如，宽带因特网服务通常使用的**电缆调制解调器**（cable modem）和**数字用户线**（Digital Subscriber Line，简称 DSL）连接实现了桥接——在住所发送的以太网帧被桥接到因特网服务提供者（ISP），反之亦然。

关于桥接，最应该理解的是：

网桥隐藏了互连的细节：一系列用网桥连接的网段就像同一个以太网一样运行着。

既然这样，桥接的网络可被认为是**透明的**（transparent），因为网络中的计算机不知道有多少个网桥连接着各个网段。一台计算机与跨网桥的另一台计算机进行通信时所用的硬件、帧格式和过程，与它和本地集线器上的一台计算机通信时所使用的完全相同。

大多数网桥所做的工作不仅仅是把帧从一段线路复制到另一段线路，它们还能够智能地决定把帧转发到何处。这种网桥称为**自适应**（adaptive）或**学习**（learning）网桥。一个自适应网桥由一台有两个以太网接口的计算机组成。自适应网桥中的软件维护着两个地址表，每个接口各有一个。当一帧从以太网 E₁ 到达时，自适应网桥把一个 48 比特的以太网源地址添加到与 E₁ 相关的表中。类似地，当一帧从以太网 E₂ 到达时，网桥也把源地址添加到与 E₂ 相关的表中。因此，经过一段时间后，自适应网桥就知道了在 E₁ 上有哪些机器，在 E₂ 上有哪些机器。

在记录了一帧的源地址后，自适应网桥根据该帧的目的地址决定是否继续转发该帧。如果从地址表中看出该帧的目的地址就在帧来自的那个以太网中，网桥就不转发该帧。如果目的地址不在地址表中（例如，目的地址是一个广播或多播地址，或者网桥还不知道目的站的位置），则网桥把该帧转发到其他以太网。

自适应网桥的优点非常明显。因为网桥使用正常通信流中发现的地址，所以它是完全自动的，

不需要人为地用特定地址配置网桥。由于网桥不转发不必要的通信流，所以可通过在特定的网段间隔离通信流，帮助提高超负载网络的性能。如果一个网络可以在物理上划分成两段，并且每段都包含一些频繁通信的计算机（例如，每段都有一组工作站和一个服务器，而工作站的大部分通信流量都来自与服务器的直接通信），那么网桥工作的效果就会特别好。总而言之：

自适应以太网网桥连接两个以太网段，把帧从一个网段转发到另一个网段。它利用源地址了解每个网段上有哪些机器，并把了解到的信息与目的地址相结合，以排除不必要的转发。

从 TCP/IP 的角度看，桥接的以太网仅仅是另一种形式的物理网络连接。要点在于：

因为由网桥提供的以太网之间的连接对使用以太网的机器来说是透明的，所以可把用网桥连接起来的多个以太网网段看成是单个物理网络系统。

许多商用网桥比这里描述的网桥更为复杂和健壮。第一次加电启动时，商用网桥会检测其他网桥，了解网络的拓扑结构。它们使用一种分布式生成树算法来决定如何转发帧。需要特别指出的是，网桥决定如何传播广播分组，即向每条线路只交付广播帧的一个副本。假如没有这样的算法，连接在同一回路上的以太网和网桥将会产生灾难性后果，因为它们将同时在两个方向上转发广播分组。

2.5 交换的以太网

以太网交换机 (Ethernet switch) 是一种融合并扩展了桥接概念的设备。一个交换机可提供多个连接端口 (port)，每个端口连接一个设备。像网桥一样，交换机在转发帧时使用每个到达帧中的源地址和目的地址。交换机使用源地址来判断交换机的各个端口连接到哪台计算机，使用帧中的目的地址来判断要把帧发到哪里。因此，可以认为以太网交换机类似于多端口的网桥。

2.6 异步传递方式

异步传递方式 (Asynchronous Transfer Mode, 简称 ATM) 是一种面向连接的网络技术，既可用于局域网，也可用于广域网。ATM 的设计允许高速数据交换。为了实现高速传输，ATM 使用专用的硬件和软件技术。第一，ATM 网络包括一个或多个高速交换机，每个交换机都连接到计算机和其他 ATM 交换机上；第二，ATM 网络的最底层使用固定长度的帧，称为**信元** (cell)。因为每个信元的长度相同，所以 ATM 交换机硬件可以迅速处理信元。

2.6.1 ATM 信元大小

令人惊讶的是，每个 ATM 信元的长度只有 53 八位组长。每个信元包含 5 八位组的首部和后面 48 八位组的数据。不过，后面的章节将说明，当使用 ATM 发送 IP 通信流时，八位组长度 53 与发送无关。ATM 网络可接收和传递更大的分组。

2.6.2 面向连接的连网

ATM 与前面描述的分组交换网络不同，因为它提供面向连接的服务。在一台连接到 ATM 交换机的计算机发送信元之前，必须手动建立连接，或者主机必须先与交换机交互，以指明目的地。这种交互类似于进行一次电话呼叫^①。发出请求的计算机指明远端计算机的地址，并等待 ATM 交换机找到一条网络通路并建立连接。如果远端计算机拒绝该请求、不响应或者发送方和接收方之间的

^① 由于 ATM 出自对运载话音和数据感兴趣的电话公司，所以 ATM 设计和电话系统之间有着很密切的关系。

ATM 交换机当前无法建立通路，则建立通信的请求失败。

一旦成功建立了一条连接，本地 ATM 交换机就为该连接选择一个标识符，并把标识符随同一个通知计算机成功建立连接的报文传给计算机。计算机在发送或接收信元时都要使用这个连接标识符。

在结束一个连接的使用时，计算机再次和 ATM 交换机通信，请求中断连接。交换机将断开两台计算机的连接。断开连接等同于在电话呼叫后挂断电话；断开连接后，计算机只有在建立新连接后才能进行通信。此外，连接所用的标识符可被回收；一旦发生断连，交换机可以把回收的连接标识符用于新的连接。

2.6.3 广域点到点网络

虽然存在一些诸如 ATM、**交换的多兆比数据服务**（Switched Multimegabit Data Service，简称 SMDS）和 X.25 等广域网服务，但现代的广域网是通过从电话公司租借数据电路构建的，并使用这些电路把一些网络元素互连起来。电话公司最初设计这些数字电路是为了承载数字化话音呼叫，它们在数据网中的使用只是到后来才显示出重要性。因此，这些电路可用的数据率不是 10 的幂。电路只能选择承载 64 kbps 的倍数，因为数字化话音呼叫使用的是称为**脉码调制**（Pulse Code Modulation，简称 PCM）的编码方案，每秒会产生 8000 次采样，其中每个采样是 8 比特。

图 2.3 中的表列出了北美和欧洲使用的几种常见数据率。

Name	Bit Rate	Voice Circuits	Location
-	0.064 Mbps	1	
T1	1.544 Mbps	24	North America
T2	6.312 Mbps	96	North America
T3	44.736 Mbps	672	North America
T4	274.760 Mbps	4032	North America
E1	2.048 Mbps	30	Europe
E2	8.448 Mbps	120	Europe
E3	34.368 Mbps	480	Europe
E4	139.264 Mbps	1920	Europe

图 2.3 从电话公司租借数字电路的可用数据率示例。所选的速率用于编码多路话音呼叫

更高速率的数字电路需要使用光纤。除了一些在铜缆上高速传输数据率的标准，电话公司还提出了在光纤上传输数据率的标准。图 2.4 中的表里有几个例子。当然，在如此高的数据率上工作的电路，其费用也比低速率的电路昂贵得多。

Standard Name	Optical Name	Bit Rate	Voice Circuits
STS-1	OC-1	51.840 Mbps	810
STS-3	OC-3	155.520 Mbps	2430
STS-12	OC-12	622.080 Mbps	9720
STS-24	OC-24	1,244.160 Mbps	19440
STS-48	OC-48	2.488 Gbps	38880
STS-96	OC-96	4.976 Gbps	64512
STS-192	OC-192	9.952 Gbps	129024
STS-256	OC-256	13.271 Gbps	172032

图 2.4 可从电话公司租借的高容量电路的数据率举例。光纤用于在长距离上获得高速率

从 TCP/IP 的角度看，任何确实连接两台计算机的通信系统都可称为**点到点网络**（point-to-point network）。因此，两台计算机之间的租借数据电路就是点到点网络的一个例子。当然，使用术语“网络”来描述两台计算机之间的连接是对这一概念的延伸。但读者以后会认识到把连接看成网络有助于保持一致性。目前，只需要记住点到点网络与常规网络的一个显著区别：由于只有两台计算机连接在一起，硬件地址不是必需的。在讨论互联网地址绑定时，缺少硬件地址将使点到点网络成为例

外。

2.6.4 拨号 IP

拨号互联网接入提供了另一种点到点网络的例子。计算机中的一个拨号调制解调器被用来建立到另一个调制解调器的电话呼叫。通常情况下，呼叫从某个住所发出，止于某个 ISP。一旦电话连接建立，两个调制解调器使用音频声调来发送数据。从 TCP/IP 的角度看，一次电话呼叫等同于让一条线路运转。一旦另一端的调制解调器答复了电话呼叫，就建立了一台计算机到另一台计算机之间的直接连接，只要需要，该连接就会一直保持着。

2.7 小结

前面已经回顾了 TCP/IP 协议使用的几种网络硬件技术，从费用不高的局域网技术（如以太网）到费用高昂的租用数字电路（用于广域连接）。已经看到，使用隧道技术可在其他一般目的的网络协议之上运行 TCP/IP 协议。某个特殊网络技术的细节对我们而言并不重要，但应该记住下面这个一般性的观点：

TCP/IP 协议是极其灵活的，几乎所有底层技术都可以用于传送 TCP/IP 通信流。

2.8 深入研究

早期的计算机通信系统使用点到点互连，通常使用 McNamara [1988] 描述的通用串行线路硬件。Metcalfe and Boggs [1976] 介绍了以太网的 3 Mbps 原型。Digital et. al. [1980] 描述了最初的 10 Mbps 标准，IEEE 标准号为 802.3。有关以太网布线的信息可在线获得。Shoch, Dalal, and Redell [1982] 概述了以太网演变的历史。Abramson [1970] 报道了有关 ALOHA 网络的工作，Cotton [1980] 对各种技术进行了综述。

有关 ARPANET 的更多信息可参见 Cerf [1989] 和 BBN [1981]。Lanzillo and Partridge [January 1989] 描述了拨号 IP。De Prycker [1995] 描述了异步传递方式以及它在广域网服务中的使用。Partridge [1994] 对包括 ATM 在内的多种吉比特技术进行了综述，还描述了高速交换机的内部结构。

2.9 习题

1. 找出你的网点所使用的网络技术。
2. 如果以太网帧在一条 OC-192 租用电路上发送，传输最大可能长度的帧需要花多长时间？对于最小可能长度的帧呢？注：在计算时排除帧的前同步码。
3. 研究以太网交换机技术。交换机与集线器有哪些不同？
4. 如果你的网点使用交换以太网技术，看看一个交换机能有多少连接，以及交换机之间如何互连。
5. 阅读有关 VLAN 以太网交换机的内容。VLAN 交换机与常规的交换机相比，增加了哪些能力？
6. 阅读以太网的标准，找出分组间空隙以及前同步码长度的详细描述。什么是以太网可以传输数据的最大稳定状态速率？
7. 卫星通信信道的特性中的哪一个是我们最想要的？哪个是最不想要的？
8. 如果网络分别以下列速率运行，找出各网络上传输一个 5 MB 文件所需时间的下限：28.8

kbps, 1.54 Mbps, 10 Mbps, 100 Mbps 和 2.4 Gbps。

9. 你的计算机的处理器、磁盘和内部总线运行的速率是否足够快，以便用 2 Gbps 的速率发送某个磁盘文件中的数据。

第3章 网际互连的概念和结构模型

3.1 引言

到目前为止，我们已经查看了各个数据网络上的传输的底层细节，这是所有计算机通信的基础。本章在概念上有一个巨大的跳跃，将描述一种方案，允许把多种网络技术集成成为一个协调的整体。其根本目标是隐藏所有底层网络硬件的细节，同时提供通用的通信服务。其主要结果是一个高层抽象，为所有的设计决策提供框架。后面的章节讨论如何用此抽象建立互联网通信软件所需的各层，以及软件如何隐藏底层的物理传输机制。后面的各章还要讲解应用程序如何使用形成的通信系统。

3.2 应用层互连

设计人员曾经采用两种不同的方法来隐藏网络细节，使用应用程序来处理异构，或隐藏操作系统的细节。早期的异构网络互连通过称为**应用网关**(application gateway)的应用层程序提供统一性。在这样的系统中，网络中每台计算机上执行的某个应用层程序了解本机网络连接的细节，并通过那些连接与其他计算机上的应用程序进行互操作。例如，某些电子邮件系统包括一些邮件程序，这些程序经过配置，把一个函件转发到下一台计算机上的邮件程序。从源站到目的站的路径可能经过许多不同的网络，但只要所有计算机上的邮件系统都能通过转发每个报文相互合作，就不会有什么问题，邮件发送会顺利进行。

使用应用程序来隐藏网络细节初看起来很自然，但这种方法会造成通信烦琐而且受限。给系统添加一个新功能，则意味着需要为系统中每台计算机构建一个新的应用程序。在系统中添加新的网络硬件，则意味着要为每个可能的应用修改现有的程序（或创建新程序）。在某一台机器上，每个应用程序都必须了解本机的网络连接，从而导致代码重复。

有过组网经验的用户都明白，一旦要互连成百上千的网络，没有人能写出所需的全部应用程序。此外，**一次一步**(step-at-a-time)通信模式的成功，要求沿着路径执行的所有程序都必须正确。若某个中间程序出现故障，源站和目的站都无法检测或控制这些问题。因此，使用中间应用程序的系统无法保证可靠地通信。

3.3 网络层互连

用应用层互连的一种替代方法是基于网络层互连的系统。网络层互连提供一种机制，把小数据分组从它们的源站交付到最终目的站，而不必使用中间的应用程序。交换的是小数据单元而不是文件或很长的报文，这样做有多个优点。第一，这种方案直接映射到底层网络硬件，使其非常高效；第二，网络层互连把数据通信活动从应用程序中分离出来，允许中间的计算机不必知道发送或接收通信量的应用程序，直接处理网络通信量；第三，网络连接使整个系统更加灵活，因此可以建造通用的通信工具；第四，这种方案允许网络管理员通过修改或增加一个新的网络层软件来添加新的网络技术，同时保持应用程序不发生变化。

网际互连(internetworking)是一个抽象的通信系统概念，从中可以找到设计普遍网络层互连的关键。网际互连或互联网是一个强有力的概念，它把通信这个观念从网络技术细节中分离出来，并且对用户隐藏了底层的细节。更重要的是，它引导了所有软件设计的决策，并解释了如何处理物

理地址和选择路由。在回顾网际互连的基本动机后，我们还将更详细地讨论一下互联网的性质。

我们从两个有关设计通信系统的基本观测结论着手：

- 没有一种单一的网络硬件技术可以彻底摆脱所有的约束
- 用户期待普遍互连

第一个观测结论既是经济方面的，也是技术方面的。价格低廉的局域网只能提供短距离的高速通信；长距离跨度的广域网不能提供低费用的本地通信。没有哪种网络技术可以满足所有的需求，所以我们被迫考虑多种底层硬件技术。

第二个观测结论是不言而喻的。用户最终希望在任何两点之间进行通信，尤其希望有一个不受任何物理网络边界限制的通信系统。

最终目标是建立一个支持通用通信服务的统一、协调的互连网络。在每个网络内，计算机将使用由下层技术决定的通信设施，如第 2 章所述；新软件被插入到与下层技术相关的通信机制与应用程序之间，它隐藏了底层细节，并使多个网络的集合看起来像是一个大的网络。这种互连方式称为**网际互连**（internetwork）或**互联网**（internet）。

建立一个互联网的思想符合一种系统设计的标准模式：研究人员设想实现一个高层级的计算工具，他们从现有的计算技术着手，不断加上软件层，直到得到一个能高效实现设想的高层工具。下一节通过更精确地定义目标，展示了第一步设计过程。

3.4 互联网的性质

通用服务的观念很重要，但仅用它并不能概括我们对统一互联网的想法，因为通用服务可以有很多种实现。在我们的设计中，希望对用户隐藏下层互联网的结构。也就是说，不想要求用户或应用程序在使用互联网时清楚地了解硬件互连的细节，也不想强行规定某一种网络互连拓扑。特别是在互联网中加入一个新网络时，这既不意味着要连到一个中心的交换结点上，也不意味着要在新网络和现有的所有网络间都建立直接的物理连接。我们希望能跨过多个中间网络传输数据，即使这些网络并没有直接连到源机器或目的机器上。我们要使互联网上所有机器共享一个机器标识符的通用集合（标识符可以看成是名字或地址）。

统一的互联网观念还包括用户接口中的网络无关性的思想。也就是说，我们希望用于建立通信或传输数据的一系列操作，保持与下层网络技术和目的机器的无关性。当用户创建或使用应用程序进行通信时，应当不需要了解网络互连的拓扑结构。

3.5 互联网结构

我们已经看到计算机如何连接到各个网络上。问题是：“多个网络如何互连形成一个互联网？”答案包括两个部分。从物理上讲，两个网络只能通过与它们都相连的一台计算机连在一起。但是，物理连接并不提供我们所想的互连，因为这种连接不保证这台计算机能够和其他希望与之通信的计算机相互协作。为使互联网切实可行，需要使用一些特殊的计算机，它们愿意把分组从一个网络传递到另一个网络。把两个网络互连起来并在它们之间传递分组的计算机称为**互联网网关**（internet gateway）或**互联网路由器**（internet router）^①。

考虑图 3.1 所示的由两个物理网络组成的例子。在图中，路由器 R 既连到网络 1 又连到网络 2。要使 R 充当一个路由器，它必须捕获网络 1 上那些发往网络 2 中机器的分组，并传递它们。同样，R 必须捕获网络 2 上那些发往网络 1 中机器的分组，并传递它们。

^① 原来用术语**IP 网关**（IP gateway），但是厂家已接受了**IP 路由器**（IP router）这个术语，所以本书中这两个术语可互换使用。

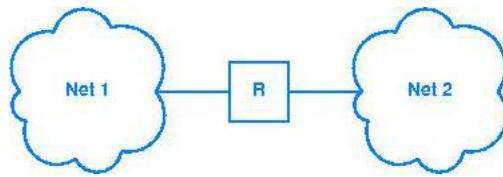


图 3.1 由一个路由器 R (IP 网关) 互连的两个物理网络

在图 3.1 中, 用云状框来表示物理网络, 因为确切的硬件信息并不重要。每个网络可以是局域网或广域网, 而且每个网络既可以连很多台计算机, 也可以只连几台计算机。

3.6 通过 IP 路由器互连

尽管图 3.1 展示了基本的连接策略, 但它还是过于简化化。在实际的互联网中包含许多网络和路由器, 每个路由器除需要了解它们所连接的网络以外, 还需要知道互联网的拓扑结构。图 3.2 显示了通过两个路由器互连的三个网络。



图 3.2 由两个路由器互连的三个网络

在这个例子中, 路由器 R₁必须把网络 1 中所有目的地是网络 2 或网络 3 的分组都传递到网络 2。对一个由许多网络组成的大型互联网来说, 路由器为分组的去向做决策的任务就变得更加复杂。

路由器的概念看来很简单, 但十分重要, 因为它提供了一种互连网络, 而不仅是互连机器的办法。事实上, 我们已经发现了在互联网中使用的互连原理:

在一个 TCP/IP 互联网中, 被称为 IP 路由器或 IP 网关的计算机提供了物理网络之间的所有互连。

读者可能会猜测, 路由器必须知道怎样把每个分组转发到目的地, 那么它们一定都是有足够的主存或辅存的大机器, 以保存互联网上连接的所有机器有关的信息。实际上, TCP/IP 互联网使用的路由器通常是小型的计算机。它们通常只有很少的磁盘存储和不大的主存。建造一个小型互联网路由器的技巧在于:

路由器在转发分组时使用的是目的网络, 而不是目的主机。

如果分组转发是基于网络的, 那么路由器需要保存的信息量与互联网中的网络数量成正比, 而不是与计算机的数量成正比。

因为路由器在互联网通信中起着关键的作用, 所以在后面的章节中还会继续讨论路由器如何工作, 以及路由器如何获得路由信息。现在, 假定互联网上每个路由器中有适用于全部网络的正确路由是可能和可行的, 还假定一个互联网上只用路由器提供物理网络之间的连接。

3.7 从用户的角度看

记住, 设计 TCP/IP 的目的是提供计算机之间的普遍互连, 独立于计算机所连接的那些特定网

络。因此，我们希望用户把互联网看成是单一的虚拟网络，所有计算机都与它相连，而不管实际的物理连接如何。图 3.3(a)告诉我们如何看待一个互联网，而不把互联网看成是由多个网络组成，这样就能简化许多细节，使用户容易把通信概念化。除了互连物理网络的路由器以外，每台计算机上还需要软件。以允许应用程序把互联网当成一个单独的物理网络来使用。

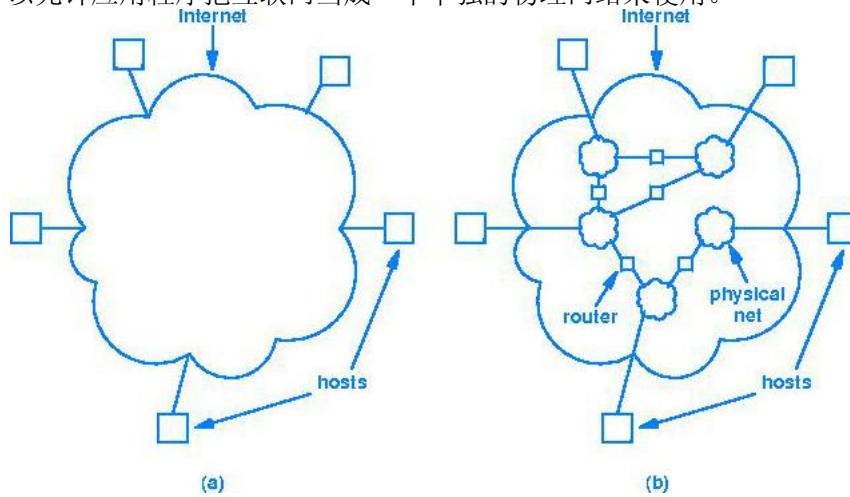


图 3.3 (a) 从用户的角度看，TCP/IP 互联网中的每台计算机看上去都是连接到了一个大型网络上；(b) 物理网络的结构和提供互连的路由器

现在，在网络层提供互连的优点就很明显了。在互联网上通信的应用程序并不知道下层连接的细节，因此它们不做任何改变就能运行在任何机器上。因为每台机器的物理网络连接细节隐藏于互联网软件中，所以在增加新的物理连接或除去现有物理连接时，只需改动互联网软件。事实上，在应用程序运行时，可以通过改变物理连接来优化互联网的内部结构。

在网络层进行通信还有一个更微妙的优点：用户不需要知道、记住或指定网络如何连接，或者网络运载什么通信量。用户可以编写出独立于下层物理连接进行通信的应用程序。事实上，网络管理员可以自由地改变下层互联网结构的内在成分，而不必改变互联网连接的大多数计算机中的应用软件（当然，当一台计算机移入一个新网络时，必须重新配置网络软件）。

如图 3.3(b)所示，路由器并不在每对网络之间提供直接的连接。通信量从一台计算机传送到另一台时，随着通信量跨越几个中间网络，可能必须通过几个路由器。因此，一个互联网涉及的网络类似于美国州际系统的高速公路：每个网络都同意处理网络间交换的通信量，从而获得在整个互联网上发送通信量的能力。一般用户不会受到影响，也不会注意到在其本地网上存在额外的通信量。

3.8 所有网络是相同的

第 2 章回顾了一些建立 TCP/IP 互联网所用的网络硬件的例子，并说明了各种不同的技术。我们已经把互联网描述成互相协作、互相连接的多个网络的集合。现在，理解一个基本概念非常重要：从互联网的角度看，任何能够传输分组的通信系统都可视为一个单一的网络，与网络延迟和吞吐量特性、最大分组长度或地理范围无关。特别是，图 3.3(b)用同样的小型云状框表示各个物理网络，这是因为 TCP/IP 把这些网络视为相同的，不管网络间存在什么差异。这个观点是：

TCP/IP 互联网协议把所有网络都看成是相同的。诸如以太网的局域网、作为主干网的广域网或者两台计算机之间的点到点链路，都可被视为一个网络。

不熟悉互联网结构的读者可能会觉得难以接受这样一个简单化的网络观点。从本质上讲，TCP/IP 定义了一个隐藏物理网络细节的抽象“网络”；我们将会认识到，这种抽象有助于使 TCP/IP 具有强大的功能。

3.9 未解答的问题

我们对互联网的概括性描述还留下了许多未解答的问题。例如，你可能想知道分配给计算机的互联网地址的确切形式，或者这种地址与第2章中描述的物理硬件地址（如以太网地址）之间有什么关系。接下来的两章将正视这些问题，描述了IP地址的格式，并举例说明计算机上的软件如何对互联网地址和物理地址进行映射。你可能还想知道当一个分组在互联网上传送时到底是什么样子；或者当分组太快到达，以至于某台计算机或路由器来不及处理时会发生什么情况。第6章将会回答这些问题。最后，你也许想知道在一台计算机上并行运行的多个应用程序，怎样才能发送和接收多个目的地不同的分组，而不会在各个传输过程中产生混乱；或者还想知道互联网路由器如何获得路由信息。所有这些问题也将得到解答。

尽管现在看来方向不太清晰，但沿着这个方向将会获悉互联网协议软件的结构及其用法。我们将研究互联网协议软件的各个部分，学习相关的概念、原理以及技术细节。首先描述的是构建互联网所基于的物理通信层。后续的各章将探讨互联网软件的各个部分，最后让读者明白如何把各个部分的内容融会贯通起来理解。

3.10 小结

互联网不仅仅是由许多计算机互连形成的网络集合。网际互连意味着互连的系统都要遵循允许每台计算机与其他各台计算机通信的约定。需要特别指出的是，即使两台计算机之间的通信路径要经过与它们都不直接相连的网络，互联网仍允许这两台计算机相互通信。只有所有计算机都接受一组通用标识符，并接受把数据运送到最终目的地的一组规程，才可能实现这种协作。

在互联网中，网络间的互连是由称为IP路由器或IP网关的计算机实现的，路由器或网关本身连接到两个或多个网络上。路由器从一个网络接收分组，并将其发送到另一个网络上，从而实现网络间分组的转发。

3.11 深入研究

我们给出的网际互连模型来自于Cerf and Cain [1983]以及Cerf and Kahn [1974]，其中将互联网描述成一组由路由器互连起来的网络，并且勾画了一个互联网协议，类似于最后发展成TCP/IP协议族的协议。有关已连接的互联网结构的更多信息可从Postel [1980]和Postel, Sunshine, and Cohen [1981]以及Hinden, Haverty, and Sheltzer [1983]中找到。Shoch [1978]讨论了同际互连命名和编址的问题。Boggs et. al. [1980]描述了Xerox PARC开发的互联网，与要研究的TCP/IP互联网相比，这是另一种互联网实现的选择方案。Cheriton [1983]描述了在V-system中涉及的网际互连。

3.12 习题

1. 在已连接的因特网上采用了什么处理器作为路由器？你对早期的路由器硬件的大小和速率感到意外吗？为什么？
2. 在你的网点上，大约有多少个网络组成互联网？大约有多少个路由器？
3. 考虑图3.3(b)所示的互联网的内部结构，哪个路由器是最关键的？为什么？
4. 改变一个路由器中的信息时需要慎重处理，因为不可能同时改变所有路由器中的信息。试研究一些算法，以保证改变或者在一组计算机上都生效，或者在任何计算机上都不改变。
5. 在一个互联网上，路由器定期交换路由表中的信息，从而使新路由器一旦出现就开始为分

组选择路由是可行的。试研究用来交换路由信息的算法。

6. 对比 TCP/IP 互联网和 Xerox 公司设计的互联网的风格。

第4章 分类的因特网地址

4.1 引言

第3章把TCP/IP互联网定义成一个虚拟网络，该网络是用路由器把多个物理网络互连起来构成的。本章开始讨论编址问题，在有助于TCP/IP软件隐藏物理网络细节并使构成的互联网看起来是一个统一实体的设计中，编址问题是其中一个基本组成部分。

4.2 通用标识符

如果一个通信系统允许任意一台主机与其他任意主机进行通信，则称该系统提供了**普遍通信服务**（universal communication service）。为了使通信系统普遍通用，需要一个全世界都能接受的方法来标志系统连接的每台计算机。

通常，主机标识符可分为**名字**（name）、**地址**（address）和**路由**（route）这三类。Shoch [1978]提议：名字标志一个对象是什么；地址标志它在哪里；路由指出怎样到达那里^①。虽然这种定义比较直观，但可能产生误导。名字、地址和路由实际上是主机标识符由高到低的三级表示方法。一般来讲，虽然软件使用地址标识符这种紧凑表示时工作效率更高，但是人们通常更愿意使用可读的名字来标志机器。地址或名字都可以用来作为TCP/IP的通用主机标识符。研究人员决定把紧凑的二进制地址标准化，这使得路由选择之类的计算更加高效。现在仅讨论二进制地址的问题，二进制地址与可读名字之间的映射问题以及如何将地址用于分组转发的问题都推迟到后面再讨论。

4.3 最初的分类编址方案

让我们把互联网想像成一个类似其他任何物理网络的大网络。当然，它们的区别在于，互联网是一个由其设计者想像出来的虚拟结构，并完全由软件实现。因此，设计人员可以自由地选择分组格式、分组大小、地址以及交付技术等，不受硬件的支配。对于地址，TCP/IP的设计人员选择了一种类似于物理网络编址的方案。在这个方案中，互联网上的每台主机都分配了一个32比特的整数地址，称为**网际协议地址**（Internet Protocol address）或**IP地址**（IP address）。互联网编址的巧妙之处在于：为了使转发更高效，用来作为地址的整数是经过精心选择的。特别是，一个IP地址既为某台主机所连网络的标志编码，又为该网络上某个唯一主机的标志编码。可以总结如下：

TCP/IP互联网上的每台主机都分配了一个唯一的32比特互联网地址，该地址用在与该主机的所有通信中。

IP地址的细节有助于澄清一些抽象的概念。目前先给出一个简化的概念，以后再加以扩充。在最简单的情况下，连接到互联网的每台主机都分配了一个32比特的通用标识符作为其互联网地址。IP地址的前缀标志出一个网络，也就是说，同一网络上所有主机的IP地址共享同一前缀。

从概念上讲，每个地址都是成对出现的，表示为（网络号，主机号），即（netid, hostid），其中netid标志某个网络，hostid标志该网络上的某台主机。但实际上，在整个互联网中，IP地址的

^① 指明对象可在哪里找到的标识符又称为**定位符**（locator）。

前缀和后缀的划分是不统一的，因为设计者没有给出单一的划分界限。在最初被称为**分类**(classful)的编址方案中，每个IP地址具有图4.1所示的前三种形式之一^①：

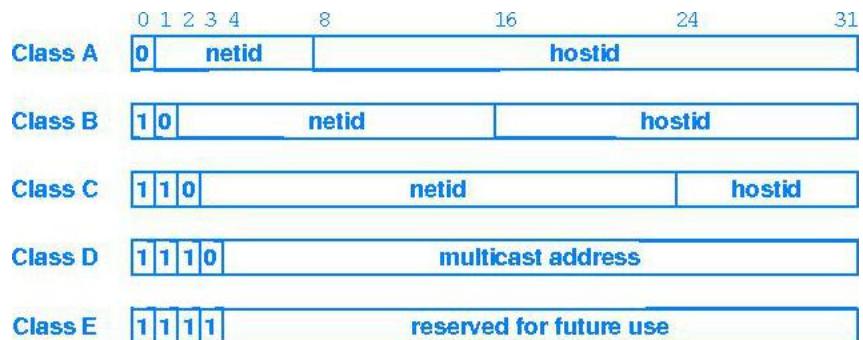


图4.1 最初的分类编址方案中因特网（IP）地址的五种形式。三种主要类别，即A类、B类和C类，可以用前三个比特进行区分

在分类编址机制中，每个地址可被认为是**自标志**(self-identifying)的，因为仅从地址本身就可以计算出前缀和后缀之间的边界，不必再参考其他外部信息。而且，可从地址的最高三个比特确定该地址属于哪一类，用最高两位就足以区分三种主要类别。A类地址，用于拥有超过 2^{16} (即65,536)台主机的网络，该类地址将7比特用于网络号，24比特用于主机号。B类地址用于中等规模的网络，它的主机数量在 2^8 (即256)到 2^{16} 之间，该类地址将14比特用于网络号，16比特用于主机号。最后，C类地址，用于主机数量少于 2^8 的网络，该类地址将21比特用于网络号，只有8比特用于主机号。注意，因为IP地址就是用这种方法定义的，所以可以很快从地址中抽取出主机号或网络号部分。高效率对于路由器而言特别重要，而路由器在决定把一个分组发往何处时使用了地址的网络号部分，地址的快速抽取有助于提高路由器的效率。下面将讨论近年来对编址方案的改动和扩充，然后再返回来讨论高效的路由查找。

4.4 用地址指明网络连接

为了简化讨论，可以说一个互联网地址标志了一台主机，但这样讲并不准确。考虑一个连接着两个物理网络的路由器。如果地址是一个网络标识符和一个主机标识符的编码，那么怎样给这个路由器分配单一的IP地址呢？事实上无法做到。类似的情况还有，例如当普通的计算机有两个或多个物理连接时，这些计算机被称为**多归属主机**(multi-homed host)，机器的每个网络连接都必须分配一个地址。这一思想是IP编址不可缺少的部分：

因为IP地址是对某个网络及该网络上某台主机的编码，所以一个地址并非指明了某台计算机，而是指明了计算机到网络的某条连接。

因此，一个连接着n个网络的路由器有n个不同的IP地址，每个地址对应着一条网络连接。

4.5 网络地址和定向广播地址

前面已经提到了在互联网地址中网络信息编码的主要优势：它使高效的转发成为可能。另一优势是，互联网地址既可以指向主机，也可以指向网络。按照约定，主机号“0”从来不分配给某个单独的主机，而主机号部分为0的IP地址被用来指向网络本身。总而言之：

^① 第四种形式是为互联网多播保留的，将在以后讨论，现在仅说明那些指明个体对象的地址形式

互联网地址可以用来指向单个主机，也可以用来指向网络。按照约定，主机号的所有比特都为“0”的地址是保留给该网络本身的。

互联网编址方案的另一个重要优势是：它包含一个指向网络中所有主机的**定向广播地址** (directed broadcast address)。根据标准，主机号全由“1”比特组成的地址被保留用于定向广播^①。当分组发送到这样的地址时，从源站沿互联网传输的是该分组的一份副本。沿路径的路由器在选择路径时使用的是地址的网络号部分，它们不查看主机部分。一旦该分组到达了连在目的网络上的路由器，该路由器检查地址的主机部分，以确定如何交付该分组。如果它发现地址为全“1”，则路由器会把该分组广播给该网络上的所有主机。

在一些网络技术中（如以太网），广播和单播传输一样有效；而在另外一些网络中，广播是通过由软件发送各个副本给网络上的每台主机来实现的。因此，使用 IP 定向广播地址并不能保证广播交付的可用性或高效性。总而言之：

IP 地址可被用来指明一个定向广播，这种形式的广播可把分组发送给网络上的所有计算机；如果可能，这种地址会被映射成硬件广播。按照约定，一个定向广播地址有一个有效的网络号，并且其主机号的所有比特都为“1”。

4.6 受限广播

刚才描述的广播地址称为**定向的** (directed)，因为它既包含一个有效的网络 ID，又包含广播主机号。定向广播可跨越一个互联网发送，因为传输路径沿线的路由器只使用地址的网络号部分来判断如何转发数据报。定向广播地址提供了一种强大（但也有点危险）的机制，允许某个远端系统在指定的网络上广播发送一个分组。为避免潜在问题的发生，许多网点对路由器进行配置，让其拒绝所有定向广播的分组。

从寻址的角度来看，定向广播的主要缺点是它需要理解网络地址。另一种形式的广播地址称为**受限广播地址** (limited broadcast address) 或**本地网广播地址** (local network broadcast address)，它为本地网提供了一个独立于分配 IP 地址的广播地址。本地广播地址由 32 个“1”组成（因此，它有时被称为“全 1”广播地址）。下面将会看到，一台主机在知道自己的 IP 地址或本地网络的 IP 地址前缀之前，就可以在启动过程中使用受限广播地址。但是，主机一旦知道了本地网络的正确 IP 地址，更愿意使用定向广播。

作为一般的规则，TCP/IP 协议会把广播限制在范围尽可能小的一组机器上。在有关子网编址的章节中将讨论这个规则如何影响多个共享地址的网络。

4.7 全零地址

32 个“0”比特组成的地址被保留用于这样的情况：某个主机需要通信，但它还不知道自己的 IP 地址（即短暂的启动期间）。特别是我们将看到，主机为获得一个 IP 地址，将发送一个数据报给受限广播地址，并用全零地址来标志自己。接收方知道发送数据报的主机还没有一个 IP 地址，就会用一种特殊方法来发送回答。

^① Berkeley UNIX 带有的 TCP/IP 代码早期版本不恰当地将全零地址用于广播。由于这个错误仍然存在，所以 TCP/IP 软件常包含一个选项。允许使用全零代表定向广播。

4.8 子网和无类型扩展

到目前为止，所描述的编址方案要求每个物理网络都有一个唯一的网络前缀。尽管最初的设计确实是这样的，但它持续的时间并不长。在 20 世纪 80 年代，随着局域网技术的逐渐流行，每个物理网络需要一个唯一的前缀显然会迅速耗尽地址空间。因此，研究人员开发了一种扩展编址方案来节省使用网络前缀。这种方法称为**子网编址** (subnet addressing)，允许多个物理网络共享一个网络前缀。

20 世纪 90 年代，研究人员设计出了第二种扩展方式，它不受分类层次的约束，并允许在地址中的任意位置进行前缀和后缀划分。这种方法称为**无类型编址** (classless addressing)，能够更彻底地利用地址空间。

第 9 章将讲解子网和**超网编址** (supernet addressing) 扩展的细节。目前只需要知道编址方案已经进行了扩展，并且本章描述的最初分类方案已不再是最广泛使用的方法。

4.9 IP 多播地址

除了**单播交付** (unicast delivery，即一个分组被交付给一台计算机) 和**广播交付** (broadcast delivery，即一个分组被交付给某个指定网络上的所有计算机) 以外，IP 编址方案还支持一种特殊形式的多点交付，称为**多播** (multicasting)。在这种形式的交付中，分组被交付给一个特定的主机子集。在硬件技术支持多播交付的网络中，IP 多播尤其有用。第 16 章将详细讨论多播编址和交付。目前，我们知道 D 类地址被保留用于多播就足够了。

4.10 互联网编址中的缺陷

在互联网地址中对网络信息进行编码具有某些缺点。最明显的缺点是该地址指向网络连接，而不是指向主机：

如果一台主机从一个网络转移到另一个网络，它的 IP 地址必须改变。

为了便于理解这种编址方案的结果，设想旅行者遇到的这样一种情况：他们想让自己的计算机断连，以便在旅途中携带，到达目的地之后再重新把计算机连到因特网上。不能给这台个人计算机分配永久 IP 地址，因为 IP 地址标识了机器所连到的网络。第 18 章表明了 IP 编址方案怎样使移动性 (mobility) 成为一个复杂问题。

互联网编址方案的另一个缺陷来自早绑定，即一旦选择了某个前缀长度，就确定了网络上主机的最大数量。如果网络增长超过了主机数的上界，就必须选择一个新前缀，而网络上的所有主机就必须重新编号。虽然**重编号** (renumbering) 看起来可能像是小问题，但改变网络地址可能会非常费时，而且难以调试。

在研究转发时，互联网编址方案中最重要的漏洞才会变得非常明显。但是，这里先要对其进行简单介绍。我们已经讲过，转发将基于互联网地址，地址的网络号部分用于做出转发选择。考虑一个与互联网有两条连接的主机，这种主机拥有的 IP 地址必须超过一个。下面这句话是正确的：

因为转发使用了 IP 地址的网络部分，所以在分组被发送到具有多个 IP 地址的主机时，分组传送采用的路径与所使用的地址有关。

这句话暗含的意思令人不可思议。人们把每台主机想成是一个实体，并希望用一个名字标志它。但是，人们常会惊讶地发现仅知道名字还不够，甚至更令他们惊讶的是，使用多个名字发送的分组，

其行为并不相同。

互联网编址方案的另一个令人惊奇的结果是：仅知道目的站的一个 IP 地址可能还不够，使用那个地址可能根本无法到达目的站。考虑图 4.2 所示的互联网例子。图中，主机 A 和 B 都连到网络 1 上，它们通常使用这个网络直接进行通信。因此，主机 A 上的用户通常应该用 IP 地址 I_3 来访问主机 B。从 A 到 B 还存在另一条通过路由器 R 的路径，而且只要 A 把分组发送到 IP 地址 I_5 (B 在网络 2 上的地址)，就要使用这条路径。现在假设 B 到网络 1 的连接出现了故障，但主机 B 本身还在运行（例如，B 和网络 1 之间的一根电缆断了）。那么，虽然主机 A 上指定地址 I_5 为目的地的用户可以到达 B，但是指定 IP 地址 I_3 为目的地的用户却无法到达 B。在后面讨论转发和名字绑定时，这些由命名和编址产生的问题还会再次出现。

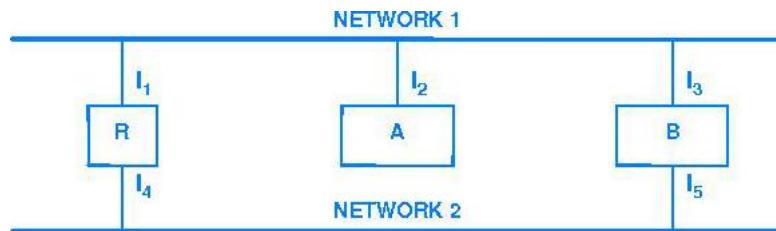


图 4.2 含有一个多归属主机 B 的互联网的例子，演示 IP 编址方案中存在的一个缺陷。如果接口 B 断连，A 必须使用地址 I_5 到达 B，通过路由器 R 来发送分组

4.11 点分十进制记法

在向人们传递信息的应用程序或技术文档中，把 IP 地址写成由小数点分隔的 4 个十进制整数，每个整数给出了 IP 地址的一个八位组的值^①。因此，32 比特互联网地址

10000000 00001010 00000010 00011110

写成

128.10.2.30

本书后面的部分都将使用点分十进制记法来表示 IP 地址。事实上，大多数显示或要求输入 IP 地址的 TCP/IP 软件都使用点分十进制记法。例如，诸如 Web 浏览器之类的应用程序，允许用户输入一个点分十进制值替代计算机名。作为点分十进制表示法的例子，图 4.3 中的表总结了每个地址类的点分十进制值的范围。

Class	Lowest Address	Highest Address
A	1.0.0.0	127.0.0.0
B	128.0.0.0	191.255.0.0
C	192.0.0.0	223.255.255.0
D	224.0.0.0	239.255.255.255
E	240.0.0.0	255.255.255.254

图 4.3 对应于每个 IP 地址类的点分十进制值的范围。某些值是为特殊用途保留的

4.12 环回地址

从图 4.3 中的表可以看出 IP 地址空间分成了几类，但每个类中的地址值并没有全部参与分配。例如，B 类地址中的最低地址前缀（128.0.0.0）不会被分配；B 类地址中最先被分配的地址前缀是 128.1.0.0。类似地，C 类地址中最先被分配的地址前缀是 192.0.1.0。特别需要说明的是网络前缀

^① 点分十进制记法有时又称为点分四组记法 (dotted quad notation)。

127.0.0.0，这个值属于 A 类范围，却保留用于环回（loopback），并用于测试 TCP/IP 以及本机进程间的通信。如果某个程序使用环回地址作为目的地址，计算机中的协议软件在处理数据时不会把通信量发送到任何网络；发送到网络号“127”的分组永远不会出现在任何网络上。此外，主机或路由器应该永远不为网络号“127”传播路由或可达性信息，它不是一个网络地址。

4.13 特殊地址约定小结

在实际应用中，IP 只使用少数由若干个 0 或若干个 1 组合成的特殊地址。图 4.4 列出了组合的可能形式。如图中注释所示，只有在启动过程中才可以使用全 0 的网络地址，这样做是为了允许计算机在不知道自己地址的情况下发送数据报。一旦机器知道了自己的正确网络地址和 IP 地址，就不能再使用网络前缀“0”。在任何情况下，全 0 地址永远都不能用来作为目的地址。

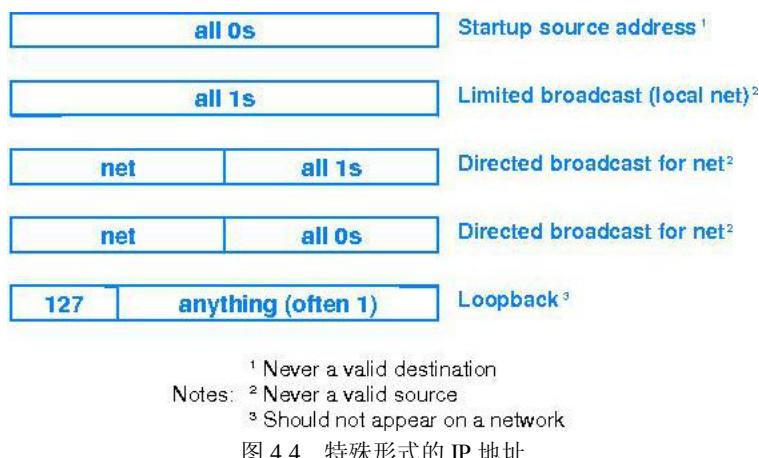


图 4.4 特殊形式的 IP 地址

4.14 因特网编址管理机构

在某 TCP/IP 互联网中使用的每个网络地址前缀必须是唯一的。某个单位使用 TCP/IP 技术建立彻底私用的互联网（例如，不与全球因特网连接的互联网）时，可以自己分配地址前缀，不必考虑其他单位分配地址的情况。但是，如果某个单位要把自己的互联网连接到全球因特网，它就一定不能使用分配给其他单位的地址前缀。

为了确保地址的网络部分在全球因特网上是唯一的，所有因特网地址都由一个中央管理机构进行分配。最初，**因特网赋号管理局**（Internet Assigned Number Authority，简称 IANA）控制着所分配的编号，并制定政策。从因特网出现开始到 1998 年秋天，一直是由 Jon Postel 一个人负责 IANA 的运转并分配地址。1998 年底，当他去世以后，组建了一个新组织来处理地址分配问题。这个组织名为**因特网名字与号码指派协会**（Internet Corporation For Assigned Names and Numbers，简称 ICANN），它负责制定政策，并为协议中使用的名字和其他常量分配值，还要分配地址。

大多数单位从不会直接和中央管理机构联系。这些单位为了把自己的网络连接到全球因特网，通常只会和一个本地的**因特网服务提供者**（Internet Service Provider，简称 ISP）联系。ISP 除了给这些单位提供与因特网其余部分的连接，还要为每个客户的网络获得一个有效地址前缀。实际上，许多本地 ISP 是更大型的 ISP 的客户，当某个客户申请一个地址前缀时，本地 ISP 只是从更大型的 ISP 那里获得一个前缀。因此，只有最大型的 ISP 需要和 ICANN 授权管理成块地址的地址注册商（ARIN, RIPE, APNIC, LATNIC 或 AFRINIC）之一联系。

需要注意的是，中央管理机构只分配地址的网络部分。这就意味着，一旦某个单位获得了一个

网络的前缀，该单位就可以自主决定如何给网络上的每台主机分配唯一的网络地址，而不必再与中央管理机构联络。

4.15 保留的地址前缀

我们应该如何给某个专用的内联网分配地址呢（例如，在一个没有连接全球因特网的互联网上）？从理论上讲，我们可以使用任何地址。例如，在全球因特网上，A类地址 9.0.0.0 已分配给 IBM 公司，地址 12.0.0.0 已分配给 AT&T 公司。虽然这些地址可以用在专用的网络上，但经验表明这样做很危险，因为泄漏到全球因特网的分组看起来都像是来自于有效的源站。为了避免专用互联网上使用的地址和全球因特网上使用的地址之间存在冲突，IETF 保留了几个地址前缀，并建议在专用互联网上使用这些前缀。因为保留的这组前缀既包括分类地址值，也包括无类型地址值，这些内容将在第 9 章里详细阐述。

4.16 示例

为了阐明 IP 编址方案，我们以普度（Purdue）大学计算机科学系的两个网络为例。图 4.5 显示了两个网络的地址，并描绘了路由器是如何互连网络的。

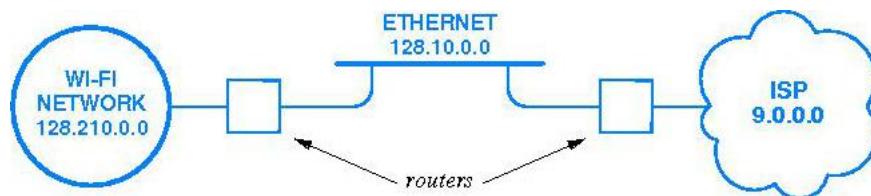


图 4.5 两个网络到因特网主干网的逻辑连接。每个网络都分配有一个 IP 地址

这个例子中有三个网络，分配给它们的网络号为：因特网服务提供者（9.0.0.0）、一个以太网（128.10.0.0）和一个 Wi-Fi 网络（128.210.0.0）。按照图 4.3 中的表，ISP 地址属于 A 类，另外两个地址都属于 B 类。

图 4.6 显示了这两个网络所连接的主机以及给每条网络连接分配的因特网地址。在图中，连到网络上的四台主机分别标记为 Arthur, Merlin, Guenevere 和 Lancelot；Taliesyn 是连接以太网和 Wi-Fi 网络的路由器；Glatisant 是把网络结点连接到一个 ISP 的路由器。

主机 Merlin 既有与以太网的连接，又有与 Wi-Fi 网络的连接，所以它可以直接到达这两个网络上的目的站。路由器（如 Taliesyn）和多归属主机（如 Merlin）之间的区别在于配置不同：路由器被配置用于在两个网络之间转发分组，而主机可以使用两个网络但不转发分组。

如图 4.6 所示，必须给每个网络连接分配一个 IP 地址。Lancelot 只与以太网连接，只给它分配了 128.10.2.26 作为 IP 地址。Merlin 的 128.10.2.3 地址用于主机和以太网的连接，128.210.0.3 地址用于主机和 Wi-Fi 网络的连接；无论由谁分配地址，通常都会给地址的低位字节选择相同的值。但是，分配给路由器 Taliesyn 的地址却没有遵守这个惯例。例如，Taliesyn 的地址 128.10.2.6 和 128.210.0.50 就是两个不相关的数字串。IP 并不关心一台计算机的不同地址的点分十进制格式中是否有一些相同字节。但是，网络技师、管理员和系统管理员可能在维护、测试和调试时需要使用地址。选择让一台计算机的所有地址以相同的八位组结束，会使人更易于记忆或猜测某个特定接口的地址。

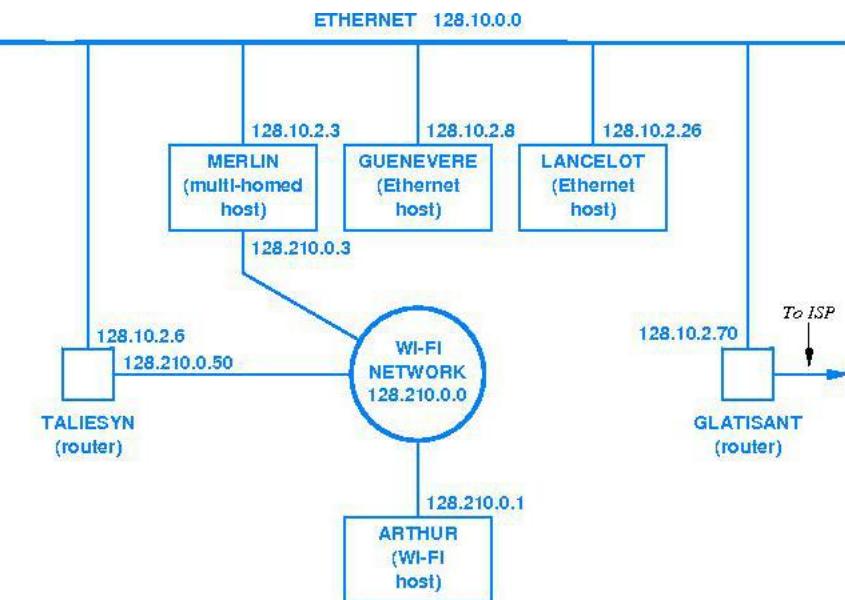


图 4.6 图 4.5 中三个网络所连主机和路由器的 IP 地址分配

4.17 网络字节顺序

为了创建一个独立于任何特定厂家机器结构或网络硬件的互联网，软件必须为数据定义一种标准表示形式。例如，当一台计算机上的软件向另一台计算机发送一个 32 比特的二进制数时，物理传输硬件把比特序列从第一台机器传送到第二台机器上，序列的顺序没有发生改变。但是，并非所有体系结构都按相同的方式来存储 32 比特整数。在一些机器（称为 Little Endian，即小数在前）中，最低存储地址装有整数的低位字节。在其他机器（称为 Big Endian，即大数在前）中，最低存储地址装有整数的高位字节。还有其他一些机器以若干组 16 位字的方式存储整数，最低存储地址容纳低位字，但要交换字中的两个字节。因此，直接把字节序列从一台机器复制到另一台机器上可能会改变数值。

由于互联网分组携带的一些二进制数指出了目的地址和分组长度等信息，因此在互联网中把整数的字节顺序标准化就显得尤为重要。发送方和接收方必须搞清楚分组中的这些量。TCP/IP 协议定义了一种**网络标准字节顺序** (network standard byte order)，所有机器都必须使用这种顺序来表示互联网分组中的二进制字段，从而解决了字节顺序的问题。每台主机或路由器在发送一个分组之前，必须把二进制数据项从本地表示转换为网络标准字节顺序，并且在分组到达目的站后，又把它们从网络标准字节顺序转换为特定主机的顺序。很自然地，分组中的用户数据字段不受这个标准的支配，因为 TCP/IP 协议并不知道要运送的是什么数据——应用程序的编写者可自由设计自己的数据表示方式和转换方式。当发送整数值时，许多程序员还是选择遵循 TCP/IP 字节顺序。当然，只调用应用程序的用户不需要直接处理字节顺序问题。

在关于字节顺序的互联网标准中，规定最先发送整数的最高有效字节（即大数在前风格）。思考一下，当一个分组从一台机器传送到另一台机器时，该分组中的二进制整数的最高有效字节距离分组的开头最近，而最低有效字节距离分组的末尾最近。关于究竟采用哪种数据表示有很多争论，互联网标准仍不时受到抨击。特别是有一些支持修改标准的人指出，虽然在过去定义标准时大多数计算机还是大数在前的，但现在大多数计算机是小数在前的。但是，大家都公认有一个标准是非常重要的，至于标准究竟采用哪种格式则是较为次要的问题。

4.18 小结

TCP/IP 使用 32 比特的二进制地址作为通用的机器标识符，这种标识符称为**网际协议**（Internet Protocol）地址或 IP 地址，可划分为两部分：前缀可识别计算机所连到的网络，后缀为该网络上的计算机提供唯一的标识符。最初的 IP 编址方案称为分类编址方案，每个前缀被分配为三个主要类别之一。地址最前面的比特定义了地址所属的类，各个地址类的容量不同。分类方案可为 127 个各包含上百万台主机的网络编址，或为几千个各包含上千台主机的网络编址，或为上百万个最多包含 254 台主机的网络编址。为了使地址更容易被人理解，可用点分十进制记法书写地址，把 4 个八位组的值写成十进制数，每个数之间用小数点分开。

因为 IP 地址对网络标识符以及该网络上特定主机的标识符进行了编码，所以转发效率很高。IP 地址的一个重要属性是它们与网络连接有关。有多个连接的主机有多个地址。互联网编址方案的一个优点是，这种形式包含的地址可用于特定主机、某个网络或者某个网络上的所有主机（广播）。IP 编址方案的最大缺陷是，如果某台机器有多个地址，当这台机器到指定接口的路径不存在时（例如个别网络无法使用），只知道该机器的一个地址可能还不够，这可能导致分组无法到达该机器。

为了能够在机器之间交换二进制数据，TCP/IP 协议要求字段内的整数采用一种标准的字节排序。主机在发送分组之前必须把所有二进制数从内部格式转换成网络标准字节顺序，并在接收到分组后再将其从网络标准字节顺序转换成内部格式。

4.19 深入研究

本章讨论的互联网编址方案可以在 Reynolds and Postel [RFC 1700] 中找到；更多的信息可以在 Kirkpatrick et. al. [RFC 1166] 中找到。

这些年来对因特网编址方案进行过几次重要补充，后面的章节将详细讨论这些内容。第 9 章讨论了一种重要的扩展方案，称为**无类型编址**（classless addressing），它允许在地址中的任意位置划分前缀和后缀。另外，第 9 章还考察了因特网地址标准的一个重要组成部分，称为**子网编址**（subnet addressing）。子网编址允许在多个物理网络中使用一个网络地址。第 16 章继续探讨 IP 地址，描述怎样为**互联网多播**（internet multicast）分配 D 类地址。

Cohen [1981] 详细阐述了比特和字节排序，并介绍了术语“大数在前”和“小数在前”。

4.20 习题

1. 精确地计算一下可以存在多少个 A 类、B 类和 C 类网络？在每个类的一个网络中确切地可以有多少台主机？要注意排除广播地址、D 类地址和 E 类地址。
2. 机器可读的地址分配表有时又称为互联网主机表（host table）。如果你的网点有一个主机表，查一查已分配的 A 类、B 类和 C 类网络号有多少？
3. 在你的网点上，连接到每个局域网的主机有多少台？在你的网点上有没有 C 类地址不够哪个局域网使用？
4. IP 编址方案和美国的电话编号方案之间的主要区别是什么？
5. 全世界有多个地址注册商合力分发成块的 IP 地址。查一查这些地址注册商如何保证分发给某个 ISP 的地址不会与其他 ISP 得到的地址产生重叠？
6. 网络标准字节顺序与你的本地机器的字节顺序是否相同？
7. 在你的国家中，如果给每个家庭都分配一个唯一的 IP 地址，共需要多少个 IP 地址？全世界呢？IP 地址空间够用吗？

第5章 因特网地址到物理地址的映射（ARP）

5.1 引言

第4章描述了给每台主机分配一个32比特地址的TCP/IP编址方案，还指出互联网的行为就像一个虚拟网络，在发送和接收分组时只使用分配的地址。第2章回顾了几种网络硬件技术，特别指出在某个物理网络上的两台机器，只有当它们相互知道对方的物理网络地址时，才能进行通信。当主机或路由器需要在一个物理网络上发送分组时，它们如何把一个IP地址映射为正确的物理地址，这一内容目前还未提及。本章将讨论这一映射，并说明如何针对两种最常用的物理网络编址方案加以实现。

5.2 地址解析问题

考虑连接到同一物理网络的两台机器A和B。它们分配得到的IP地址分别是 I_A 和 I_B ，物理地址分别是 P_A 和 P_B 。无论下层网络硬件提供什么样的物理地址方案给物理网络使用，最终都必须由物理网络来实现通信。然而，我们的目标是设计这样的软件，把物理地址隐藏起来，并让更高层的程序只利用因特网地址进行工作。例如，假设机器A和机器B都连接到一个物理网络上，A要在该网络上把分组发送给B，但是A只知道B的因特网地址 I_B 。由此产生的问题是：A怎样把B的因特网地址映射为B的物理地址 P_B ？

在从源站到最终目的站的路径上，每一步都要进行地址映射。特别要指出两种情况。第一，在交付分组的最后一步，分组必须通过一个物理网络送达它的最终目的站。发送分组的计算机在可能实现传输前，必须把最终目的站的因特网地址映射为目的站的物理地址；第二，沿着从源站到目的站的路径，除了最后一步，在任意一点都必须把分组发送到一个中间路由器。因此，发送方必须把中间路由器的因特网地址映射为一个物理地址。

把高层地址映射为物理地址的问题称为**地址解析问题** (address resolution problem)，它已经有了几种解决方法。一些协议族在每台机器上保存高层地址和物理地址的对照表，其他协议则把硬件地址编码到高层地址中。无论使用哪种方法，多少会使我们感到高层寻址有点麻烦。本章讨论TCP/IP协议所采用的两种地址解析技术，并且说明了它们各自适用的场合。

5.3 两种类型的物理地址

物理地址有两种基本类型：固定的长地址（如以太网所用的地址）和易配置的短地址。对于这两种地址，TCP/IP都能够处理，但以太网的广为流行意味着目前几乎所有网络硬件都使用长地址。因此，本章只简单讨论短地址的解析，而将重点放在长地址上。

5.4 通过直接映射进行解析

考虑使用可配置的小整数作为硬件地址的网络硬件。每当一台新计算机加入这样的网络，系统管理员可以选择一个硬件地址来配置计算机的网络接口卡。唯一需要重视的规则是不能让两台计算机具有相同的地址。为了使地址分配容易进行并且安全，管理员通常会顺序分配地址：给网络上的

第一台计算机分配地址 1，给第二台计算机分配地址 2，以此类推。

对于这种网络硬件，其地址解析易于实现的关键是，只要留意到可以自由选择 IP 地址和物理地址，那么总可以选择让两个地址中的某些部分相同，特别是可以让一台计算机的 IP 地址的主机号部分与硬件地址相同。例如，假定有一个网络的前缀是 192.5.48.0，网络前缀占了前三个八位组。给网络上的第一台计算机指派的是硬件地址 1 和 IP 地址 192.5.48.1，给第二台计算机指派的是硬件地址 2 和 IP 地址 192.5.48.2，以此类推。这样做的本质是，网络上每个 IP 地址已把计算机的硬件地址编码到低位的八位组中。

如果 IP 地址含有硬件地址的编码，地址解析就变得微不足道了。在上面的例子中，如果软件获得网络上某台计算机的 IP 地址（如 192.5.48.3），可以通过提取低位的八位组计算相应的硬件地址。我们把这样完成的解析称为直接映射（direct mapping）。由于此处示范的映射只需要少数几条机器指令，因此它的计算效率较高，而且不涉及外部数据的引用。最后，不需要改变现有的地址分配或把地址分配信息传播给现有的计算机，就可以在网络中添加新的计算机。

从数学上讲，直接映射意味着选择一个函数 f ， f 把 IP 地址映射为物理地址，而解析 IP 地址 I_A 意味着计算

$$P_A = f(I_A)$$

尽管可以选择多种不同于上面例子的映射，但我们希望 f 的计算是高效的，并且希望 f 的选择容易让人理解，因此更偏爱 IP 地址和硬件地址之间的关系显而易见的方案。

5.5 通过动态绑定进行解析

虽然直接映射是高效的，但它不能用于使用以太网编址的硬件技术。若想知道原因，可以回顾第 2 章的内容，每个以太网接口在设备生产时都被指派了一个 48 比特物理地址。这样做的后果是，当硬件出现故障并需要替换某个以太网接口时，机器的物理地址就会发生改变。此外，因为以太网地址的长度为 48 位，所以无法把它编码成 32 比特 IP 地址^①。

以太网是具有广播能力的网络，为解决类似网络的地址解析问题，TCP/IP 协议的设计人员找到了一种创新办法。使用这个办法，当把新主机或路由器添加到网络中时，不需要重新编译代码，也不需要维护一个中央数据库。为了避免维护中央数据库，设计人员选择使用一个低层协议来动态地绑定地址。这个协议称为**地址解析协议**（Address Resolution Protocol，简称 ARP），它提供了一种既比较高效又易于维护的机制。

如图 5.1 所示，使用 ARP 进行动态解析的思路非常简单：当主机 A 要解析 IP 地址 I_B 时，它广播一个特殊的分组，请求 IP 地址为 I_B 的主机用其物理地址 P_B 做出响应。包括 B 在内的所有主机都接收到这个请求，但只有主机 B 识别出它的 IP 地址，然后发出一个含有自己的物理地址的回答。当 A 收到回答后，就使用该物理地址把互联网分组直接发送给 B。我们可以概括为：

通过地址解析协议，即 ARP，主机只要知道同一物理网络上某个目的主机的 IP 地址，就可以找到该目的站的物理地址。

^① 由于直接映射比动态绑定更便捷和高效，研究人员正在设计下一代的 IP 地址，允许 48 位硬件地址被编码到 IP 地址中。

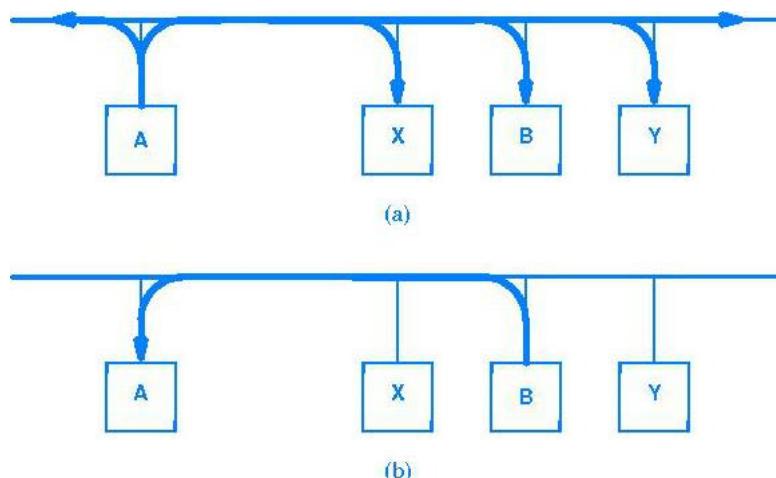


图 5.1 ARP 协议。为了根据 B 的 IP 地址 I_B 确定它的物理地址 P_B , (a) 主机 A 向网上的所有机器广播一个包含 I_B 的 ARP 请求; (b) 主机 B 用一个包含 (I_B, P_B) 对的 ARP 回答做出响应

5.6 地址解析高速缓存

A 向 B 发送一个分组之前先发出一个到达 B 的广播, 这种做法看起来似乎有些愚蠢。或者, A 广播一个问题“我怎么才能和你取得联系?”而不是直接广播要发送的分组, 这种做法看起来似乎更愚蠢。但是, 这样做有一个重要的理由: 如果一台机器每次都使用广播把分组发送给另一台机器, 由于网络上的每台机器都必须接收和处理广播分组, 这样做付出的代价过于高昂。

为了降低通信费用, 使用 ARP 的计算机维护着一个高速缓存, 存放最近获得的 IP 到物理地址的绑定。也就是说, 当一台计算机发送一个 ARP 请求并接收到一个 ARP 回答后, 它就在自身的高速缓存中保存 IP 地址及对应的物理地址信息, 以便于日后的查询。当计算机要传输一个分组时, 在它发出 ARP 请求之前, 总是先在高速缓存中寻找所需的绑定。如果计算机在它的 ARP 高速缓存中找到了所要的绑定, 就不需要在网络上广播 ARP 请求。这样, 当一个网络上的两台计算机进行通信时, 先以 ARP 请求和回答开始, 然后就会反复传送分组, 不再需要为任何一方使用 ARP。经验表明, 由于大多数网络通信要传送的分组不止一个, 即使是一个小规模的高速缓存也很值得使用。

5.7 ARP 高速缓存超时

ARP 高速缓存提供了**软状态** (soft state) 的一个例子, 这种技术常用在网络协议中。软状态描述了这样一种情况: 信息可能在没有任何警告的情况下变得“陈旧”。在 ARP 的应用场合下, 假设有两台计算机 A 和 B 都连接到一个以太网上, A 已经发送了一个 ARP 请求, B 已经做出了回答。再假设 B 在发送回答后出现故障。计算机 A 不会接到任何关于 B 出现故障的通知。而且, 因为在 A 的 ARP 缓存中已经有了 B 的地址绑定信息, A 将继续把分组发送给 B。以太网硬件没有提供 B 不在线的指示, 因为以太网并不能保证交付。这样, A 无法知道自己的 ARP 缓存中的信息何时变得不正确。

为了能适应软状态, 协议让信息的拥有者负责绑定信息的正确性。一般情况下, 实现软状态的协议使用了计时器, 当计时器超时后将删除状态信息。例如, 当地址绑定信息被放入 ARP 缓存时, 协议需要设置一个计时器, 其典型的超时时间是 20 分钟。当计时器超时后, 必须删除这些信息。删除后有两种可能性。如果没有更多分组要发送到相应的目的站, 则什么事也没有。如果有一个分组必须发送到该目的站, 而在缓存中又找不到绑定, 则计算机将按照正常的过程, 广播一个 ARP

请求并获得绑定。如果目的站仍可达，绑定将再次被放入 ARP 缓存。如果目的站不可达，发送方就会发现目的机器已下线。

在 ARP 中使用软状态既有优点也有缺点。主要的优点在于自治性。第一，一台计算机能够决定自己 ARP 缓存中的信息应在何时重新变得有效，与其他计算机无关。第二，发送方不需要通过与接收方或第三方的成功通信来确定绑定已变得无效；如果目标方不响应 ARP 请求，发送方将宣告目标机器已宕机。第三，这种方法不依靠网络硬件提供可靠传送。软状态的主要缺点在于时延，如果计时器的超时时间是 N 秒，发送方直到 N 秒以后才能检测到接收方已出现故障。

5.8 ARP 的改进

协议中包括了几处对 ARP 的改进。第一，如果主机 A 因为需要给主机 B 发送分组而准备使用 ARP，那么很有可能主机 B 在不久之后需要给 A 发送分组。由于预感到 B 的这种需要，为避免额外的网络通信量，A 在发给 B 的请求中包含了 A 的 IP 到物理地址的绑定。B 从请求中提取出 A 的绑定，把它保存在自己的 ARP 缓存中，然后给 A 发送回答；第二，因为 A 广播它的首次请求，所以网络上的所有机器都接收到该请求，并且都可从中提取出 A 的 IP 到物理地址的绑定，然后用此绑定来更新自己的 ARP 缓存中的相应绑定；第三，当一台计算机的某个主机接口被替换时（例如，由于硬件发生故障），它的物理地址也就变了。如果网络上有些计算机的 ARP 缓存中含有该机地址的绑定，必须把此信息通知给那些计算机，以便那些机器修改相应的绑定项。地址变更的计算机可以在启动时，通过广播一个无回报的（gratuitous）ARP 请求，把新地址通知给其他主机^①。

下面的规则对这些改进做了总结：

每个 ARP 广播分组中都包含发送方的 IP 到物理地址的绑定；接收方在处理 ARP 分组（对目标地址进行解析）之前，先在自己的缓存中更新发送方 IP 到物理地址的绑定信息。

5.9 ARP 与其他协议之间的关系

ARP 提供一种可以把 IP 地址映射到物理地址的机制；我们已经知道，支持直接映射的网络技术不需要 ARP。可以设想，如果能使所有网络硬件都能识别 IP 地址，则 ARP 完全可以是不必要的。因此，无论硬件使用的是哪种底层地址机制，ARP 只是把一个新的地址方案施加在底层地址机制上。概而言之：

ARP 是一个隐藏底层网络物理编址并允许给每台机器分配任意 IP 地址的底层协议。我们把 ARP 看成是物理网络系统的一部分，而不是互联网协议的一部分。

5.10 ARP 的实现

从功能上讲，ARP 有两部分功能。第一部分在发送数据报时把 IP 地址映射成物理地址，第二部分回答来自其他机器的请求。给发送出去的数据报进行地址解析看似简单，但在具体实现细节上稍有些复杂。一旦给予软件某个目的站的 IP 地址，软件将查询本站的 ARP 缓存，看是否知道从该 IP 地址到物理地址的映射。如果知道，则软件提取该物理地址。把数据放到包含该地址的帧中，并发送该帧。如果不知道映射，则软件必须广播一个 ARP 请求并等待回答。

通过广播 ARP 请求找到地址映射有时可能变得更复杂。目标机器可能宕机，也可能太忙而来不及接受请求。如果是这样，发送方可能收不到回答或者回答可能被延迟。因为以太网是一个尽最

^① 通常，启动的机器为自己的 IP 地址发送一个请求，用此办法来验证没有其他计算机恰好分配了同一 IP 地址。

大努力交付系统，所以最初的 ARP 广播也可能会丢失（此时发送方至少应重发一次）。同时，主机必须保存原先外发的分组，以便在地址解析成功后就可以发送^①。事实上，主机必须决定，当它处理一个 ARP 请求时，是否允许其他应用程序继续执行（大多数是允许的）。如果允许，软件还必须能处理这样的情况：当应用程序生成了多个需要解析同一地址的数据报时，它不会为解析该目标地址而广播多个请求。

最后考虑这种情况：机器 A 已经获得机器 B 的绑定，但随后 B 的硬件就出现故障并被替换了。尽管 B 的地址已经改变了，但 A 缓存的绑定还没有改变，因此 A 如果使用已不存在的硬件地址，将导致无法成功接收。这一情况表明了让 ARP 软件把它的绑定表处理成高速缓存并在一段固定时间后清除表项的重要性。当然，一旦某个含有绑定的 ARP 广播到达，高速缓存中相应表项的计时器就要被复位（但当该表项用于发送分组时，计时器不进行复位）。

ARP 代码的第二部分处理来自网络的 ARP 分组。当 ARP 分组抵达时，软件首先提取发送方的 IP 地址和物理地址对，并检查本地的高速缓存，查看缓存中是否已存有该发送方对应的表项。如果缓存中有该发送方 IP 地址的表项，处理程序就用从分组中获得的物理地址覆盖缓存中的物理地址，从而更新该表项。然后接收方才处理 ARP 分组的其余部分。

接收方要处理到达的两种 ARP 分组。如果到达的是一个 ARP 请求，则接收方机器必须查看它自己是否是请求的目标（即其他某个机器为获得该接收方的物理地址而广播了一个请求）。如果是这样，则 ARP 软件用本机的物理地址形成回答，并把回答直接发送回请求方。如果发送方的地址对不在接收方的高速缓存中，则接收方还要把这个地址对添加到缓存中。如果在 ARP 请求中出现的 IP 地址与本地 IP 不匹配，则该分组是请求给网络上的其他机器进行地址映射，可忽略该分组。

另一种有意思的情况出现在一个 ARP 回答到达时。根据实现方式的不同，处理程序可能需要创建一个高速缓存表项，该表项也可能在请求产生时已创建。在任何一种情况下，一旦缓存被更新后，接收方将尝试把 ARP 回答与自己以前发出的请求进行匹配。通常，机器有分组要发送就会产生请求，而到达的回答是对请求的响应。从机器广播它的 ARP 请求至机器收到回答期间，应用程序或高层协议可能会对同一地址产生其他请求；ARP 软件必须记住它以前已发送过一个请求，而且不会再发请求。通常，ARP 软件把额外的请求分组放入一个队列。一旦回答到达并且获得地址绑定，ARP 软件就把分组从队列中取出，放到一个帧中，并使用该地址绑定来填写帧中的物理目的地址。如果软件以前没有为回答中的 IP 地址发出过请求，则机器更新其缓存中发送方的对应表项，然后停止处理分组。

5.11 ARP 的封装与标识

当 ARP 报文从一台机器传到另一台机器时，必须将其放在物理帧中运送。图 5.2 显示了由一个帧的数据部分所携带的 ARP 报文。

为了识别携带 ARP 报文的帧，发送方给帧首部的类型字段分配了一个特殊值，并把 ARP 报文放在该帧的数据字段中。当每一帧到达计算机时，网络软件通过帧类型确定其内容。在大多数技术中，所有携带 ARP 报文的帧都使用一个单一类型值，接收方的网络软件必须进一步查看 ARP 报文，以区分 ARP 请求和 ARP 回答。例如，在以太网中，携带 ARP 报文的帧类型字段值是 0806_{16} 。这是以太网管理机构分配的一个标准值，其他网络硬件技术使用的是其他值。

^① 如果延迟相当大，主机可以选择放弃外发的数据报。



图 5.2 封装在一个物理网络帧中的 ARP 报文

5.12 ARP 协议格式

与大多数协议不同，ARP 分组中的数据没有固定格式的首部。为了使 ARP 适用于多种网络技术，含有地址的字段长度取决于网络类型。但是，为了能理解任意一个 ARP 报文，在帧首部开头附近包含一些固定字段，用于指定可在后续字段中找到的地址的长度。事实上，ARP 报文格式已相当通用，能够适用于任何物理地址和任何协议地址。图 5.3 中的例子显示了解析 IP 协议地址（4 八位组长）时，在以太网硬件（其物理地址为 48 位或 6 八位组长）上使用的 28 八位组 ARP 报文格式。

图 5.3 显示的 ARP 报文每行为 4 八位组，这是本书中所用的标准格式。遗憾的是，与其他大多数协议不同，ARP 分组中的变长字段没有与 32 位边界对齐，使图表难以阅读。例如，标记为 SENDER HA 的发送方硬件地址占据了 6 个连续八位组，以至于在图中占了两行。

0	8	16	24	31		
HARDWARE TYPE		PROTOCOL TYPE				
HLEN	PLEN	OPERATION				
SENDER HA (octets 0-3)						
SENDER HA (octets 4-5)		SENDER IP (octets 0-1)				
SENDER IP (octets 2-3)		TARGET HA (octets 0-1)				
TARGET HA (octets 2-5)						
TARGET IP (octets 0-3)						

图 5.3 用于 IP 到以太网地址解析的 ARP/RARP 报文格式。字段的长度取决于硬件和协议地址长度，以太网的地址长度为 6 八位组，IP 地址长度为 4 八位组

硬件类型 (HARDWARE TYPE) 字段指明发送方想要知道的硬件接口类型；对于以太网，该字段含有的值为“1”。类似地，协议类型 (PROTOCOL TYPE) 字段指明发送方提供的高层协议地址类型；对于 IP 地址，这个字段含有的值为 0800_{16} 。操作 (OPERATION) 字段指明是 ARP 请求 (1)、ARP 响应 (2)、RARP^①请求 (3) 或 RARP 响应 (4)。硬件地址长度 (HLEN) 字段和协议地址长度 (PLEN) 字段允许 ARP 在任意网络中使用，因为它们分别指出了硬件地址和高层协议地址的长度。发送方如果知道自己的硬件地址和 IP 地址，则会在发送方硬件地址 (SENDER HA) 和发送方 IP 地址 (SENDER IP) 字段中给出其硬件地址和 IP 地址。

当发送方发出请求时，在目标硬件地址 (TARGET HA) 或目标 IP 地址 (TARGET IP) 字段中填入目标硬件地址 (RARP) 或目标 IP 地址 (ARP)。在目标主机发送响应之前，它填入报文中所缺的地址，并交换目标和发送方地址对，然后把操作 (OPERATION) 字段的值改成响应。因此。一个回答携带了原始请求方的 IP 和硬件地址，以及为目标机器寻找绑定所得到的 IP 和硬件地址。

^① 下一节介绍 RARP，这种协议使用相同的报文格式。

5.13 ARP 缓存自动重新确认

我们可以使用一种技术来避免抖动 (jitter) 现象（即分组传输时间的差异）的产生。为了理解这种情况，可以观察一下，只要某个地址的 ARP 定时器超时，发送给那个地址的下一个数据报就会经历额外的延迟，因为该数据报将在队列中等待，直到 ARP 软件发送一个请求并收到一个响应。此外，超时可能发生在任意时刻，可能出现在一次平稳的分组传输期间。虽然这种延迟通常可以忽略不计，但它们确实造成了抖动现象。

避免抖动产生的关键在于**提早重新确认** (early revalidation)。也就是说，其实现把 ARP 缓存中的每个表项与两个计数器关联起来：传统的定时器和一个重新确认计时器。当重新确认计时器超时，软件将对表项进行检查。如果一些数据报最近还在使用该表项，软件将发送一个 ARP 请求并继续使用该表项。当目标站收到请求并做出回答以后，这两个定时器都被复位。如果没有收到回答，传统的定期器将会超时，而数据报在 ARP 尝试获得一个响应期间将保持等待。但是，在大多数情况下，重新确认会不断地复位定时器。

5.14 逆地址解析协议

如上所述，ARP 分组中的操作字段可以指定一种**逆地址解析** (Reverse Address Resolution) 报文。RARP 在因特网中的重要性目前已不再存在，但它过去曾是没有固定存储的**自引导** (bootstrap) 系统所使用的重要协议。实质上，RARP 允许系统在启动时获得一个 IP 地址。这个过程简单明了：在系统启动时，它广播发送一个 RARP 请求，然后等待响应。网络上的另一台计算机必须被配置成监听 RARP 请求，并生成含有请求方 IP 地址的 RARP 回答。一旦应答到达，系统继续启动，并在所有通信中使用获得的 IP 地址。

当系统发出一个 RARP 请求时，它必须给出自己的标识，以便接收请求的计算机可以把正确的 IP 地址放在回答中。虽然可以使用任何一种唯一识别硬件的方法（如 CPU 序列号），但 RARP 使用一个显而易见的 ID：系统的 MAC 地址。也就是说，正在启动的系统把它的 MAC 地址放在 RARP 请求中，然后在 RARP 回答中接收它的 IP 地址。

值得注意的是，RARP 使用的分组格式与 ARP 一样^①。RARP 请求是通过填充目标协议地址字段形成的，被授权提供 RARP 服务的机器把报文类型从请求 (request) 改成回答 (reply)，并把回答直接返回给发送请求的机器。

与 ARP 报文类似，RARP 报文封装在一个网络帧的数据部分，从一台机器发送到另一台机器。例如，一个携带 RARP 请求的以太网帧，在帧的前部包含常有的前同步码、以太网源地址和目的地址以及分组类型字段等。帧类型字段包含的值为 8035_{16} ，标志着帧的内容是一个 RARP 报文。

5.15 小结

IP 地址的分配与机器的物理硬件地址无关。为了通过物理网络把互联网分组从一台计算机发送到另一台计算机，网络软件必须把 IP 地址映射成一个物理硬件地址，并用这个硬件地址传输帧。如果硬件地址比 IP 地址短小，通过把机器的物理地址编码到 IP 地址中即可建立直接的映射，否则必须动态完成映射。地址解析协议 (ARP) 完成动态地址解析，它只使用底层的网络通信系统。ARP 允许机器解析地址，并且不保存绑定的永久记录。

通过广播 ARP 请求，一台机器可使用 ARP 找到另一台机器的硬件地址。请求中包含了需找到其硬件地址的主机的 IP 地址。网络上的所有机器都会收到 ARP 请求。如果一台机器的 IP 地址与

^① ARP 分组格式见 5.12 节。

该请求匹配，则此机器发出一个包含所需硬件地址的回答来做出响应。回答是直接发给一台机器的，不是广播发送的。

为了提高 ARP 的效率，每台机器都把 IP 到物理地址的绑定保存在高速缓存中。因为互联网通信量倾向于由成对机器之间的系列交互组成，所以缓存排除了大多数 ARP 广播请求；提早重新确认可用于消除抖动现象。

RARP 是一种与 ARP 有关的旧协议，它允许计算机在系统启动时获得一个 IP 地址。第 22 章将讨论替代它的一种协议。

5.16 深入研究

本章所讲的地址解析协议由 Plummer [RFC 826] 给出，并已成为一个 TCP/IP 互联网协议标准。在 Finlayson et. al. [RFC 903] 中给出了 RARP 的细节。Dalal and Printis [1981] 描述了以太网地址和 IP 地址之间的关系，Clark [RFC 814] 一般性地讨论了地址和绑定。Parr [RFC 1029] 讨论了容错地址解析。Kirkpatrick and Recker [RFC 1166] 说明了在因特网编号（Internet Numbers）文档中用于标志网络帧的值。本系列教材的第二卷给出了一个 ARP 实现的例子，并讨论了高速缓存策略。

5.17 习题

1. 给出一个小的物理地址集合（正整数），能否找到一个函数 f 和一种 IP 地址分配方法，使 f 把 IP 地址一一映射到物理地址上，并且使 f 的计算效率高？提示：查找有关成熟的哈希技术的文献。
2. 在什么特殊情况下，连到以太网的主机在传输 IP 数据报之前不需要使用 ARP 或 ARP 高速缓存？
3. 有一种管理 ARP 缓存的常用算法，在添加一个新表项时替换最近最少使用的表项。在什么情况下，这个算法会产生不必要的网络通信量？
4. 仔细阅读标准。对于一个给定的 IP 地址，如果缓存中已存在相应的旧表项，ARP 是否应更新缓存？为什么？
5. ARP 软件是否需要修改缓存，即使它接收到的信息并不是它专门请求的信息？为什么？
6. 在一个拥有许多主机和大量 ARP 通信量的网络上，任何使用固定容量缓存的 ARP 实现都会失败。试解释原因。
7. ARP 常被当成一个安全弱点提及，试解释原因。
8. 假设机器 C 收到一个来自 A 要查找目标 B 的 ARP 请求，而且假设 C 的高速缓存中有从 I_B 到 P_B 的绑定。C 是否应该回答这个请求？为什么？
9. ARP 可以为一个以太网上的所有可能的主机预建缓存，做法是通过遍历这些可能的 IP 地址的集合，为每台主机都发送一个 ARP 请求。这样做是个好主意吗？为什么？
10. 提早重新确认是否应该给局域网上所有可能的 IP 地址或 ARP 缓存中的所有表项都发送请求？或者是否应该只给一些近期有过通信量的目标站发送请求？试解释原因。
11. 当一台工作站启动时想找一找网络上是否有机器在模拟它，应该怎样使用 ARP？这种方法的缺点是什么？
12. 怎样向远程以太网上一个不存在的地址发送 IP 分组，才能在该网络上产生广播通信量？

第6章 网际协议：无连接数据报交付（IPv4）

6.1 引言

前面回顾了互联网通信所需的一些网络硬件和软件，阐述了底层网络技术和地址解析。本章讲解无连接交付的基本原理，并讨论**网际协议**（Internet Protocol，简称 IP）如何提供这种无连接交付。网际协议是网际互连中使用的两个主要协议之一（TCP 是另一个协议）。本章将学习 IP 分组的格式，并理解分组如何构成所有互联网通信的基础。后续两章将继续研究网际协议，讨论分组转发和差错处理。

6.2 虚拟网络

在第 3 章讨论的互联网体系结构中，一些路由器把多个物理网络连接起来。从表面上看这个体系结构可能会产生误导，因为人们的注意力往往放在互联网给用户提供的接口上，而不是网络互连的技术。

用户把互联网想像成一个虚拟网络，它把所有主机都互连起来，并通过该网络让主机间的通信成为可能；互联网的底层体系结构被隐藏起来，并且无关紧要。

在某种意义上，互联网是对物理网络的抽象，因为它在最底层提供了相同的功能集：接受分组并进行交付。大多数用户感知到的丰富功能是由更高层的互联网软件补充提供的。

6.3 互联网体系结构和基本原理

从概念上讲，TCP/IP 互联网提供了三组服务，如图 6.1 所示，图中三组服务的排列方式暗示了它们之间的相互依赖关系。在最底层，无连接交付服务提供了其他服务赖以存在的基础。在第二层，可靠的传输服务提供了应用所依赖的较高层平台。接下来将探讨每一种服务，搞清楚它们各自提供了什么，以及与它们有关的协议。**互联网服务的三个概念层是指应用服务，可靠的运输服务，无连接的分组交互服务。**



图 6.1 互联网服务的三个概念层

6.4 概念化服务的组织结构

尽管可以把图 6.1 中的每种服务与协议软件关联起来，但我们还是将其作为互联网的概念化组

成部分来理解，原因就在于它们清楚地指出了互联网设计的思想支柱。这个观点是：

互联网软件是围绕着一个分层结构中的三个概念化网络服务来设计的；这个结构相当健壮且具有很强的适应性，使互联网软件取得了很大成功。

这种概念划分最主要的优点是：在不干扰其他服务的情况下，使替换某个服务成为可能。因此，这三种服务的研究和开发可以并行开展。

6.5 无连接交付系统

最基本的互联网服务包括一个分组交付系统。从技术角度来讲，该服务被定义为不可靠的、尽最大努力交付的、无连接分组交付系统，类似于按照尽最大努力交付模式运行的网络硬件所提供的服务。这种服务不能保证交付，因此认为其**不可靠**（unreliable）。分组可能丢失、重复、延迟或不按次序交付等，但服务检测不到这些情况，也不会通知发送方或接收方。认为**服务无连接**（connectionless），是因为每个分组都被独立处理。一台计算机发送给另一台计算机的分组序列可能途经不同的路径，可能有的分组丢失而有的分组到达。最后，认为服务使用了**尽最大努力交付**（best effort delivery），是由于互联网软件努力尝试发送每个分组。也就是说，互联网不会随便丢弃哪个分组；只有当资源耗尽或底层网络出现故障时才可能出现不可靠性。

6.6 网际协议的目的

这种定义不可靠、无连接交付机制的协议称为**网际协议**（Internet Protocol）。由于协议的当前版本是版本 4，我们通常用其首字母缩写词 IPv4 代表它；如果版本是不确定的，就用 IP 代表它。IP 提供了三个重要的定义。第一，IP 定义了在整个 TCP/IP 互联网中使用的数据传送基本单元。因此，它规定了数据在互联网上传递时的确切格式；第二，IP 软件完成**转发**（forwarding）的功能，选择分组发送的路径；第三，除了精确、正式的数据格式规范和转发之外，IP 还包含一组体现不可靠分组交付思路的规则。这些规则刻画了主机和路由器应当如何处理分组，何时及如何产生差错报文，以及在什么情况下可以丢弃分组。IP 是设计的最基本的部分，以至于因特网有时又称为**基于 IP 的技术**（IP-based technology）。

本章从查看规范的分组格式开始研究 IPv4。在以后的章节中再讨论转发和差错处理的问题。

6.7 IPv4 数据报

物理网络和 TCP/IP 互联网十分相似。在一个物理网络上，传送单元是一个包含首部和数据的帧，首部给出了（物理）源地址和目的地址之类的信息。互联网把它的基本传送单元称为**Internet 数据报**（Internet datagram），有时简称为**IP 数据报**（IP datagram），或仅简称为**数据报**（datagram）。与典型的物理网络帧类似，数据报划分成首部和数据区。同样与帧类似，数据报首部也包含了源地址和目的地址以及一个标志数据报内容的类型字段。当然，它们的不同之处在于，数据报首部包含的是 IP 地址，而帧首部包含的是物理地址。图 6.2 给出了数据报的一般格式。



图 6.2 IP 数据报的一般格式，与网络帧类似。IP 规定了数据报首部的格式，包括源 IP 地址和目的 IP 地址。IP 不规定数据区的格式，可用来运输任意数据

6.7.1 数据报格式

我们已经描述了 IP 数据报的一般格式，现在更详细地看看其中的内容。图 6.3 显示了数据报中各字段的排列方式。

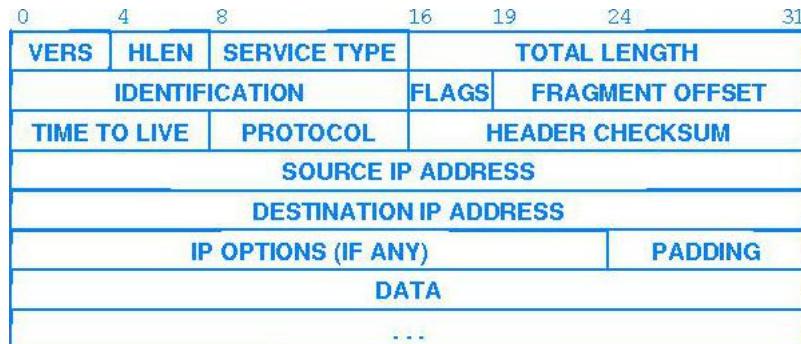


图 6.3 Internet 数据报格式，Internet 数据报是 TCP/IP 互联网中的基本传送单元

由于数据报是在软件中进行处理的，所以它的格式和内容都不受任何硬件限制。例如，数据报中前 4 位的字段，即版本（VERS）字段，包含了创建数据报所用的 IP 协议的版本信息，用来确认发送方、接收方和它们之间的所有路由器就数据报格式达成一致。所有 IP 软件在处理一个数据报以前都要检查版本字段，确保它与软件预期的格式匹配。如果标准改变了，机器就会拒绝版本信息与协议版本不同的数据报，以避免按照过时的格式错误地解释数据报内容。对于 IPv4，版本字段的值为 4。

首部长度（header length field，简称 HLEN）字段也是 4 位，它给出用 32 位字来度量的数据报首部长度。正如将要看到的，除了 IP 选项（IP OPTIONS）及相应的填充（PADDING）字段以外，首部的所有字段长度都是固定的。最常见的首部不含选项和填充字段，长度为 20 八位组，其首部长度字段等于 5。

总长度（TOTAL LENGTH）字段给出用八位组来度量的 IP 数据报长度，包括首部和数据区中的八位组。数据区的大小可以从总长度中减去首部长度计算得到。因为总长度字段长 16 位，所以 IP 数据报的最大可能长度为 2^{16} ，即 65,535 八位组。在大多数应用程序中，这个限制还不算苛刻。但将来如果更高速网络可以运送长度超过 65,535 八位组的数据分组，这个长度限制就可能成问题了。

6.7.2 数据报的服务类型和区分服务

8 位的服务类型（SERVICE TYPE）字段，在非正式场合称为 TOS（Type Of Service），它指明应当如何处理数据报。这个字段最初分成几个子字段，指明数据报的优先级和期望的路径特征（低时延或高吞吐量）。在 20 世纪 90 年代后期，IETF 重新定义了字段的含义，以提供一组**区分服务**（Differentiated Service，简称 DiffServ）。图 6.4 显示的是最后得到的定义。



图 6.4 IP 数据报中服务类型字段的区分服务（DiffServ）解释

根据区分服务的解释，前 6 位组成**码点**（codepoint）字段，有时缩写为 DSCP，后 2 位保留未用。一个码点值被映射到一个底层服务定义，这通常是通过一个指针数组实现的。尽管可以定义 64 个不同的服务，但设计人员建议某个给定的路由器只需要几个服务，并且多个码点将映射到各

个服务。另外，为了保持与最初定义向后兼容，标准把码点的前 3 位（以前用于优先级的位）和后 3 位区别对待。当后 3 位包含 0 时，优先级的位定义 8 大类服务，定义遵循的指导原则与最初的定义相同：优先级字段中数值较高的数据报比数值较低的数据报更受优待。也就是说，8 个有序类别用下列形式的码点值定义：

XXX000

其中 X 代表 0 或 1。

区分服务设计还适用于另一种应用场合：优先级 6 或 7 被广泛用于优先权高的路由选择通信量。为了处理这些优先级数值，标准中包含了一个特例。在此特例中，路由器需要实现至少两个优先级方案：一个用于普通通信量，一个用于高优先级通信量。当码点字段的后 3 位为 0 时，路由器必须把优先级为 6 或 7 的码点映射成高优先级的类别，把其他码点值映射成低优先级的类别。这样，如果到达的数据报是使用最初的 TOS 方案发送的，使用区分服务方案的路由器就会像数据报发送方期望的那样，对具有优先级 6 和 7 的数据报给予优待。

64 个码点值划分为 3 个管理集合，如图 6.5 所示。

Pool	Codepoint	Assigned By
1	xxxxx0	Standards organization
2	xxxx11	Local or experimental
3	xxxx01	Local or experimental

图 6.5 码点值的三个管理池

如图所示，一半的码点值（即池 1 中的 32 个值）必须由 IETF 来分配解释。目前，池 2 和池 3 中的所有值可用于实验或在本地使用。但是，如果标准组织把池 1 中的值都用完了，也可能选择指派池 3 中的值。

这种把码点值分成几个池的做法似乎有些不寻常，因为它依靠码点值的低位比特来区分不同的池。这样，池 1 得到的不是一段连续的数值，而是包含一些间隔分开的数字（即 2~64 之间的偶数）。选择这种划分方式是为了让值 XXX000 对应的 8 个码点处于同一个池中。

无论使用最初的 TOS 解释还是修改后的区分服务解释，转发软件必须在当前可用的底层物理网络技术中进行选择，并且必须符合本地策略，认识到这一点很重要。因此，在数据报中指明某种服务级别，并不能保证路径沿线的路由器都同意接受这种服务级别的请求。概而言之：

我们把指明服务类型当成是给转发算法的提示，帮助转发算法根据本地策略和对路径上，可用的硬件技术的了解，在抵达一个终点的不同路径中进行选择。一个互联网不能保证提供任何一种特定类型的服务。

6.7.3 数据报的封装

在理解数据报的其他字段以前，我们有必要先了解一下数据报与物理网络帧之间有怎样的关联。先思考一个问题：“一个数据报可以有多长？”数据报与物理网络帧不同，物理网络帧必须被硬件识别，而数据报是由软件处理的。数据报的长度可以由协议设计者任意选择。前面已提到，IPv4 数据报格式给总长度字段分配了 16 位，限制数据报的长度最多为 65,535 八位组。

对数据报大小的更多基本限制是在实践中产生的。我们知道，数据报从一台机器移到另一台，总是要通过底层的物理网络进行传输。为了使互联网传输更高效，我们更愿意让每个数据报在一个独立的物理帧中传输。也就是说，如果可能，希望把一个物理网络分组的抽象直接映射到一个实际的分组。

在网络帧中携带一个数据报的想法称为封装（encapsulation）。对底层网络来说，一个数据报与从一台机器发送到另一台机器的其他任何报文类似。硬件不识别数据报的格式，也不明白 IP 目的

地址。因此，如图 6.6 所示，当一台机器把一个 IP 数据报发送到另一台机器时，整个数据报是放在网络帧的数据部分中运送的^①。

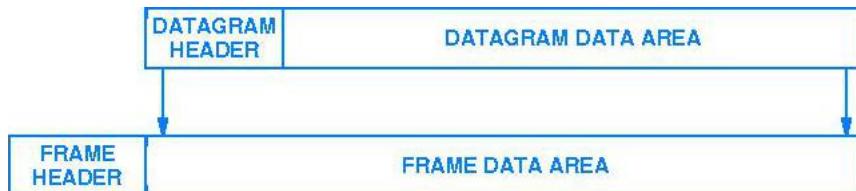


图 6.6 在帧中封装 IP 数据报。物理网络把包括首部在内的整个数据报都当成数据处理

6.7.4 数据报的大小、网络 MTU 及分片

在理想情况下，整个数据报封装在一个物理帧中，使物理网上的传输效率很高。为了获得这样的高效率，IP 的设计者应该选择一个最大数据报长度，使数据报始终能完整地放入一个帧。但是，应该选择多大的帧呢？毕竟，在一个数据报通过互联网传送到最终目的站的过程中，它可能会穿过多种类型的物理网络。

为了搞清楚这个问题，我们需要知道一个有关网络硬件的事实：每种分组交换技术都给一个物理帧可传送的数据量规定了一个固定的上界。例如，以太网限制传输 1500^② 八位组的数据。我们把这些限制称为网络的最大传送单元（maximum transfer unit，简称 MTU）。MTU 的长度可能超过 1500，也可能更小，有些硬件技术的传送限制为 128 八位组。如果把数据报大小限制成互联网中最小可能的 MTU，当数据报经过一个能够运载更大长度帧的网络时，就会使传输效率不高。但是，如果允许数据报的大小比互联网中最大网络 MTU 更大，则意味着数据报可能无法始终封装在单个网络帧中。

要做出的选择应该是显然的：互联网的要点是隐藏底层网络技术和方便用户通信。因此，TCP/IP 软件并没有设计符合物理网络限制的数据报，而是选择一个合适的初始数据报大小，同时提供一种机制，在数据报需要经过小 MTU 的网络时，把长数据报分解成若干更小的片。数据报分解得到的这些小片称为数据报片（fragment），数据报的分解过程称为分片（fragmentation）。

如图 6.7 所示，分片通常发生在路由器上，发生分片的路由器位于数据报的源站到目的站之间的路径上，路由器从一个大 MTU 的网络上接收数据报，并且要把它发送到一个 MTU 小于数据报大小的网络上。

在图 6.7 中，两台主机都直接连到 MTU 为 1500 八位组的以太网上。因此，每台主机都可以产生和发送长度最多为 1500 八位组的数据报。但是在它们之间的路径上，有一个 MTU 为 620 八位组的网络。如果主机 A 给 B 发送了一个长度超过 620 八位组的数据报，则路由器 R₁ 将把数据报分片。同样，如果 B 给 A 发送了一个长数据报，则路由器 R₂ 将把数据报分片。

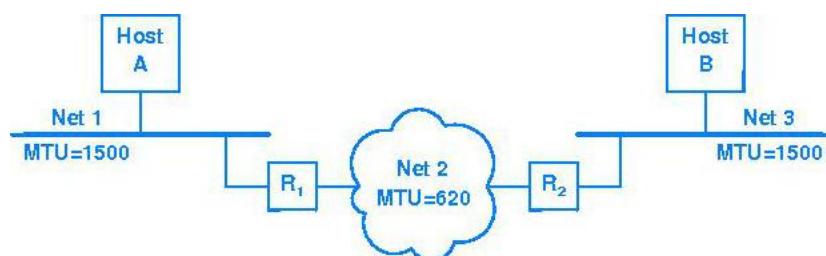


图 6.7 发生分片的示意图。路由器 R₁ 把从 A 发到 B 的长数据报分片；路由器 R₂ 把从 B 发到 A 的长数据报分片

^① 帧首部的一个字段通常标出帧中携带的数据；以太网使用类型值 0800₁₆ 指明数据区含有一个封装的 IP 数据报。

^② 1500 的限制来自以太网的规范；当与 SNAP 首部一起使用时，IEEE 802.3 标准限制数据为 1492 八位组。

对数据报片的大小应进行选择，以便让每个数据报片都能装在单个帧内在底层网络上传输。另外，因为 IP 用 8 八位组的倍数表示数据的偏移量，所以数据报片的大小必须是 8 的倍数。当然，选择最接近网络 MTU 的 8 八位组的倍数，不一定能把数据报分成大小相同的片，最后一片往往比其他片小。为了生成原先数据报的完整副本，必须赶在目的站处理以前，对收到的数据报片进行重装（reassembled）。

IP 协议并没有把数据报限制得很小，但也不保证大的数据报交付时不被分片。源站可以任意选择合适的数据报大小；分片和重装自动进行，源站不必采取任何特殊动作。IP 规范指出：路由器必须能够接受所连网络中最大 MTU 大小的数据报。另外，它必须随时能够处理大小不超过 576 八位组的数据报（要求主机也随时能够接受大小至少为 576 八位组的数据报，并在必要时重装数据报）。

数据报分片意味着把一个数据报分成几片。每片的格式都与原来的数据报相同，这一点可能会让你觉得奇怪。图 6.8 展示了分片的结果。

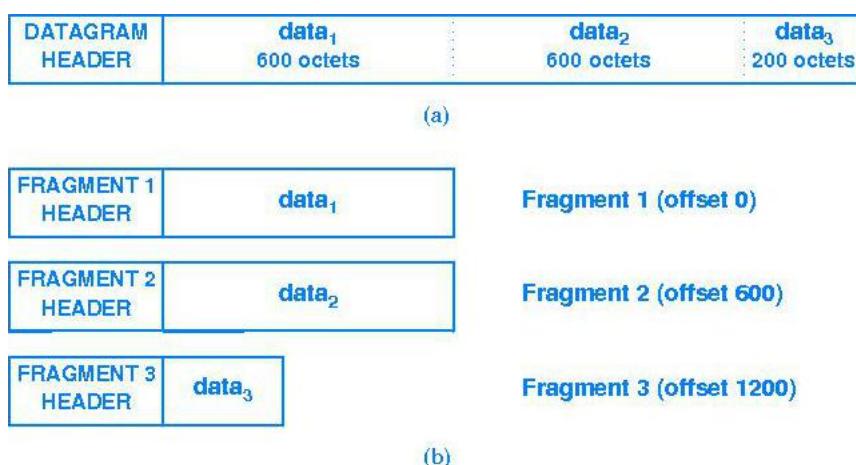


图 6.8 (a) 携带 1400 八位组数据的原始数据报；(b) 三个数据报片，用于 MTU 为 620 八位组的网络。首部 1 和首部 2 设置了**更多分片位** (more fragment bit)。显示的偏移量是十进制的八位组数，必须将其除以 8 才能得到数据报片首部中存储的值

每个数据报片都包含一个数据报首部，该首部基本上复制了原始数据报的首部（标志字段中表示数据报片的一个比特除外），首部后面是数据报片携带的数据，同时让数据报片的总长度始终保持小于传输它的底层网络的 MTU 大小。

6.7.5 数据报片的重装

数据报片应该在经过一个网络传递后重装吗？或者，在重装前应该将数据报片传给最终的主机吗？在 TCP/IP 互联网中，一旦对数据报进行了分片，数据报片一路上就作为独立的数据报传送，直到抵达必须对其进行重装的最终目的站。在到终点站的沿线上保留数据报片有两个缺点。第一，由于数据报片没有在经过一个小 MTU 的网络传递后立即重装，所以必须把小数据报片从分片的地方开始一直运送到最终目的站。在最终目的站重装就会导致效率不高：即使在分片的位置之后遇到某些大 MTU 的网络，在那些网络上也只有小的数据报片经过；第二，如果任何一个数据报片丢失了，数据报就无法重装。接收机在它收到第一个数据报片后就启动一个**重装计时器** (reasembly timer)。如果在所有数据报片到达以前计时器就超时了，那么接收机会丢弃已收到的数据报片，不处理数据报。因为丢失一个数据报片意味着丢失整个数据报，所以当分片发生后，数据报丢失的概率就会增加。

尽管有这些小缺点，在最终目的站重装数据报片工作得还不错。它允许每个数据报片独立转发，而且，不需要中间路由器存储或重装数据报片。

6.7.6 分片控制

数据报首部中的标识 (IDENTIFICATION)、标志 (FLAG)、片偏移量 (FRAGMENT OFFSET) 这三个字段，控制着数据报的分片和重装。标识字段含有一个唯一识别该数据报的整数。回顾前述内容，路由器对数据报分片时，把数据报首部中的大部分字段都复制到每个数据报片的首部中。因此，必须复制标识字段。这样做的主要目的是让目的主机知道每个到达的数据报片属于哪个数据报。一旦某个数据报片到达，目的主机通过数据报片的标识字段及源站地址来识别数据报。发送 IP 数据报的计算机必须为每个数据报的标识字段生成一个唯一的值^①。IP 软件使用的一种技术是在内存中保持一个全局计数器，每当产生一个新数据报，计数器累加 1，并把累加的结果指派给数据报的标识字段。

前面讲过，每个数据报片的格式与完整的数据报相同。数据报片的片偏移量字段指明数据报片所携带数据在原始数据报中的偏移量（以 8 八位组为单位）^②，从零偏移的位置开始算起。目的站必须收到从偏移量 0 到最高偏移量的所有数据报片，才能重装数据报。数据报片不必按序到达，并且在对数据报进行分片的路由器和试图重装数据报的目的主机之间也不必进行通信。

在 3 个比特位的标志字段中，低位的 2 比特控制分片。通常，使用 TCP/IP 的应用软件并不关心分片的问题，因为分片和重装都在低层操作系统中自动进行，对于终端用户是不可见的。但是，为了测试互联网软件或调试操作中的问题，可能需要测试进行分片的数据报大小。在这种测试中会用到第一个控制比特位，它指明数据报是否可分片。如果把它设成“1”，则说明数据报不应该分片，因此又将它称为**不分片** (do not fragment) 位。当只有整个数据报才有用时，应用程序可以选择不允许分片。例如，思考一下自引导过程：一个小型嵌入式系统执行 ROM 中的一个程序，该程序通过互联网向另一台机器发出一个请求，那台机器发回一个内存映像作为响应。如果在设计嵌入式系统时，让它或者需要整个映像或者全部不要，就应该把数据报的不分片位设为“1”。当路由器需要对不分片位设为“1”的数据报进行分片时，就会丢弃该数据报，并向源主机发回一个差错报文。

标志字段的低位比特称为更多分片 (more fragments) 位，指明数据报片包含的数据出自于原始数据报的中间，还是出自于原始数据报的末尾。为了搞清楚为什么需要这个比特位，我们不妨想一想最终目的主机上试图重装数据报的 IP 软件。它将接收若干数据报片（可能顺序不对），还要知道自己是否已接收完一个数据报的全部数据报片。当一个数据报片到达时，其首部中的总长度字段指明的是数据报片的长度，而不是原始数据报的长度，所以目的主机无法根据该字段来判断自己是否已收集到所有数据报片。更多分片位轻易地解决了这个问题：一旦目的主机收到更多分片位关闭（值为“0”）的数据报片，就知道这个数据报片携带的是原始数据报的尾部数据。目的主机利用片偏移量和总长度字段，可以计算出原始数据报的总长度。通过查看已到达的全部数据报片的片偏移量和总长度字段，接收方就可以知道现有的数据报片是否包括重装原始数据报所需的全部数据报片。

6.7.7 生存时间

从原理上讲，**生存时间** (Time To Live, 简称 TTL) 字段指明数据报在互联网系统中允许保留多长时间（以秒为单位）。这个概念既简单又重要：每当一台机器把数据报送入互联网时，就为数据报设置了一个最大生存时间。随着时间流逝，处理数据报的主机和路由器要递减生存时间字段的值；一旦超过时限，就把该数据报从互联网中清除。

由于路由器通常不知道物理网络的传输时间，所以它很难进行确切的估计。有一些规则可以简化处理，使数据报处理变得更容易，并且不需要同步时钟。首先，从源站到目的站的路径沿线的每

^① 在理论上讲，重传的数据报可以携带与原始数据报相同的标识字段；但在实际运用中，重传由更高层协议完成，从而导致新的数据报具有自己的标识。

^② 为了节省首部的空间，用 8 八位组的倍数来指明偏移量。

个路由器在处理数据报首部时，需要把生存时间字段的值减 1。此外，由于路由器最初出现时速度慢，原来的标准规定：如果路由器让一个数据报滞留了 K 秒，路由器就应把生存时间字段的值减 K。

路由器让数据报延迟许多秒的情况虽然一度受到人们的重视，但这样的年代已经一去不复返了——现在设计的路由器和网络最多在几毫秒内即可完成每个数据报的转发。如果时延过长，路由器会简单地丢弃数据报。因此，在实际运用中，生存时间起着“跳数限制”的作用，而不是延迟时间的估计。路径沿线的每个路由器（即每一跳）将生存时间字段值递减 1。

只要生存时间字段值减到 0，路由器就会丢弃该数据报，并向源站发回一个差错报文。给数据报设置计时器的做法很有意义，因为它保证数据报不会在一个互联网内无休止地游历，即使路由表遭到破坏而使路由器循环转发了数据报。因此，**生存时间字段可被看成是一种自动防范故障的机制**。

6.7.8 其他数据报首部字段

协议 (PROTOCOL) 字段类似于网络帧中的类型字段，其值指明数据报中的数据 (DATA) 区携带的报文是使用哪个高层协议创建的。实质上，协议字段的值指明了数据区的格式。高层协议和协议字段中的整数值之间的映射必须由一个中央权力机构管理，以确保在整个因特网内的统一。

首部检验和 (HEADER CHECKSUM) 字段保证首部字段值的完整性。IP 检验和的计算是把首部看成一个 16 位整数的序列 (按网络字节顺序)，分别计算每个整数的二进制反码，然后相加，再对结果求一次二进制反码。为了计算检验和，假定首部检验和字段包含零。

注意，**检验和只用于 IP 首部字段值的检验，而不用于数据**。首部和数据的检验分开既有优点也有缺点。因为首部通常比数据所占的八位组数量少，所以在只需要计算首部检验和的路由器上，分开检验将减少处理时间。分开检验也允许更高层协议自己给数据选择检验和算法。主要的缺点是高层协议必须添加自己的检验和，否则就会有检测不出已遭破坏数据的危险。

源 IP 地址 (SOURCE IP ADDRESS) 和目的 IP 地址 (DESTINATION IP ADDRESS) 字段含有数据报的发送方和指定接收方的 32 位 IP 地址。虽然数据报可能经过许多中间路由器转发，但这两个字段始终不变，它们指明的是最初源站和最终目的站的 IP 地址^①。

图 6.3 中的数据 (DATA) 字段显示了数据报中数据区的开头部分。当然，数据字段的长度取决于数据报中发送的内容。下面要讨论的 IP 选项 (IP OPTIONS) 字段长度是可变的。填充 (PADDING) 字段与选项的选择有关。为了保证数据报首部的长度延长为 32 位的整数倍，可能需要让填充字段包含一些“0”比特 (前面讲过，首部长度字段值的度量单位是 32 位字)。

6.8 Internet 数据报选项

目的 IP 地址后面的 IP 选项 (IP OPTIONS) 字段并非每个数据报都需要，数据报包含的选项主要用于网络测试或调试。但是，选项处理是 IP 协议的一个组成部分，所以所有标准实现都必须包含它。

IP 选项字段的长度变化取决于选择了哪个选项。有的选项只有 1 八位组长，它们由 1 八位组的**选项码** (option code) 组成。其他选项的长度是可变的。当一些选项出现在一个数据报中时，这些选项看起来是连续的，中间没有任何特殊分隔符。每个选项包含 1 八位组的选项码，后面可能跟有该选项对应的 1 八位组长度和一串数据八位组。选项码八位组分成三个字段，如图 6.9 所示。



图 6.9 选项码八位组分成 1 位、2 位和 5 位长的三个字段

^① 当数据报包含源路由选项 (在后面介绍) 时也存在例外。

选项码由 1 位的复制(COPY)标志、2 位的选项类(OPTION CLASS)以及 5 位的选项号(OPTION NUMBER)组成。复制标志控制路由器在分片过程中如何处理选项。当复制标志置为“1”时，说明该选项应复制到所有数据报片中。如果复制标志置为“0”，则意味着应该只把该选项复制到第一个数据报片中，而不是复制到所有数据报片中。

选项类和选项号指明选项所属的一般类和该类中的某个具体选项。图 6.10 所示的表显示了选项类是如何分配的。

Option Class	Meaning
0	Datagram or network control
1	Reserved for future use
2	Debugging and measurement
3	Reserved for future use

图 6.10 在选项码八位组的选项类比特中的 IP 选项的类型编码

图 6.11 中的表列出 Option Number 数据报中的可能选项，并给出其选项类和选项号的值。如表中所示，大多数选项用于控制目的。

Option Class	Option Number	Length	Description
0	0	-	End of option list. Used if options do not end at end of header (see header padding field for explanation).
0	1	-	No operation. Used to align octets in a list of options.
0	2	11	Security and handling restrictions (for military applications).
0	3	var	Loose source route. Used to request routing that includes the specified routers.
0	7	var	Record route. Used to trace a route.
0	8	4	Stream identifier. Used to carry a SATNET stream identifier (obsolete).
0	9	var	Strict source route. Used to specify an exact path through the internet.
0	11	4	MTU Probe. Used for path MTU discovery.
0	12	4	MTU Reply. Used for path MTU discovery.
0	20	4	Router Alert. Router should examine this datagram even if not an addressee.
2	4	var	Internet timestamp. Used to record timestamps along the route.
2	18	var	Traceroute. Used by traceroute program to find routers along a path.

图 6.11 IP 选项的数字类和编号代码举例。“长度”一栏中的 var 代表可变 (variable)

6.8.1 记录路由选项

路由和时间戳选项是最受关注的选项，因为它们提供了一种办法来监测或控制互联网的路由器如何转发数据报。**记录路由**(record route) 选项允许源主机创建一个 IP 地址的空表，以备处理数据报的各个路由器将其 IP 地址添加到这个表中。图 6.12 显示了记录路由选项的格式。

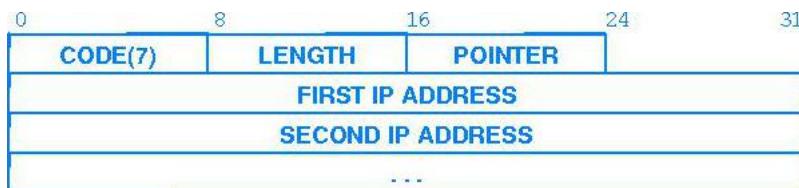


图 6.12 IP 数据报中记录路由选项的格式。选项最前面是 3 个八位组，后面紧跟着一个地址表。尽管图中显示的地址长度都为 32 位，但实际上它们不与数据报中的任何八位组边界对齐

正如前面所述，代码(CODE)字段包括选项类和选项号(对于记录路由，选项类为 0，选项

号为 7)。长度 (LENGTH) 字段指明选项在 IP 数据报中出现时的总长度，其中包括前面 3 个八位组。从第一个 IP 地址 (FIRST IP ADDRESS) 字段标记开始的一些字段组成的区域，保留用于记录互联网地址。指针 (POINTER) 字段指明下一个用于存放地址的空位在选项中的偏移量。

每当机器处理设置了记录路由选项的数据报时，它把自己的地址添加到记录路由表中（最初的源主机必须在选项中分配足够的空间，以容纳以后需要添加的所有表项）。为了把自己的地址添加到表中，机器首先比较指针字段和长度字段。如果指针比长度大，则表已满，机器不插入自己的表项，而是直接转发该数据报；如果表不满，机器就在由指针指明的位置上插入它自己的 4 八位组 IP 地址，并把指针值加 4。

当数据报到达时，目的主机可以提取和处理 IP 地址表。通常，接收数据报的计算机会忽略记录的路由。要想使用记录路由选项，需要两台机器一致同意协作；计算机在发出的数据报中打开记录路由选项以后，不会自动接收进来的数据报中记录的路由。源主机必须同意启用记录路由选项，同时目的主机也必须同意处理得到的表。

6.8.2 源路由选项

源路由 (source route) 选项是另一个受到网络构建者关注的概念。源路由为发送方指明一条经过互联网的路径提供了一种办法。例如，为了测试在某个物理网络 N 上的吞吐量，系统管理员可以使用源路由选择，强制要求 IP 数据报通过网络 N 传送，即使路由器在正常情况下可能会选择一条不包括 N 的路径。在生产环境中能够进行这种测试相当重要，因为它可以让网络管理员在已知运行正常的网络上自由转发用户数据报，同时还可以测试其他网络。当然，源路由选择只对那些知道网络拓扑的人有用，一般用户不需要知道或使用它。

IP 支持两种形式的源路由选择：一种形式是**严格的源路由选择** (strict source routing)，通过在选项中包含一个 IP 地址序列来指明一条路由选择路径，如图 6.13 所示。

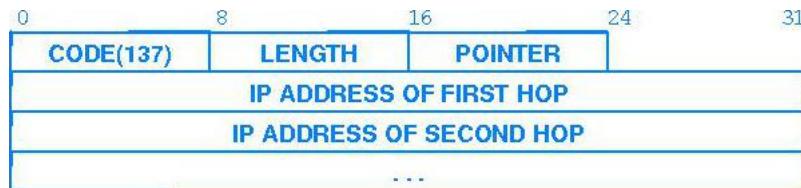


图 6.13 严格的源路由选项，通过给出数据报必须经过的 IP 地址列表来指明一条确切路由

严格的源路由选择意味着，数据报必须沿着地址表所指明的确切路由到达目的站。地址表中相邻两个地址之间的路径必须处于同一物理网络；如果路由器无法遵循严格的源路由转发数据报，就会出现错误。另一种形式，称为**不严格的源路由选择** (loose source routing)，它也包含一个 IP 地址序列。它要求数据报必须沿着 IP 地址序列传输，但是允许表中相邻两个地址之间有多个网络跳。

两种源路由选项都要求路径沿线的路由器用它们的本地网络地址覆盖地址表中的表项。因此，当数据报到达目的站时，地址表中就包含了数据报到访过的所有地址，与记录路由选项产生的表十分类似。

源站路由选项的格式类似于前面介绍的记录路由选项。每个路由器都要查看指针字段和长度字段，确定地址表的空间是否已用完。如果已用完，即指针值比长度值大，路由器照常把数据报转发到目的站。如果表中的空间还没有用完，则路由器根据指针值取出 IP 地址，用路由器地址代替它^①，并使用从表中取得的地址转发数据报。

^① 路由器的每个接口都有一个地址，记录数据报被转发到的那个网络所对应的接口地址。

6.8.3 时间戳选项

时间戳 (timestamp) 选项包含一个初始为空的表，从源站到目的站的路径上每一个路由器各在表中填入一项，在这一点上时间戳选项与记录路由选项的工作方式类似。表中每项都包含两个 32 比特的条目：填写表项的路由器的 IP 地址以及一个 32 位整数时间戳。图 6.14 显示了时间戳选项的格式。

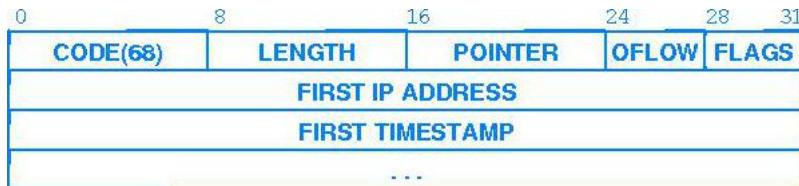


图 6.14 时间戳选项的格式。标志字段中的比特位控制路由器处理此选项的确切格式和规则

在图 6.14 中，长度字段和指针字段分别用于指明为选项所保留的空间大小及下一个未使用的空间位置（与记录路由选项完全相同）。4 位的溢出（OFLOW）字段含有一个整数计数器，它负责统计因为选项空间太小而无法提供时间戳的路由器个数。

4 位的标志字段中的值用于控制选项的确切格式，并告诉路由器应该如何提供时间戳。这些值是：

Flags value	Meaning
0	Record timestamps only; omit IP addresses.
1	Precede each timestamp by an IP address (this is the format shown in Figure 6.14).
3	IP addresses are specified by sender; a router only records a timestamp if the next IP address in the list matches the router's IP address.

图 6.15 时间戳选项的标志字段中一些值的解释

时间戳给出了路由器处理数据报的时间和日期，其表示采用自**世界时** (Universal Time) 的午夜开始计算的毫秒数^①。如果时间的标准表示不可用，假如路由器设置了时间戳字段的高位比特，则路由器可以使用任意一种本地时间表示。当然，即使时间都采用世界时表示，各自独立的计算机给出的时间戳也可能不一致；每台机器都按自己的本地时钟报告时间，但时钟可能不同。因此，无论采用哪种时间表示，时间戳表项总是被视为估计时间。

时间戳选项让路由器在记录时间戳的同时也记录 IP 地址，而记录路由选项已经提供了地址记录的能力，这一重合似乎有些令人费解。但是，把时间戳和 IP 地址一起记录，可以消除时间表示不一致的影响。让每个地址伴随一个时间戳记录也很有用，因为这样就允许接收方知道数据报是沿着哪条路径而行的。

6.8.4 分片时对选项的处理

选项的代码 (CODE) 字段中的复制 (COPY) 比特的意义现在应该很清楚了。在给数据报分片时，路由器把一些选项复制到所有数据报片中，而把其他选项只放到一个数据报片中。例如，考虑用来记录数据报路由的选项。我们知道，所有数据报片都作为独立的数据报处理，所以不能保证它们都沿着同一条路径到达目的站。如果每个数据报片都包含记录路由选项，那么目的主机可能会从各个数据报片中收到不同的路由表。目的主机无法给重装的数据报产生一个有意义的路由表。因此，IP 标准规定，记录路由选项只能复制到其中一个数据报片中。

^① 世界时以前称为格林尼治标准时间 (Greenwich Mean Time)，指的是在本初子午线上每天的时间。

并非所有 IP 选项都可以只复制到一个数据报片中。以源路由选项为例，它指明数据报应该如何通过互联网传播。源路由选择信息必须复制到所有数据报片的首部，否则这些数据报片将不会沿着指明的路由传输。因此，源路由的代码字段指明该选项必须复制到所有数据报片。

6.9 小结

TCP/IP 互联网软件所提供的基本服务是一种无连接的、不可靠的、尽最大努力的分组交付系统。网际协议（IP）正式规定了互联网分组的格式，称其为**数据报**（datagram），并且非正式地表述了无连接交付的思想。本章重点讨论了数据报的格式，后面几章将讨论 IP 转发和差错处理。

类似于一个物理帧，IP 数据报划分成首部和数据区。在数据以外的其他信息中，数据报首部包含源和目的 IP 地址、分片控制、优先级以及用来捕捉传输差错的检验和。除了固定长度的字段，每个数据报首部都可包含一个选项字段。选项字段的长度可变，与选项号、选项类以及分配给每个选项的数据区的大小有关。通过使用选项，有助于监视和控制互联网，允许指明或记录路由信息，或随着数据报在互联网上的传输收集时间戳。

6.10 深入研究

Postel [1980]讨论了互联网协议、编址和转发的可能实现方法。在后来发表的著作中，Postel [RFC 791]给出了网际协议的标准。Braden [RFC 1122]进一步改进了标准。Hornig [RFC 894]规范了在以太网上传输 IP 数据报的标准。Clark [RFC 815]描述了数据报片的高效重装；Kent and Mogul [1987]讨论了分片的缺陷。

Nichols et. al. [RFC 2474]给出了数据报首部中服务类型比特的区分服务解释，Blake et. al. [RFC 2475]讨论了区分服务的一种体系结构。除了分组格式以外，网络协议中需要的许多常量也进行了标准化，这些值可在定期发布的官方网际协议 RFC 中找到。

在 Xerox [1981]中给出了另一种互联网协议族，即 XNS。Boggs et. al. [1980]描述了**PARC 通用分组**（PARC Universal Packet，简称 PUP）协议，它是从 XNS 中抽象出来的，与 IP 数据报密切相关。

6.11 习题

1. 让 IP 检验和的计算只包括数据报首部而不包括数据，这样做的一个最大好处是什么？缺点是什么？
2. 在以太网上发送分组时，使用 IP 检验和究竟有没有必要？试解释原因。
3. 帧中继网络的 MTU 是多大？802.11 网络呢？**超信道**（Hyperchannel）网络呢？
4. 你希望高速局域网的 MTU 比广域网的更大还是更小？
5. 论证数据报片应该具有小的非标准首部。
6. 找出 IP 协议版本最近在何时做过修改。协议版本号有没有用？
7. 在前面习题的基础上，试论证，如果 IP 版本改变了，指派一个新的帧类型比把版本号编码到数据报中更有意义。
8. 你知道为什么给 IP 选择一个二进制反码的检验和，而不采用循环冗余检验吗？
9. 在最终目的站重装数据报，而不是在数据报经过一个网络传送后重装，有什么好处？
10. 要发送至少包含 1 八位组数据的 IP 数据报，所需的最小网络 MTU 是多少？
11. 假设需要你用硬件实现 IP 数据报处理。你是否有办法重新组织首部中的各字段，以使硬

件更高效？更容易构造？

12. 如果你已取得一个 IP 实现，试修改它并测试本地的 IP 实现，看看它们是否会拒绝那些具有过期版本号的 IP 数据报。
13. 当一个最小长度的 IP 数据报在以太网上传输时，帧有多大？
14. 服务类型字段的区分服务解释允许多达 64 个不同的服务级别。试论证只需要少数几个服务级别（也就是说，列举出用户可能会用到的所有可能服务）。
15. 选择区分服务定义是为了保持与最初的服务类型优先级比特向后兼容。向后兼容会使实现的效率比其他替代方案更低吗？试解释原因。

第7章 网际协议：转发 IP 数据报

7.1 引言

我们已经知道，所有互联网服务都使用底层的无连接分组交付系统，并且在 TCP/IP 互联网上的基本传送单元是 IP 数据报。本章进一步描述无连接服务，说明路由器如何转发 IP 数据报，以及如何把它们交付给最终目的站。我们认为第 6 章介绍的数据报格式刻画的是网际协议静态的一面。本章中描述的转发是有关操作方面的；下一章将阐述如何处理差错，从而结束有关 IP 的基本陈述。然后，第 9 章将描述无分类编址和子网编址扩展，后面的章节讲解其他协议如何使用 IP 来提供更高层服务。

7.2 互联网中的转发

按过去的传统说法，术语**路由选择** (routing) 用在因特网之类的分组交换系统中，指的是为要发送的分组选择一条路径的过程，术语**路由器** (router) 则是指做出这种选择的一台计算机。近来，工程人员采用术语**转发** (forwarding) 来表示为分组选择路径的过程，但保留了术语“路由器”来表示做出选择的系统。我们将遵循目前流行的用法，使用术语“转发”。

转发发生在几个层级。例如，在连接多个物理网段的一个交换式以太网内，从以太网帧开始进入网络直到交付给目的主机，帧的转发都是由交换机负责的。这种内部转发完全自包含在网络内部。外部的机器无法参与决策，它们仅仅把网络看成是一个接受并交付分组的实体。

需要记住的是：IP 的目的就是提供包含多个物理网络的一个虚拟网络，并提供无连接的数据报交付服务。因此，我们将重点关注**IP 转发** (IP forwarding)，其传统的称法是**IP 路由选择** (IP routing)。用于做出转发决策的信息称为**IP 转发信息** (IP forwarding information)。与单个物理网络内部的转发类似，IP 转发是为要发送的数据报选择路径。与单个网络内的转发不同的是，IP 转发算法必须决定如何通过多个物理网络发送数据报。

在一个互联网内进行转发可能很困难，在那些具有多个物理网络连接的计算机之间转发更是如此。理想状态下，转发软件在选择最佳的路径时，要查看网络负载、被运载的数据或在数据报首部指明的服务类型。但是，大多数互联网转发软件远没有这么复杂，它们往往根据有关最短路径的固定假设来选择路由。

为了完全理解 IP 转发，必须回顾 TCP/IP 互联网的体系结构。首先，互联网由多个物理网络组成，这些物理网络通过称为路由器的系统互连起来。每个路由器与两个以上的物理网络有直接的连接。相比之下，计算机主机通常直接与一个物理网络连接。但是，也有直接与多个网络相连的多归属主机。

主机和路由器都参与了把 IP 数据报转发到目的站的过程。当一个主机上的应用程序试图进行通信时，TCP/IP 协议将产生一个或多个 IP 数据报。主机在决定把数据报发到何处时，必须做出最初的转发决策。如图 7.1 所示，即使主机只有一个网络连接，也必须做出转发决策。

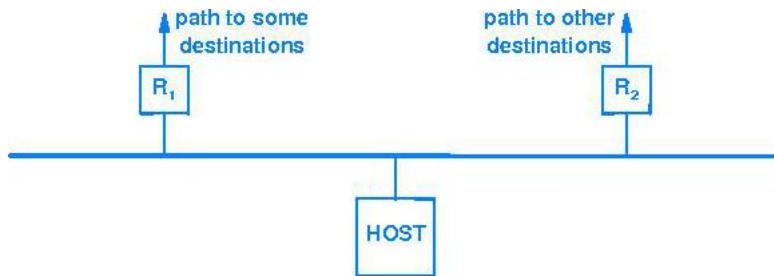


图 7.1 必须转发数据报的单归属主机举例。该主机必须选择将数据报发给路由器 R_1 还是 R_2 ，因为每个路由器都提供通往某些目的站的最佳路径

IP 路由器的主要目的是做出 IP 转发决策。那么，多归属主机如何呢？任何具有多个网络连接的计算机都可以充当路由器，并且正如我们将看到的，运行 TCP/IP 的多归属主机具有数据报转发所需的所有软件。实际上，那些不能提供独立路由器的网点，有时将通用计算机既当成主机又当成路由器。但是，TCP/IP 标准明确规定了主机与路由器的功能之间的区别，并且那些试图在单个机器上混用主机和路由器功能的网点，有时会发现它们的多归属主机忙于处理一些意想不到的交互。目前，我们会把主机和路由器区分开，并假定主机不完成路由器的功能——把分组从一个网络传递到另一个网络。

7.3 直接和间接交付

不严格地讲，我们可以把转发分成两种形式：**直接交付** (direct delivery) 和**间接交付** (indirect delivery)。直接交付是指把数据报从一台机器通过物理网络直接传输到另一台机器，这是所有互联网通信的基础。只有当两台机器同时连接到同一底层物理传输系统时（例如，一个以太网），才能进行直接交付。间接交付发生在目的站不在一个直接相连的网络中时，从而强迫发送方把数据报传递给一个路由器进行交付。

7.3.1 单个物理网络上的数据报交付

我们知道，在一个给定物理网络上的机器可以把物理帧直接发送到同一网络的另一台机器上。为了传送 IP 数据报，发送方把数据报封装在物理帧中，如第 6 章所述，并使用第 5 章所述的 ARP 协议把目的 IP 地址映射成物理地址，然后使用网络硬件传送帧。因此，理解直接交付所需的各部分内容，前面都已研讨过了。总而言之：

同一物理网络上两台机器之间传输 IP 数据报与路由器无关。发送方把数据报封装在一个物理帧中，把目的 IP 地址绑定到一个物理硬件地址，并把封装生成的帧直接发送到目的站。

发送方怎样知道目的站是否位于直接相连的网络上呢？测试方法很简单。每个 IP 地址被划分成一个特定网络的前缀和一个特定主机的后缀。为了判断目的站是否在同一直接相连的网络上，发送方从目的 IP 地址中把网络部分抽取出来，并把抽取出来的比特与自己 IP 地址中的网络部分进行比较。如果匹配，则意味着数据报可以直接发送。从这里可以看出因特网编址方案的一个优点：

因为同一网络上所有机器的互联网地址都有一个相同的网络前缀，而且抽取前缀只需几条机器指令，所以测试某一机器是否可以到达是非常高效的。

从互联网的角度来看，即使数据报要穿越许多网络和中间路由器，我们也很容易理解直接交付

是任何数据报传输的最后一步。在数据报的源站到目的站的路径上，最后一个路由器与目的站直接连接在同一个物理网络上。因此，最后一个路由器将使用直接交付来交付数据报。我们也可以把源站和目的站之间的直接交付看成是一般意义的转发的一个特例，即数据报在一个直通的路由中并不经过任何中间路由器。

7.3.2 间接交付

间接交付比直接交付更困难一些，因为发送方必须识别第一个路由器，从而可以把数据报发送给它。然后，路由器必须朝着目的网络继续转发数据报。

为了使间接转发的工作方式更直观，我们可以想象这样一个大型互联网：许多网络通过一些路由器互连起来，但只有两台主机位于相隔很远的两端。当一台主机要向另一台主机发送数据报时，它把数据报封装起来并发送到最近的路由器上。我们知道它肯定可以到达一个路由器，因为所有物理网络都是互连的，因此每个网络肯定至少与一个路由器相连。这样，发送方主机可以使用单个物理网络到达某个路由器。一旦这个帧到达该路由器，软件把封装的数据报提取出来，然后 IP 软件在通往目的站的路径上选择下一个路由器。数据报再次被放入一个帧，并通过下一个物理网络发送到下一个路由器，以此类推，直到它能够被直接交付。此概念可总结为：

TCP/IP 互联网中的路由器形成一个相互协作的互连结构。数据报从一个路由器传递到下一个路由器，直到抵达一个可直接交付数据报的路由器。

路由器怎样知道把每个数据报发往何处呢？主机怎样知道对于某个指定目的站究竟使用哪一个路由器呢？这两个问题是相关的，因为它们都与 IP 转发有关。我们将分两步来回答这个问题。本章先介绍基本的表驱动转发算法，关于路由器如何获得有关目的站的信息推迟到第 13 章至第 15 章讨论。

7.4 表驱动 IP 转发

IP 转发算法使用每台机器上的一个数据结构来存储有关可能的目的站及怎样到达目的站的信息。该数据结构的正式名称是**网际协议路由表**(Internet Protocol routing table)或**IP 路由表**(IP routing table)，非正式的简称为**路由表**^①。

由于主机或路由器都为数据报选择路由，因此它们都有 IP 路由表。当主机或路由器中的 IP 转发软件需要传输数据报时，它就查询路由表来决定把数据报发往何处。

什么信息应该保存在路由表中呢？如果每个路由表都含有所有可能目的站的地址信息，则不太可能让表始终符合当前的实际情况。此外，一些小型的专用系统没有足够的空间来存储信息，然而由于可能的目的站的数量很大，从而导致它们不能连接到因特网。

从概念上讲，我们可以使用信息隐藏的原理，允许机器使用最少的信息做出转发决策。例如，我们把特定主机有关的信息隔离在它们存在的本地环境中，并设法让远端的机器在不知道这些信息细节的情况下，可以把分组转发给这些主机。幸运的是，IP 编址方案有助于实现这个目标。前面讲过，IP 地址的分配使所有连接到某个物理网络上的机器共享一个相同的前缀（地址的网络部分）。我们已经知道这种分配使直接交付的测试非常高效。这也意味着路由表中仅需要包含网络前缀的信息而不需要完整的 IP 地址。

^① 工程人员有时使用术语**IP 转发表**(IP forwarding table)作为 IP 路由表的同义词。

7.5 下一跳转发

使用目的地址的网络部分而不用完整的主机地址，使转发效率很高，同时也可以让路由表保持较小。更重要的是，它有助于隐藏信息，把特定主机的信息局限在这些主机运行的本地环境中。典型的情况是，一个路由表包含许多对 (N, R) ，其中 N 是目的网络的 IP 地址， R 是通往网络 N 的路径上“下一个”路由器的 IP 地址^①。路由器 R 称为**下一跳** (next hop)，用路由表为每个目的站存储下一跳的做法称为**下一跳路由选择** (next-hop routing) 或**下一跳转发** (next-hop forwarding)。因此，路由器 R 中的路由表仅仅指明了从 R 到某个目的网络的路径上的下一步——路由器并不知道到目的站的完整路径。

路由表中每个表项指向一个可通过单个网络到达的路由器，理解这一点很重要。也就是说，机器 M 的路由表中列出的所有路由器必须位于 M 直接相连的网络上。当一个数据报准备好离开 M 时，IP 软件找到目的 IP 地址，并提取出地址的网络部分。然后 M 使用网络部分做出转发决策，选择一个可直接到达的路由器。

在实践中，我们也把信息隐藏的原理应用到主机上。我们强烈建议，尽管主机有 IP 路由表，但它们必须在自己的表中保存最少的信息。我们的想法是强迫主机依靠路由器进行大多数转发。

图 7.2 显示的具体例子有助于我们解释路由表。示例的互联网由 4 个网络组成，用 3 个路由器相互连接起来。图中的表对应路由器 R 所用的路由表。由于 R 直接连接到网络 20.0.0.0 和 30.0.0.0，它可以用直接交付把数据报发送到这两个网络上的任一台主机（可能使用 ARP 查找物理地址）。假定有一个数据报的目的站在网络 40.0.0.0 上， R 为它选择路由为地址 30.0.0.7 的路由器 S 。然后， S 将直接交付数据报。 R 可以到达地址 30.0.0.7，因为 R 和 S 都直接连到网络 30.0.0.0。

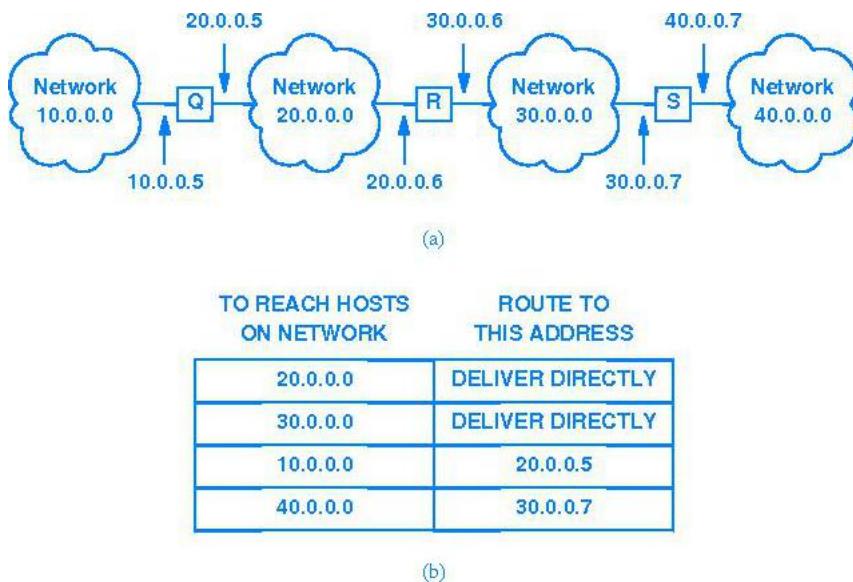


图 7.2 (a) 有 4 个网络和 3 个路由器的互联网例子；(b) R 中的路由表

如图 7.2 所示，路由表的大小取决于互联网中网络的数量；仅在增加新网络时表才会增大。也就是说，表的大小和内容与网络上连接的单个主机数量无关。我们可以把基本原理总结如下：

为了隐藏信息，应该保持路由表较小并使转发决策效率更高，IP 转发软件保存的只是有关目的网络地址的信息，而不是有关个体主机地址的信息。

仅仅根据目的网络 ID 来选择路由有几个后果。第一，在大多数实现中，这意味着所有目的地

^① 实际上，路由表中的一个表项还指明了数据报被发送到下一个路由器时所用的网络接口，以及已使用的时间计时之类的管理信息。

在某个网络的通信量都沿着同一条路径。因此，即使存在多条路径，它们也不可能被并发使用。而且，在最简单的情况下，所有通信量都沿着同一条路径通过，不考虑物理网络的时延或吞吐量。第二，因为只有路径上的最后一个路由器才和目的主机通信，所以只有它能确定主机是否存在或正在运转。因此，我们需要设法让最后一个路由器可以向初始源站发回交付情况的报告。第三，因为每个路由器独立转发通信量，所以数据报从主机 A 运送到主机 B 所沿的路径，可能与数据报从主机 B 回到主机 A 的路径完全不同。我们需要确保路由器相互协作，以保证始终可以进行双向通信。

7.6 默认路由

另一种用来隐藏信息和保持路由表容量较小的技术是把多个表项合并成一个默认表项。这种思路是让 IP 转发软件首先在路由表中查找目的网络。如果表中没有路由，则转发软件把数据报发给一个**默认路由器**（default router）。

当一个网点的本地地址集很小，并且只有一条到互联网的连接时，默认转发尤其有用。例如，对于那些连接到单一物理网络的主机，它们通过唯一的路由器通向互联网其余部分，默认路由可以很好地发挥作用。转发决策由两种测试组成：一个用于本地网络，另一个默认指向唯一的路由器。即使网点中包含几个本地网络，转发也很简单，因为它只包括对几个本地网络的测试，外加一个针对所有其他目的站的默认测试。

7.7 特定主机的路由

尽管我们知道所有的转发是基于网络而不是基于个别主机的，但是多数 IP 转发软件允许指明每个主机的路由作为特例。指明每个主机的路由，使本地网络管理员能够更好地控制网络的使用，允许测试，还可出于安全目的来控制访问。在调试网络连接或路由表时，为单个主机指定一条特殊路由的能力尤其有用。

7.8 IP 转发算法

考虑到上述的所有情况，用来转发分类编址 IP 数据报的原型算法如图 7.3 所示^①。

Algorithm:

ForwardDatagram (Datagram , RoutingTable)

```
Extract destination IP address, D, from the datagram;
if the table contains a host-specific route for D
    send datagram to next-hop specified in table and quit;
compute N, the network prefix of address D;
if N matches any directly connected network address
    deliver datagram to destination D over that network;
    (This involves resolving D to a physical address,
     encapsulating the datagram, and sending the frame.)
else if the table contains a route for network prefix N
    send datagram to next-hop specified in table;
else if the table contains a default route
    send datagram to the default router specified in table;
else declare a forwarding error;
```

^① 第 9 章讨论了目前与无分类 IP 地址一起使用的改进算法。

图 7.3 用来转发 IP 数据报的原型算法。给定一个 IP 数据报和一个路由表，算法选择数据报应该发往的下一跳。
所有路由都必须指明下一跳，而下一跳位于直接连接的网络上

7.9 利用 IP 地址转发

除了减少生存时间及重新计算检验和以外，IP 转发并不改变原始的数据报，理解这一点是十分重要的。特别是，数据报的源地址和目的地址始终保持不变，指明了最初源站及最终目的站的 IP 地址^①。当 IP 执行转发算法时，它总是选择一个新的 IP 地址，即数据报下一步将发送到的机器的 IP 地址。这个新地址很可能是一个路由器的地址，但如果数据报能直接交付，则新地址与最终目的站的 IP 地址相同。

我们说 IP 转发算法所选择的 IP 地址是“下一跳”地址，是因为它指明了下一步应该把数据报发往何处。IP 把下一跳地址存储在哪里呢？不在数据报中，数据报中没有为它预留空间。事实上，IP 根本不保存下一跳地址。在执行转发算法后，IP 把数据报及下一跳地址传递给一个网络接口软件，该接口软件负责数据报必须发往的那个物理网络。这个接口软件把下一跳地址绑定到一个物理地址，使用这个物理地址形成一个帧，把数据报放在该帧的数据部分，然后把形成的帧发送出去。在使用下一跳地址找到一个物理地址后，网络接口软件就会丢弃下一跳地址。

路由表存储每个目的网络的下一跳的 IP 地址，在数据报能够发送之前，必须把这些地址转换成相应的物理地址，这似乎有点奇怪。设想一台主机向同一个目的地址发送一个数据报序列，使用 IP 地址似乎会使效率变得相当低。IP 尽职尽责地抽取每个数据报中的目的地址，并用路由表产生一个下一跳地址。然后，它把数据报及下一跳地址传递给网络接口，在那里重新计算得到一个物理地址的绑定。如果路由表使用物理地址，那么下一跳的 IP 地址与物理地址的绑定可以一次完成，能够省去不必要的计算。

那么，为什么 IP 软件在存储和计算路由时避免使用物理地址呢？如图 7.4 所示，有两个重要的原因。

首先，路由表在转发数据报的 IP 软件和操纵路由的高层软件之间提供了一个特别清晰的接口。为了调试转发问题，网络管理员通常需要检查路由表。在路由表中只使用 IP 地址，使管理人员很容易理解并判断软件是否已正确更新了路由。其次，网际协议的目标就是建立一个隐藏底层网络细节的抽象。

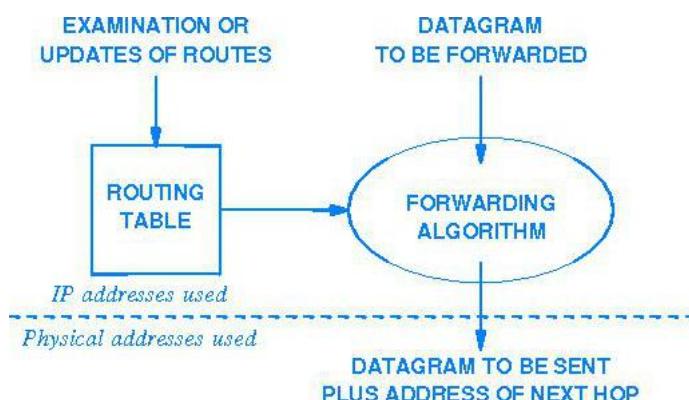


图 7.4 IP 软件及路由表通常位于地址边界的上面。仅使用 IP 地址更易于检查或改变路由，并且隐藏了物理地址的细节

图 7.4 显示了**地址边界** (address boundary)，这是低层软件和互联网软件之间的重要概念性划

^① 唯一的例外发生在数据报含有源路由选项的情况下。

分，低层软件知道物理地址，而互联网软件只使用高层地址。在这个边界的上面，所有软件都可以编写成使用互联网地址通信；对物理地址的理解被移交给几个小的低层例程。我们将看到，保持这种界限也有利于让其余 TCP/IP 协议的实现更易于理解、测试和修改。

7.10 处理传入的数据报

到目前为止，我们已经通过描述如何就传出的分组做出转发决策讨论了 IP 转发。但是，很明显，IP 软件还必须对传入的数据报进行处理。

当一个数据报到达主机时，网络接口软件就把它交付给 IP 模块进行处理。如果数据报的目的地址与主机的 IP 地址匹配，则主机上的 IP 软件就接受该数据报，并把它传递给相应的高层协议软件进一步处理。如果目的 IP 地址不匹配，则要求主机丢弃该数据报（也就是说，禁止主机试图转发偶然错误转发给它的数据报）。

与主机不一样，路由器会对数据报进行转发。当一个数据报到达路由器时，它被交付给 IP 软件。这又产生两种可能：数据报已到达它的最终目的站，或者可能需要继续传递。与主机相同的是，如果数据报的目的 IP 地址与路由器自己的 IP 地址匹配，则 IP 软件把数据报传递给高层协议软件进行处理^①。如果数据报还没有到达最终目的站，IP 就使用标准算法和本地路由表中的信息转发数据报。

判断一个 IP 数据报是否到达它的最终目的站并不像看起来那么简单。要记住，路由器有多个物理连接，每个连接都有自己的 IP 地址。当一个 IP 数据报到达时，接收的机器必须把目的互联网地址与它的每个网络连接的 IP 地址进行比较。如果有任何一个匹配，就保留该数据报并对它进行处理。如果数据报的目的 IP 地址是受限 IP 广播地址，或目标为该网络的定向 IP 广播地址，机器也必须接受在物理网络上广播的数据报。在第 9 章和第 16 章中还将看到。无分类地址、子网地址和多播地址使地址识别变得更加复杂。在任何情况下，如果数据报的目的地址与所有本机地址都不匹配，IP 就会把数据报首部中的生存时间字段值减 1，如果该字段值减到“0”，则丢弃该数据报；如果该字段值仍为正数，则计算一个新的检验和并转发数据报。

每台机器是否都应该转发它收到的 IP 数据报呢？很明显，路由器必须转发传入的数据报，因为这就是路由器的主要功能。我们已经讲过，一些网点还配置了通用的计算机充当路由器的角色，而这些计算机必须加以配置，以完成数据报的转发。但是，其他没有被指派作为路由器的主机不应该转发数据报；如果这样的主机收到一个与所有主机地址都不匹配的数据报，它必须丢弃数据报。

没有被指派作为路由器的主机应该避免完成任何路由器的功能，其原因有四点。第一，当这种主机接收到原本打算发送给其他主机的数据报时，就说明互联网的编址、转发或交付在某个地方已经出了点问题。如果主机通过转发数据报加以纠正，这些问题就会被掩盖；第二，转发会造成多余的网络通信量（而且会占用主机合法使用 CPU 的时间）；第三，一些简单的差错可能会引起混乱。假定每个主机都转发通信量，如果一台机器偶然广播了一个目的站为主机 H 的数据报，想象一下会发生什么情况。因为数据报已经被广播了，所以网络上的每个主机都会收到一个该数据报的副本。每个主机都会把它的副本转发给 H，从而使 H 遭受众多副本的轰击；第四，正如在后面章节中所说明的，路由器不仅仅转发通信量。在下一章中将会阐述，路由器使用一种特殊的协议报告差错，而主机却不是这样（同样是为了避免多个差错报告轰击源站）。路由器还会传播信息，以保证它们的路由表是一致和正确的。如果主机转发数据报但又不具备全部的路由器功能，可能就会出现难以预料的异常情况。

^① 通常，目的站为路由器的数据报被用来测试连通性或携带路由器管理命令，但路由器也接收任何一个在网络上广播的数据报副本。

7.11 建立路由表

我们已经讨论了 IP 如何根据路由表的内容转发数据报，但并没有说明系统如何初始化路由表，或者如何随着网络的改变而更新路由表。后面几章里将答复这些问题，并讨论允许路由器保持路由一致的协议。目前，最重要的是理解 IP 软件在决定如何转发数据报时就要使用路由表，所以改变路由表将改变数据报所经过的路径。

7.12 小结

IP 软件负责转发数据报，所需的计算包括根据目的 IP 地址确定数据报发往何处。如果目的机器位于发送方相连的那个网络上，则可以使用直接交付；我们把这种方式看成是数据报传输过程中的最后一步。如果发送方无法直接连到目的站，它必须把数据报转发给一个路由器。一般的模式是主机间接把要转发的数据报发给最近的路由器；数据报通过互联网从一个路由器移动到另一个路由器，直到数据报可以通过一个物理网络直接交付。

IP 把转发所需的信息保存在一个表中，该表称为路由表。当 IP 查找一个路由时，算法会生成数据报要发送到的下一台机器的 IP 地址（即下一跳的地址）；IP 把数据报和下一跳地址传递给网络接口软件。数据报从一台机器传输到下一台机器，该过程通常包括把数据报封装在一个物理帧中，把下一跳互联网地址映射成一个物理地址，以及使用底层硬件发送帧。

互联网转发算法只使用 IP 地址；IP 地址和物理地址间的绑定不属于 IP 转发功能的一部分。尽管路由表可以包含一个特定主机的目的地址，但大多数路由表都只含有网络地址，以保持路由表较小。使用默认路由也有助于保持路由表较小，特别是对于那些只接入一个路由器的主机。

7.13 深入研究

转发是一个重要的课题。Frank and Chou [1971]一般性地讨论了转发；Postel [1980]讨论了因特网转发。Baker [RFC 1812]对因特网路由器如何处理 IP 数据报进行了总结。Narten [1989]对因特网转发进行了综述。Fultz and Kleinrock [1971]分析了自适应转发方案；McQuillan, Richer, and Rosen [1980]描述了 ARPANET 自适应转发算法。

人们常常考虑用策略语句 (policy statement) 将转发有关的规则公式化地加以表示，Leiner [RFC 1124]深入考察了互连网络所用的策略。Braun [RFC 1104]讨论了互联网的转发策略模型，Rekhter [RFC 1092]把转发策略和第二个 NSFNET 主干网联系起来，Clark [RFC 1102]描述了 IP 的策略转发用法。

7.14 习题

1. 把图 7.2 中的所有路由器的路由表都补充完整。哪个路由器使用默认路由获益最多？
2. 研究你的本地系统所用的转发算法。它有没有涵盖本章中提到的所有转发情况？该算法允许其他没有提到的情况吗？
3. 路由器如何处理 IP 首部的生存时间 (time to live) 值？
4. 假设一台机器有两条物理网络连接和对应的两个 IP 地址 (I_1 和 I_2)。它是否可能在含有地址 I_2 的网络上收到一个目的站为 I_1 的数据报？试解释原因。
5. 在上题中，如果发生了这种情况，应该做出什么响应？
6. 假设有两个主机 A 和 B。它们都连到同一物理网络 N。当使用图 7.3 中讨论的转发算法时，

-
- A 是否可能收到一个目的站为 B 的数据报？试解释原因。
7. 修改转发算法，使它适用于第 6 章中讨论的 IP 源路由选项。
 8. 每当 IP 路由器处理一个数据报时，它必须完成一次计算，计算所需的时间与该数据报首部长度成正比。试解释原因。
 9. 一位网络管理员认为，为了使自己对本地网络的监视和调试更容易，要重写转发算法，从而让转发算法在测试直接交付之前先测试特定具体主机的路由。他应该怎样用修改过的算法建立一个网络监视器？
 10. 是否可能把数据报的地址指定为某个路由器的 IP 地址？这样做是否有意义？
 11. 假设有一个修改过的转发算法，该算法在测试直接相连网络上的交付之前先检查特定主机的路由。在什么情况下这种算法是可取的，在什么情况下是不可取的？
 12. 有一个人在某天晚上对一个局域网进行探测，他在对局域网上的 IP 通信量监视了 10 分钟后，注意到：所有目的站为机器 A 的帧，其携带的 IP 数据报的目的地址等于 A 的 IP 地址；然而所有目的站为机器 B 的帧，其携带的 IP 数据报的目的地址却不等于 B 的 IP 地址。用户报告 A 和 B 都能够进行通信。试解释原因。
 13. 如何改变 IP 数据报格式才能支持路由器上的高速分组交换？提示：一个路由器在生存时间字段值减 1 后，必须重新计算首部的检验和。

第8章 网际协议：差错与控制报文（ICMP）

8.1 引言

前几章讲解了网际协议软件如何让各个路由器转发数据报，以提供可靠的无连接数据报交付服务。数据报从一个路由器传到另一个路由器，直到数据报到达某个能够直接将其交付到最终目的地的路由器。如果路由器不能转发或交付数据报，或者如果它检测到影响转发数据报的异常条件（例如网络拥塞），则需要通知最初源站采取措施，以避免或纠正问题。本章讨论因特网主机和路由器用来传递这种控制或差错信息的一种机制。我们将看到路由器使用该机制来报告问题，主机使用该机制来测试目的站是否可达。

8.2 网际控制报文协议

到目前为止，在我们所描述的无连接系统中，每个路由器都是自主运行的，转发或交付到达的数据报都没有与最初发送方协调。如果所有主机和路由器运转正常，并就路由达成一致，这个系统将工作得很好。遗憾的是，没有一个大型通信系统能在任何时候都正常运转。除了通信线路和处理器故障以外，当目的机器临时或永久断开网络连接、生存时间计时器超时或者中间路由器拥塞得无法处理进入的通信量时，IP 交付数据报就会失败。用同构的专用硬件实现单一网络，与用多个独立系统实现一个互联网相比，两者之间的重要区别是：在单一网络中，设计者可以设法让底层硬件在产生问题时通知与其相连的主机，而在互联网中则没有这种硬件机制，发送方无法辨别某一次交付失败是由于本地故障还是远端故障造成的。调试变得极为困难。IP 协议本身没有任何机制可以帮助发送方测试连通性或了解这种故障。

为了让互联网中的路由器报告差错或提供关于意外情况的信息，设计人员给 TCP/IP 协议补充了一个特殊用途的报文机制。这种机制称为**网际控制报文协议**（Internet Control Message Protocol，简称 ICMP），它是 IP 不可缺少的一部分，在每个 IP 实现中都必须包含它^①。

与所有其他通信量类似，ICMP 报文被放在 IP 数据报的数据部分，通过互联网进行传送。ICMP 报文的最终目的地不是一个应用程序，也不是目的机器上的用户，而是目的机器上的网际协议软件。也就是说，当一个 ICMP 报文到达后，ICMP 软件模块对它进行处理。当然，如果 ICMP 确定是某个特定的高层协议或应用程序引起的问题，它会通知相应的模块。我们可以总结如下：

网际控制报文协议允许路由器向其他路由器或主机发送差错或控制报文；ICMP 在一台机器上的网际协议软件与另一台机器上的网际协议软件之间提供通信。

最初的设计是为了允许路由器向主机报告交付出错的原因，但是 ICMP 的使用并不局限于路由器。尽管一些原则限制了某些 ICMP 报文的使用，但是任何一台机器都可以向任何其他机器发送 ICMP 报文。因此，一个主机可以用 ICMP 与路由器或另一个主机通信。允许主机使用 ICMP 的主要优点是：它为所有控制报文和信息报文提供一种单一的使用机制。

^① 当有必要指出 ICMP 是 IP 版本 4 的一部分时，我们将其写为 ICMPv4。

8.3 差错报告与差错纠正的对比

从技术上讲，ICMP 是一种**差错报告机制**（error reporting mechanism）。它为遇到差错的路由器提供了向最初源站报告差错的方法。尽管协议规范概述了 ICMP 的用途并建议了针对差错报告可能采取的措施，但 ICMP 并没有完全为每个可能的差错指明采取何种措施。简而言之：

当一个数据报产生差错时，ICMP 只能向数据报的最初源站报告差错情况；源站必须把差错告诉给一个单独的应用程序或采取其他措施来纠正问题。

大多数差错出自最初源站，但有的差错并非如此。由于 ICMP 向最初源站报告问题，它不能用来把问题通知给中间路由器。例如，设想一个数据报沿着一条通过路由器序列 R_1, R_2, \dots, R_k 的路径传输。如果 R_k 有不正确的路由信息，并且错误地把数据报转发到路由器 R_E ， R_E 就不能用 ICMP 把差错报告返回给 R_k ；ICMP 只能向最初源站发回一个报告。遗憾的是，最初源站不对行为异常的路由器负责，也不对它们进行控制。事实上，源站无法确定是哪个路由器导致了问题的产生。

为什么要限制 ICMP 只和最初源站通信呢？从前几章对数据报格式和转发的讨论中可以找到答案。数据报只含有指明最初源站和最终目的站的字段，并不包含它在互联网上的行程的完整记录（除非数据报使用了记录路由选项）。此外，由于路由器只能建立和改变自己的路由表，所以不了解路由的全局情况。因此，当一个数据报到达指定的路由器时，无法知道该数据报到达这里所走的是哪条路径。如果路由器检测到出现的问题，由于 IP 无法知道哪些中间机器处理过数据报，因而无法把问题通知给它们。但是，路由器并不会悄悄地丢弃数据报，而是使用 ICMP 通知最初源站发生了问题，并相信主机管理员会与网络管理员协作找到问题并进行修复。

8.4 ICMP 报文交付

ICMP 报文需要两级封装，如图 8.1 所示。每个 ICMP 报文被放在 IP 数据报的数据部分，通过互联网传送，而 IP 数据报本身被放在帧的数据部分，通过物理网络传送。携带 ICMP 报文的数据报完全就像携带用户信息的数据报一样转发，没有额外的可靠性或优先级。因此，差错报文本身可能会丢失或被丢弃。此外，在一个已出现拥塞的网络中，差错报文可能会引起额外的拥塞。如果携带 ICMP 报文的 IP 数据报产生了差错，则差错处理过程会产生异常。产生异常是为了避免出现有关差错报文的差错报文，即携带 ICMP 差错报文的数据报中出现差错时不会生成 ICMP 报文。

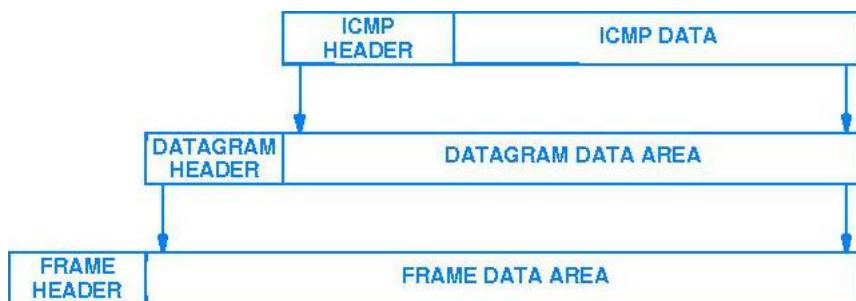


图 8.1 两级 ICMP 封装。ICMP 报文封装在 IP 数据报中，IP 数据报封装在帧中传输。为了标识 ICMP，数据报协议字段包含的值为“1”

即使 ICMP 报文是使用 IP 封装和发送的，也不要把它当成高层协议，它是 IP 的一个必要组成部分。记住这一点很重要。ICMP 报文使用 IP 进行交付，是因为它们可能需要经过几个物理网络才能到达其最终目的地。因此，它们不能单独通过物理传输进行交付。

8.5 ICMP 报文格式

尽管每个 ICMP 报文有自己的格式，但它们开头的三个字段都是相同的：一个 8 位整数报文类型 (TYPE) 字段用来标识报文；一个 8 位代码 (CODE) 字段提供有关报文类型的更多信息，还有一个 16 位检验和 (CHECKSUM) 字段 (ICMP 使用与 IP 相同的加法检验和算法，但 ICMP 检验和的计算只包括 ICMP 报文)。此外，报告差错的 ICMP 报文总是包括从产生问题的数据报中取出来的首部和一些八位组^①。

ICMP 不仅仅返回出错数据报的首部的原因是，让接收方能够更准确地判断是哪个 (些) 协议以及哪个应用程序负责该数据报。正如将在后面看到的，TCP/IP 协议族中的更高层协议被设计为将最重要的信息编码到 IP 首部之后的前 64 位。

ICMP 类型 (TYPE) 字段定义了报文的格式及含义。图 8.2 列举了一些可能的 ICMP 报文类型：

Type Field	ICMP Message Type
0	Echo Reply
3	Destination Unreachable
4	Source Quench
5	Redirect (change a route)
6	Alternate Host Address
8	Echo Request
9	Router Advertisement
10	Router Solicitation
11	Time Exceeded for a Datagram
12	Parameter Problem on a Datagram
13	Timestamp Request
14	Timestamp Reply
15	Information Request
16	Information Reply
17	Address Mask Request
18	Address Mask Reply
30	Traceroute
31	Datagram Conversion Error
32	Mobile Host Redirect
33	IPv6 Where-Are-You
34	IPv6 I-Am-Here
35	Mobile Registration Request
36	Mobile Registration Reply
37	Domain Name Request
38	Domain Name Reply
39	SKIP
40	Photuris

图 8.2 ICMP 报文的类型字段中可能出现的值及其各自的含义。没有列举的值是未分配或保留的值

下面几节将描述每种报文，详细介绍每种报文的格式及其含义。

8.6 测试目的站的可达性和状态

TCP/IP 协议提供了一些工具来帮助网络管理员或用户识别网络问题。最常用的调试工具之一使用 ICMP 的回送请求 (echo request) 和回送回答 (echo reply) 报文。主机或路由器向某个指明的目的站发送 ICMP 回送请求报文。任何收到回送请求的机器形成一个回送回答，并把它返回给最初的发送方。回送请求包含一个可选的数据区，回答中含有在请求中所发送数据的一个副本。回送请求及其关联的回答可用来测试目的站是否可达和是否响应。因为请求和回答都是在 IP 数据报中运送的，所以回答的成功接收就证实传输系统大体上是正常的。第一，源站计算机上的 IP 软件必须转发数据报；第二，在源站和目的站之间的中间路由器必须在运行状态，并且必须正确地转发数

^① 为了保持差错报文长度小且避免分片，不太可能把产生问题的整个数据报都包含在内。

据报；第三，目的站机器必须正常运行（至少它必须响应中断），而且 ICMP 和 IP 软件必须都在工作；最后，在返回路径上所有路由器的路由表中，必须含有形成可行路径所需的信息。

在许多系统上，用户用来发送 ICMP 回送请求的命令是 ping^①。一些版本的 ping 发送固定数量的 ICMP 请求分组并等待回答；其他一些版本发送一连串 ICMP 回送请求，捕获响应，并提供关于数据报丢失情况的统计。大多数版本允许用户指明要发送数据报的长度，以及各个请求之间的间隔时间。发送一个大的 ping 分组有助于测试数据报的分片和重装。

8.7 回送请求和回答报文的格式

图 8.3 显示了回送请求和回答报文的格式。

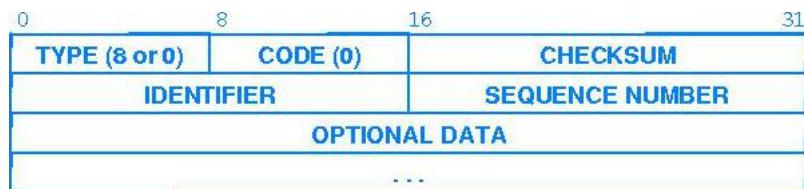


图 8.3 ICMP 回送请求或回答报文的格式

图 8.3 中的可选数据（OPTIONAL DATA）字段是一个可变长度的字段，包含将要返回给发送方的数据。回送回答返回的数据总是与收到的请求中的数据完全相同。标识符（IDENTIFIER）字段和序号（SEQUENCE NUMBER）字段被发送方用来匹配回答和请求。类型（TYPE）字段的值指明报文是一个请求（8）还是一个回答（0）。

8.8 目的站不可达的报告

当路由器无法转发或交付一个 IP 数据报时，它使用图 8.4 所示的格式向最初源站发回一个**目的站不可达**（destination unreachable）的报文。

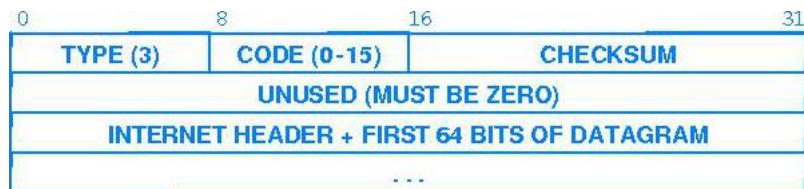


图 8.4 ICMP 目的站不可达报文的格式

目的站不可达报文中的代码（CODE）字段包含一个整数，可以进一步描述问题。图 8.5 给出了它的可能取值。

^① Dave Mills 曾经提出，ping 是 Packet InterNet Groper（分组互联网探索者）的首字母缩写词。

Code Value	Meaning
0	Network unreachable
1	Host unreachable
2	Protocol unreachable
3	Port unreachable
4	Fragmentation needed and DF set
5	Source route failed
6	Destination network unknown
7	Destination host unknown
8	Source host isolated
9	Communication with destination network administratively prohibited
10	Communication with destination host administratively prohibited
11	Network unreachable for type of service
12	Host unreachable for type of service
13	Communication administratively prohibited
14	Host precedence violation
15	Precedence cutoff in effect

图 8.5 目的站不可达报文中代码字段的可能取值

虽然 IP 是尽最大努力交付的机制，但它不应轻率地丢弃数据报。只要某个差错阻碍了路由器转发或交付一个数据报，路由器就向源站发回一个目的站不可达报文，然后丢掉（即丢弃）那个数据报。网络不可达的差错意味着在一些中间位置出现了转发故障；主机不可达的差错意味着交付失败^①。因为 ICMP 差错报文包含了发生问题的数据报的一个短前缀，所以源站会准确地知道是哪个地址不可达。

由于硬件临时不工作，或者发送方指定了一个不存在的目的站地址，或者（在一些罕见的情况下）路由器没有到达目的网络的路由，目的站可能不可达。注意，尽管路由器报告了遇到的故障，但可能并不知道所有的交付故障。例如，如果目的机器连接到一个以太网上，网络硬件并不提供确认信息。因此，在一个目的站已掉电之后，路由器由于没有接收到分组无法交付的任何提示，而会继续向那个目的站发送分组。总之：

尽管路由器遇到不能转发或交付的数据报时，会发送一个目的站不可达的报文，但它不能检测到所有这样的差错。

当研究了高层协议如何使用称为**端口**（port）的抽象目的站**端点**（point）以后，就会明白协议和端口不可达报文的含义。其余的大部分报文含义可从字面理解。如果数据报包含一个路由不正确的源路由选项，可能会激发一个**源路由**（source route）故障报文。如果路由器需要对数据报分片，但数据报的不分片位的值为“1”，路由器就向源站发回一个**需要分片**（fragmentation needed）报文。

8.9 拥塞和数据报流量控制

因为 IP 是无连接的，所以路由器在收到数据报之前不能预留内存或通信资源。结果，路由器可能出现通信量泛滥，这种情况称为**拥塞**（congestion）。拥塞可能是由于两个完全不同的原因，理解这一点很重要。第一，一个高速计算机产生通信量的速度比网络所能传输的更快。例如，想象一台超级计算机所产生的互联网通信量。尽管超级计算机本身是连在一个高速局域网上的，但数据报可能最终需要通过一个低速拨号连接。在连接局域网与广域网的路由器上就会发生拥塞，因为数据报到达的速率比发送出去的速率更快。第二，如果许多计算机同时需要通过某一个路由器发送数据

^① IETF 建议只向最初源站报告主机不可达的报文，并使用路由协议来处理其他转发问题。

报，即使没有单独哪个源站导致拥塞的发生，但该路由器仍可能会发生拥塞。

当一些数据报到达得太快，以至于主机或路由器无法处理时，它们就临时在内存中排队。如果这些数据报只是一部分零星的突发通信量，那么这种缓冲机制就可以解决问题。然后，如果通信量继续产生，主机或路由器最终会耗尽内存，从而必须放弃到达的其他数据报。机器使用 **ICMP 源站抑制**（source quench）报文向最初的源站报告数据报被丢弃的情况。源站抑制报文是发给源站的一个请求，请求它减慢当前的数据报传输速率。通常，发生拥塞的路由器为每个放弃的数据报都发送一个源站抑制报文。路由器也可能使用一些更复杂的拥塞控制技术。有些技术监视进入的通信量，并对数据报传输速率最高的源站进行抑制^①，其他一些技术则在队列开始变长但未溢出之前，就让机器发出抑制请求，从而完全地避免拥塞。

没有一个 ICMP 报文可以逆转源站抑制的作用。主机收到针对目的站 D 的源站抑制报文后，就会降低向 D 发送数据报的速率，直到不再收到源站抑制报文；然后，只要没有进一步收到源站抑制请求，它就会逐渐提高数据报发送速率。实际上，大多数实现不怎么使用源站抑制，而依靠更高层协议（如 TCP）来对拥塞做出反应。

8.10 源站抑制的格式

除了常用的 ICMP 类型（ICMP TYPE）、代码（CODE）、检验和（CHECKSUM）字段，以及一个未用的 32 位字段以外，源站抑制报文还有一个包含数据报前缀的字段。图 8.6 显示了该格式。正如大多数报告差错的 ICMP 报文一样，数据报前缀字段包含了激发源站抑制请求的数据报的前缀。

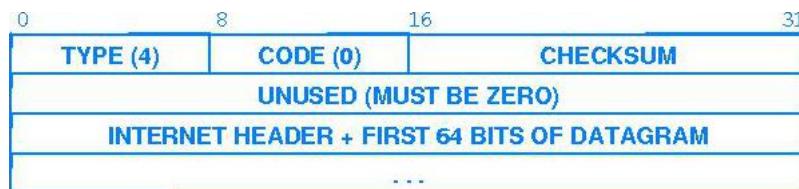


图 8.6 ICMP 源站抑制报文的格式。发生拥塞的路由器每丢弃一个数据报就发送一个源站抑制报文；数据报前缀标识了被丢弃的数据报

8.11 来自路由器的改变路由请求

互联网的路由表通常在很长一段时间内保持不变。主机在系统启动时根据一个配置文件对路由表进行初始化，而系统管理员在正常的操作过程中偶尔才会改变路由。正如将在后面章节中看到的，路由器比主机的动态性强一些，路由器定期地交换路由信息，以适应网络改变并保持它们的路由总是最新的。因此，我们将以下规则作为一般规则：

我们假定路由器知道正确的路由；主机一开始只有最少的路由信息，它从路由器那里了解新的路由。

为了有助于遵守这个规则，并避免在每个主机的配置文件中复制同样的路由信息，初始的主机路由配置只指明通信所需的最少可能的路由信息（例如，单独一个路由器的地址）。因此，主机从最少的信息开始，依靠路由器来更新它的路由表。在一种特殊情况下，当路由器检测到一个主机在使用非优化的路由时，它向该主机发送一个称为**重定向**（redirect）的 ICMP 报文，请求该主机改变其路由。路由器也把原来的数据报转发给它的目的站。

^① 第 12 章讨论了另一种称为 RED 的拥塞管理机制。

ICMP 重定向机制的优点是简单：它允许主机启动时只知道本地网络上的一个路由器地址。当主机把一个数据报发送到初始知道的那个路由器，而该路由器知道数据报有另外一条更佳的路由时，路由器就会向主机发送一个 ICMP 重定向报文。主机的路由表可以保持很小，但它仍然包含关于所有使用中的目的站的最优路由。

重定向报文仅限于在直接连到同一网络上的路由器和主机之间交互，因此它并没有一般性地解决路由信息传播的问题。图 8.7 显示了这种局限性。在图中，假设源站 S 向目的站 D 发送一个数据报。假设路由器 R₁不正确地把数据报转发到路由器 R₂而不是路由器 R₄（即 R₁ 不正确地选择了一条更长的路径）。当路由器 R₅收到该数据报时，它不能向 R₁发送 ICMP 重定向报文，因为它不知道 R₁的地址。后面的章节将探讨如何通过多个网络来传播路由信息的问题。

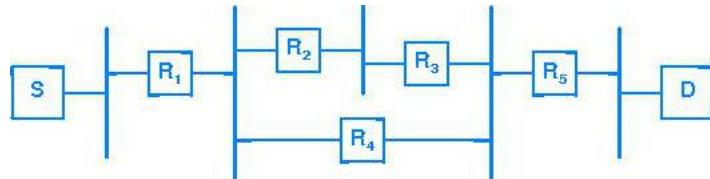


图 8.7 ICMP 重定向报文不在路由器之间传播改变路由的信息。在这个例子中，路由器 R₅不能让 R₁重定向，以使从 S 到 D 的更短路径

除了必须包含的类型、代码和检验和字段以外，每个重定向报文都包含一个 32 位路由器互联网地址 (ROUTER INTERNET ADDRESS) 字段以及一个互联网首部 (INTERNET HEADER) 字段，如图 8.8 所示。

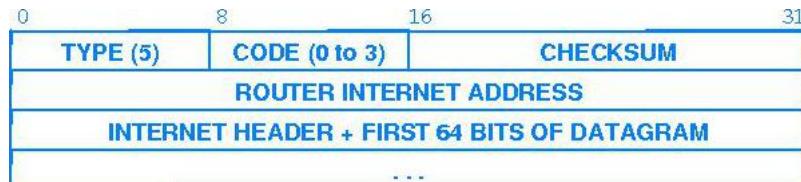


图 8.8 ICMP 重定向报文的格式

路由器互联网地址字段包含一个路由器的地址，主机将用这个地址使数据报抵达其首部中给出的目的站。互联网首部字段包含激发重定向报文的数据报的 IP 首部，再加上首部之后紧接着的 64 位。因此，收到 ICMP 重定向报文的主机，通过检查数据报的前缀来确定该数据报的目的站地址。ICMP 重定向报文的代码字段，进一步指出了如何根据分配的代码值来解释目的地址，代码值的含义如下：^①

作为一般的规则，路由器只向主机而不向其他路由器发送 ICMP 重定向请求。在以后的章节中，我们将看到路由器怎样使用其他协议来交换路由信息。

^① 回顾前面的内容，每个 IP 首部都指明了服务类型，服务类型可用于转发。

8.12 检测循环或过长的路由

因为互联网路由器使用本地的表来计算下一跳，所以对于某个目的站 D，路由表中的差错可能导致**路由选择循环** (routing cycle)。路由选择循环可能只包括两个路由器，它们都把发往目的站 D 的数据报转发给对方，路由选择循环也可能包括多个路由器。当几个路由器形成一个路由选择循环时，它们各自都把发往目的站 D 的数据报转发给循环中的下一个路由器。如果数据报进入一个路由选择循环，将会在循环中无休止地传递下去。正如前面提到的，为了避免数据报在 TCP/IP 互联网上无休止地循环，每个 IP 数据报都有一个生存时间计数器，有时称为**跳计数器** (hop count)。只要路由器处理了数据报，就会将数据报的生存时间计数器的值减 1，当计数器的值达到 0 时就丢弃数据报。

一旦路由器因为某个数据报的跳计数器递减到 0，或因为等待数据报分片时出现超时而丢弃它，路由器就向数据报的源站发回一个 ICMP 超时 (time exceeded) 报文，报文的格式如图 8.9 所示。

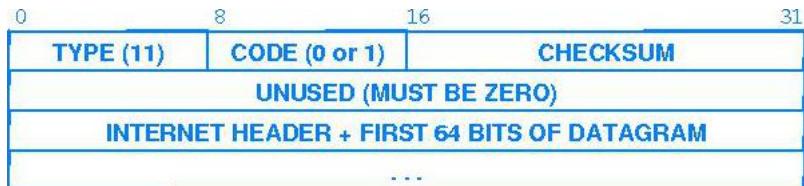


图 8.9 ICMP 超时报文的格式。一旦路由器因为数据报首部中的生存时间字段递减到 0，或因为等待分片时重装计时器超时而将数据报丢弃，路由器就会发送这个报文

ICMP 使用每个超时报文中的代码字段（值为 0 或 1），解释路由器所报告的超时的性质：

分片重装是指收集一个数据报的所有分片的任务。当数据报的第一个分片到达时，接收主机就启动一个计时器，如果在该数据报的所有分片都到达之前计时器超时，主机就会认为这是一个差错。代码值 1 用于向发送方报告这种差错；主机为每个差错发送一个报文。

8.13 报告其他问题

当路由器或主机发现一个数据报出现问题，而该问题又不属于前面提到的 ICMP 差错报文的范畴（例如，不正确的数据报首部），它就向最初源站发送一个**参数问题** (parameter problem) 报文。当某个选项的参数值不正确时，就可能导致发生这种问题。报文的格式如图 8.10 所示，只有当问题严重到必须把数据报丢弃时才发送这种报文。

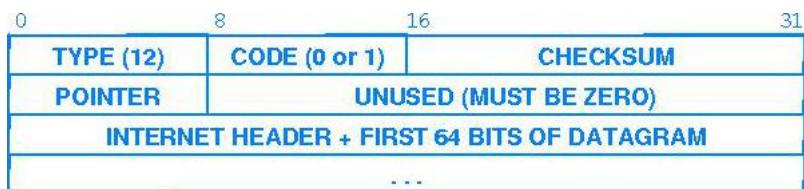


图 8.10 ICMP 参数问题报文的格式。只有当问题导致必须丢弃数据报时才发送这种报文

为了让这种报文没有二义性，发送方用报文首部的指针（POINTER）字段来标识数据报中产生问题的八位组。代码 1 用于报告缺少了一个需要的选项（例如，军事通信中的安全选项）；代码为 1 时，不使用指针字段。

8.14 时钟同步和传输时间估计

虽然互联网上的机器能够进行通信，但它们通常是独立运行的，每个机器都有自己的当前时间表示。如果时钟差异很大，就会让用户或分布式系统软件感到困惑。TCP/IP 协议族包含几个能够用来同步时钟的协议。一种最简单的技术是使用 ICMP 报文从其他机器获取时间。请求的机器向另一台机器发送一个 ICMP **时间戳请求**（timestamp request）报文，请求它返回当前日期的时间值。接收的机器给发出请求的机器返回一个**时间戳回答**（timestamp reply）报文。图 8.11 显示了时间戳请求及回答报文的格式。

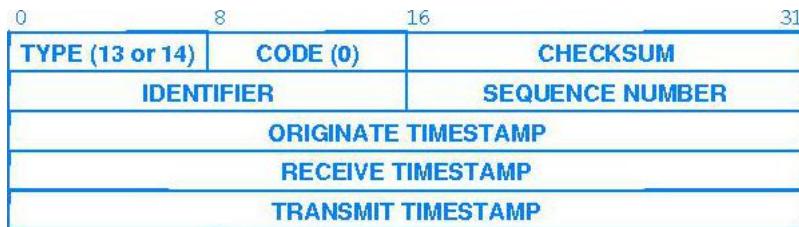


图 8.11 ICMP 时间戳请求或回答报文的格式

类型（TYPE）字段标识报文是一个请求（13）还是回答（14）；标识符（IDENTIFIER）和序号（SEQUENCE NUMBER）字段被源站用来在回答和请求之间建立关联。剩下的字段指明时间，给出的是从世界时（Universal Time）的午夜开始算起的毫秒数^①。起始时间戳（ORIGINATE TIMESTAMP）字段是由最初的发送方在传输分组前填写的，接收时间戳（RECEIVE TIMESTAMP）字段是收到请求后立即填写的，而传输时间戳（TRANSMIT TIMESTAMP）是在传输回答之前填写的。

主机使用这三个时间戳字段，计算三者之间时延的估计量，并同步它们的时钟。因为回答包含起始时间戳字段，所以主机可以计算出一个请求传到目的地、转换成一个回答并返回所需的总时间。由于回答携带了请求进入远端机器以及回答离开的时间，主机可以计算出网络传输时间，由此估计出远端时钟和本地时钟的差异。

在实践中，精确地估计往返时延可能很困难，而且它实质上限制了 ICMP 时间戳报文的使用。当然，为了得到往返时延的精确估计，必须采取多种测量，然后计算平均值。但是，在连接到一个大型互联网的一对机器之间，即使在一个短期时间内，往返时延也可能差别很大。另外，回顾前面的内容，由于 IP 是一种尽最大努力交付的技术，数据报有可能被丢弃、延迟或不按顺序交付。因此，仅仅采取多种测量也无法保证一致性；为了得到精确的估计，需要复杂的统计分析。

8.15 不再需要的较早 ICMP 报文

起初，ICMP 定义了一组报文，主机在启动时使用这些报文确定其 IP 地址、路由器的地址以及

^① 世界时以前称为**格林尼治标准时间**（Greenwich Mean Time），是在本初子午线上当前日期的时间。

在网络上使用的地址掩码。现在，称为 DHCP 的协议提供了一次交换中所需的全部信息^①，较早的 ICMP 报文现在已不再使用。

信息请求和回答报文 (Information Request And Reply Message)。ICMP 的信息请求和回答报文 (类型 15 和类型 16)，原本用来使主机在系统启动时发现自己的互联网地址。IETF 已声明它们作废，因此不应再使用这些较早的 ICMP 报文。

地址掩码请求和回答报文 (Address Mask Request And Reply Message)。ICMP 的地址掩码请求和回答报文 (类型 17 和类型 18) 原本用来使主机获得本地网络上使用的地址掩码。广播一个请求以后，网络上的路由器发送回答。

路由器恳求和通告报文 (Router Solicitation And Advertisement Message)。路由器恳求和通告报文原本用来使主机发现本地网络上的当前可用路由器。它和 DHCP 不同，DHCP 使用一个配置文件提供路由器的地址，而 ICMP 路由器发现机制提供直接的通信——主机直接从路由器接收通告。

尽管不再使用，但 ICMP 路由器发现给出了两点不同于 DHCP 的概念性差异。第一，由于信息是直接从路由器本身获得的，因此信息永远不会失效。第二，ICMP 路由器发现使用带有计时器的软状态 (soft state) 技术，防止主机在路由器崩溃后仍保留一个路由表项，路由器会定期通告自己的信息，如果一个路由的计时器超时，则主机会丢弃这个路由。

8.16 小结

网际控制报文协议 (Internet Control Message Protocol) 是 IP 的一个必要组成部分，用于正常情况以外的通信 (即用于报告异常情况或发送控制信息)。大多数情况下，ICMP 差错报文来源于因特网中的路由器；ICMP 报文总是被发回到引起差错的数据报的最初源站。

ICMP 包括降低传输速率的源站抑制报文，请求主机改变其路由表的重定向报文，以及主机可用来确定目的站是否可达的回送请求 / 回答报文。以前还有一组 ICMP 报文，它们原本用于给启动的主机提供信息，现在已不再使用。

ICMP 报文放在 IP 数据报的数据区中运送，在报文开头有三个固定长度的字段：ICMP 类型字段、代码字段及 ICMP 检验和字段。报文类型决定了报文其余部分的格式及其含义。

8.17 深入研究

关于时钟同步协议的讨论可参见 Mill [RFCs 956, 957, 1305]。

这里描述的网际控制报文协议是一个 TCP/IP 标准，由 Postel [RFC 792] 定义并由 Braden [RFC 1122] 改进。Nagle [RFC 896] 和 Prue and Postel [RFC 1016] 讨论了 ICMP 源站抑制报文，以及路由器应该如何使用它们来处理拥塞控制。Nagle [1987] 论证了在分组交换网络中拥塞总是令人关注的问题。Mogul and Postel [RFC 950] 讨论了子网掩码请求和回答报文，Deering [RFC 1256] 讨论了在路由器发现中使用的恳求和通告报文。

8.18 习题

1. 设计一个实验，记录一天内每种 ICMP 报文在你的本地网络上出现的次数。
2. 做个实验，看你是否能够以足够快的速率发送分组通过一个路由器，以激发一个 ICMP 源站抑制报文。
3. 设计一个使用 ICMP 时间戳报文同步时钟的算法。

^① 第 9 章将阐述地址掩码，第 22 章将阐述 DHCP 如何把掩码随同其他信息送回来。

-
4. 查看你的本地计算机系统是否包含 ping 命令。此程序与操作系统中的协议如何接口？尤其是，该机制是否允许任意用户运行 ping 程序的一个副本，或者说这样的程序是否需要特殊的权限？试解释原因。
 5. 假设所有的路由器都发送 ICMP 超时报文，而且你的本地 TCP/IP 软件将把这些报文返回给一个应用程序。使用这样的功能构造一个跟踪路由 (traceroute) 命令，让它报告源站和一个特定目的站之间的所有路由器的清单。
 6. 如果你连接到因特网上，尝试 ping 主机 128.10.2.1 (普度大学的一台机器)。
 7. 路由器是否应该给予 ICMP 报文比正常通信量更高的优先权？为什么？
 8. 假定一个以太网上有一个常规的主机 H，还有 12 个路由器和它相连。找到一个携带 IP 分组的单个（有点不合法）帧，当主机 H 发送它时，会使 H 刚好接收到 24 个分组。
 9. 把 ICMP 源站抑制分组与在 DECNET 中使用的 Jain 的 1 比特机制进行比较。对拥塞的处理来说，哪一个更有效？为什么？
 10. 没有一个 ICMP 报文可以允许机器通知源站：传输差错造成到达的数据报被破坏了。试解释原因。
 11. 在前一个问题中，在什么情况下那种报文可能是有用的？
 12. ICMP 差错报文是否应该包含一个指明它们何时发送的时间戳？为什么？
 13. 如果在你的网点上路由器参与了 ICMP 路由器发现，查找一下每个路由器在每个接口上通告了多少个地址。
 14. 尝试在你的本地网络上与一个根本不存在的主机上的服务器联系。也尝试与远程网络上一个不存在的主机通信。在哪种情况下你会收到一个差错报文？为什么？
 15. 尝试 ping 一个网络广播地址。有多少计算机做出回答了？阅读协议文档，判断对广播的请求做出回答是需要的、推荐的、不推荐的还是禁止的？

第9章 无分类和子网地址扩展（CIDR）

9.1 引言

第 4 章讨论了最初的因特网编址方案，并介绍了 A 类、B 类和 C 类单播地址。本章研究 IP 编址方案的四种扩展形式，它们是为了节省网络前缀而设计的，包括异步点对点链路、代理 ARP、子网编址和无分类编址。本章讨论了每种地址扩展的动机，并描述了地址扩展的基本机制。子网和无分类编址两种技术特别重要，因为它们目前广泛用于整个因特网中。

9.2 相关情况的回顾

第 4 章讨论了互联网中的编址，并给出了 IPv4 使用的最初地址方案。每个地址被分成两个部分；设计人员把前缀看成是互联网地址的网络部分，把其他部分看成是主机部分。对我们而言的重要结论是：

在最初的 IP 编址方案中，每个物理网络都被指派了一个唯一的网络地址；一个网络上每个主机的地址都把网络地址作为各个主机地址的前缀。

把 IP 地址分成两部分的主要优点在于路由器中所需路由表的大小。路由器不必为每个目的主机维护一个路由表项，而是为每个网络保留一个路由表项，而且路由器在做出转发决策时，只查看目的地址的网络部分。

前面讲过，最初的 IP 编址方案使用分类编址来确定前缀和后缀之间的边界。A 类把一个地址划分成 8 比特的网络部分和 24 比特的主机部分，B 类把地址划分成 16 比特的网络和 16 比特的主机部分，而 C 类把地址划分成 24 比特的网络部分和 8 比特的主机部分。

9.3 使网络数量最少

最初的 IP 分类编址方案似乎能够处理所有可能性，但是它有一个小缺陷。因为设计人员起初工作在昂贵的大型计算机时代，没有预见到今后互联网的增长。当时预见因特网上只有几百个网络和几千台主机；在 TCP/IP 被设计出来后的几年内，个人计算机还不曾出现。自从互连的因特网出现后，每隔 9~15 个月，它的大小就会翻一番。最终，IPv4 地址空间将会用完^①。在 20 世纪 80 年代初期，随着以太网的广泛流行，分类编址方案没有足够的网络地址变得显而易见，特别是 B 类地址前缀。由此产生的问题是：“在不摒弃原有编址方案的情况下，如何让技术能够适应网络增长的需要？”

设计人员提出用几种技术来解答这个问题，包括目前留存下来的三种技术：无编号的点对点链路、代理 ARP 和子网编址。在每种情况下，基本动机是相同的：减少使用的网络前缀数量。在因特网发展阶段的后期，一些在子网编址中使用的想法被延伸到网络前缀中，创造了无分类编址的概念。接下来将分别阐述这几种技术。

^① 尽管许多人曾预言 IPv4 地址空间在 2000 年就会用完，但现在看来，只要谨慎地分配地址并利用本章中描述的技术，到 2019 年左右地址空间还是够用的。

9.4 代理 ARP

术语**代理 ARP** (proxy ARP)、**混杂 ARP** (promiscuous ARP) 和 **ARP 窃用** (ARP hack)，都指的是把一个网络前缀用于两个物理网络时用到的技术。这一技术，只适用于那些使用 ARP 把互联网地址绑定到物理地址的网络，我们最好用一个例子加以解释，如图 9.1 所示。

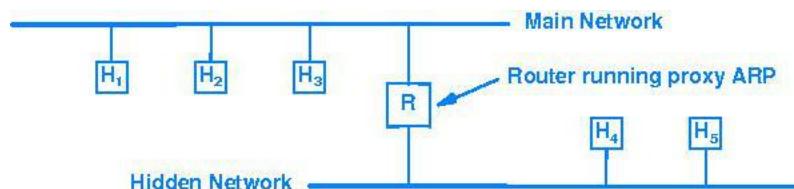


图 9.1 代理 ARP 技术（ARP 窃用）允许两个物理网络共享一个网络地址。路由器 R 响应每个网络上的 ARP 请求（为了获得另一个网络上主机的物理地址），给出它自己的硬件地址，然后在数据报到达时正确地转发它们。实质上，R 制造了关于 IP 地址到物理地址绑定的假象

在图 9.1 中，两个网络共享了一个 IP 网络地址。不妨设想一下，标记为“主网络”的网络是原来的网络，标记为“隐蔽网络”的网络是后来添加的。路由器 R 连接这两个网络，通过配置得知哪个主机在哪个物理网络上，并使用 ARP 来维持只存在一个网络的假象。为了做到这一点，R 把主机的位置完全隐藏起来，允许网络上的所有其他机器相互通信，就像它们直接连接在一个网络上。在我们的例子中，当主机 H₁ 需要与主机 H₄ 通信时，它首先使用 ARP 把 H₄ 的 IP 地址映射成物理地址。一旦有了物理地址，H₁ 就可以把数据报直接发送到那个物理地址。

由于 R 运行代理 ARP 软件，它捕获 H₁ 广播的 ARP 请求，判断 ARP 请求所询问的主机是否位于另一个物理网络上，并通过发送 R 自己的物理地址来响应这个 ARP 请求。H₁ 收到 ARP 响应后，把地址映射存放到自己的 ARP 表中，然后使用这个映射把目的站为 H₄ 的数据报发送给 R。当 R 接收到数据报时，它搜索一个特殊的路由表，以确定如何转发该数据报。R 必须把目的站为 H₄ 的数据报转发到隐蔽网络上。为了使隐蔽网络上的主机能到达主网络上的主机，R 同样也执行了主网络上的代理 ARP 服务。

使用代理 ARP 技术的路由器充分利用了 ARP 协议的一个重要特性，即**信任** (trust)。ARP 基于这样一种思想：所有的机器都相互协作，并且任何响应都是合法的。大多数主机直接保存通过 ARP 得到的映射，而不去检查它们的有效性，也没有保持一致性。因此，可能发生 ARP 表把几个 IP 地址映射成同一个物理地址的情况，但这并没有违反协议规范。

有些 ARP 实现并非如此宽松。特别是一些 ARP 实现设计成能够提醒管理员一些可能违反安全的行为，只要有两个不同的 IP 地址映射到同一个物理硬件地址，就会通知管理员。提醒管理员的目的是为了发出关于**欺骗** (spoofing) 的警告，即一台机器冒名顶替了另一台机器来截获分组。这种警告管理员可能存在欺骗的 ARP 主机实现，不能用在含有代理 ARP 路由器的网络上，因为代理 ARP 软件会频繁地产生欺骗报文。

代理 ARP 的主要优点是，可以把它添加到网络中的单个路由器上，但不会干扰网络上其他主机或路由器的路由表。因此，代理 ARP 完全隐藏了物理连接的细节。

代理 ARP 的主要缺点是，除非网络使用 ARP 进行地址解析，否则代理 ARP 就无法适用于该网络。此外，它不能推广到更复杂的网络拓扑（例如，多个路由器互连了两个物理网络），也不支持合理形式的转发。事实上，大多数代理 ARP 的实现依赖于管理员手工维护机器和地址的表格，既费时又容易出错。

9.5 子网编址

允许一个网络地址涵盖多个物理网络的第二种技术称为**子网编址** (subnet addressing)、**子网转发** (subnet forwarding) 或**划分子网** (subnetting)。由于划分子网是最通用的，而且已经被标准化了，所以它成为三种技术中应用最广泛的一种。事实上，划分子网是 IP 编址中不可缺少的一部分。

各个网点都有修改地址和路由的自由，只要这种修改保持对其他网点不可见。认识到这一点，对于我们理解划分子网的概念很重要。也就是说，一个网点可以选择与众不同的方式来分配和使用 IP 地址，只要：

- 网点上的所有主机和路由器一致认同该网点的编址方案
- 因特网上的其他网点可以把地址处理为一个网络前缀和一个主机后缀

最容易理解子网编址的办法是，设想一个网点分配得到一个 B 类的 IP 网络地址，但它有两个或更多的物理网络。只有本地的路由器知道有多个物理网络，而且知道如何在它们之间转发通信量；因特网中的所有其他路由器在转发通信量时，感觉这个网点似乎只有单独一个物理网络。图 9.2 显示了一个例子。

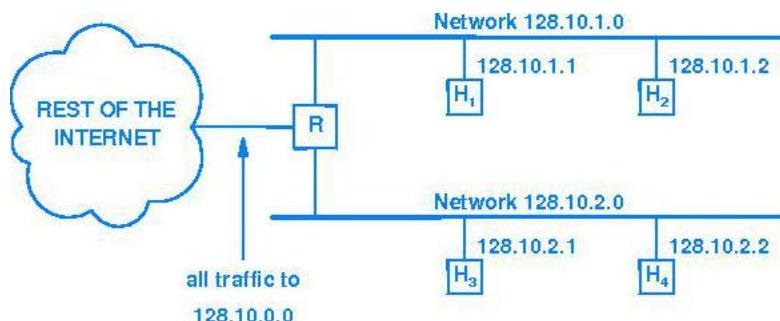


图 9.2 含有两个物理网络的一个网点使用子网编址，用一个 B 类网络地址来标志它们。路由器 R 接受所有到网络 128.10.0.0 的通信量，并根据地址的第三个八位组选择一个物理网络

在这个例子中，网点把一个 B 类网络地址 128.10.0.0 用于两个网络。除了路由器 R，因特网上的其余路由器在转发通信量时，感觉此网点仿佛只有一个物理网。一旦有分组到达 R，分组必须通过正确的物理网络送达它的目的站。为了高效地选择物理网络，本地的网点选择了使用地址的第三个八位组来区分这两个网络。管理员给一个物理网上的机器分配形式为 128.10.1.X 的地址，给另一个物理网上的机器分配形式为 128.10.2.X 的地址，其中 X 是地址的最后一个八位组，它包含一个标志特定主机的小整数。为了选择一个物理网络，R 检查目的地址的第三个八位组，把其值为“1”的数据报发送到标记为 128.10.1.0 的网络，把其值为“2”的数据报发送到标记为 128.10.2.0 的网络。

从概念上讲，增加子网只轻微地改变了对 IP 地址的解释。划分子网不是把 32 比特的 IP 地址划分成网络前缀和主机后缀，而是划分成**网络部分** (network portion) 和**本地部分** (local portion)。网络部分的解释与那些没有划分子网的网络一样。因特网中的路由器在做出转发决策时，照常只查看网络前缀。这样，地址的本地部分的解释留给了本地网点（受到子网编址的正式标准的约束）。总而言之：

在使用子网编址时，我们把一个 32 比特的 IP 地址看成有一个互联网部分和一个本地部分，其中互联网部分标志某个网点，该网点可能有多个物理网络，而本地部分标志了该网点中的一个物理网络和主机。

图 9.2 中例子所示的子网编址使用了一个 B 类地址，它有一个 2 八位组的互联网部分和一个 2 八位组的本地部分。为了使物理网络之间的转发高效，网点的管理员选择了使用本地部分的一个八位组来标志物理网络，另一个八位组标志物理网络上的一台主机，如图 9.3 所示。

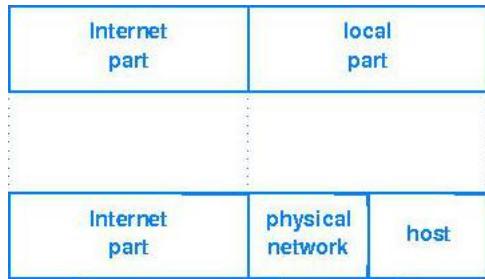


图 9.3 (a) 在最初的 IP 编址方案中的 32 比特 IP 地址的概念性解释; (b) 使用图 9.2 所示的子网方案时地址的概念性解释。本地部分被分成两个部分，分别标志一个物理网络和该网络上的一台主机

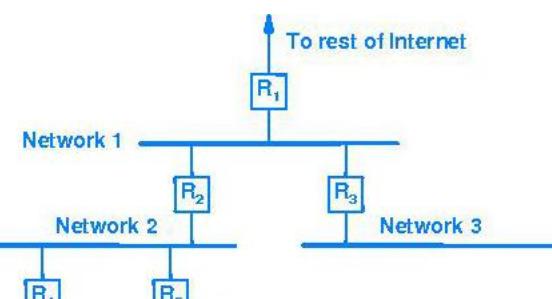
结果就是**分层编址**(hierarchical addressing)的形式，它导致了相应的**分层路由选择**(hierarchical routing)。层次结构的最顶层(即因特网中的其他自治系统)在转发时使用前两个八位组，下一层(即本地的网点)额外再使用一个八位组，而最底层(即通过一个物理网络交付)使用整个地址。

分层编址的思想并不是新的，许多系统过去就曾使用过它。最好的例子是美国的电话系统，其中，10 位数字的电话号码被分成 8 位数字的地区代码、3 位数字的交换机代码和 4 位数字的连接代码。使用分层编址的优点是，它非常适合于处理大量增长的对象，因为这意味着路由器不必像了解本地目的站的情况那样，了解远端目的站的太多细节。它的一个缺点在于很难选择合适的层次结构，而且一旦层次结构建立起来后就很难修改。

9.6 子网地址分配的灵活性

TCP/IP 的子网编址标准认识到并非每个网点都有同样的地址分层需求，它允许网点灵活地选择如何分配地址。为了理解这种灵活性的好处，想像一个由 5 个网络互连起来的网点，如图 9.4 所示。假定网点有一个 B 类网络地址，这个地址将用于它的所有物理网络。我们应该如何划分该地址的本地部分，才能使转发效率更高呢？

在我们的例子中，网点将根据预期的增长情况，选择一种划分 IP 地址本地部分的方法。如图 9.3 所示，它把 16 比特的本地部分划分成一个 8 比特的网络标识符和一个 8 比特的主机标识符，最多允许有 256 个网络，每个网络最多有 256 台主机^①。如果网点使用如上所述的**定长划分子网**(fix-length subnetting) 方案，并且避免使用全 0 和全 1 的子网和主机地址，那么一些可能的选择方案如图 9.5 所示。



^① 实际上，限制为 254 个子网，每个子网有 254 台主机。因为全 1 和全 0 主机地址被保留用于广播，并且不建议使用全 1 和全 0 子网。

图 9.4 有 5 个物理网络的网点被排列成三“层”。对于此类情况，把地址分成物理网络和主机两个部分的简单划分可能不是最优的

Subnet Bits	Number of Subnets	Hosts per Subnet
0	1	65534
2	2	16382
3	6	8190
4	14	4094
5	30	2046
6	62	1022
7	126	510
8	254	254
9	510	126
10	1022	62
11	2046	30
12	4094	14
13	8190	6
14	16382	2

图 9.5 一个 B 类号可能对应的定长子网容量，子网比特数最常见的选择是 8；一个单位必须选择表中的某一行

如图所示，采用定长划分子网的单位必须选择一种折中方案。如果该单位有大量的物理网络，那么这些网络各自不能包含太多的主机；如果一个网络上的主机数很多，则物理网络数必须较小。例如，分配 3 个比特来标志物理网络，最多可支持 6 个物理网络，每个物理网络最多可有 8190 台主机。分配 12 个比特来标志物理网络，最多可支持 4094 个网络，但每个网络最多只能有 14 台主机。

9.7 变长子网

前面讲过，选择子网编址方案，实际上是选择如何把 IP 地址的本地部分划分成物理网络和主机部分。实际上，大多数实现了子网划分的网点都采用了定长的分配方案。我们应该明白这一点：设计人员无法为子网的划分选择某种具体方法，因为没有哪一种划分地址本地部分的方案能够适用于所有单位，因为有些单位需要许多网络且每个网络上主机数很少，而另一些单位的网络数很少，但每个网络上连接了许多主机。设计人员还认识到，在一个单位内部也存在同样的问题。为了允许最大程度的自治，TCP/IP 子网标准提供了更大的灵活性。一个单位可以根据每个网络的情况来选择子网的划分方案。这种技术称为**变长划分子网**（variable-length subnetting），不过这个名字可能容易引起误解，因为长度值并不能随时间而变化，一旦为某个网络选择了一种划分方法，这种划分方法就不能再改变。连到该网络的所有主机和路由器都必须遵循这个抉择；如果不遵循，数据报就可能丢失或选择错误路由。总而言之：

为了在选择如何划分子网地址时获得最大限度的灵活性，TCP/IP 子网标准允许变长划分子网，可以独立地为每个物理网络选择一种划分方法。一旦选定了某种划分子网方法，该网上的所有机器都必须遵守它。

变长划分子网的主要优点是其灵活性：一个单位可以既有大型网络又有小型网络，并且能够更高效地利用地址空间。但是，变长划分子网有一些严重的缺点。最重要的是，必须十分小心地给子网分配值，以免出现**地址二义性**（address ambiguity），在这种情况下，根据不同的物理网络，同一个地址可能得到不同的解释。例如，一个地址可能与两个不同的子网匹配，其结果是这些无效的变长子网可能造成所有主机之间都无法通信。路由器不能解决这样的二义性问题，这意味着一个无效的分配只能通过重新编号来修复。因此，我们不鼓励网络管理员使用变长划分子网。

9.8 带掩码的子网的实现

子网技术使定长或变长的配置都很容易实现。标准要求用一个 32 比特的掩码来指示划分。因而，使用子网编址的网点必须为每个网络选择一个 32 比特的**子网掩码** (subnet mask)。如果网络上的机器把 IP 地址中的一些比特看成是网络前缀的部分，则子网掩码中那些对应的比特就被置为“1”；而如果把一些比特看成是主机标识符的部分，则把它们置为“0”。例如，一个 32 比特的子网掩码：

11111111 11111111 11111111 00000000

表示的是，前三个八位组标志网络，而第四个八位组标志该网络上的一个主机。一个子网掩码中对应地址网络部分的比特都应为 1（例如，一个 B 类网络的子网掩码，前两个八位组都为 1，后两个八位组中可能有一个或多个比特为 1）。

由于标准并没有要求子网掩码连续选择地址中的相邻比特，在子网编址中可能出现意想不到的有趣变化。例如，一个网络可能分配得到这样的掩码：

11111111 11111111 00011000 01000000

它选择了前两个八位组、第三个八位组中的两个比特和第四个八位组中的一个比特。尽管这种灵活性可以让这种有趣的地址分配运用在机器上，但这样做会使主机地址的分配及路由表的理解变得麻烦。因此，我们推荐网点使用连续相邻的子网掩码，并且在所有共享同一 IP 地址的物理网络集合中使用相同的掩码。

9.9 子网掩码的表示

用二进制表示子网掩码既笨拙又容易出错。因此，大多数软件允许采用其他表示方法。有时，子网掩码的表示遵循本地操作系统表示二进制量时所用的约定（例如，十六进制表示法）。

大多数 IP 软件将点分十进制记法用于子网掩码；当网点选择子网划分的边界与八位组边界对齐时，最适合使用这种方法。例如，许多网点选用了 B 类子网地址，用第三个八位组标志物理网络，用第四个八位组标志网络上的主机，如前一节所述。在这种情况下，子网掩码的点分十进制表示为 255.255.255.0，既容易书写又容易理解。

文献中也有用括号中的三元组来表示子网地址和子网掩码的例子：

$\{ <\text{网络号}>, <\text{子网号}>, <\text{主机号}> \}$

在这种表示法中，值为 -1 意味着全“1”。例如，如果一个 B 类网络的子网掩码是 255.255.255.0，则它可以写成 {-1, -1, 0}。

三元组表示法的主要缺点是没有精确指明地址的每个部分使用了多少比特；其优点是从比特字段的细节中抽象出来，而重点突出地址的三个部分的值。为了说明地址值有时比比特字段更重要的原因，考虑一个三元组：

$\{ 128.10, -1, 0 \}$

它表示一个地址的网络号为 128.10，子网字段为全“1”，主机字段为全“0”。如果用其他表示方法来表示同一地址值，则需要一个 32 比特的 IP 地址和一个 32 比特的子网掩码，并要求读者在其能够推算出每个字段的值之前先对比特字段进行解码。此外，三元组表示法与 IP 地址类型或子网字段的大小无关。因此，三元组可用来表示一些地址的集合或抽象概念。例如，三元组：

$\{ <\text{网络号}>, -1, -1 \}$

表示这样的地址：具有一个有效网络号并含有全“1”的子网字段和全“1”的主机字段。在本章后面将看到更多的例子。

9.10 子网存在时的转发

标准的 IP 转发算法必须进行修改，才能适用于子网地址。在一个使用子网编址的网络上，网络相连的所有主机和路由器都必须使用修改过的算法，称为**子网转发**（subnet forwarding）或**子网路由选择**（subnet routing）。除非给子网划分的使用增加一些限制，否则网点上的其他主机和路由器可能也需要使用子网转发，这可能不太明显。为了搞清楚在没有限制的情况下会出现怎样的问题，让我们考虑一下图 9.6 所示的一组网络。

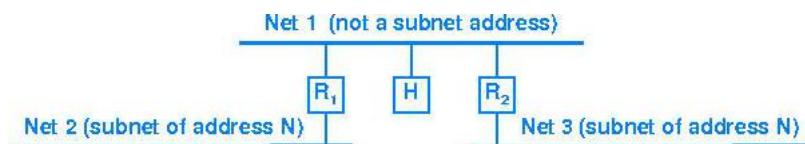


图 9.6 一个（不合法的）拓扑结构举例，它含有三个网络，其中网络 2 和 3 是同一个 IP 网络地址 N 的两个子网。

如果允许这种拓扑，即使网络 1 没有子网地址，主机 H 也需要使用子网转发

在图 9.6 中，物理网络 2 和 3 已经（不合法地）分配得到同一个 IP 网络地址 N 的不同子网地址。尽管主机 H 没有直接连到含有子网地址的网络上，但它必须使用子网转发来决定把发往网络 N 的数据报发送给路由器 R₁ 还是 R₂。或许有人会说 H 可以发给任何一个路由器，让路由器来处理这个问题，但这样做意味着有些通信量不能沿着一条最短路径行进。在更复杂些的例子中，优化与非优化路径之间的区别可能会非常明显。

从理论上讲，有个简单的规则决定了机器何时需要使用子网转发。这个子网规则是：

为了获得最优的转发，机器 M 必须为一个 IP 网络地址 N 而使用子网转发，除非已在机器 M 和 N 的每个子网所对应物理网络之间找到了一条最短路径 P。

遗憾的是，理解这个理论上的限制对于子网地址的分配并没有帮助。第一，如果硬件出现故障，或者如果路由选择协议重定向通信量来避开拥塞，则会改变最短路径。这种动态的改变使子网规则难以使用，除非是在很简单的情况下；第二，子网规则没有考虑在传播子网掩码信息时网点的边界或遇到的困难。子网路由的传播不可能超出某个给定单位的边界，因为稍后讨论的路由选择协议没有提供此类传播方式。实际上，把子网信息传播到一个给定的物理网络之外是极端困难的。因此，设计人员提出建议，如果一个网点使用子网编址，那么这个网点应该让子网的划分尽可能地简单。特别是，网络管理员应该坚持下面的原则：

一个给定网络 IP 地址的所有子网应该是相邻的^①，子网掩码在所有网络上应该一致，并且所有机器都应该参与子网转发。

一些大公司拥有多个网点，这些网点都需要连接到因特网上，而且这些网点彼此不直接相连，对于这样的大公司而言，这个原则会带来特别的困难。这个公司不能为其所有网点使用同一个地址的子网，因为这些网点的物理网络并没有相邻。

9.11 子网转发算法

与标准的 IP 转发算法类似，子网转发算法也要搜索路由表。前面讲过，在标准算法中，每个

^① 这里的“相邻”是指这些子网号都是一个紧接着一个的二进制数字，而不是指地理上的相邻——译者注。

主机的路由和默认路由属于特例，必须明确地进行检查；其他路由都使用表进行查询。传统的路由表包含如下形式的表项：

(网络地址, 下一跳地址)

其中**网络地址** (network address) 字段指明了目的网络的 IP 地址 N, **下一跳地址** (next hop address) 字段指明了一个路由器的地址，这个路由器把要传送到目的地址 N 的数据报转发出去。标准转发算法把目的地址的网络部分与路由表中每一表项的网络地址字段进行比较，直到找到一个匹配。因为**下一跳地址** (next hop address) 字段限制用于指明直接相连网络上的一个可达机器，所以在任何时候只需要查一次表。

标准算法知道地址是如何划分成网络部分和本地部分的，因为前三个比特对地址类型和格式进行了编码（即类型 A, B, C 和 D）。使用子网编址时，单从地址上不可能判断出哪些比特对应网络部分，哪些比特对应主机部分。因此，修改过的用于子网的算法要在路由表中维护一些附加信息。每个表项都包括一个附加的字段，指明该表项中的网络所使用的地址掩码：

(地址掩码, 网络地址, 下一跳地址)

在选择路由时，修改过的算法使用**地址掩码** (address mask) 提取目的地址的一些比特，与表项中的相应字段进行比较。也就是说，它把一个 32 比特目的 IP 地址与一个表项中的地址掩码字段按位进行布尔与运算，再把结果与该表项中的网络地址字段相比较，看是否相等。若相等，则把数据报转发到该表项中下一跳地址字段^①所指明的地址。

9.12 统一的转发算法

细心的读者们可能已经想到了，如果允许使用任意的掩码，那么子网转发算法就能涵盖标准算法的所有特例。只要使用与子网用法一样的掩码技术，就能处理到单个主机的路由、默认路由以及到直接相连网络的路由。另外，掩码还能处理传统的分类地址的路由。这种灵活性来自于子网掩码字段中任意 32 位值与网络地址字段中任意 32 位值组合的能力。例如，如果要装入单个主机的路由，可以使用全“1”的掩码和一个等同于主机 IP 地址的网络地址；如果要装入默认路由，可以使用全“0”的地址掩码和全“0”的网络地址（因为任意目的地址和“0”进行“与”运算还等于 0）。如果要装入一条到某个（非子网型）B 类网络的路由，可以指明掩码为全“1”的两个八位组和全“0”的两个八位组。如果给表中的表项排序，最长的掩码排在最前面，则转发算法包含的特例较少，如图 9.7 所示。

Algorithm:

Forward_IP_Datagram (datagram, routing_table)

```
Extract destination IP address,  $I_D$ , from datagram;
If prefix of  $I_D$  matches address of any directly connected
    network send datagram to destination over that network
    (This involves resolving  $I_D$  to a physical address,
     encapsulating the datagram, and sending the frame.)
else
    for each entry in routing table do
        Let  $N$  be the bitwise-and of  $I_D$  and the subnet mask
        If  $N$  equals the network address field of the entry then
            forward the datagram to the specified next hop address
        endforloop
    If no matches were found, declare a forwarding error;
```

^① 与标准转发算法中的情况类似，下一跳路由器必须能够通过一个直接相连的网络可达。

图 9.7 统一的 IP 转发算法。给出一个 IP 数据报的按掩码长度排序的路由表，本算法要选择出数据报将被发往的下一跳路由器。下一跳必须位于一个直接相连的网络上

事实上，大多数实现方法都去掉了对直接相连网络上目的站的显式测试。为了进行显式测试，必须在表中为每个直接相连的网络添加一个表项。与其他表项类似，每个直接相连网络对应的表项都包含一个掩码，掩码指明了前缀中的比特数。

9.13 子网掩码的维护

管理员如何分配子网掩码呢？子网掩码如何传播给主机和路由器呢？后面会给出第二个问题的答案。我们将会看到，主机在启动时获得子网掩码信息，路由器在使用路由转发协议交换路由信息时，把子网掩码信息传递给其他路由器。

第一个问题较难回答。每个网点可以为它自己的网络自由地选择子网掩码。在分配子网掩码时，管理员要在网络大小、物理网络数目、预期的增长和容易维护之间进行平衡。不统一的掩码可以提供最大的灵活性，但却可能导致二义性路由的产生，因而给子网掩码的分配带来了困难。或者，在更坏的情况下，当在网络中添加更多主机时，会使有效的分配变得无效。由于没有简单的规则可循，所以大多数网点都会做出保守的选择。通常，网点在地址的本地部分中选择连续相邻的比特来标志网络，并在该网点的所有本地物理网络上使用同一种划分（即同样的掩码）。例如，许多网点在对 B 类地址进行子网编址时只使用一个子网八位组。

9.14 广播到子网

在子网环境中进行广播比较困难。前面讲过，在最初的 IP 编址方案中，主机部分为全“1”的地址表示广播到指定网络上的所有主机。从子网型网点外部的一个观察者角度来看，广播到一个网络地址仍然是有意义的。也就是说，地址：

$$\{ \text{<网络>} , -1, -1 \}$$

意味着，把一个副本交付给以“<网络>”作为其网络地址的所有机器，即使这些机器位于不同的物理网络上。从操作上讲，只有把子网互连起来的路由器都同意向所有物理网络传播数据报，广播到这种地址才有意义^①。当然，必须谨慎处理，避免出现转发回路。需要特别指出的是，路由器不能仅仅把到达一个接口的广播分组传播给共享子网前缀的所有接口。为了避免这种回路的产生，路由器使用**反向路径转发**（Reverse Path Forwarding，简称 RPF）。路由器把广播数据报的源站提取出来，在它的路由表中查找该源站。除非数据报到达的接口是转发给源站时所用的接口（即数据报将从最短路径到达），否则路由器将丢弃这个数据报。

在一组划分子网的网络内部，向一个特定的子网进行广播成为可能（即，在分配得到子网地址之一的物理网络上，可以向该网络上的所有主机进行广播）。子网地址标准使用一个全“1”的主机字段表示子网广播。也就是说，子网广播地址变成：

$$\{ \text{<网络>} , \text{<子网>} , -1 \}$$

仔细研究子网广播地址和子网广播，就会明白为什么要在共享同一子网型 IP 地址的所有网络上，推荐使用一致的子网掩码。只要子网与主机字段相同，子网广播地址就不会有二义性。更复杂的子网地址分配可能允许，也可能不允许广播到一些选出来另构成一个子网的物理网络子网。

^① 出于安全原因，大多数网点禁止定向广播。

9.15 匿名的点对点网络

在最初的 IP 编址方案中，给每个网络分配一个唯一的前缀。特别地，因为 IP 把每对机器之间的点对点连接都看成是一个“网络”，所以给这个连接分配了一个网络前缀。给每台计算机分配了一个主机后缀。当地址变得不够用时，每个点对点连接还使用前缀似乎就不太合理了。对于那些有很多节点对点连接的单位。问题就会变得尤其严重。例如，一个单位有多个网点，它可能使用了租用数字线路（例如 OC3 线路）来组成一个主干网，主干网把每个网点的路由器和其他网点的路由器相互连接起来。

为了避免给每条点对点的连接分配一个前缀，人们发明了一种简单的技术，称为**匿名组网**（anonymous networking）。这种技术通常用在通过租用数字线路连接一对路由器时。这种技术简单地避免了给租用线路编号，并且不用给每一端的路由器分配主机地址。因为不需要硬件地址，所以在发送数据报时把接口软件配置成忽略下一跳地址。于是，在 IP 路由表中可以选择任意值作为下一跳地址。

当匿名组网技术应用于点对点连接时，把这种连接称为**无编号网络**（unnumbered network）或**匿名网络**（anonymous network）。图 9.8 中的例子有助于解释无编号网络中的转发。

为了搞清楚为什么可能出现无编号网络，必须要记住，用于点对点连接的硬件并不像共享媒体的硬件那样操作。因为只有一个可能的目的地，即位于线路另一端的计算机，所以底层硬件在传输帧时不使用物理地址。于是，当 IP 把一个数据报交给网络接口时，可以给下一跳指定任何值，因为硬件会忽略它。因此，IP 路由表的下一跳字段可以包含任意值（例如“0”）。

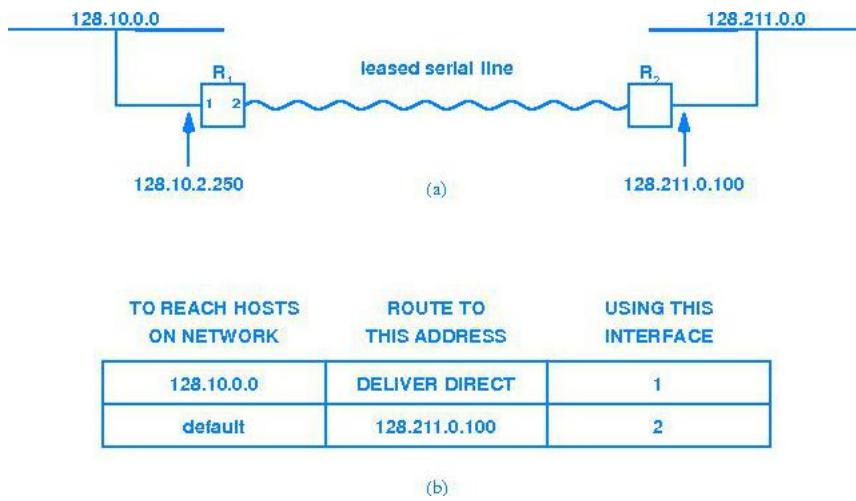


图 9.8 (a) 两个路由器之间的无编号点对点连接；(b) 路由器 R₁ 中的路由表

图 9.8(b)中的路由表的下一跳字段都不是“0”。然而，这个例子说明了通常与无编号网络一起使用的一种技术。下一跳字段不为空，而是填入了分配给下一跳路由器的 IP 地址之一（即，分配给路由器的另一个接口的地址）。在这个例子中，使用的是 R₂ 的以太网连接的地址。

我们讲过，硬件忽略了下一跳地址，因此分配这个值看上去有些奇怪。更奇怪的是，下一跳指向的是从 R₁ 无法直接到达的网络。实际上，IP 或网络接口代码都不会以任何方式使用这个值。指定非零内容的唯一原因是，使人们更容易理解和记住点对点连接另一端的路由器的地址。在这个例子中，我们选择了给 R₂ 的以太网接口分配的地址，因为 R₂ 的租用线路接口没有对应的地址。

9.16 无分类编址和构造超网

前面说过，子网编址和无编号点对点链路的出现源于节省 IP 地址空间的动机。到 1993 年，显然这些技术都无法阻止因特网的增长将导致的地址空间耗尽，人们开始定义一个含有更多地址的新版本 IP。但是，在新版本 IP 被标准化并得到采纳之前，为了适应网络增长，人们发明了一个临时的解决办法。

这种编址方法称为**无分类编址** (classless addressing)，它扩展了子网编址中所用的思路，允许网络前缀具有任意长度。除了一种新的编址模型以外，设计人员还发明了新的转发和路由传播技术。由此形成的技术整体称为**无分类域间路由选择** (Classless Inter-Domain Routing，简称 CIDR)：

要理解 CIDR 带来的影响，需要知道三个事实。第一，分类方案没有把网络地址分成相同大小的三类。虽然可以分配的 B 类网络号少于 1,700 个，但 C 类网络号超过 2,000,000 个。第二，因为 C 类前缀足够小型网络使用，并且不怎么受划分子网的影响，C 类前缀的需求比 B 类前缀的需求少得多。第三，研究表明，按照 B 类号分配的速率，所有的 B 类前缀很快就会用完。

一种无分类编址的最初用法称为**构造超网** (supernetting)。为了理解超网是如何构造的，考虑连接到因特网的一个中等规模的单位。采用分类方案时，这种单位可以请求一个 B 类前缀。构造超网的方案允许一个 ISP 给该单位分配一块 C 类地址，而不是分配一个 B 类网络号。这块地址必须多到足以给单位中的所有网络编号，而且（正如我们所见）地址块必须按 2 的幂来划分。例如，假设那个单位期望有 200 个网络。通过构造超网，可以给单位分配 256 个连续 C 类号构成的地址块。

9.17 CIDR 地址块和比特掩码

当把 CIDR 看成是用多个 C 类地址替代一个 B 类地址的方法时，比较容易理解提出 CIDR 的动机。尽管如此，CIDR 的提议者当初设计它的目的，是为了把它应用在更广泛的场合中。他们想象有这样一种分层编址模型，其中的每个商业**因特网服务提供者** (Internet Service Provider，简称 ISP) 可以得到一大块因特网地址，然后 ISP 再把这些地址分配给用户。由于允许网络前缀出现在任意比特的边界上，CIDR 允许某个 ISP 根据每个用户的需求给用户分配一小块地址。

与子网编址类似，CIDR 使用一个 32 比特**地址掩码** (address mask) 来指明前缀和后缀之间的边界。掩码中的连续相邻“1”比特决定了前缀的大小，掩码中的“0”比特与后缀相对应。例如，假定给一个单位分配了一块地址，包括起始于 128.211.168.0 地址的 2048 个连续地址。图 9.9 中的表给出了这个地址段内的地址二进制值。

	Dotted Decimal	32-bit Binary Equivalent
lowest	128.211.168.0	10000000 11010011 10101000 00000000
highest	128.211.175.255	10000000 11010011 10101111 11111111

图 9.9 包括 2048 个地址的 CIDR 块举例。表中给出了地址段的最低和最高地址，地址用点分十进制和二进制值表示

对于图中所示的地址段，CIDR 地址掩码有 21 个比特置为“1”，这意味着前缀和后缀的分界线出现在第 21 个比特之后：

11111111 11111111 11111000 00000000

9.18 地址块和 CIDR 记法

由于标志一个 CIDR 块既需要地址也需要掩码，因此人们发明了一种简写的记法来表示这两项。这种简写方法称为 CIDR 记法 (CIDR notation)，非正式场合下可称为斜线记法 (slash notation)，它用一个十进制数表示掩码的长度，并使用一个斜线把它和地址隔开。这样，采用 CIDR 记法，图 9.9 中的地址块可表示为：

128.211.168.0/21

其中，/21 表示地址掩码中有 21 个比特置“1”。图 9.10 中的表则列出了所有可能的 CIDR 掩码对应的点分十进制值。/8, /16 和/24 前缀对应于传统的 A 类，B 类和 C 类划分。

CIDR Notation	Dotted Decimal	CIDR Notation	Dotted Decimal
/1	128.0.0.0	/17	255.255.128.0
/2	192.0.0.0	/18	255.255.192.0
/3	224.0.0.0	/19	255.255.224.0
/4	240.0.0.0	/20	255.255.240.0
/5	248.0.0.0	/21	255.255.248.0
/6	252.0.0.0	/22	255.255.252.0
/7	254.0.0.0	/23	255.255.254.0
/8	255.0.0.0	/24	255.255.255.0
/9	255.128.0.0	/25	255.255.255.128
/10	255.192.0.0	/26	255.255.255.192
/11	255.224.0.0	/27	255.255.255.224
/12	255.240.0.0	/28	255.255.255.240
/13	255.248.0.0	/29	255.255.255.248
/14	255.252.0.0	/30	255.255.255.252
/15	255.254.0.0	/31	255.255.255.254
/16	255.255.0.0	/32	255.255.255.255

图 9.10 所有可能的 CIDR 前缀对应的点分十进制掩码值

9.19 无分类编址举例

图 9.10 中的表说明了无分类编址的一个主要优点：完全能够灵活分配各种大小的块。使用 CIDR。ISP 可以选择给每个客户分配一块适当大小的地址。如果 ISP 拥有一个 N 比特的 CIDR 块，它可以选择给客户分配大于 N 比特的任意地址块。例如，如果给 ISP 分配的是 128.211.0.0/16，则 ISP 可以选择给自己的一个客户分配如图 9.9 所示的地址块，即/21 地址段中的 2048 个地址。如果同一个 ISP 还有一个只有两台计算机的小客户，它可以选择分配另一个块 128.211.176.212/30，它包括的地址范围如图 9.11 所示。

	Dotted Decimal	32-bit Binary Equivalent
lowest	128.211.176.212	10000000 11010011 10110000 11010100
highest	128.211.176.215	10000000 11010011 10110000 11010111

图 9.11 CIDR 地址块 128.211.176.212/30 的示例。使用任意的比特掩码，可在分配地址块大小时获得比分类编址方案更大的灵活性

我们可以用下面这种方法来理解无分类地址：似乎一个 ISP 的每个客户都获得了该 ISP 的 CIDR 块的一个（变长）子网。这样，一个给定的地址块可按任意比特边界进一步细分，每种子划分可从一个单独的路由器进入。结果，尽管某个给定网络上的一组计算机将被赋予一段连续的地址，但这

个地址段并不需要对应某个预定义的地址类。由于允许指定前缀对应的准确比特数，这种方案使得子划分十分灵活。总而言之：

目前在因特网中使用的无分类编址方案，把 IP 地址看成是任意整数，并且允许一个网络管理员把地址划分成若干连续的块，一块中的地址数是 2 的幂。

9.20 无分类查找所用的数据结构和算法

评判路由表所用算法和数据结构的基本指标是速度。有关速度的考虑有两点：主要考虑的是为到达给定目的站而查找下一跳的速度，其次考虑的是改变表中数值的速度。

引入无分类编址对因特网中的转发有深远的影响，因为它从根本上改变了查找。在分类编址的情况下，每个地址是**自标志的**（self-identifying），即前缀的长度可以从地址本身计算出来，IP 查找使用**散列**（hashing）方法非常奏效。但是，在无分类编址的情况下，散列方法不能很好地发挥作用。因此，我们必须使用另一种方法。

9.20.1 按掩码长度进行搜索

理解 CIDR 的关键是，观察 ISP 给其用户分配的地址块，会发现用户地址块的地址掩码总是比 ISP 自己拥有的地址块长。于是，路由查找的目标是**最长前缀匹配**（longest prefix match，简称 LPM）。也就是说，给出了一个目的地址 D，在路由表中查找出某个表项，该表项必须含有与 D 的比特匹配的最长比特前缀。

最简单的 LPM 算法是遍历前缀和后缀之间的所有可能划分。也就是说，给出一个目的地址 D，算法首先尝试匹配 D 的 32 个比特，然后尝试 31 个比特，以此类推。对于每个可能的比特数 M，路由器从 D 中提取 M 个比特，设想提取出的比特组成了一个网络前缀，然后在表中查找这个前缀。一旦找到某个匹配的前缀，算法才会停止搜索。

尝试所有可能的长度，缺点是很显然的：这样做的速度相当慢，因为算法要为每个数据报进行多达 32 次的查找。当没有路由存在时会出现最坏的情况，但即使能够找到路由，遍历方法也要很多次搜索路由表。即使不是在路由表含有特定主机的路由的情况下，算法仍会浪费很多时间去查看主机比特。更为严重的是，这种算法在判定默认路由匹配前执行了 31 次不必要的搜索（许多路由表中大量使用了默认路由）。

9.20.2 二叉线索结构

为了避免效率低的搜索，无分类查找使用了一种分层的数据结构。使用最广泛的数据结构是一种**二叉线索**（binary trie）的变形。在这种数据结构中，地址中的连续比特值决定了从根向下的路径。

二叉线索是一棵树，树中包含的路径由存储数据决定。为了可视化地表示一个二叉线索，我们可以想象有一组 32 比特的地址，地址书写成二进制字符串，并去掉了冗余的后缀。此时剩下的一组唯一标志每个地址的前缀。图 9.12 显示了一组二进制形式的 7 个地址以及各个地址对应的唯一前缀。

32-Bit Address	Unique Prefix
00110101 00000000 00000000 00000000	00
01000110 00000000 00000000 00000000	0100
01010110 00000000 00000000 00000000	0101
01100001 00000000 00000000 00000000	011
10101010 11110000 00000000 00000000	1010
10110000 00000010 00000000 00000000	10110
10111011 00001010 00000000 00000000	10111

图 9.12 一组 32 比特的二进制地址，对应的是一组唯一标志各个地址项的前缀

如图 9.12 所示，识别地址所需的比特数取决于集合中的地址值。例如，图中的第一个地址可以用 2 比特唯一地标志，因为其他地址的前两个比特都不是“00”。但是，标志表中最后一个地址项需要用 5 比特，因为集合中具有前缀“1011”的地址项不止一个。

一旦计算出一组唯一的前缀，就可以使用这组前缀来定义二叉线索。图 9.13 显示了图 9.12 中的 7 个前缀所对应的树。

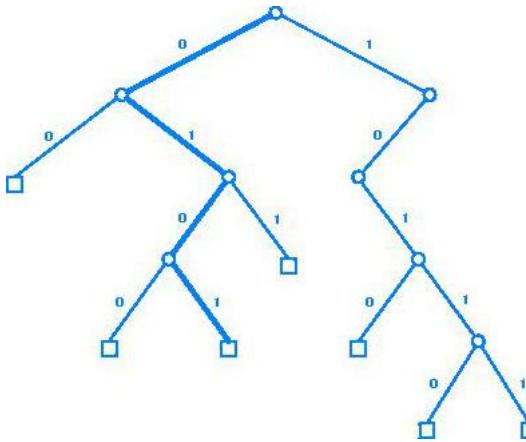


图 9.13 图 9.12 中所列的 7 个二进制前缀构成的二叉线索。加粗显示的是前缀“0101”对应线索的路径

线索中的每个内部节点（显示为圆圈）对应两个或更多的前缀，每个外部节点（显示为方块）对应一个唯一的前缀。当到达一个外部节点，或对某个指明的前缀不存在路径时，搜索算法就会停止搜索。例如，对于如下地址的搜索：

10010010 11110000 00000000 00000001

会失败，因为对应“10”的节点处没有标记为“0”的分支。

为了使查询更高效，处理无分类路由的转发软件必须使用不同于分类查找的数据结构和算法。许多系统使用了一种基于二叉线索的机制来适应无分类查找。

9.21 最长匹配转发和混合路由类型

我们对二叉线索的简要描述只给出了实际应用中所用数据结构的概貌。例如，我们说过二叉线索只需要为表中的每个路由存储一个唯一前缀，而没有说明这个前缀必须囊括路由的整个网络部分。为了保证路由器只在目的地址中的整个网络前缀与路由匹配时才转发数据报，二叉线索中的每个外部节点都必须包含一个 32 比特地址 A，以及一个 32 比特掩码 M，其中 M 囊括了 A 的整个网络部分。当搜索到一个外部节点时，该算法将 M 与目的地址进行逻辑“与”运算，然后像常规查找算法所做的那样，将结果与 A 进行比较。如果比较失败，则数据报被拒绝（也和常规的查找算法类似）。换句话说，我们可以把二叉线索看成是一种迅速标志路由表中潜在候选项的机制，而不是查找准确匹配的机制。

即使认为二叉线索是一种标志潜在匹配的机制，在我们的描述中还是遗漏了另一个重要细节。我们假设的是路由表中的每个表项都有唯一的二进制前缀，但实际应用中的大多数路由表中的表项没有唯一的二进制前缀，因为路由表混合含有同一目的站的一般路由和具体路由。例如，假设有这样一个路由表，表中包含一个特定网络的路由，对于同一网络某个特别的子网还包含一条不同的路由。或者假设有这样一个路由表，表中既包含一个特定网络的路由，还包含该网络上一个主机的特

殊路由。网络路由的二进制前缀也是子网的前缀或特定主机路由的前缀。图 9.14 给出了一个例子。

Prefix	Next Hop
128.10.0.0 / 16	10.0.0.2
128.10.2.0 / 24	10.0.0.4
128.10.3.0 / 24	10.1.0.5
128.10.4.0 / 24	10.0.0.6
128.10.4.3 / 32	10.0.0.3
128.10.5.0 / 24	10.0.0.6
128.10.5.1 / 32	10.0.0.3

图 9.14 没有唯一前缀的一组路由的例子。这种情况频繁发生，因为许多路由表混合含有同一网络的一般路由和具体路由

为了允许前缀相互重叠，必须对前面描述的二叉线索数据结构进行修改，以便在选择路由时遵循**最长匹配** (longest-match) 原则。一种可能的实现方法允许每个内部节点包含一个地址 / 掩码对，并修改搜索算法，在每个节点检查是否匹配。搜索中较晚发现的匹配（即对应一个更具体路由的匹配）必须覆盖任何较早发现的匹配，因为较晚的匹配对应一个更长的前缀。

9.21.1 PATRICIA 和层压缩线索

对二叉线索的描述还省略了与优化查找相关的细节。最重要的细节包括线索中可“**跳过**” (skipping) 的层，这些层中的不同路由之间没有区别。例如，考虑图 9.14 中的一组路由所对应的二叉线索。由于列表中每个路由的前 16 比特相同（即为值 10000000 00001010），这些路由的二叉线索在根以下的最初 16 层中，每层都只有一个节点。

在这种情况下，与一次只提取一个比特并在二叉线索中沿路使用它们相比，一次性地检查目的地址的前 16 比特会更快些。线索的两个修改版本都使用了优化性能的方法。第一种是 PATRICIA 树，允许每个节点指明一个要测试的值以及要跳过的比特数。第二种是**层压缩线索** (level compressed trie)，通过去掉线索中沿任意路径可跳过的一层或多层，提供了进一步的优化。

当然，数据结构的优化代表着一种折中。尽管优化提高了搜索速度，但在创建或修改路由表时需要进行更多的计算。但是，大多数情况下这样的优化是值得的，因为人们认为路由表的修改远不如路由表的搜索频繁。

9.22 为专用网络保留的 CIDR 块

第 4 章中已经讲过，IETF 已经指定把一组前缀保留用于专用网络。保留的前缀从不会分配给全局因特网中的网络。保留的前缀总称为**专用地址** (private address) 或**不可转发地址** (nonroutable address)。由于全局因特网中的路由器能够发现这些地址是保留的，于是有了后面这种叫法。如果发往一个专用地址的数据报意外地转发到了全局因特网上，因特网中的路由器将能检测到这个问题。

除了与分类地址相对应的块，那组保留的 IPv4 前缀包含跨越多个地址类的 CIDR 块。图 9.15 列出了 CIDR 表示的值，以及块中最高地址和最低地址的点分十进制值。列表中的最后一个地址块 169.254.0.0/16 不太常见，因为这是**自动配置** (autoconfigure) IP 地址的系统所使用的。

Prefix	Lowest Address	Highest Address
10.0.0.0 / 8	10.0.0.0	10.255.255.255
172.16.0.0 / 12	172.16.0.0	172.31.255.255
192.168.0.0 / 16	192.168.0.0	192.168.255.255
169.254.0.0 / 16	169.254.0.0	169.254.255.255

图 9.15 保留用于未连接到全局因特网的专用互联网前缀。如果发往这些地址之一的数据报意外地到达因特网上，则会产生错误

9.23 小结

本章研究了过去提出的四种用于保留 IP 地址的技术。第一种技术称为代理 ARP，即安排一个路由器顶替其他物理网络上计算机的身份，作为这些计算机的代理来回答 ARP 请求。代理 ARP 只有在使用 ARP 进行地址解析的网络上才有用，而且只对那些不会抱怨多个 IP 地址被映射到同一硬件地址的 ARP 实现才有用。第二种技术是一个 TCP/IP 标准，称为子网编址，允许一个网点的多个物理网络共享一个 IP 网络地址。在划分子网的网络上连接的所有主机和路由器必须使用修改过的转发算法，其中路由表的每个表项都包含一个子网掩码。修改后的算法可以看成是原来转发算法的一般化，因为它能处理类似默认路由或特定主机路由的特殊情况。第三种技术允许点对点连接不编号（即没有 IP 前缀）。

第四种技术称为无分类编址（CIDR），代表了 IP 技术的一次大转变。无分类编址没有坚持最初的编址方案，而是允许以任意比特为界划分前缀和后缀。CIDR 允许把地址空间划分成若干块，其中每个块的大小是 2 的幂。提出 CIDR 的一个动机源自人们希望把多个 C 类前缀组合成一个超网块。与最初的分类地址不同，无分类编址不是自标志的，因此 CIDR 需要对主机和路由器上的 IP 软件所用的算法和数据结构进行重大修改，以便存储和查找路由。CIDR 的许多实现方式都使用了基于二叉线索数据结构的方法。

9.24 深入研究

子网编址标准来自 Mogul [RFC 950]，并在 Braden [RFC 1122] 中进行了改进。Clark [RFC 932]，Karels [RFC 936]，Gads [RFC 940] 和 Mogul [RFC 917] 都含有子网编址方案的早期提案。Mogul [RFC 922] 讨论了子网存在时的广播问题。Postel [RFC 925] 思考了代理 ARP 对于子网的用法。Atallah and Comer [1998] 为变长子网分配提出了一种可证明的优化算法。Carl-Mitchell and Quarterman [RFC 1027] 讨论了用代理 ARP 实现透明子网路由器。Rekhter and Li [RFC 1518] 详细说明了无分类 IP 地址的分配。Fuller, Li, Yu, and Varadhan [RFC 1519] 详细说明了 CIDR 和超网构造。Rekhter et. al. [RFC 1918] 详细说明了保留用于专用网络的地址前缀。

9.25 习题

1. 如果使用代理 ARP 的路由器用一个主机地址表来判断是否回答 ARP 请求，只要给其中一个网络添加一台新主机，就必须修改路由表。试解释如何分配 IP 地址才能在添加主机后不必改变路由表。提示：考虑子网。
2. 尽管标准允许把全“0”地址分配成一个子网号，但是一些厂家的软件却不能正确地运行。尝试在你的网点上分配一个零子网，看看路由是否能够正确地传播。
3. 示范说明：代理 ARP 可用于通过两个路由器互连的三个物理网络。
4. 假设有一个 B 类网络号的定长子网划分，它能适应至少 76 个网络。每个网络上可以有多少台主机？
5. 对于一个 C 类网络地址，划分子网是否有意义？为什么？
6. 某个网点为其 B 类地址划分子网时，把第三个八位组用于物理网络，但是却发现不能适应 255 个或 256 个网络的情况。试解释原因。

-
7. 假设你有一个 B 类地址可用，为你的单位设计一个子网编址方案。
 8. 在一个路由器上同时使用代理 ARP 和子网编址是否合理？如果合理，说明如何做到这一点；如果不合理，解释为什么。
 9. 试论证：任何使用代理 ARP 的网络很容易受到“欺骗”（即任意一台机器都能顶替其他任何机器的身份）。
 10. 能否设计一种（非标准的）ARP 实现，使它支持正常的使用，但是却禁止代理 ARP？
 11. 一个使用网桥来连接两个以太网段的以太网，和一个使用代理 ARP 连接两个以太网段的系统，两者之间有什么区别，试进行分析。
 12. 一个厂家决定在其 IP 软件中加入子网编址技术，为所有的 IP 网络地址分配一个子网掩码。厂家修改了它的标准 IP 路由选择软件，将子网检查作为一个特例来处理。找一个简单的例子说明这个实现不能正确地工作。提示：考虑多归属主机。
 13. 描述在何种（受限的）情况下，前面习题中所讨论的子网实现能够正确地工作。
 14. 阅读标准，找到有关在子网存在时广播的更多内容描述。对于那些允许为所有可能的子网指明一个广播地址的子网地址分配，你能否描述其特点？
 15. 对于一个划分子网的 IP 地址包含的若干网络，标准允许为网络任意分配子网掩码。标准是否应该限制让子网掩码包括地址中的连续相邻比特？为什么？
 16. 给出一个变长子网分配并且主机地址产生二义性的例子。
 17. 仔细思考子网存在时的默认转发。如果到达的分组发往一个不存在的子网，将会发生什么情况？
 18. 使用子网编址和路由器互连多个以太网的结构，与第 2 章所描述的使用网桥的结构进行比较。在什么情况下，一种结构比另一种结构更可取？
 19. 假设有一个网点给 B 类网络地址划分子网，却让某些物理网络使用本地部分的 6 比特来标志物理网络，而其他物理网络将用 8 比特进行标志。找出一种主机地址的分配方案，使目的地址存在二义性。
 20. 图 9.7 所示的子网转发算法使用了顺序扫描路由表中表项的方法，允许管理员把特定主机的路由放在特定网络的路由或特定子网的路由之前。设计一种可获得同样灵活性的数据结构，但要使用散列算法来提高查找效率（这个习题是由 Dave Mills 建议的）。
 21. 尽管人们已经付出很多努力让路由器能够快速操作，但无分类路由查找所用的软件仍比分类查找所用的散列方法慢。试研究找出一些比二叉线索操作更快的数据结构和查找算法。
 22. 二叉线索使用 1 个比特从每个节点的两个子节点中选择其一。假设有这样一棵线索树，它使用 2 个比特从每个节点的 4 个子节点中选择其一。在什么条件下这样的线索会使查找更快？什么条件下会使查找更慢？
 23. 如果所有因特网服务提供者都使用无分类编址，并从其地址块中给用户分配地址号码，那么当一个用户从某个提供商转移到另一个时，会产生什么问题？

第10章 协议的分层

10.1 引言

前面几章回顾了网际互连的构造基础，描述了主机和路由器如何转发 Internet 数据报，并阐述了如何将 IP 地址映射成物理网络地址所用的机制。在主机和路由器中可以找到实现网络通信的软件，本章将讨论这些软件的结构。本章先给出一个通用的分层原理，说明分层技术如何使网际协议软件更容易理解和构造，并通过数据报在穿越 TCP/IP 互联网时遇到的协议软件，跟踪数据报的路径。

10.2 对多个协议的需求

前面已经讲过，通过协议就能详细说明和掌握通信的情况，而不必知道某个具体厂家的网络硬件细节。协议对于计算机通信，就像编程语言对于计算一样。至此，这两者之间有多么相似应该显而易见。与汇编语言类似，一些协议描述的是一个物理网络上的通信过程。例如，以太网帧格式的细节、网络访问策略和帧的差错处理，它们组成了描述以太网通信的协议。同样地，与高级语言类似，网际协议描述了更高层的抽象（例如，IP 编址、数据报格式以及不可靠的无连接交付的概念）。

复杂的数据通信系统不会使用单一的协议来处理所有传输任务。与此相反，它们需要一整套相互合作的协议，这些协议有时称为**协议族**（protocol family 或 protocol suite）。为什么需要这些协议呢？我们思考一下计算机在一个数据网络中通信时可能出现的问题：

- **硬件故障**（Hardware Failure）。主机或路由器可能因硬件故障或操作系统崩溃而不能工作。网络传输链路可能出故障或偶尔断开连接。协议软件需要检测出这种故障，如果有可能，还需要从故障中恢复过来。
- **网络拥塞**（Network Congestion）。即使所有硬件和软件都工作正常，但网络的容量也是有限的，这个限度有可能被超过。协议软件需要采取一定的策略，使已发生拥塞的机器能够抑制更多的通信量。
- **分组延迟或丢失**（Packet Delay Or Loss）。有时，分组会经历特别长的时延或者丢失。协议软件需要了解分组传输失败的情况，或适应长的时延。
- **数据损坏**（Data Corruption）。电磁干扰和硬件的故障可能导致传输的差错，从而破坏传输数据的内容。协议软件需要检测出这种差错并从差错中恢复过来。
- **数据重复或无序到达**（Data Duplication Or Inverted Arrivals）。提供多个路由的网络可能会无序地交付数据，也可能交付重复的分组。协议软件必须能将分组重新排序，并抛弃重复的分组。

上述这些问题组合在一起就显得太过复杂。怎样编写一个单独的协议来解决所有这些问题，似乎是难以想象的。从编程语言进行类推，我们就会明白如何克服这种复杂性。程序的**转换**（translation）问题从概念上可划分为四个子问题，每个子问题分别由编译器、汇编器、链接编辑器和加载器进行处理。这样的划分使设计者有可能每次只关注一个子问题，从而可以独立地构造和测试软件的各个部分。我们会看到协议软件也进行了类似的划分。

与协议相似的编程语言还有两个现象，有助于阐明协议的组织结构。首先，转换软件的各部分必须就它们之间传递数据的精确格式达成一致，这一点应该很明确。例如，从编译器传递到汇编器

的数据包括用汇编语言定义的程序。转换过程包含多种表示方式。在通信软件中有类似这样的情况，因为多个协议定义了在通信软件模块之间传递数据的表示方式。其次，解释器的四个部分构成一个线性序列，编译器的输出成为汇编器的输入，以此类推。协议软件也使用了一个线性序列。

10.3 协议软件的概念分层

可以把每台计算机上的协议软件模块想象成垂直堆叠的**多层**（layer）结构，如图 10.1 所示。每一层负责处理问题的一个部分。

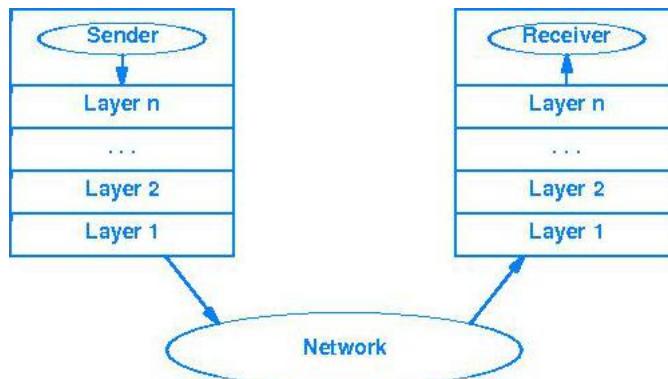


图 10.1 分层协议软件的概念构成

在概念上，从一台机器上的应用程序向另一台机器上的应用程序发送一个报文，意味着这样一个过程：报文通过发送方机器上的协议软件层逐层向下传送，再通过网络转发报文，然后通过接收方机器上的协议软件层逐层向上传送。

在实际应用中，协议软件比图 10.1 所示的简单模型复杂得多。每层都要对报文的正确性进行判断，并根据报文类型或目的地址选择采取合适的动作。例如，接收方机器上的一个协议层必须决定是否保留报文，或把它转发给另一台机器。另一层必须判断哪个应用程序应当接收这个报文。

为了理解协议软件的概念构成与实现细节的不同之处，图 10.2 给出了两者之间的对比。图 10.2(a)中的概念模型表明，在高层协议层和网络接口层之间有一个互联网层。图 10.2(b)中的真实模型表明，IP 软件能够与多个高层协议模块和多个网络接口进行通信。

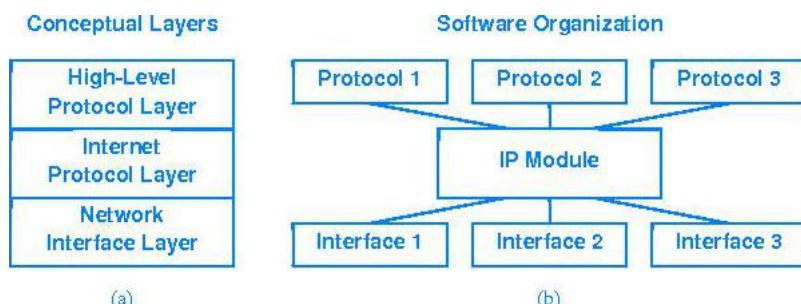


图 10.2 对比图(a) 概念上的协议分层和图(b) 软件组织结构的真实视图。真实视图表明在 IP 下面有多个网络接口，

虽然概念上的协议分层图并没有给出所有细节，但它确实有助于解释通用的分层概念。例如，图 10.3 显示了穿过三个网络的报文所用的协议软件层。图中只显示出路由器中的网络接口层和网际协议层，因为接收、查找下一跳和发送数据报只需要这两层。虽然概念分层图显示每个机器中只有一个网络接口层，但我们知道，任何与两个网络相连的机器必须有两个网络接口模块。

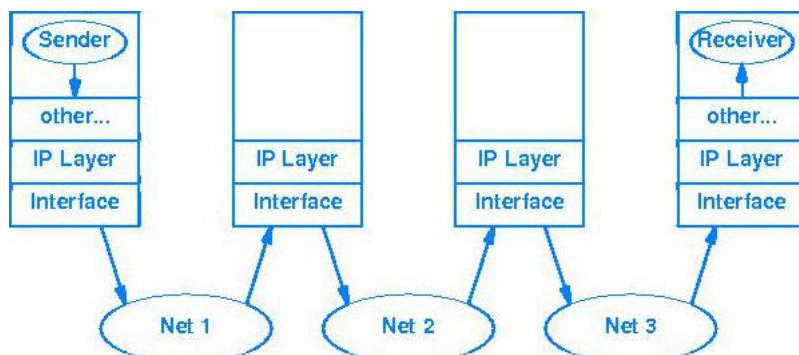


图 10.3 报文在因特网上从发送方穿过两个中间路由器到达接收方的路径。中间路由器只把数据报发送到 IP 软件层

如图 10.3 所示，源主机上的发送方传输一个报文，IP 层将报文放入一个数据报，并通过网络 1 发送。在中间路由器上，这个数据报上行到 IP 层，然后 IP 层再将它发送出去（到另一个网络上）。只有当它到达最后的目标机器时，IP 把报文抽取出来，然后将其上交给更高层的协议软件。

10.4 各层的功能

一旦已决定对通信问题进行划分，并把协议软件以模块的形式组织起来，每个模块负责处理其中一个子问题，就会出现这样的问题：“各个模块应具备什么功能？”这个问题不容易回答的原因有几点。第一，只有给出一个特殊通信问题的一组目标和约束，才可能为该问题的协议软件选择一种组织方式，这种组织方式将使协议软件最优化。第二，即使在考虑诸如可靠传输之类的通用网络级服务时，也有可能从一些根本不同的方法中选择出解决问题的方法。第三，网络（或互联网）体系结构的设计与协议软件的组织方式是相互关联的；任何一方都不能离开对方而单独进行设计。

10.4.1 ISO 七层参考模型

关于协议分层有两种思路占据该领域的主导地位。第一种思路基于**国际标准化组织**（International Organization for Standardization，简称 ISO）早期所做的工作，称为 ISO 的**开放系统互连参考模型**（Reference Model of Open System Interconnection），通常称为**ISO 模型**（ISO Model）。图 10.4 显示了 ISO 模型的七个概念层。

ISO 模型是构造用来描述单一网络的协议，它没有用一个特定的层来描述网际互连转发，这一点与 TCP/IP 协议不一样。

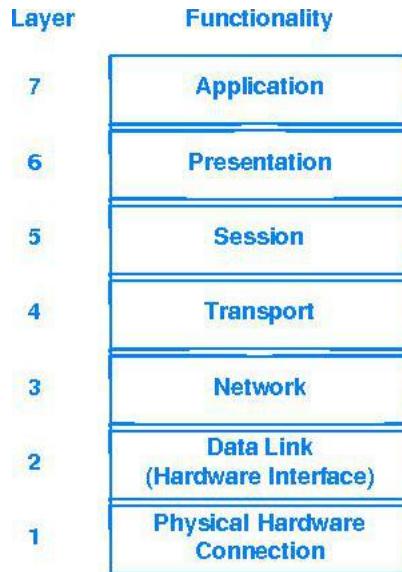


图 10.4 协议软件的 ISO 七层参考模型

10.5 X.25 及其与 ISO 模型的关系

虽然 ISO 分层方案当初的设计是为了提供一个概念上的模型而非实现的指南,但它也成为了某些协议实现的基础。在与 ISO 模型相关的协议中,最著名且应用最广泛的是称为 X.25 的协议族。X.25 是由国际电信联盟 (International Telecommunications Union, 简称 ITU) 推荐制定的,ITU 的前身是国际电报电话咨询委员会 (CCITT),CCITT 是推荐国际电话业务标准的国际组织。X.25 已被公用数据网采纳,尤其在欧洲十分流行。研究 X.25 有助于阐述 ISO 的分层思想。

从 X.25 的角度来看,一个网络的运行很像一个电话系统。我们可以设想 X.25 网络由复杂的、具有选择分组路由能力的分组交换机组成。每台主机并非直接连到网络的通信线路上,而是使用串行通信线路与其中一个分组交换机相连。从某种意义上讲,主机与 X.25 分组交换机之间的连接是一个微缩网络,网络中含有一条串行链路。主机必须遵循复杂的操作流程才能将分组传送到网络上。

- **物理层** (Physical Layer)。X.25 描述了主机与网络分组交换机之间的物理连接标准,也定义了一台机器向另一台机器传送分组的流程。在参考模型中,第一层详细说明了物理互连,包括电压和电流等电气特性。相应的协议 X.21 给出了公用数据网使用的规范细节。
- **数据链路层** (Data Link Layer)。X.25 协议的第二层规范了主机与其相连的分组交换机之间如何运送数据。X.25 使用了术语帧 (frame) 来表示主机和分组交换机之间传递的数据单元 (X.25 定义的帧与我们前面定义过的帧有细微的差别,明白这一点非常重要)。由于硬件本身只能交付比特流,第二层协议必须定义帧的格式,并给出机器识别帧边界的方法。由于传输差错可能会破坏数据,第二层协议还包括了差错检测机制 (如帧的检验和)。最后,由于传输是不可靠的,第二层协议定义了两台机器交换确认的机制,以便在帧传输成功时让两台机器知道。

一种常用的第二层协议是高级数据链路通信 (High-Level Data Link Communication),其缩写 HDLC 广为人知。HDLC 有几个不同的版本,最新出现的版本是 HDLC/LAPB。务必记住,在第二层中“传送成功”的含义是指帧已传递给网络分组交换机进行交付,但这并不保证分组交换机接受这个分组或者能够转发这个分组。

- **网络层** (Network Layer)。ISO 参考模型指出,第三层含有的功能需完成主机和网络之间交互的定义。第三层称为网络或通信子网层 (network or communication subnet layer),它定义了通过网络传送的基本数据单元,还包括目的站寻址和转发的概念。记住,在 X.25

网络环境中，主机和分组交换机之间的通信与它们传递的通信量在概念上是相互隔离的。因此，网络可以允许第三层协议定义的分组比第二层上能够传输的帧长一些。第三层软件按照网络所期望的形式组装分组，再由第二层把它（可能分成若干片）传送给分组交换机。第三层还要处理网络拥塞的问题。

- **运输层** (Transport Layer)。第四层让目的主机与源主机通信，从而提供**端到端** (end-to-end) 的可靠性。即使较下层的协议在每次传输时都提供可靠性检查，端到端层还是要再次进行检查，以确保在两端中间没有机器出故障。
- **会话层** (Session Layer)。ISO 模型中的更高层描述了应该如何组织协议软件，以提供应用程序所需的全部功能。ISO 委员会认为远程终端接入的问题是一个如此基本的问题，以至于指派了第五层来处理它。实际上，早期的公用数据网提供的核心业务就包括了终端到主机的连接。在公用数据网上，给专用的主计算机提供拨号访问服务的一个载体设备称为**分组组装拆器** (Packet Assembler And Disassembler，简称 PAD)。用户，通常是携带便携计算机的旅行者，他们使用调制解调器拨叫本地的 PAD，与主计算机建立网络连接，然后再以终端的方式登录到主计算机上。
- **表示层** (Presentation Layer)。ISO 第六层企图包括许多应用程序使用网络时所需的功能。典型的例子包括一些标准例程，如压缩文本，或将图像图片转换成在网络上传输的比特流。例如，一个称为**抽象语法表示法 1** (Abstract Syntax Notation 1，简称 ASN.1) 的 ISO 标准，提供了应用程序所用数据的表示方法。TCP/IP 协议之一的 SNMP 也使用了 ASN.1 来表示数据。
- **应用层** (Application Layer)。ISO 第七层包括了一些使用网络的应用程序，例如电子邮件或文件传输程序。

10.5.1 TCP/IP 的五层参考模型

第二个重要的分层模型并非出自哪个标准组织，而是出自关于 TCP/IP 协议族的研究。现在 TCP/IP 协议已广为流行，较早的 ISO 模型支持者曾进行过尝试，要把 ISO 模型扩展用于 TCP/IP，但原先的 ISO 模型没有提供网际互连的事实依然存在。

广义地讲，TCP/IP 软件组织成五个概念层——四个新的层建立在第五层（常规的硬件层）上。图 10.5 显示了这些概念层和数据在各层之间传递时的形式。

- **应用层** (Application Layer)。在最高层上，用户调用应用程序来访问通过 TCP/IP 互联网可用的服务。应用程序与一个运输层协议交互，以接收和发送数据。每个应用程序选择所需的传输样式，可能是一些独立的报文序列，也可能是连续的字节流。应用程序把要交付的数据按照要求的格式传递给运输层。
- **运输层** (Transport Layer)。运输层的基本任务是提供一个应用程序到另一个应用程序的通信。这种通信通常称为**端到端** (end-to-end) 通信。运输层可以控制信息的流动，也可以提供可靠的传输，以确保数据无差错地顺序到达。为了达到这个目的，运输协议软件设法让接收方发回确认信息，并让发送方重传丢失的分组。运输协议软件把要传输的数据流划分为小片（有时把这些小片称为分组），并把每个分组连同目的地址传递给下一层进行传输。

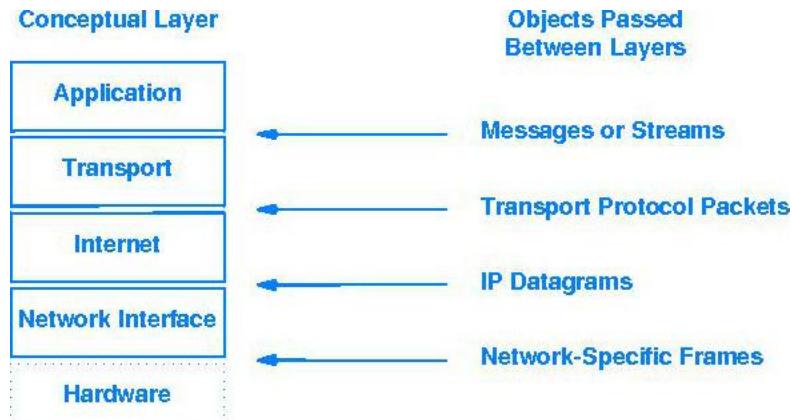


图 10.5 TCP/IP 软件的四个概念层在硬件层之上，以及各层之间传递的对象的形式。网络接口层有时也称为数据链路层

虽然图 10.5 只使用单个方框来表示应用层，但通用计算机可以有多个应用程序一起访问互联网。运输层必须从几个应用程序那里接收数据，并把数据发送到较低的下一层。为此，它给每个分组添加了附加信息，包括标识该分组由哪个应用程序发送的代码、标识应由哪个应用程序来接收分组的代码，还包括一个检验和。接收的机器使用检验和来验证数据是否完好无损地到达，并使用目的代码来标识分组应交付给哪个应用程序。

- **互联网层 (Internet Layer)**。正如前面已经讲过的，互联网层处理一台机器到另一台机器的通信。它从运输层接受发送分组的请求，以及分组要发送到的那个机器的身份标识。该层把分组封装到 IP 数据报中，填入数据报的首部，使用转发算法来决定是直接交付数据报，还是把它发送给路由器，然后把数据报传递给适当的网络接口进行传输。该层还要处理进入的数据报，检验其有效性，使用转发算法来决定是否应在本地处理数据报，还是应转发数据报。如果数据报的目的站是本地的机器，互联网层的软件就会删除数据报的首部，再从几个运输协议中选择其一来处理这个分组。最后，互联网层根据需要可发送和接收 ICMP 差错和控制报文。
- **网络接口层 (Network Interface Layer)**。TCP/IP 软件的最低层是网络接口层，它负责接受 IP 数据报并通过特定的网络传输数据报。网络接口层可能包括一个设备驱动程序（例如，当机器与一个局域网直接相连时），也可能包括一个复杂的子系统，该子系统使用它自己的数据链路协议。

10.6 智能的位置

TCP/IP 要求主机上的协议软件几乎参与所有网络协议的工作。前面已经提到，主机要主动进行端到端的差错检测及恢复工作。主机也参与了数据报的转发，因为它必须在发送数据报时选择路由器。另外，主机必须对 ICMP 控制报文做出响应，因而它也参与了网络的控制。由此可知，与电话网不同，TCP/IP 互联网可被看成一个相对简单的分组交付系统，该系统与一些智能主机相连。结论很重要：

TCP/IP 把大量网络智能放在主机中——因特网中的路由器转发数据报，但没有参与较高层的服务。

10.7 协议分层的原理

无论采用何种特殊的分层方案以及各层的功能是什么，分层协议软件的操作都基于一个基本的思想。这一思想就是所谓的**分层原理**（layering principle），可简要总结如下：

设计分层协议是为了让目的站上第 n 层收到的正是源站第 n 层所发送的对象。

分层原理可以解释分层思想如此有用的原因。它允许协议的设计者每次只关注一层，而不必考虑其他层如何工作。例如，在构造一个文件传送的应用程序时，设计者只需要考虑两台机器上所执行应用程序的两个副本，关注它们为了完成文件传送所需交换的报文。设计者假定一台主机上的应用程序收到的正是另一台主机上的应用程序所发送的数据。

图 10.6 描述了分层原理是如何工作的：

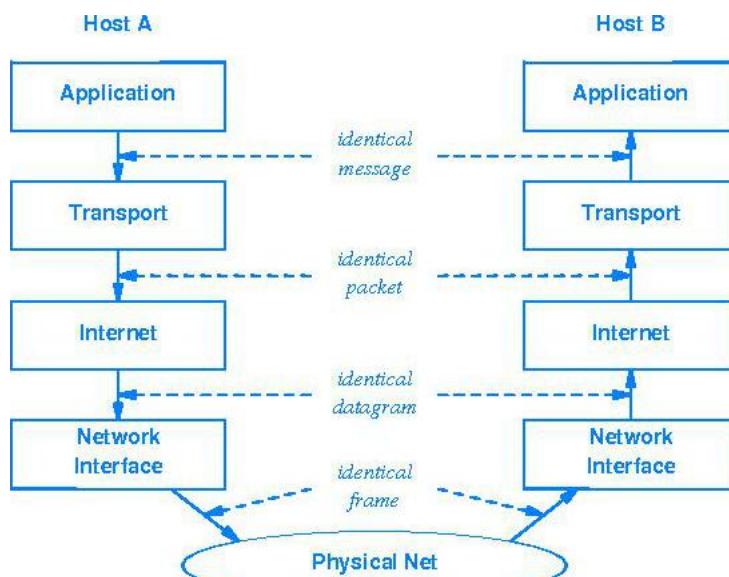


图 10.6 报文从一台主机上的应用程序传递到另一台主机上的应用程序时所经过的路径。主机 B 上第 n 层收到的正是主机 A 上第 n 层所发送的对象

10.7.1 TCP/IP 互联网环境中的分层

前面对分层原理的阐述有点模糊不清，图 10.6 忽略了一个重要的内容，它没有把从源站到最终目的站的传送和跨多个网络的传送区分开。图 10.7 说明了两者之间的区别，表明了报文从一台主机上的应用程序通过一个路由器发送到另一台主机上的应用程序的路径。

如图 10.6 所示，报文交付使用了两个独立的网络帧，一个帧用于从主机 A 传输到路由器 R，另一个用于从路由器 R 传输到主机 B。网络分层原理意味着交付给路由器 R 的帧等同于主机 A 发送的帧。相比而言，应用层和运输层处理端到端的事务，它们的设计是为了让源站软件与最终目的站的对等软件进行通信。因而，分层原理意味着最终目的站上运输层收到的分组与最初源站上运输层发送的分组一样。

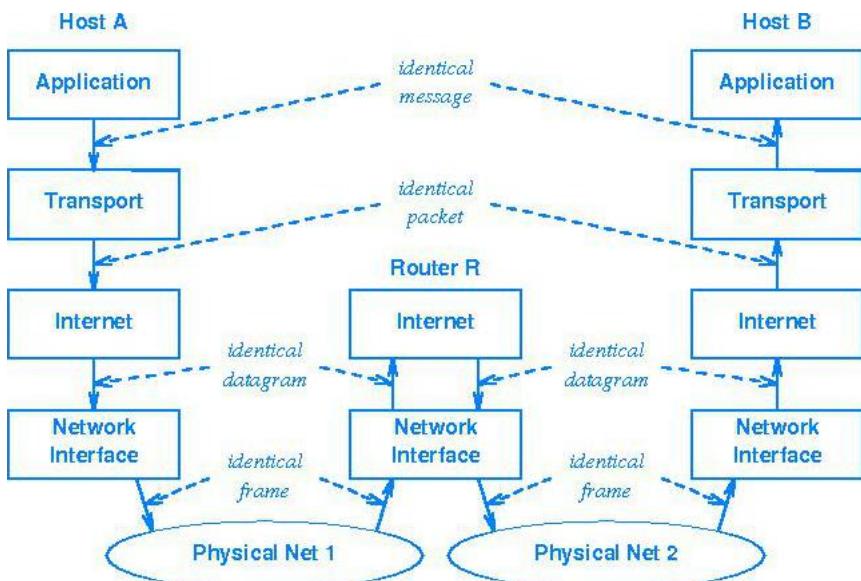


图 10.7 使用路由器时的分层原理。交付给路由器 R 的帧完全与主机 A 所发送的帧相同，但与路由器 R 发送给主机 B 的帧不同

在较高层，分层原理适用于端到端的传输；而在最低层，分层原理用于单个机器之间的传送，这一点很容易理解。但是，分层原理如何适用于互联网层就不太容易理解了。一方面，前面讲过，与因特网相连的主机应当把因特网当成一个巨大的虚拟网络，只不过用 IP 数据报来代替网络帧。从这个角度看，数据报从最初源站运送到最终目的站，且分层原理保证了最终目的站收到的正是最初源站发送的数据报。另一方面，数据报的首部包括了一些字段，如**生存时间**（time to live）计数器，数据报每次通过一个路由器，计数器的值就会变化一次。因此，最终目的站收到的数据报与源站发送的数据报并不相同。总的来说，尽管大部分数据报通过 TCP/IP 互联网传递时可保持完好无损，但分层原理只适用于数据报仅仅通过一台机器传送（到另一台机器，中间不经过路由器）的情况。准确地说，我们不应该认为互联网层提供了端到端的服务。

10.8 存在网络子结构时的分层

第 2 章曾提到，一些广域网含有多个分组交换机。例如，一个广域网既可能包含各个网上连接局域网的路由器，也可能包含使用租用串行线路与其他路由器连接的路由器。当一个路由器接收到数据报时，它通过局域网把数据报交付给目的站，或者通过一条租用串行线路把数据报传送给另一个路由器。由此产生的问题是：“在串行线路上使用的协议如何符合 TCP/IP 分层方案？”答案与设计者怎样看待串行线互连有关。

从 IP 的角度来看，一组位于路由器之间的点对点连接，可以各自像一组相互独立的物理网络一样工作，也可以集中起来像单个物理网络一样工作。在第一种情况下，每个物理链路完全像因特网中任何其他网络一样被对待处理。给链路分配一个唯一的网络号，而共享这条链路的两台主机各自为连接申请分配了一个唯一的 IP 地址^①。就像其他网络一样，链路的路由加入到 IP 路由表中。网络接口层添加了新的软件模块以控制新的链路硬件，但分层方案并没有做实质性的改动。这种独立网络方法的主要缺点是产生了大量的网络号（两台机器之间的每个连接都占用一个网络号），造成路由表的大小超过所需的大小。**串行线路 IP**（Serial Line IP，简称 SLIP）和**点对点协议**（Point to

^① 唯一的例外出现在使用第 9 章中介绍的匿名网络方案时；无编号链路不会让协议分层有什么变化。

Point Protocol, 简称 PPP) 把每条串行链路都当成一个独立的网络来处理。

用于点对点连接的第二种方法可避免给物理线路分配多个 IP 地址。它把所有连接当成单个独立的 IP 网络来集中处理，有自己的帧格式、硬件寻址方案和数据链路协议。使用这种方法的路由器只需要为所有的点对点连接申请分配一个 IP 网络号。

采用单一网络的方法，意味着要扩展协议分层方案，在网络接口层和硬件设备之间添加一个新的**内联网** (intranet) 转发层。对于只有一条点对点连接的机器，增加一个额外的层似乎没什么必要。为了搞清楚添加新层的必要性，可以考虑一个具有多条点对点连接的机器，再回想图 10.2(b)中网络接口层如何划分为多个软件模块，每个模块控制一个网络。我们需要添加新的网络接口，以处理新的点对点网络，但新接口必须能控制多个硬件设备。此外，当发送数据报时，这个新接口必须选择正确的链路进行发送。图 10.8 给出了这样的结构。

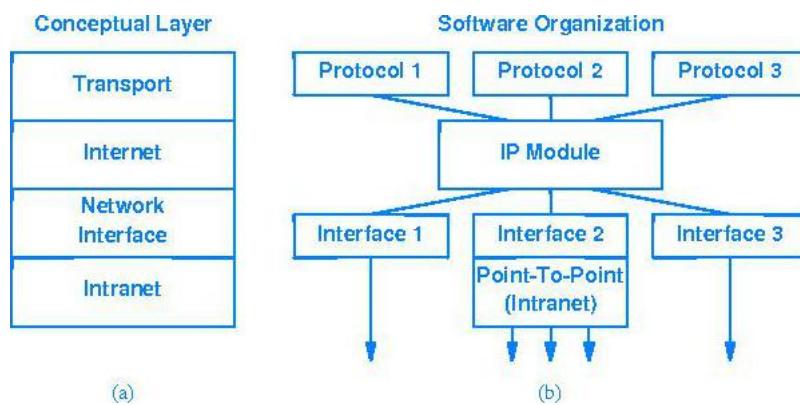


图 10.8 (a) IP 协议把多个点对点连接当成一个单独的 IP 网络处理时，点对点连接所用内联网协议的概念位置；(b) 相应软件模块的细节图。图中每个箭头对应于一个物理设备

只要数据报在其中一条点对点连接上发送，互联网层的软件就会将其全部传递给网络接口层，然后网络接口层再传递给内联网转发模块，转发模块必须进一步区分多条物理连接，把数据报转发到一条正确的物理连接上。

软件如何选择一条物理链路，这是由设计内联网转发软件的编程人员决定的。通常，算法离不开内联网路由表。内联网路由表与互联网路由表的相似之处是也指明了目的地址与下一跳的映射。路由表中包含多对表项 (D, L)，其中 D 是目的主机地址，L 指明了到达那个目的主机所用的物理线路。

互联网路由表与内联网路由表之间的区别在于：内联网路由表相当小，仅仅包含与点对点网络直接相连的主机的转发信息。原因很简单：互联网层把数据报传递给某个网络接口之前，要把任意一个目的地址映射成一个特定的路由器地址；内联网层只需要区分识别位于单个点对点网络上的那些机器。

10.9 TCP/IP 模型中的两条重要分界线

概念上的协议分层包括两条可能不太明显的分界线：一条是分隔高级和低级寻址的协议地址分界线，另一条是分隔系统和应用程序的操作系统分界线。

10.9.1 高级协议地址分界线

既然已经知道 TCP/IP 软件的分层，应该能更准确地把握第 7 章中介绍的一个观点：一个概念性的边界把使用低级（物理）地址的软件和使用高级（IP）地址的软件区分开。如图 10.9 所示，这个分界线出现在网络接口层和互联网层之间。也就是说：

应用程序和位于互联网层之上的所有协议软件只使用 IP 地址，而网络接口层处理的是物理地址。

因此，类似 ARP 那样的协议属于网络接口层的协议，它们不是 IP 的一部分。

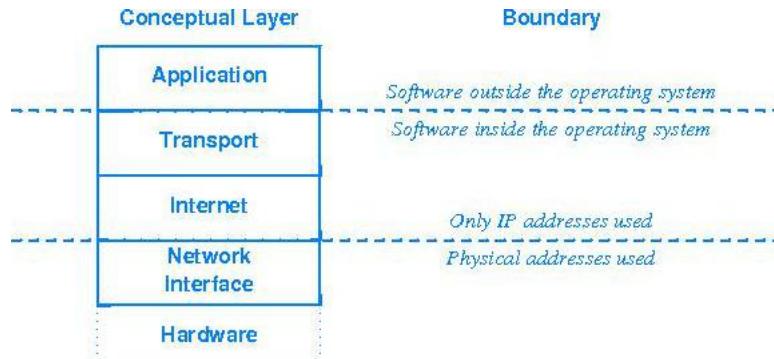


图 10.9 概念分层与操作系统分界线和高级协议地址分界线之间的关系

10.9.2 操作系统分界线

图 10.9 还显示了另一条重要的分界线，它把软件分成两类，一类为通常被认为是操作系统一部分的软件，另一类为非操作系统的软件。尽管 TCP/IP 的各种实现可自行选择如何进行划分，但大多数实现还是沿用了图示的方案。由于协议栈的较低层位于操作系统内部，在协议软件的较低层之间传递数据的代价比在应用程序和运输层之间传递要小得多。第 21 章更详细地讨论了这个问题，并且描述了一个操作系统可能提供的接口例子。

10.10 分层的缺点

我们已经说过，分层是为协议设计提供基础的基本思想。它允许设计者把一个复杂的问题分解成几个子问题，并独立地解决各个子问题。遗憾的是，严格按照分层思想设计的软件可能效率非常低。举例来说，我们思考一下运输层的工作。运输层必须从应用程序中接收一个字节流，把字节流划分为分组，再把每个分组发送到下层的互联网上。为了优化运送过程，运输层应当选择一个最大可能的分组大小，该长度允许把一个分组放入一个网络帧中运送。特别地，当目的机器和源站位于同一网络时，在传送过程中只涉及一个物理网络，因此发送方可以针对那个网络来优化分组的大小。然而，如果软件保持严格的分层，运输层就无法知道互联网模块如何转发通信量，也无法知道哪些网络直接相连。此外，运输层不知道数据报和帧的格式，也不能确定有多少八位组的首部将被添加到一个分组之前。因此，严格的分层将使运输层不能优化传送过程。

通常，协议软件的实现者在编写软件时会将严格的分层方案适当放宽，允许下层把诸如路由选择和网络 MTU 之类的信息向上层传播。高层协议软件模块在申请分配缓冲区时，为较低层的模块将要添加的首部预留了空间；而较低层模块把收到的帧上传给较高层协议时，保留了该帧的首部。这类优化措施使软件效率得到了很大提高，同时还保留了基本的分层结构。

10.11 复用和分用的基本思想

在整个层次结构中，通信协议使用了**复用**（multiplexing）和**分用**（demultiplexing）的技术。发送报文时，源计算机加入了一些额外的编码信息，如报文类型、发起传输的应用程序以及所用的

协议。最终，所有报文被放入网络帧中传送，并组合起来形成一个分组流。在接收端，目的机器使用额外的信息来引导分组的处理过程。

考虑图 10.10 所示的一个分用例子。

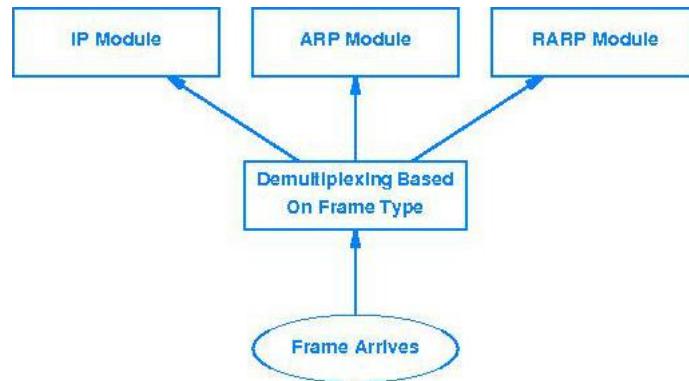


图 10.10 根据帧首部里的类型字段对收到的帧进行分用

图 10.10 举例说明了网络接口层的软件如何根据帧类型为收到的帧选择一个处理过程。我们可以这么说，网络接口根据帧的类型来**分用**（demultiplex）帧。要让这种选择成为可能，源站机器必须在传输之前设置帧类型字段的值，因此每个发送帧的软件模块使用类型字段来描述帧的内容。

几乎每个协议层中都有复用和分用出现。例如，在网络接口层对帧进行分用并把含有 IP 数据报的那些帧传递给 IP 模块后，IP 软件从帧中提取出数据报，并根据运输层协议对数据报进行进一步的分用。图 10.11 给出了互联网层分用的示意图。

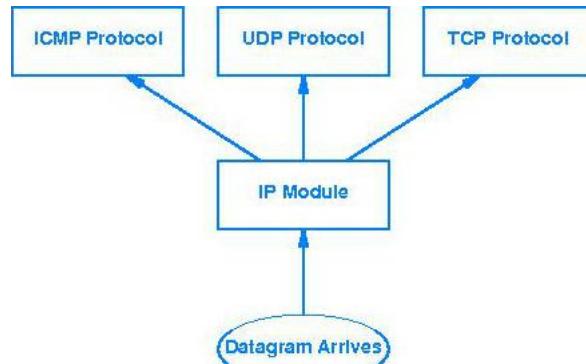


图 10.11 互联网层的分用。IP 软件根据数据报首部中的协议类型字段选择合适的处理过程

为了确定如何处理数据报，IP 软件检查数据报的首部，根据数据报类型选择一个协议处理程序。在这个例子中，数据报的类型既有前面已经讲解过的 ICMP，也有在后续章节中将会讲到的 UDP 和 TCP。

10.12 小结

协议是描述数据在机器之间传输时如何表示的标准。协议详细说明了传送如何发起、错误如何检测以及确认信息如何传递。为了简化协议的设计和实现，通信问题被划分为若干可以独立解决的

子问题。每个子问题对应着一个单独的协议。

分层的思想很重要，因为它为协议的设计提供了一个概念性的框架。在分层模型中，每一层分别处理通信问题的一个部分，并且通常对应着一个协议。协议遵循分层原理，这意味着在目的机器上第 n 层协议软件接收到的正是源站上第 n 层软件所发送的。

我们详细讨论了五层因特网参考模型和较早的 ISO 七层参考模型。在这两种情况下，分层模型都只提供了协议软件的概念框架。在实际应用中，协议软件使用复用和分用技术在指定的层次内区分多种协议，从而使协议软件比分层模型的提议更复杂。

10.13 深入研究

Postel [RFC 791]提供了网际协议分层方案的构架，Clark [RFC 817]讨论了分层实现的效果。Saltzer, Reed, and Clark [1984]论证了端到端验证的重要性。Chesson [1987]认为分层会造成极低的网络吞吐率，这一观点仍存在争议。本系列教材的第二卷深入研究了分层技术，给出了一个实现的例子，它通过严格分层与各层间传递指针的折中来获得高效率。

ISO 协议文档[2002a]和[2002b]详细描述了 ASN.1。Sun [RFC 1014]描述了 XDR，这个例子可称为 TCP/IP 表示协议。

10.14 习题

1. 更深入地研究 ISO 分层模型。这个模型如何描述类似以太网的局域网上的通信？
2. 构造一种情况，让 TCP/IP 演变成一个六层的协议体系结构，包括一个表示层。提示：许多程序使用 XDR 协议、XML 和 ASN.1。
3. 你认为最终会出现能够取代所有其他协议的一个表示协议吗？为什么？
4. 比较 ASN.1 表示方法使用的带标记数据格式和 XDR 协议中使用的无标记格式。在什么情况下，一种表示方法比另一种更好？
5. 指出 UNIX 系统如何使用 mbuf 结构来提高分层协议软件的效率。
6. 阅读了解 UNIX System V 的流机制。这种机制对于更容易地实现协议有什么帮助？它的主要缺点是什么？

第11章 用户数据报协议（UDP）

11.1 引言

前面几章描述了一个抽象的互联网，它能够提供在主机之间传送数据报的能力，每个数据报根据其目的站的 IP 地址通过互联网进行转发。在互联网协议层上，目的地址只是标志了一个主机的位置，没有对接收这个数据报的用户或应用程序更进一步加以标志。本章对 TCP/IP 协议族进行了扩展，添加了一个机制来区分某个指定主机内的不同目的地，从而允许某台指定计算机上运行的多个应用程序相互独立地发送和接收数据报。

11.2 识别最终目的地

大多数计算机中的操作系统支持**多道程序设计** (multiprogramming)，这意味着可允许多个应用程序同时执行。用操作系统的术语来讲，把每个正在运行的程序称为**进程** (process)、**任务** (task)、**应用程序** (application program) 或**用户级进程** (user level process)，把系统称为**多任务** (multitasking) 系统。一个进程是一个报文的最终目的地，这样说似乎比较自然。但是，把一台特定机器上运行的特定进程指明为某个数据报的最终目的地，有点容易让人误解。第一，由于进程的创建和撤销都是动态的，发送者几乎无法识别其他机器上的进程；第二，我们希望能够改换接收数据报的进程，同时不需要通知所有发送者（例如，重新启动机器后，所有进程都可能改变，但发送方应该不需要知道这些新进程）；第三，需要利用目的机器提供的功能来识别目的地，而不需要知道实现这个功能的进程（例如，允许发送者与一个邮件服务器联系，而不必知道服务器功能是由目的机器上的哪个进程实现的）。最为重要的是，在那些允许由一个进程处理两个或三个功能的系统中，有必要设法使进程确定发送方到底需要哪种功能。

我们并不是把进程当成报文的最终目的地，而是想象每台机器都包括一组称为**协议端口** (protocol port) 的抽象目的点。每个协议端口用一个正整数来标志。本地的操作系统提供了一个接口机制，进程使用它来指明或访问一个端口。

大多数操作系统提供了同步的端口访问。从一个特定进程的角度来看，同步访问意味着在端口访问的操作期间会停止计算。例如，如果一个进程试图在任何数据到达之前从端口提取数据，则操作系统会暂时停止（阻塞）进程，直到数据到达。一旦数据到达，操作系统把数据传递给这个进程并重新启动它。通常端口带**缓冲** (buffered)，在进程没有做好接受准备时，到达的数据就会进入缓冲区而不会丢失。为了实现缓冲，位于操作系统中的协议软件，将发往某个特定端口的分组放入一个（有限的）队列，直到某个进程把分组从队列中提取出来。

为了能够与外部端口通信，发送方不仅需要知道目的机器的 IP 地址，还需要知道该机器内目的地的协议端口号。每个报文必须携带报文送达机器的**目的端口** (destination port) 号，还必须携带回答报文应返送至源机器的**源端口** (source port) 号。这样，任何接收到报文的进程才可能回答发送方。

11.3 用户数据报协议

在 TCP/IP 协议族中，**用户数据报协议** (User Datagram Protocol，简称 UDP) 提供应用程序之

间发送数据报的基本机制。UDP 提供的协议端口可用来区分在一台机器上运行的多个程序。也就是说，每个 UDP 报文除了包含要发送的用户数据，还包含目的端口号和源端口号，这样就使目的机器上的 UDP 软件能够把报文交付给正确的接收者，而接收者也能发送回答报文。

UDP 使用下层的互联网层把报文从一台机器传输到另一台机器，与 IP 一样，提供的也是不可靠、无连接的数据报交付语义。它没有使用确认来确保报文到达，没有对收到的报文排序，也不提供反馈信息来控制机器之间信息流动的速度。因此，UDP 报文可能会出现丢失、重复或无序到达的现象。此外，分组到达的速率可能超过接收进程能够处理的速率。概括如下：

用户数据报协议（UDP）使用 IP 在机器之间传输报文，提供的是不可靠、无连接的交付服务。它使用 IP 来运载报文，但比 IP 增加了区分指定主机上的多个目的地的能力。

使用 UDP 的应用程序要全权负责处理可靠性问题，包括报文的丢失、重复、时延、无序交付以及**连通性的丢失**（loss of connectivity）。遗憾的是，应用程序开发人员在设计软件时常常会忽略这些问题。此外，由于程序员常常使用高可靠性、低时延的局域网来测试网络软件，一些潜在的故障问题难以暴露出来。于是，许多依赖 UDP 的应用程序在局域网上工作得很好，而在全局因特网上运行时却会出现各种故障问题。

11.4 UDP 报文的格式

每个 UDP 报文称为一个**用户数据报**（user datagram）。从概念上讲，用户数据报分为两个部分：UDP 首部和 UDP 数据区。如图 11.1 所示，首部可分为四个 16 位字段，分别指明报文从哪个端口来、到哪个端口去、报文的长度以及 UDP 检验和。

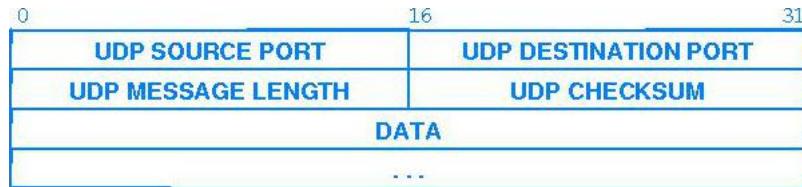


图 11.1 UDP 数据报的字段格式

源端口（SOURCE PORT）字段和目的端口（DESTINATION PORT）字段包含了一个 16 位的 UDP 协议端口号，用来在各个等待接收数据报的进程之间对数据报进行分用。源端口字段是可选的。若选用，则指明回答报文应该发往的端口；若不选用，其值应该为零。

长度（LENGTH）字段指明了该 UDP 数据报中八位组的个数，包括 UDP 首部和用户数据的长度。因此，长度字段的最小值是 8，即刚好是首部的长度。

UDP 的检验和（CHECKSUM）字段是可选的，不是所有数据报都需要使用它；如果该字段值为零，则意味着没有计算出检验和。设计者选择把这个字段作为可选的，是为了在高可靠性的局域网上使用 UDP 时，让那些实现能够以很少的计算开销运行。但是，前面讲过，IP 协议对 IP 数据报中的数据部分并不计算检验和。因此，UDP 检验和为保证数据原封不动地到达提供了唯一的验证途径，我们应当使用这个字段。

初学者往往想知道检验和计算得零时 UDP 报文会怎样。由于 UDP 使用的检验和计算方法和 IP 协议一样，即将数据划分为若干 16 个二进制位的单元，分别计算每个单元的二进制反码，再把这些值相加的结果求一次二进制反码，因此检验和有可能计算得零。因为零的二进制反码可以有两种表示方法：全“0”或全“1”，所以检验和为零并不会引起问题，这一点可能会让人意想不到。当计算得出检验和为零时，UDP 就使用全“1”来表示零。

11.5 UDP 伪首部

UDP 检验和计算涉及的范围超出了 UDP 数据报本身。为了计算检验和，UDP 把**伪首部**（pseudo-header）引入数据报中，并且为了保证数据报长度为 16 比特整数倍还添加了一个全零八位组来填充数据报，最后为整个对象计算检验和。用于填充数据报的八位组和伪首部并不随 UDP 数据报一起传输，也不计算在数据报长度之内。为了计算检验和，软件先在检验和（CHECKSUM）字段中存储零值，然后对整个对象，包括伪首部、UDP 的首部和用户数据，计算 16 位的二进制反码和。

使用伪首部的目的是验证 UDP 数据报是否已到达正确的终点。理解伪首部的关键在于认识到：一个正确的终点包括一台特定的主机和该机器上的一个特定协议端口。UDP 首部本身仅仅指明了协议端口号。因此，为了验证终点正确与否，发送端机器上的 UDP 在计算检验和时把目的机器的 IP 地址和 UDP 数据报都包括在内。在最终的目的机器上，UDP 协议软件对检验和进行验证时，用到了携带 UDP 报文的 IP 数据报首部中的 IP 地址。如果检验和一致，则说明 UDP 数据报到达了正确主机上的正确协议端口。

在 UDP 检验和的计算中用到的伪首部包括 12 八位组的数据，其排列如图 11.2 所示。标记为源 IP 地址（SOURCE IP ADDRESS）字段和目的 IP 地址（DESTINATION IP ADDRESS）的伪首部字段，分别含有发送 UDP 报文时使用的源 IP 地址和目的 IP 地址。协议（PROTO）字段含有 IP 协议类型代码（UDP 是 17），而 UDP 长度（UDP LENGTH）字段含有 UDP 数据报的长度（不包括伪首部）。接收方为了验证检验和，必须把这些字段从 IP 报文的首部中抽取出来，按伪首部的格式把它们组装起来，然后再重新计算检验和。

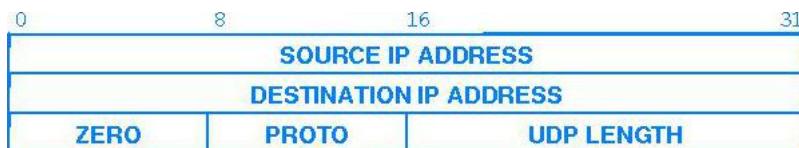


图 11.2 UDP 检验和计算时使用的 12 八位组的伪首部

11.6 UDP 的封装与协议的分层

UDP 给我们讲解运输协议提供了第一个例子。在第 10 章提到的分层模型中，UDP 位于互联网层上面一层。从概念上讲，应用程序访问 UDP 层，而 UDP 使用 IP 层来收发数据报，如图 11.3 所示。

UDP 层位于 IP 层之上，意味着一个包括 UDP 首部和数据的完整 UDP 报文，在通过互联网传输时要被封装到一个 IP 数据报中，如图 11.4 所示。

Conceptual Layering

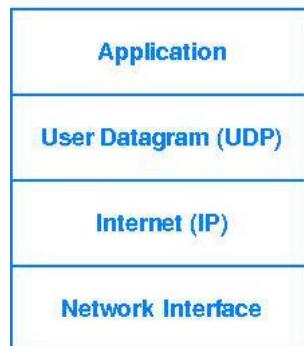


图 11.3 UDP 介于应用程序和 IP 之间的概念分层

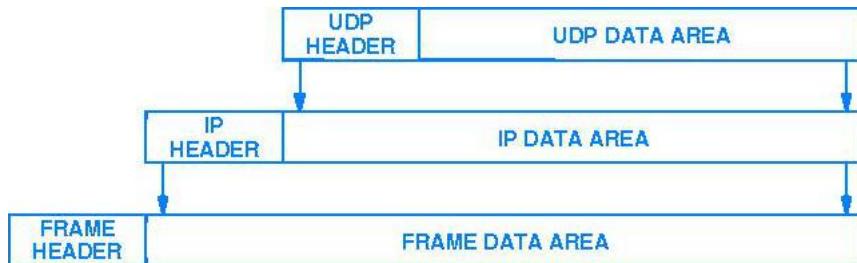


图 11.4 为了在互联网中传输，UDP 数据报被封装到 IP 数据报中。IP 数据报每次在单个网络上运送时被进一步封装成帧

对我们所讨论的协议而言，封装意味着 UDP 在用户要发送的数据前加上一个首部，然后再传递给 IP 层。IP 层又在从 UDP 层接收到的数据前加上一个首部。最后，网络接口层把数据报嵌入一个帧中，然后再把帧从一台机器发送到另一台机器。帧的格式取决于底层的网络技术。通常，网络帧还额外包括一个首部。

在接收端，分组先到达最底层的网络软件，然后开始逐层向上传递给较高层。每一层都在向上递交数据前去掉一个首部，因此，当最高层的协议软件把数据传递给相应的接收进程时，所有首部都已去掉了。这就是说，最靠外的首部对应的是最底层的协议，而最靠内的首部对应的是最高层的协议。在研究首部的插入与删除时，应该始终牢记分层原则。特别是，观察分层原则应用于 UDP 的情况可以看到，目的机器上 IP 层传给 UDP 层的数据报等同于发送端机器上 UDP 层传给 IP 层的数据报。同样，接收端机器上 UDP 层交付给用户进程的数据也正好是发送端机器的用户进程传到 UDP 层的数据。

在多层协议之间的职责划分是清楚而明确的：

IP 层只负责在互联网上的一对主机之间传送数据，而 UDP 层只负责区分一台主机内部的多个源端口或目的端口。

因此，只有 IP 层的首部标志出源主机和目的主机；只有 UDP 层标志出一台主机内部的源端口或目的端口。

11.7 分层及 UDP 检验和的计算

细心的读者可能已经注意到，分层原则与 UDP 检验和的计算看起来似乎存在着冲突。回顾前

面的内容，UDP 检验和的计算包括了一个伪首部，而这个伪首部中包括了源 IP 地址和目的 IP 地址。我们可以认为，用户在发送 UDP 数据报时必须知道目的 IP 地址，而且必须把目的 IP 地址传递给 UDP 层。这样，UDP 层就可以直接得到目的 IP 地址，而不必通过与 IP 层交互来获取。但是，源 IP 地址与 IP 为数据报选择的路由有关，因为源 IP 地址标志的是数据报传输所经过的网络接口。因此，UDP 层必须与 IP 层进行交互，才能获得源 IP 地址。

我们假定 UDP 软件要求 IP 层提供源 IP 地址，可能还有目的 IP 地址，再用这些信息构造伪首部，计算检验和，再丢弃伪首部，然后把 UDP 数据报传给 IP 层传输。另一种可替代方法会有较高的效率，让 UDP 层把 UDP 数据报封装到一个 IP 数据报中，从 IP 层获取源地址，把源地址和目的地址存入数据报首部的相应字段中，计算 UDP 检验和，然后把 IP 数据报传给 IP 层，而 IP 层只需要填充 IP 报文首部中的其余字段。

UDP 与 IP 的这种密切交互会不会破坏分层反映功能集分割的基本前提呢？回答是肯定的。UDP 与 IP 协议已紧密地结合起来了。这种做法显然是纯分割方案的一种折中，这样做完全出于实际应用的需要。之所以忽略这种做法对分层的违反，是因为在不指明目的机器的前提下不可能完全标志目的应用程序，而且我们希望对 UDP 所用的地址和 IP 所用的地址进行高效映射。本章有一道习题从另一个角度研讨了这个问题，让读者自己考虑是否应该把 UDP 与 IP 分开。

11.8 UDP 的复用、分用和端口

在第 10 章中可以看到，协议层次结构中的各层软件必须对相邻层的多个对象进行复用和分用。UDP 软件提供了复用和分用的又一个例子。它从多个应用程序接受数据报，把它们传给 IP 层进行传输，同时它从 IP 层接受传入的 UDP 数据报，并分别把它们传递给相应的应用程序。

从概念上讲，UDP 软件与应用程序之间的所有复用和分用都要通过端口机制来进行。实际上，每个应用程序在发送数据报之前必须与操作系统进行协商，以获得一个协议端口和关联的端口号^①。一旦分配好了端口，应用程序通过这个端口发送的任何数据报都要把该端口号放入源端口 SOURCE PORT 字段。

在处理输入时，UDP 接受从 IP 层软件传入的数据报，并根据 UDP 目的端口进行分用，如图 11.5 所示。

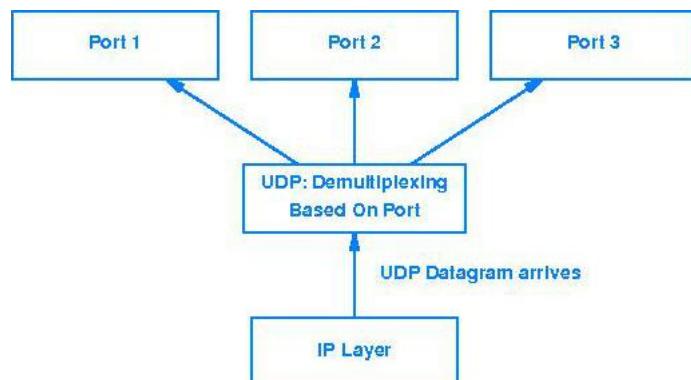


图 11.5 在 IP 的上一层分用的例子。UDP 使用 UDP 目的端口号为传入的数据报选择相应的目的端口

最容易理解 UDP 端口的方法是把它想象成一个队列。在大多数实现中，当应用程序与操作系统协商，以使用某个指定端口时，操作系统就创建一个内部队列来容纳收到的报文。通常应用程序

^① 现在只是抽象地描述端口，第 21 章将讨论用于创建和使用端口的操作系统原语的例子。

可以指定和修改这个队列的长度。当 UDP 收到数据报时，先检查该数据报的目的端口是否与当前使用的端口之一匹配。如果不匹配，则发送一个 ICMP 端口不可达 (port unreachable) 报文并丢弃这个数据报。如果有一个端口匹配，则把这个新的数据报送到该端口的队列中，等待应用程序的访问。当然，如果端口队列已满也会出错，UDP 也要丢弃这个传入的数据报。

11.9 保留和可用的 UDP 端口号

应该如何分配协议端口号呢？这个问题很重要，因为两台计算机在能够互操作之前必须协商确认一个端口号。例如，当 A 机需要从 B 机那里获得一个文件时，它就需要知道 B 机的文件传送程序使用哪个端口号。端口分配有两种基本方法。一种方法是使用中央授权机构。大家都同意让一个中央授权机构按需分配端口号，并由它来公布分配的所有端口号的列表。然后，所有软件都必须按照这个列表来构建。这种方式又称为统一分配 (universal assignment)，那些由授权机构指定的端口分配又称**熟知端口分配** (well-known port assignment)。

第二种端口分配方法使用了**动态绑定** (dynamic binding)。在使用动态绑定方法时，并非所有的机器都知道端口。而是当一个应用程序需要端口时，由网络软件给它分配一个端口。为了知道另一台机器当前的端口分配情况，就必须发送一个请求报文，询问当前的端口分配情况（如，“文件传送服务使用的是什么端口号？”）。目的主机给出所用的正确端口号进行回答。

TCP/IP 的设计者们采纳了一种混合端口分配方法，只预先分配一些端口号，同时把大量可用端口留给本地网点或应用程序，进行动态分配。预先分配的端口号从较低的值开始，向上扩展，可用的大整数值留给动态分配。图 11.6 中的表格列出了一些当前分配的 UDP 端口号，其中第二栏包含了因特网标准分配的关键字，而第三栏包含了大多数 UNIX 系统中使用的关键字。

Decimal	Keyword	UNIX Keyword	Description
0	-	-	Reserved
7	ECHO	echo	Echo
9	DISCARD	discard	Discard
11	USERS	systat	Active Users
13	DAYTIME	daytime	Daytime
15	-	netstat	Network Status Program
17	QUOTE	qotd	Quote of the Day
19	CHARGEN	chargen	Character Generator
37	TIME	time	Time
42	NAMESERVER	name	Host Name Server
43	NICNAME	whois	Who Is
53	DOMAIN	nameserver	Domain Name Server
67	BOOTPS	bootps	BOOTP or DHCP Server
68	BOOTPC	bootpc	BOOTP or DHCP Client
69	TFTP	tftp	Trivial File Transfer
88	KERBEROS	kerberos	Kerberos Security Service
111	SUNRPC	sunrpc	Sun Remote Procedure Call
123	NTP	ntp	Network Time Protocol
161	-	snmp	Simple Network Management Protocol
162	-	snmp-trap	SNMP traps
512	-	biff	UNIX comsat
513	-	who	UNIX rwho Daemon
514	-	syslog	System Log
525	-	timed	Time Daemon

图 11.6 当前已分配的部分 UDP 端口和相应的标准因特网关键字和 UNIX 关键字，本表并没有列出所有已分配的端口。在可能的范围内，提供同样服务的其他运输协议也和 UDP 使用相同的端口号

11.10 小结

大多数计算机系统允许多个应用程序同时运行。用操作系统的术语来讲，我们把每个正在执行的程序称为进程。用户数据报协议 UDP，通过让发送方和接收方在每个 UDP 数据报中添加两个名为协议端口号的 16 位整数，将同一台机器上运行的多个进程区分开。端口号标志了数据报的来源和目的地。一些 UDP 协议端口（称为熟知端口）是永久性分配的，并且在整个因特网内被大家认同，例如端口 69 保留用于第 25 章描述的简单文件传送协议（TFTP）。其他端口号可供任意应用程序使用。

UDP 并没有给 IP 层的语义增加多少内容，在这个意义上它是一个“瘦”协议。它只给应用程序提供了通信能力，使用的是 IP 的不可靠、无连接分组交付服务。因此，UDP 的报文可能会出现丢失、重复、延迟或无序交付；使用 UDP 的应用程序必须负责处理这些问题。许多使用 UDP 的程序在互联网上不能正常地工作，就是因为它们没有处理好这些问题。

在协议的分层方案中，UDP 协议处于运输层，位于互联网协议层之上、应用层之下。从概念上讲，运输层与互联网层是相互独立的，但实际上这两层之间交互密切。UDP 检验和包括 IP 源地址和目的地址，这意味着 UDP 软件必须与 IP 软件进行交互，在传输数据报之前得到这些地址。

11.11 深入研究

此处讨论的 UDP 协议是 TCP/IP 的标准，是由 Postel [RFC 768] 定义的。

11.12 习题

1. 尝试在你的本地环境上使用 UDP。对长度为 256, 512, 1024, 2048, 4096 和 8192 字节的报文，分别测试其平均传送速率。试解释测试的结果。提示：你的网络 MTU 是多少？
2. 为什么 UDP 检验和与 IP 检验和是分开的？如果有一个协议对包括 UDP 报文在内的完整 IP 数据报使用一个检验和，你是否会反对这样的协议？
3. 不使用检验和可能出现危险。如果机器 P 的一个 ARP 分组广播遭到破坏，就会造成机器 P 无法和另一台机器 Q 通信，试解释原因。
4. 通过协议端口来标志多个目的地的概念应该包括在 IP 中吗？为什么？
5. **名字登记** (Name Registry)。假设你想使用 UDP 允许任意一对应用程序建立通信，但是又不希望给它们分配固定的 UDP 端口号。你希望使用一个长度为 64 字符或稍短一些的字符串来标志可能的通信方。例如，A 机上的一个程序可能想与 B 机上的 funny-special-long-id 程序通信（你可以假定一个进程总是知道与之通信的主机的 IP 地址）。同时，C 机上的一个进程想与 A 机上的 comer's-own-program-id 进程通信。说明你只需要分配一个 UDP 端口就可以进行这样的通信，前提是设计每台机器上的软件，从而允许：(a) 一个本地进程挑选一个未用的 UDP 端口号，通信在该端口上进行；(b) 一个本地进程登记 64 字符的名字，它对此名字做出响应；(c) 一个外部进程只使用 64 字符的名字和目的站互联网地址，就能够使用 UDP 与本地进程建立通信。
6. 实现上题中的名字登记软件。
7. 使用预分配的 UDP 端口号的主要优点是什么？主要缺点呢？
8. 在一台机器内使用协议端口而不是用进程标识符来指明目的地的主要优点是什么？
9. UDP 提供了不可靠的数据报通信，因为它不确保报文的交付。要求你设计一个可靠的数据报协议，它使用超时和确认机制来确保交付。这种可靠性带来的网络开销和时延有多

少？

10. 在广域网上发送一个 UDP 数据报，测试报文丢失和重新排序的百分比。这个结果与日期时间和网络负载有关吗？

第12章 可靠的流运输服务（TCP）

12.1 引言

前几章讨论的是不可靠的无连接分组交付服务，此服务构成了所有互联网通信的基础，IP 协议对此进行了定义。本章介绍第二种最重要而且非常著名的网络级服务，即可靠的流交付服务，**传输控制协议**（Transmission Control Protocol，简称 TCP）对此进行了定义。我们将会看到，TCP 在前面讨论过的协议基础上添加了许多有价值的功能，当然它的实现也复杂得多。

虽然在此处介绍 TCP 时把它当成 TCP/IP 网际协议族的一部分，但它是一个独立的通用协议。也可适用于其他交付系统。例如，由于 TCP 几乎没有给下层网络操作提要求，它既可以在以太网之类的单个网络上使用，也可以在全球因特网上使用。

12.2 对流交付的需求

最底层的计算机通信网络提供了不可靠的分组交付。当传输差错干扰了数据时，当网络硬件出现故障时，或网络负载过大而不堪重负时，分组可能会丢失或被破坏。分组交换系统动态地改变路由，分组交付的顺序可能出现混乱，也可能经过相当长的时延才交付或重复地交付。此外，底层网络技术出于提高传输效率的需要，可能会规定一个最佳分组长度或施加其他的约束手段。

最高层的应用程序常常需要把大量的数据从一台机器发送到另一台机器。使用不可靠的无连接交付系统来传输大量数据真是苦不堪言，而且它要求程序员把差错检测和恢复机制加入到每个应用程序中。对一个能正确提供可靠性的软件而言，其设计、理解或修改都很困难，只有很少的应用程序开发人员才具备所需的技术背景知识。因此，网络协议的研究目标之一就是为可靠的流交付问题找到通用的解决方案，从而使专家们能够构建一个流交付软件的实例，以供所有应用程序使用。一个通用的协议软件，有助于应用程序从连网有关的细节中分离出来，并且使为流传送服务定义一个统一接口成为可能。

12.3 可靠交付服务的特征

应用程序与 TCP/IP 可靠交付服务之间的接口可以用五个特征来刻画：

- **面向流**（Stream Orientation）。当两个应用程序传输大量数据时，我们把这些数据看成是一个比特流，比特流又可划分为若干个**八位组**（octet）或**字节**（byte）。目的机器上的流交付服务传给接收方的八位组序列，与源机器上发送方传给流交付服务的完全一样。
- **虚电路连接**（Virtual Circuit Connection）。流传送的过程与打电话类似。在传送可以开始之前，发送应用程序和接收应用程序都要与各自的操作系统交互，通知它们需要进行流传送。从概念上讲，这就像一个应用程序打一个必须由对方来接的“电话”。在双方的操作系统中，协议软件模块通过在下层的互联网上发送报文来进行通信，核实传送是否经过认可，双方是否已做好准备。在所有的细节处置妥当之后，协议模块通知应用程序连接已经建立，传送可以开始了。在传送过程中，两台机器上的协议软件继续进行通信，以确认数据是否已正确到达。如果出于某种原因通信失败了（例如在两台机器之间的传送路径上，某个网络硬件出现故障），这两台机器都将检测到故障并报告给相应的应用程序。我们使

用术语“**虚电路**”(virtual circuit)来描述这种连接，因为尽管应用程序把这种连接当成一条专用的硬件线路，但可靠性只是流交付服务提供的一种幻象。

- **带缓冲的传送** (Buffered Transfer)。应用程序不断地给协议软件传送数据八位组，通过虚电路发送一个数据流。在传送数据时，应用程序可以使用自选大小的数据片，最小可为1八位组。在接收端，协议软件完全按数据流的发送顺序来交付流中的八位组，从而在接收完成和通过检验后，就可由接收端的应用程序使用。协议软件可以随意将流划分为若干分组，分组大小与应用程序传送的数据片大小无关。为了提高传送效率并尽可能减少网络通信量，协议软件在实现时通常会从流中收集到足够多的数据，直到填满一个适当大小的数据报，然后再传输到互联网上。这样，即使应用程序每次只产生1八位组的流，在互联网上传送的效率也比较高。与此类似，如果应用程序选择产生特别大块的数据，协议软件也可以选择把每一块划分成若干更小的片进行传输。
有些应用程序还没等到数据装满一个缓冲区就必须交付数据，流服务给它们提供了一种“推”(push)的机制，应用程序使用该机制被迫立即进行传送。在发送端，推操作强迫协议软件传送当前生成的所有数据，而不必等到装满缓冲区。当这样的数据到达接收端时，推操作会使TCP立即把数据提供给应用程序使用。不过读者也许注意到了，推功能仅仅保证所有的数据都会被传送出去，而没有规定数据记录的边界。所以，即使在强迫交付时协议软件仍可以选择对数据流进行任意划分。
- **无结构的流** (Unstructured Stream)。TCP/IP流服务没有特别地照顾到结构化的数据流，理解这一点很重要。例如，在工资表应用程序中，流服务无法区分雇员记录之间的边界，也不能识别流的内容是否为工资数据。使用流服务的应用程序在建立初始连接之前就必须了解流的内容，并就流格式达成一致意见。
- **全双工连接** (Full Duplex Connection)。TCP/IP流服务所提供的连接允许双方向的传送同时进行。这种连接称为**全双工** (full duplex) 连接。从一个应用进程的角度来看，全双工连接包括了两个相反流向的独立的流，这两个流之间没有明显的交互。流服务允许一个应用进程终止一个方向的数据流动，同时让数据继续在另一方向流动，这样的连接称为**半双工** (half duplex) 连接。全双工连接的优点在于，下层协议软件能够把一个流的控制信息放到反向传送数据的数据报中，发回到源主机。这种**捎带** (piggybacking) 减少了网络通信量。

12.4 提供可靠性

我们说过，可靠的流交付服务确保一个数据流从一台机器发送到另一台机器，且数据不会重复交付或丢失。这样就带来了一个问题，当下层的通信系统仅仅提供不可靠的分组交付时，协议软件怎样才能提供可靠的传送呢？答案很复杂，但是大部分可靠协议使用了一项基本技术，称为“**带重传的正面确认**”(positive acknowledgement with retransmission)。这项技术要求接收方与源站进行通信，在收到数据之后向源站发回一个**确认** (acknowledgement，简称ACK) 报文。发送方保存了每个分组发送的记录，在发送下一个分组之前等待确认信息。发送方还在发送分组时启动一个计时器，如果计时器超时而确认还没有收到，就会**重传** (retransmit) 刚才的分组。

图 12.1 显示了最简单的正面确认协议如何传送数据。

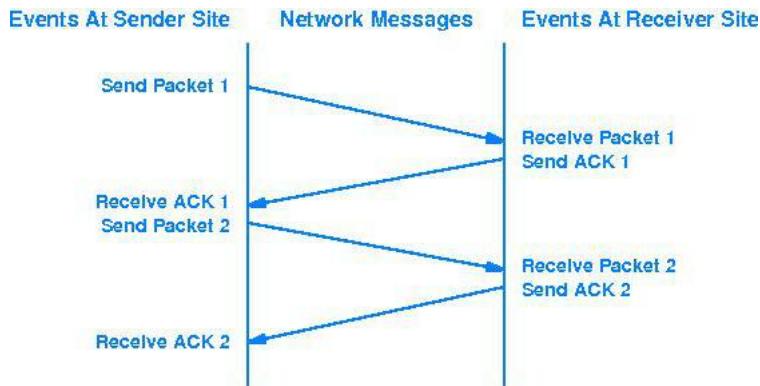


图 12.1 在使用带重传的正面确认技术的协议中，发送方等待每个已发送分组的确认。图中的垂直线段自上而下表示增长的时间，中间的斜线表示网络中分组的传输

发送方和接收方发生的事件各自出现在图的左右两边。图中每条斜线代表了一个分组在网络中的传送。

图 12.2 使用了与图 12.1 一样的格式图，显示的是分组丢失和损坏时发生的情况。发送方在传输一个分组后启动一个计时器。当计时器超时（确认还未收到），发送方就认为这个分组已丢失，并进行重传。

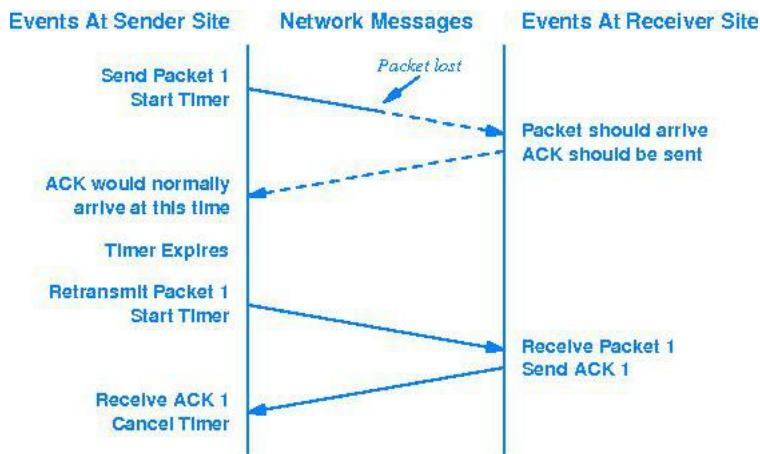


图 12.2 分组丢失时发生的超时和重传。虚线表示如果分组没有丢失，分组及其确认报文的传输可能花费的时间

最后一个可靠性问题出自于下层的分组交付系统重复递交了分组。如果网络的时延特别长，导致发送方误认为分组丢失而重传分组，也会造成重复交付。由于分组和对分组的确认都可能出现重复，因此重复交付的问题一定要谨慎处理。通常，可靠性协议通过给每个分组指定一个序号，并要求接收方记住它已收到的分组序号来检测重复分组。为了避免延迟或重复的确认造成混乱，正面确认协议把序号放在确认报文中发送回去，这样接收方就能正确地把确认与分组关联起来。

12.5 滑动窗口的概念

在研究 TCP 流服务之前，需要先探讨流传输中的另一个基本概念。这个概念称为**滑动窗口** (sliding window)，它提高了流传输的效率。为了理解滑动窗口的设计动机，我们回顾一下图 12.1 中的事件发生顺序。为了获得可靠性，发送方传输一个分组后，在传输另一个分组前要等待相应的

确认。如图 12.1 所示，尽管网络具有同时进行双向通信的能力，但在任一时刻，数据只在两台机器之间单方向流动。在机器延迟响应期间（例如当机器计算路由或检验和时），网络完全是空闲的。如果在一个传输时延很大的网络上，这个问题就更为突出：

一个简单的正面确认协议会浪费大量的网络带宽，因为它在收到前一个分组的确认之前必须推迟下一个分组的发送。

与上述简单方法相比，滑动窗口技术是正面确认和重传技术的一种更复杂形式。滑动窗口协议能更好地利用网络带宽，因为它们允许发送方在等待一个确认到达时发送多个分组。把要传输的分组序列想象成图 12.3 显示的结构，最便于想象滑动窗口的操作过程。协议在分组序列上放置了一个固定长度的小“窗口”，并传输位于窗口内的所有分组。

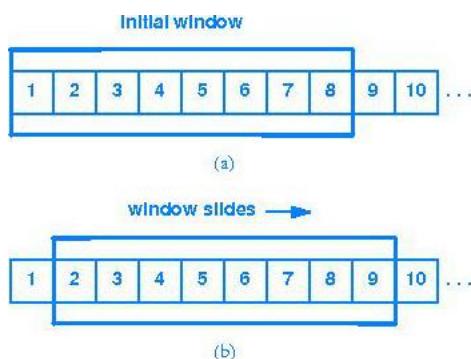


图 12.3 (a) 一个窗口中包含 8 个分组的滑动窗口协议；(b) 在分组 1 的确认收到后窗口向后滑动一格，从而使分组 9 能被发送。只有未确认的分组才会重传

如果一个分组已被传输但没有收到确认，就称这个分组是**未被确认的** (unacknowledged)。从技术上讲，在任何一个指定的时刻，未被确认的分组数受到**窗口大小** (window size) 的限制，这个窗口大小是一个固定的小整数。例如，在一个窗口大小为 8 的滑动窗口协议中，发送方在收到一个确认之前允许传输 8 个分组。

如图 12.3 所示，当发送方收到了对窗口内第一个分组的确认信息时，它可以向后滑动一格，并发送下一个分组。随着确认的不断到达，窗口也在不断地向后滑动。

滑动窗口协议的性能与窗口大小和网络接收分组的速度有关。图 12.4 就是一个滑动窗口协议软件发送三个分组时的操作示意图。注意，发送方在收到确认之前已把三个分组都传输出去了。

当窗口大小为 1 时，滑动窗口协议完全等同于简单的正面确认协议。通过增加窗口大小，就有可能完全消除网络的空闲时间。也就是说，在稳定的状态下，发送方能以网络传输分组的最快速度来发送分组。要点是：

由于调整合适的滑动窗口协议能够让网络中的分组完全饱和，因此能够获得比简单的正面确认协议高得多的吞吐率。

从概念上讲，滑动窗口协议总是要记住哪些分组已经被确认，并为每个未确认的分组分别设定单独的计时器。如果某个分组丢失，对应的计时器超时之后，发送方就要重新传输那个分组。当发送方滑动其窗口时，要把窗口移动到所有已确认的分组后面。在接收端，协议软件建立一个类似的窗口，在分组到达之后进行接收并发回确认。因此，窗口把所有分组的序列划分成三个部分：窗口左边的分组是已经成功传输、已被接收和确认过的，而窗口右边的分组是还没有传输的，窗口内的分组是已经传输了的（还没收到确认）。窗口中序号最小的分组是还没收到确认的分组序列里的第一个分组。

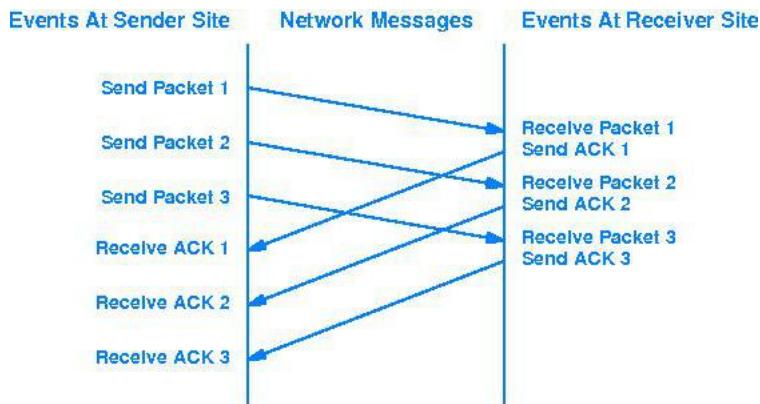


图 12.4 使用滑动窗口协议传输三个分组的例子。关键在于发送方可以发送窗口中的所有分组而不必等待确认信息

12.6 传输控制协议

理解了滑动窗口的原理之后，我们现在可以研究 TCP/IP 网际协议族如何提供可靠的流传输服务了。该服务是由**传输控制协议**（Transmission Control Protocol，简称 TCP）定义的。可靠的流服务非常重要，以至于整个协议族被命名为 TCP/IP。理解下面这一点很重要：

TCP 是一个通信协议，而不是软件的一部分。

协议与实现协议的软件之间的区别，类似于编程语言的定义与编译器之间的区别。与编程语言的情况类似，定义与实现之间的区别有时会比较模糊。大家与 TCP 软件打交道的机会远远多于与 TCP 协议规范打交道的机会，因而会很自然地把某个具体的实现当成标准。尽管如此，读者必须明确地区分两者。

到底 TCP 提供了什么东西？TCP 很复杂，所以没有一个简单的答案。该协议规定了为获得可靠的传送服务，在两台计算机之间交换的数据和确认报文的格式，还规定了计算机为确保数据的正确到达而采用的措施。协议规定了 TCP 软件如何区分某个给定机器上的多个目的地，如何对丢失或重复分组之类的差错进行恢复。协议还规定了两台计算机如何初始化一个 TCP 流传送，以及在传送完成时相互之间如何进行协商。

弄清楚协议不包括什么，这也很重要。虽然 TCP 协议规范笼统地描述了应用程序如何使用 TCP，但是并没有详细说明应用程序和 TCP 之间接口的细节。也就是说，协议文档仅仅讨论了 TCP 所能提供的操作，而并没有详细说明应用程序为访问这些操作而调用的具体过程。不详细说明应用程序接口的原因是为了实现的灵活性。特别是由于应用程序开发人员通常在计算机的操作系统中实现 TCP，无论操作系统提供的接口是什么，都需要使用这些接口。正是由于协议实现的灵活性，才使我们只需要定义单一的一种 TCP 协议规范，使用它为各种不同的机器构建协议实现的软件。

由于 TCP 几乎不给下层通信系统提什么要求，因此可与各种不同的分组交付系统一起使用。特别是，由于 TCP 不需要下层通信系统速度快或可靠，所以可在各种硬件装置系统上运行，如拨号电话线、局域网、高速光纤网、卫星连接，或者会丢失许多分组的多噪声无线连接。实际上，TCP 能够使用数目众多的交付系统，这正是其强大功能的体现。

12.7 端口、连接与端点

与第 11 章介绍的用户数据报协议（UDP）类似，TCP 在协议分层结构中位于 IP 层之上。图 12.5 给出了概念分层的组织结构。TCP 允许一台机器上的多个应用程序并发地通信，能将收到的

TCP 通信量在多个应用程序之间进行分用。与用户数据报协议类似，TCP 使用**协议端口**（protocol port）号来标志一台机器内的最终目的地。每个端口都用一个小整数来标志^①。

Conceptual Layering

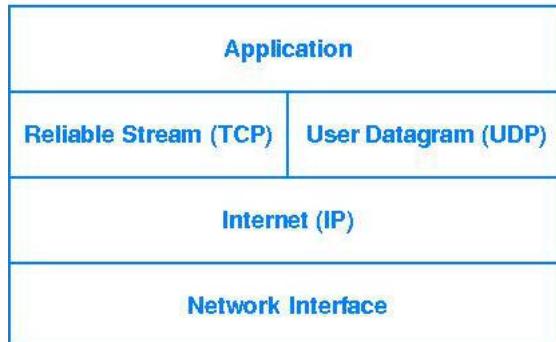


图 12.5 TCP 和 UDP 在概念分层结构中都位于 IP 层之上。TCP 提供可靠的流服务，UDP 提供不可靠的数据报交付服务。应用程序使用这两种服务

在讨论 UDP 端口时讲过，可以把每个端口想象为协议软件用来放置到达数据报的队列。TCP 的端口更复杂一些，因为一个指定的端口号并不对应于一个单独的对象。相反，TCP 是建立在**连接抽象**（connection abstraction）上的，协议端口号标志的对象不是个别端口，而是虚电路连接。明白 TCP 使用连接的概念至关重要，因为它有助于解释 TCP 端口号的含义和使用：

TCP 使用连接而不是协议端口作为它的基本抽象，连接是用一对端点来标志的。

那么，连接的“端点”究竟是什么呢？我们曾经提到，一个连接由两个应用程序之间的一条虚电路组成，所以把应用程序当成连接“端点”的想法是很自然的。然而，这种想法并不正确。实际情况是，TCP 把**端点**（endpoint）定义为一对整数，即（host, port），其中 host 是一个主机的 IP 地址，而 port 则是该主机上的一个 TCP 端口。例如，端点（128.10.2.3, 25）表示的是 IP 地址为 128.10.2.3 的主机上的 TCP 端口 25。

定义了端点之后就很容易理解连接了。一个连接是由它的两个端点定义的。也就是说，如果麻省理工学院的一台 IP 地址为（18.26.0.36）的机器与普度大学的一台 IP 地址为（128.10.2.3）的机器之间建立连接，该连接可以由两个端点定义：

（18.26.0.36, 1069）和（128.10.2.3, 25）

同时，另一个连接可能建立在信息科学研究院的地址为（128.9.0.32）的机器与普度大学的同一台机器之间，该连接用它的两个端点表示为：

（128.9.0.32, 1184）和（128.10.2.3, 53）

至此，我们举例的连接还是简单易懂的，因为所有端点中用到的端口号都是唯一的。但是，连接抽象允许多个连接共享一个端点。例如，可以再增加一个连接，从卡内基梅隆大学一台地址为（128.2.254.139）的机器连接到普度大学的那台机器：

（128.2.254.139, 1184）和（128.10.2.3, 53）

我们看到同时有两个连接使用了地址为（128.10.2.3）的那台机器上的 TCP 端口 53，这看起来有点奇怪，但是并不会引起歧义，因为 TCP 把进入的报文与连接对应起来，而不是与协议端口对应起来，它使用两个端点来标志相应的连接。要牢记这个重要的观点：

由于 TCP 使用一对端点来标志连接，同一台机器上的某个 TCP 端口号可以被多个连接共享。

^① 虽然 TCP 和 UDP 使用从 1 开始的整数端口标识符来标志端口，但两者之间不会产生混淆，因为收到的 IP 数据报除记录了端口号之外，还记录了使用的协议种类。

从程序员的角度来看，连接抽象是很重要的。这意味着程序员能够设计一个程序，同时为多个连接同时提供服务，而且不需要为每个连接分配唯一的本地端口号。例如，大多数系统都提供电子邮件服务的并发访问，允许多台计算机同时发送电子邮件。由于接收电子邮件的程序使用 TCP 进行通信，尽管它允许多个连接同时接收，也只需要使用一个本地 TCP 端口。

12.8 被动打开和主动打开

TCP 是一个面向连接的协议，与 UDP 不一样，TCP 需要两个端点都同意参与通信。也就是说，在 TCP 通信量可以通过互联网传递之前，位于连接两端的应用程序都必须同意建立连接。为此，连接一端的应用程序执行**被动打开** (passive open) 的操作，告诉操作系统自己将会接受一个传入的连接。这时，操作系统为连接的这一端分配一个 TCP 端口号。在连接的另一端，应用程序必须使用**主动打开** (active open) 的请求，告诉操作系统自己要建立一个连接。这两个 TCP 软件模块进行通信，以建立和验证一个连接。创建好连接之后，应用程序才能开始递送数据；连接两端的 TCP 软件模块进行报文交换，以确保数据的可靠交付。在详细介绍 TCP 报文的格式之后，我们再继续讨论建立连接的细节。

12.9 报文段、流和序号

TCP 把数据流当成八位组或字节的序列，而为了便于传输，又把序列划分成若干**报文段** (segment)。通常，每个报文段被放到一个 IP 数据报中，在下层互联网上运送。

TCP 使用一个专门的滑动窗口机制来解决两个重要问题：高效传输和流量控制。与前面描述的滑动窗口协议类似，TCP 窗口机制允许在收到一个确认之前发送多个报文段。这种做法使网络总是处于忙碌状态，从而提高了整个网络的总吞吐率。TCP 的滑动窗口协议还解决了端到端的**流量控制** (flow control) 问题，允许接收方对传输进行限制，直到它拥有足够的缓冲空间来容纳更多的数据。

TCP 的滑动窗口机制是按八位组操作的，而不是按报文段或分组操作的。数据流的八位组按顺序编号，发送方给每个连接保留三个指针。这些指针定义了一个滑动窗口，如图 12.6 所示。第一个指针标出了滑动窗口的左边界，把已发送并得到确认的八位组与尚未得到确认的八位组区分开。第二个指针标出了窗口的右边界，指出在收到更多确认之前序列中可以发送的最高八位组。第三个指针位于窗口的内部，把已发送的八位组和尚未发送的八位组区分开。协议软件会立刻发送窗口中的所有八位组，所以窗口内的分界线会迅速地从左向右移动。

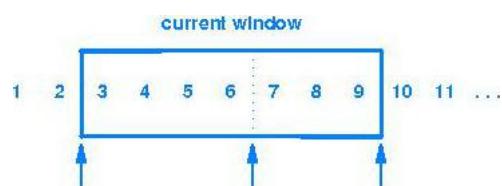


图 12.6 一个 TCP 滑动窗口的例子。2 号以前的八位组已发送并得到了确认，3 号到 6 号的八位组已经发送但尚未得到确认，7 号到 9 号的八位组还未发送出去，但将立即发送，而 10 号及更高序号的八位组在窗口移动之前不能发送

我们已经描述过发送方的 TCP 窗口如何滑动，还提到接收方必须设立一个类似的窗口把八位组流重新拼凑起来。但是，由于 TCP 连接是全双工的，在一个连接上的两个相反方向的传送可以同时进行，理解这一点非常重要。我们认为这两个方向的传送是完全独立的，因为在任意时刻数据都可以在连接上单方向流动，或者在两个方向上都有数据流动。于是，每一端的 TCP 软件都为每

一个连接维持两个窗口（一个连接总共有四个窗口），其中一个窗口在发送数据时滑动，而另一个窗口在接收数据时滑动。

12.10 可变的窗口大小与流量控制

TCP 滑动窗口协议与前面介绍的简单滑动窗口协议有一个区别，因为 TCP 允许窗口大小随时间而改变。在每个确认中，除了指明已收到多少八位组之外，还包括了一个**窗口通告**（window advertisement），指出接收方准备再接受多少八位组的数据。我们可以认为窗口通告指出了接收方缓冲区当前可用的空间大小。为了响应窗口通告值的增加，发送方扩大滑动窗口的大小，继续发送新窗口右边界内尚未发送的八位组。为了响应窗口通告值的减少，发送方缩小发送窗口的大小，并停止发送超过新窗口右边界的八位组。TCP 软件通过让收缩后的窗口位于流中以前已被接受的八位组位置之后，应该不使当前窗口通告和以前的通告产生冲突。实际上，较小的窗口通告是随着确认送达的，也就是说，在窗口向前滑动时，窗口的大小才发生改变。

使用大小可变的窗口的优点是：它不仅提供了可靠传输，而且还提供了流量控制。如果接收方的缓冲区快要满了，为避免收到的数据存储不下，就会发出较小的窗口通告。在极端的情况下，接收方通告窗口大小为零，从而停止所有的传输。而在缓冲区空间再次可用之后，接收方通告一个非零的窗口大小，再次激活数据的流动^①。

如果有多种速率和型号的机器通过不同速率、容量的网络和路由器进行通信，在这种互联网环境中具备流量控制机制十分重要。这包括两个独立的问题。第一，协议需要在源站和最终目的站之间提供端到端的流量控制。例如，当一个手持式 PDA 与一个速度快的 PC 通信时，PDA 需要调节数据的流动速率，否则协议软件很快就会超载。因此，TCP 必须实现端到端的流量控制来确保可靠的交付。第二，当源站发送的通信量超过了中间系统机器的承受能力时，需要有一种机制允许中间系统（如路由器）来控制源站的发送速率。

当中间的机器超载时，这种状况称为**拥塞**（congestion），解决这个问题的机制称为**拥塞控制**（congestion control）。TCP 使用滑动窗口的方案来解决端到端的流量控制问题，稍后将讨论拥塞控制，但应该注意到，一个设计很好的协议能够检测出拥塞并从拥塞恢复正常，而一个设计低劣的协议却会让拥塞变得更严重。特别要注意，一个精心选择的重传方案有助于避免拥塞，但选择不好的方案只会造成大肆重传，反而加剧拥塞。

12.11 TCP 报文段的格式

两台机器上的 TCP 软件之间传送的数据单元称为**报文段**（segment）。两台机器通过报文段的交互来建立连接、传送数据、发送确认、通告窗口大小以及关闭连接。由于 TCP 使用了捎带技术，尽管 A 机运送给 B 机的确认是对 B 机发给 A 机的数据加以确认，但该确认可以和 A 机运送给 B 机的数据一起放入同一个报文段中传送^②。图 12.7 给出了 TCP 报文段的格式。

^① 当窗口大小为零时有两种例外的传输情况。一种情况允许发送方传输**紧急位**（urgent bit）置 1 的报文段，通知接收方有紧急数据可用。另一种情况，在窗口大小到零后，为了避免非零窗口通告丢失时可能造成的死锁，发送方定期发出零窗口探询报文段。

^② 实际上不会经常发生捎带，因为大多数应用程序不同时在两个方向上发送数据。

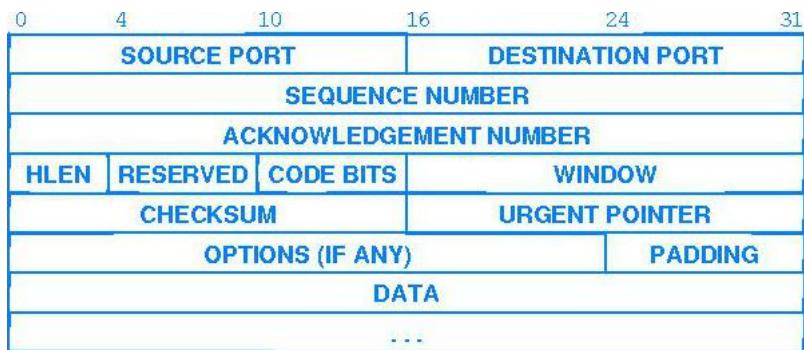


图 12.7 TCP 报文段的格式，包括一个 TCP 首部以及数据部分。报文段用来建立连接、运载数据和确认

每个报文段分为两个部分：首部和后面的数据。首部，也就是 TCP 首部（header），携带了所需的标志和控制信息。源端口（SOURCE PORT）和目的端口（DESTINATION PORT）字段包含了 TCP 端口号，用于标志位于连接两端的应用程序。序号（SEQUENCE NUMBER）字段指出这个报文段的数据在发送方的字节流中的位置。确认序号（ACKNOWLEDGEMENT NUMBER）字段指明源站希望收到的下一个八位组的编号（此编号之前的八位组已确认收到）。注意，与序号相关的流与报文段的流向相同，而与确认序号相关的流与报文段流向相反。

首部长度（HLEN）字段^①包含一个整数，指明了报文段首部的长度，长度的度量单位是 32 位。由于选项（OPTIONS）字段的长度根据所含选项的不同而变化，因此需要用这个字段来指明首部长度。也就是说，TCP 报文段的长度随着所选择的选项而变化。6 位保留（RESERVED）字段是为将来的应用而保留的。

一些报文段只携带确认信息，而另一些报文段携带数据。还有一些报文段携带了建立或关闭连接的请求。TCP 软件使用名为代码比特（CODE BITS）的 6 位字段来表示报文段的目的和内容。这 6 个比特位告诉我们如何解释报文段首部中的其他字段，如图 12.8 所示。

Bit (left to right)	Meaning if bit set to 1
URG	Urgent pointer field is valid
ACK	Acknowledgement field is valid
PSH	This segment requests a push
RST	Reset the connection
SYN	Synchronize sequence numbers
FIN	Sender has reached end of its byte stream

图 12.8 TCP 首部中代码比特字段的 6 个比特位

TCP 软件每次发送一个报文段时，通过在窗口（WINDOW）字段中指明可用缓冲区大小，通告自己愿意接受多少数据。该字段包含一个 16 位无符号整数，整数按网络标准字节顺序存储。窗口通告提供了捎带的另一个例子，因为通告可以附加在任何报文段中，既包括那些携带数据的报文段，也包括只携带确认的报文段。

12.12 带外数据

虽然 TCP 是一个面向流的协议，但有时位于连接一端的应用程序需要发送**带外数据**（data out of band），而不必等待另一端的程序处理完流中已有的八位组，这种做法很有意义。例如，当 TCP 用

^① 协议规范表明，HLEN 字段是数据区在报文段内的偏移量。

于远程登录会话服务时，用户可能决定发送一个键盘字符序列来**中断** (interrupt) 或**异常终止** (abort) 在另一端运行的程序。通常，当远端主机上的程序不能正常运行时，就非常需要这些信号。这种信号必须立即发送出去，而不必等待程序读取 TCP 流中已有的全部八位组（否则就不能终止已停止读取输入的程序）。

为了适应这些带外信号的发送，TCP 允许发送方把数据指明为**紧急的** (urgent)，这意味着接收端的程序应该能尽快收到紧急数据到达的通知，而不考虑紧急数据在流中处于什么位置。协议规定，当发现紧急数据后，接收方的 TCP 应当通知与连接相关的应用程序进入“紧急模式”。当所有紧急数据用完之后，TCP 软件告诉应用程序返回到正常操作状态。

TCP 如何通知应用程序有紧急数据到达的具体细节与计算机的操作系统有关。当把紧急数据放在一个报文段中传输时，紧急数据的标记包括报文段首部中的 URG 位和紧急指针 (URGENT POINTER) 字段。当 URG 位被置为“1”之后，紧急指针字段指明了紧急数据的末尾在报文段中的位置。

12.13 TCP 选项

如图 12.7 所示，TCP 报文段的首部中可能包含零个或多个选项。每个选项的开头是 1 八位组字段，指明选项的**类型** (type)^①；紧接着是 1 八位组长度字段，指明选项的长度（八位组的个数）；最后是选项本身。如果选项所占的位数不是 32 的倍数，则在首部的末尾添加填充 (PADDING) 字段。

12.13.1 最大报文段长度的选项

发送方可以选择在每个报文段中放置的数据量。但是，一个 TCP 连接的两端必须在传送前就最大报文段达成一致意见。TCP 使用选项字段和另一端的 TCP 软件进行协商，TCP 使用一个**最大报文段长度** (Maximum Segment Size, 简称 MSS) 选项，允许接收方指出自己愿意接受的最大报文段长度。于是，一个仅有几百字节缓冲空间的嵌入式系统，可以规定一个 MSS 来限制报文段的大小，使之能装入缓冲区中。由于允许异构的系统相互通信，即超级计算机可以与小型嵌入式系统通信，所以 MSS 协商就变得特别有意义。为了最大限度地提高吞吐率，当两台计算机连接到同一个物理网络时，TCP 通常会计算出一个最大的报文段长度，从而使形成的 IP 数据报能够与网络的 MTU 相适应。如果连接的两端不在同一个物理网络上，它们可尝试找到两者之间路径沿线上的最小 MTU，或选择把 536 作为最大报文段长度 (IP 数据报的默认长度是 576，还要减去 IP 和 TCP 首部的标准长度)。

在一般的互联网环境中，选择合适的大报文段长度可能很困难，MSS 取值过大或过小都可能使网络性能变差。一方面，如果报文段长度小，网络的利用率就会比较低。前面讲过，TCP 报文段是封装在 IP 数据报中运送的，而 IP 数据报又封装在物理网络帧中。于是，每个报文段除了数据之外，还有至少 40 八位组的 TCP 和 IP 首部，因此携带 1 八位组数据的数据报，只使用了至多 1/41 的底层网络带宽来运送用户数据；而在实际应用中，分组之间的最小间隙和形成网络硬件帧额外所需的比特，还会使这个比率更低。

另一方面，太大的报文段长度也会降低网络性能。大的报文段形成大的 IP 数据报。当这样的数据报在 MTU 较小的网络上传输时，IP 必然把数据报分片。与 TCP 报文段不一样，IP 数据报片不能独立确认或重传；所有数据报片必须正确到达，否则就要重传整个数据报。因为丢失某个数据报片的概率总是存在的，TCP 报文段长度增大到超过数据报分片的阈值后，就会降低数据报成功到达的概率，从而降低网络的吞吐率。

^① RFC 使用术语 kind 而不用 type。

从理论上讲，在 IP 数据报从源站到目的站的路径上都不需要分片的前提下，让 IP 数据报携带尽可能长的报文段，那么这个报文段的长度就是最佳长度 S。在实际应用中，找到 S 值是很困难的，原因有几点：首先，大多数 TCP 的实现并不提供相应的机制^①；其次，由于互联网中的路由器可以动态改变路由，在通信的一对计算机之间，数据报经过的路径可以动态改变，因此必须进行分片的数据报长度也会改变；此外，最佳长度 S 还取决于较低层协议的首部，例如报文段的长度必须减小才能适应 IP 选项的情况。

12.13.2 窗口扩大比例选项

由于 TCP 首部中的窗口字段长度为 16 位，因此最大的窗口大小为 64 KB。虽然这个窗口对早期网络是足够用的，但对于像卫星信道那样的网络，由于有一个大延迟、大带宽的通道——非正式地称为**长粗管道**（long fat pipe），为了获得高吞吐率，需要更大的窗口大小。

为适应更大的窗口大小，提议在 TCP 中使用**窗口扩大选项**（window scaling option）。此选项包括三个八位组：类型、长度和**移位**（shift）值 S。实际上，移位值表示应用在窗口字段值上的一个二进制扩大因子。当窗口扩大有效时，接收方从窗口字段中提取数值 W，然后把 W 向左移动 S 位后获得实际的窗口大小。

有几个细节问题使设计复杂化了。窗口扩大选项可以在初始建立连接时进行协商，在这种情况下，接收方会以为所有后续窗口通告都使用已协商过的扩大因子。或者，可以在每个报文段中指明选项，在这种情况下，每个报文段的扩大因子可以不一样。此外，如果连接的某一端实现了窗口扩大，当它不需要扩大其窗口时，可以发送值为零的选项，使扩大因子变为“1”。

12.13.3 时间戳选项

发明 TCP 时间戳选项是为了帮助 TCP 计算下层网络上的时延，也用于处理 TCP 序号超过 2^{32} 的情况，后者称为**防止序号绕回**（Protect Against Wrapped Sequence numbers，简称 PAWS）。除了所需的类型和长度字段，时间戳选项还包括两个值：一个是时间戳值，另一个是回送回答时间戳值。发送方在发送一个分组时把当前时钟的时间值放入时间戳字段；接收方在返回这个分组的确认前，把时间戳字段值复制到回送回答字段。因此，在确认到达后，发送方可以准确地计算出自报文段发送以来总共经过了多长时间。

12.14 TCP 检验和的计算

TCP 首部中的检验和（CHECKSUM）字段包含一个 16 位整数的检验和，用于检验数据和 TCP 首部的完整性。为计算检验和，发送端机器上的 TCP 软件采用了类似于第 11 章中 UDP 检验和的计算过程。它给报文段引入了一个**伪首部**（pseudo header），添加足够多的零比特，使报文段的长度为 16 位的整数倍，然后在这些操作的结果上计算 16 位的检验和。TCP 并不把伪首部和填充位计入报文段长度，也不传输它们。而且，TCP 出于计算检验和的目的，假定检验和字段本身为零。类似于其他校验和的计算，TCP 使用 16 位运算，计算各个 16 位单元的二进制反码和的二进制反码。在接收端，TCP 软件进行同样的计算，以检验收到的报文段是否完整。

TCP 使用伪首部的目的与 UDP 完全一样。伪首部中包括了主机的 IP 地址和一个协议端口号，这样就使接收方能够核实报文段是否到达了正确的目的地。源 IP 地址和目的 IP 地址对 TCP 都很重要，因为 TCP 必须使用它们来识别报文段属于哪个连接。因此，一旦收到了一个携带 TCP 报文段的数据报，IP 层必须从数据报中取出源 IP 地址和目的 IP 地址，并把它们随同报文段本身一起传给

^① 为了获得路径 MTU 的值，发送方通过发送设置了 IP 不分片位的数据报来探测路径。然后，如果 ICMP 差错报文报告需要分片，则减小数据报长度。

TCP 层。图 12.9 显示了检验和计算中使用的伪首部的格式。

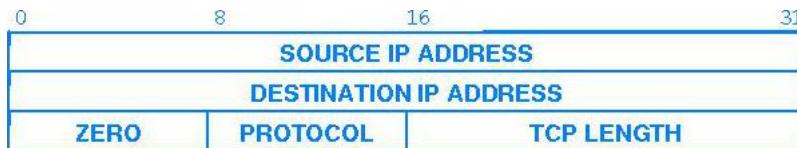


图 12.9 TCP 计算检验和中使用的伪首部的格式。在接收端，从携带报文段的 IP 数据报中提取出这些信息

发送端的 TCP 给协议 (PROTOCOL) 字段赋值，下层交付系统将在协议类型字段中使用这个值。对于携带 TCP 报文段的 IP 数据报，这个值为 6。TCP 长度 (TCP LENGTH) 字段给出了包括 TCP 首部在内的 TCP 报文段的总长度。在接收端，伪首部中使用的信息是从携带报文段的 IP 数据报中提取出来的，并用于检验和的计算，以验证报文段是否已完好无损地到达了正确的目的地。

12.15 确认、重传和超时

由于 TCP 使用可变长度的报文段来发送数据，而且重传的报文段中可能比原报文段包含更多的数据，所以不能简单地对数据报和报文段进行确认。实际上，TCP 使用流序号对流中的一个位置进行确认。接收方从到达的报文段中收集数据八位组，并重构出一个发送流的副本。因为报文段是在 IP 数据报中运送的，它们可能丢失或无序交付；接收方使用序号将报文段重新排序。在任一时刻，接收方可能已重构了始于流起点的零个或多个连续的八位组，但是也可能因报文段的无序到达而收到流中其他部分的八位组。接收方总是对已正确收到流的最长连续前缀进行确认。每个确认给出一个序号值，其值比收到的连续前缀中最高八位组的位置大 1。这样，发送方就能在继续发送流的同时，从接收方得到连续的反馈信息。我们可以把这个重要的观点概括为：

一个 TCP 确认指出了接收方期望收到的下一个八位组的序号。

TCP 的确认方法称为**累积确认** (cumulative acknowledgement)，因为它报告了接收方已累积收到流中的多少个八位组。累积确认既有优点也有缺点。一个优点是这种确认容易产生，不会有二义性。还有一个优点，若确认丢失，发送方没有必要进行重传。但是，累积确认也有一个大缺点，发送方无法收到所有成功传输的相关信息，而只能知道接收方已收到的八位组在流中的位置信息。

缺乏所有成功传输的相关信息会使累积确认的效率降低，要知道原因，可以设想一个包括 5000 个八位组的窗口，窗口从流的位置 101 开始。假设发送方通过发送 5 个报文段把窗口中的八位组全都传输出去了。再假设第一个报文段丢失，但其他报文段都完好无损地到达了。在每个报文段到达时，接收方都发送一个确认，但每个确认指向的却是八位组 101，即它希望收到的下一个八位组的位置。接收方无法告诉发送方，当前窗口中的大部分数据都已收到。

若发送方一方出现超时，它必须在两种可能的低效策略中进行选择。可以选择重传一个报文段，也可以选择重传全部 5 个报文段。在这个例子中重传 5 个报文段效率很低。当重传的第一个报文段到达之后，接收方就会获得窗口中的全部数据，然后会发送确认 5101。如果发送方遵循已接受的标准，只重传第一个未被确认的报文段，它在决定发送什么报文段和发送多少报文段之前，必须要等到确认。这样它又退化成一个简单的正面确认协议，从而可能失去使用大窗口的优点。

TCP 中最重要并且最复杂的概念之一在于其处理超时和重传的方式。与其他可靠的协议类似，TCP 希望目的站成功地从流中收到新的八位组时能够发送确认。TCP 每发送一个报文段，就启动一个计时器并等待确认。如果在报文段中的数据未确认之前计时器超时，TCP 就认为该报文段已经丢失或损坏，就会重传该报文段。

要弄清楚 TCP 重传算法与其他网络协议所用的算法为什么有区别，需要记住 TCP 是针对互联

网环境使用的协议。在互联网中，在一对机器之间运送的报文段可能只穿过一个低时延的网络（例如高速局域网），也可能经过多个路由器，穿过多个中间网络。因此，不可能预先知道确认信息多快能返回发送方。此外，每个路由器上的时延与通信量有关，所以报文段送到目的站以及确认返回源站所需的总时间每时每刻都在变。图 12.10 显示的是在全球因特网上连续传输 100 个分组的往返时间测量结果，由此可以说明时延变化的问题。报文段到达不同目的站所需的时间有很大差异，到达同一个目的站所需的时间也会随通信量负载的变化而出现很大差异，TCP 软件必须适应这两个方面的差异。

TCP 使用**自适应重传算法**（adaptive retransmission algorithm）来适应互联网时延的变化。该算法的实质是：TCP 监视每条连接的性能，由此推算出适当的超时时限值。当连接的性能变化时，TCP 随即修改时限值（也就是说，它能自动适应时延的变化）。

为了搜集自适应算法所需的数据，TCP 记录每个报文段发送出去的时间和该报文段数据确认到达的时间。从这两个时间，TCP 计算出传输过程所需的时间，即**样本往返时间**（sample round trip time）或**往返时间样本**（round trip sample）。每当获得一个新的往返时间样本，TCP 就修改这个连接的平均往返时间。通常，TCP 软件计算往返时间的加权平均值，作为往返时间的估计值（RTT），并使用新的往返时间样本逐步修改这个平均值。例如，在计算新的加权平均值时，早期的一种平均技术使用一个常数权重因子 α ， $0 \leq \alpha < 1$ ，对旧的平均值和最新的往返时间样本进行加权：

$$RTT = (\alpha * Old_RTT) + ((1 - \alpha) * New_Round_Trip_Sample)$$

选用接近 1 的 α 值会使加权平均值对短暂的时延变化不敏感（例如，仅有一个报文段遇到长时延），而选择接近 0 的 α 值则会使加权平均值很快地响应时延的变化。

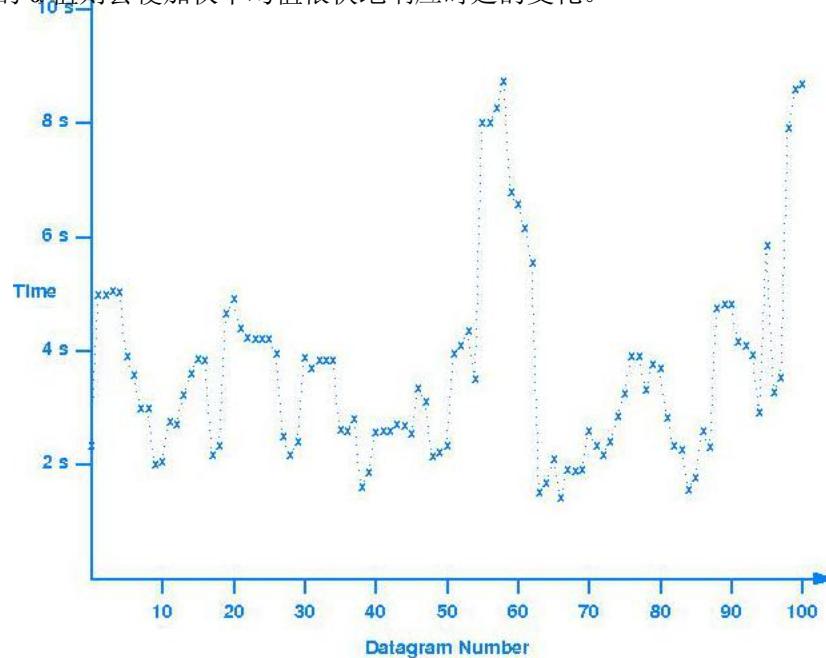


图 12.10 因特网上相继传输的 100 个 IP 数据报的往返时间标绘图。虽然因特网的时延现在低了不少，但整体时延仍随时间而变化

当发送报文段时，TCP 计算出一个超时时限值，它是当前的往返时间估计值的函数。早期实现的 TCP 协议都使用常数加权因子 β ($\beta > 1$)，使时限值大于当前的往返时间估计值：

$$Timeout = \beta * RTT$$

早期规范推荐设置 $\beta = 2$ ，本章后面介绍的一些近期著作提出了更好地调节超时时限的技术。

至此可总结如下：

为了适应互联网环境中遇到的时延变化，TCP 使用自适应重传算法来监测每个连接的时延，并相应地调整各连接的超时参数。

12.16 往返时间样本的精确测量

从理论上讲，测量一个往返时间样本是很简单的，只需用报文段确认到达的时刻减去报文段发送完成的时刻即可。但 TCP 使用累积确认方式，把问题复杂化了，因为确认针对的是收到的数据，而不是针对携带这些数据的某个数据报实例。下面看一个重传的例子。TCP 生成了一个报文段，把它放到数据报里发送出去，然后计时器超时了，TCP 就在第二个数据报中再次发送这个报文段。由于这两个数据报携带了完全相同的数据，发送方无法知道确认是针对原来的数据报还是重传的数据报。这种现象称为**确认二义性** (acknowledgement ambiguity)，我们可以称这样的 TCP 确认是有**二义性的** (ambiguous)。

TCP 应该认为确认是针对最早的（即原来的）那次传输，还是最近的（即最新的重传）那次传输呢？令人意外的是，两种假设都不成立。如果把确认与原来的传输关联起来，那么在容易丢失数据报的互联网环境中，可能会使往返时间估计值无限增大^①。如果确认在一次或多次重传后到达，TCP 从原来的传输中测量往返时间样本，并使用这个很长时间值的样本计算新的 RTT，从而使 RTT 变大一点。TCP 在下一次传输报文段时，变大的 RTT 会使超时时限值变得稍长一些。因此，如果确认在一次或多次重传后到达，下一个样本往返时间将会变得更大，以此类推下去。

如果把确认与最近的重传关联起来，也可能导致错误。考虑端到端时延突然增大时发生的情况。当 TCP 发送一个报文段时，它使用旧的往返时间估计值来计算超时时限，现在（时延已突然增加）这个值就太小了。该报文段已正确到达且开始往回发送确认，可是时延增加使计时器在确认还未到时就超时了，于是发送方重传这一报文段。TCP 重传报文段之后不久，对原来传输报文段的确认就到达了，但这被当成是对重传的确认，因而这个往返时间样本就会很小，从而使计算出的往返时间估计值减小一些。遗憾的是，减小 RTT 估计值会使 TCP 设定的超时时限对于下一个报文段而言太小。最后，往返时间估计值将稳定在一个值 T 上，而真正的往返时间比 T 的某个倍数稍长一些。观察那些把确认与最近重传相关联的 TCP 实现，会发现在稳定的状态下 RTT 值略小于正确往返时间值的一半（也就是说，尽管没有发生丢失，TCP 也会把每个报文段发送两次）。

12.17 Karn 算法与计时器补偿

如果原来的传输和最近的重传都不能提供精确的往返时间，那么 TCP 应该怎么办呢？解决的方法也很简单：TCP 不应该为重传的报文段更新往返时间估计值。这个思想，即**Karn 算法** (Karn Algorithm)，只对没有二义性的确认（即报文段只发送一次就收到的确认）调整往返时间的估计值，从而避免了有二义性的确认带来的问题。

当然，有一种简单实现的 Karn 算法只忽略了从重传的报文段得到的往返时间，也可能导致错误的发生。考虑在时延突然增大后 TCP 发送一个报文段时出现的情况。TCP 使用已有的往返时间估计值来设置超时时限。而在时延增大后，这个超时时限相比之下就太小了，从而造成报文段重传。如果 TCP 忽略重传对往返时间的影响，它就不能修改估计值，从而使重传反复重复下去。

为了适应这种出错的情况，Karn 算法要求发送方在对重传的超时时限进行设置时结合使用**计时器退避** (timer backoff) 策略。退避技术使用类似前面介绍的公式来计算最初的超时时限。但是，当计时器出现超时并导致重传时，TCP 就加大超时时限值。实际上，每当 TCP 必须重传一个报文段时，它就增加超时时限值（为避免时限值无限增加，大多数实现都规定了时限值的上限，这个值比互联网中任何一条路径上的时延都大）。

TCP 实现使用多种不同的技术来计算退避，它们大多使用一个乘法系数 γ ，将新的时限值设置为：

^① 如果每个报文段至少丢失一次，估计值只可能会任意增大。

$$\text{new_timeout} = \gamma * \text{timeout}$$

通常 γ 的值为 2 (已证明 γ 值小于 2 时将导致不稳定)。其他一些实现使用一个乘法系数表，从而允许每一步的退避不一样^①。

Karn 算法把退避技术与往返时间的估计结合起来，解决往返时间估计值永不增加的问题：

Karn 算法：在计算往返时间估计值时，忽略重传报文段对应的样本，但要使用退避策略，从重传的分组计算得到超时时限值，并且这个值对后续的分组保持不变，直到获得一个新的有效样本时，才会更新超时时限值。

一般来说，当互联网行为出现异常时，Karn 算法把超时时限值的计算与当前的往返时间估计值区分开。它使用往返时间估计值来计算初始的时限值，再在每次重传时对时限值进行退避处理，直到能成功地传送一个报文段为止。当它发送后续的报文段时，保持退避计算得到的时限值不变。最后，当一个无需重传的报文段对应的确认到达后，TCP 重新计算往返时间估计值，并重置相应的超时时限。实践表明，Karn 算法在分组丢失率很高的网络上也能很好地工作^②。

12.18 对高方差时延的响应

对往返时间估计值进行深入研究，会发现上述计算方法都不能适应时延变化范围很大的情况。由排队论可知，往返时间的增长与 $1 / (1 - L)$ 成比例， L 是当前网络负载， $0 \leq L \leq 1$ ，而往返时间的均方差 σ 与 $1 / (1 - L)^2$ 成比例。如果互联网运行负载为其网络容量的 50%，我们预测往返时延相对于平均往返时间变化的因子为 4。当网络负载为 80% 时，我们预测变化因子为 25。早期的 TCP 标准使用的往返时间估计技术在前面已讨论过。使用这种技术并限制 β 为推荐值 2，意味着往返时间的估计值最多能适应网络负载为 30% 的情况。

1989 年的 TCP 协议规范要求，协议在实现时既要估计往返时间的平均值，也要估计往返时间的方差，并使用方差的估计值替代常数 β 。这样，TCP 的新实现就能适应更大范围变化时延的情况，还能实现较高的吞吐率。所幸这些估计值所需的计算量并不大，由下列简单的方程式可以导出高效率的程序：

$$\begin{aligned}\text{DIFF} &= \text{SAMPLE} - \text{Old_RTT} \\ \text{Smoothed_RTT} &= \text{Old_RTT} + \delta * \text{DIFF} \\ \text{DEV} &= \text{Old_DEV} + \rho(|\text{DIFF}| - \text{Old_DEV}) \\ \text{Timeout} &= \text{Smoothed_RTT} + \eta * \text{DEV}\end{aligned}$$

其中 DEV 是平均偏差 (mean deviation) 的估计值， δ 是一个 0~1 之间的分数，用于控制新样本对加权平均值影响的快慢程度， ρ 也是一个 0~1 之间的分数，用于控制新样本对平均方差影响的快慢程度，而 η 是控制方差对往返超时时限影响程度的因子。为提高计算效率，TCP 把 δ 和 ρ 都选择为 2 的幂的倒数，这样，为 δ 和 ρ 选择合适的 n 值，把计算变为缩小 2^n 倍的整数运算。研究表明，当 $\delta = 1/2^3$, $\rho = 1/2^2$, $\eta = 3$ 时，算法就工作得很好了。在 4.3 BSD UNIX 中 η 的初值为 2，而在 4.4 BSD UNIX 中这个值变为 4。

图 12.11 使用一组随机生成的值，显示了超时时限的计算结果如何随往返时间的变化而变化。尽管往返时间是模仿出来的，但它们遵循实际应用中观测到的模式：随着总体上平均值的起伏，相邻分组间的时延变化很小。

注意，往返时间的频繁变化，包括反复地增长和减少，都可能使重传计时器的值增大。此外，尽管在时延升高时计时器的值增长趋势很快，但随着时延的下降计时器的值并未如此迅速地降低。

^① Berkeley UNIX 是最著名的使用乘法系数表的系统，但表中的值相当于使用 $\gamma = 2$ 。

^② Phil Karn 是一个业余无线电爱好者，他提出了这个算法，使 TCP 通信可以在分组丢失率高的无线连接上进行。

图 12.12 使用了图 12.10 中的数据点说明 TCP 如何响应时延变化的极端情况。前面讲过，我们的目标是让重传计时器尽可能准确地估计出实际往返时间，而不能低估。该图表明，尽管计时器反应迅速，但它还是可能低估。例如，在箭头标记的两个相邻数据报之间，时延从不足 4 s 加倍增加到 8 s。更重要的是，在相对稳定（时延变化较小）的一段时间之后出现了急剧的变化，这样的变化让任何算法都无法预测。在这个例子的 TCP 算法中，由于超时时限（大约为 5）大量低估了长时延，所以会发生不必要的重传。但是，估计值能够对时延的增加迅速做出响应，这意味着连续到达的分组不必重传。

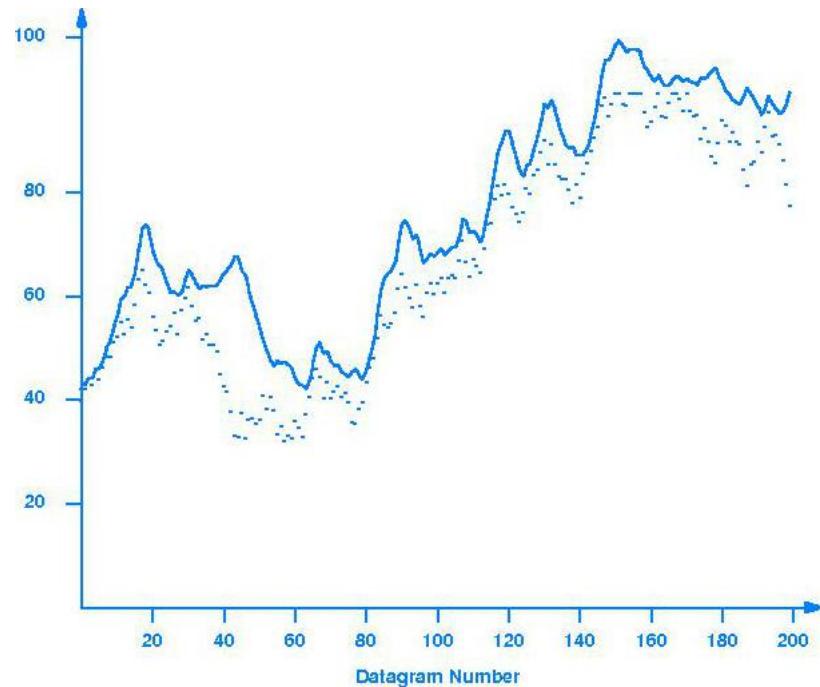


图 12.11 一组 200 个（随机生成的）往返时间的标绘点，TCP 重传计时器的时间用连续的线条表示。当时延变化时，超时时限会增加

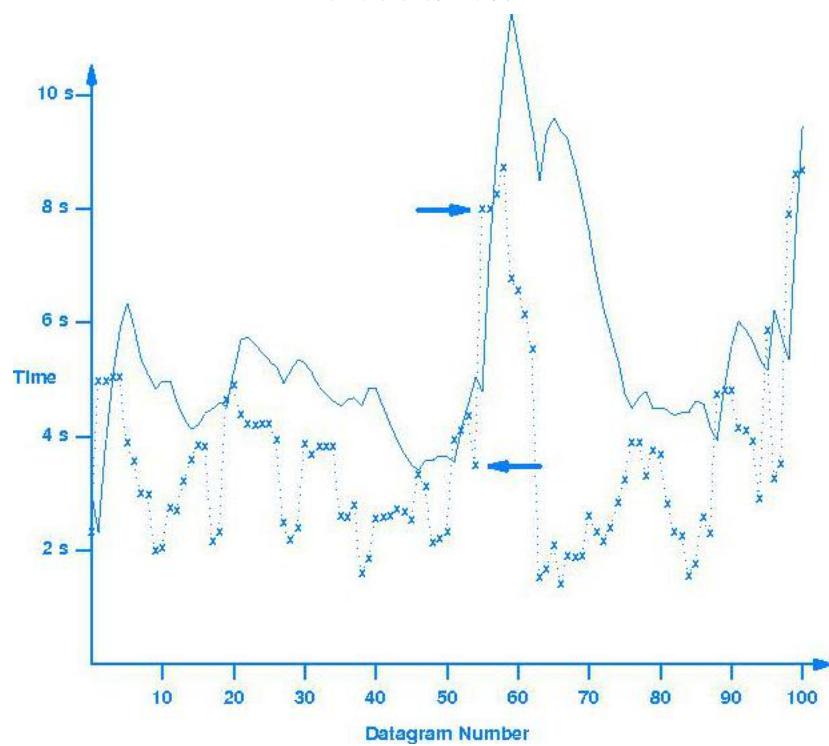


图 12.12 图 12.10 中数据对应的 TCP 重传计时器。箭头标记的是时延加倍的两个相邻数据报

12.19 对拥塞的响应

从表面上看，在考虑了连接的两个端点之间的交互以及通信时延以后，就可以设计 TCP 软件了。但实际上，TCP 还必须应付互联网中的**拥塞**（congestion）现象。拥塞是指一个或多个交换点（如路由器）的数据报超载而导致时延剧烈增加的现象。当发生拥塞时，数据报的时延增加，路由器开始把数据报放在队列中排队，直到能够对它们进行转发。值得注意的是，每个路由器的存储能力有限，其中的存储空间由多个数据报来竞争（也就是说，在一个基于数据报的互联网中，各个 TCP 连接没有预先分配的资源）。在最坏的情况下，到达拥塞路由器的数据报总数不断增加，直至路由器容量达到饱和，并开始丢弃报文。

通常，端点并不知道拥塞发生在哪，也不知道为什么会发生拥塞。对于端点而言，拥塞只是表现为时延的增加。遗憾的是，大多数运输协议使用超时和重传机制，它们对时延增加的响应就是重传数据报。重传只会加剧拥塞，而不会减轻拥塞。如果不加抑制，通信量的增加将导致时延的增加，时延增加又会进一步加大通信量，如此恶性循环，直至网络变得无法使用。这种现象称为**拥塞崩溃**（congestion collapse）。

为了避免出现拥塞崩溃，TCP 必须在拥塞发生时减小传输速率。路由器观察队列的长度，使用类似于 ICMP 源站抑制（source quench）的技术通知主机已发生拥塞^①。类似 TCP 的运输协议，在时延较大时可以自动降低传输速率，这样有助于避免拥塞的发生。当然，我们必须精心地构建避免拥塞的算法，因为即使在正常的操作环境中，互联网的往返时延仍会出现大范围的变化。

为了避免拥塞，现在的 TCP 标准推荐使用两种技术：**慢开始**（slow-start）和**乘法减小**（multiplicative decrease）。这两种技术是有关联的，也容易实现。我们知道，TCP 必须记住每个连接的接收方的窗口大小（即在确认中通告的可用缓冲区大小）。为了控制拥塞，TCP 使用第二个窗口限制，即**拥塞窗口限制**（congestion window limit）或称**拥塞窗口**（congestion window）。当发生拥塞时，可使用此限制让数据流量小于接收方的缓冲区大小。也就是说，在任何时候，TCP 通信把窗口的大小当成：

$$\text{Allowed_window} = \min(\text{receiver_advertisement}, \text{congestion_window})$$

在没有拥塞的连接上，稳定状态下的拥塞窗口与接收方的窗口一样大。缩小拥塞窗口，就会减少 TCP 注入连接的通信量。为了估算拥塞窗口的大小，TCP 假设报文的丢失大多数是由于拥塞造成的，并使用下面的策略来改变窗口大小：

乘法减小的拥塞避免策略：一旦发现报文段丢失，就把拥塞窗口的大小减半（直到减至最小的窗口，窗口中应包括至少一个报文段）。对于留在许可窗口内的那些报文段，采用指数退避算法计算重传计时器的时限值。

由于 TCP 每次发现报文段丢失时就将拥塞窗口的大小减半，如果继续丢失报文段，拥塞窗口的大小将呈指数级递减。换句话说，如果可能发生拥塞，TCP 将按指数级递减通信量和重传速率。如果继续发生丢失，最终 TCP 将限制每次只能传输一个数据报，同时在每次重传时继续把超时时限值加倍。这个策略的意图是迅速地大量减少通信量，从而让路由器有足够的时间来清除其队列中已有的数据报。

拥塞结束后，TCP 如何恢复正常通信呢？你可能猜测 TCP 会把加倍减小策略反过来，在通信量重新流通后把拥塞窗口的大小加倍。然而这样做会造成系统状态不稳定，系统会在没有通信量和拥塞的状态之间剧烈振荡。实际上，TCP 使用**慢开始**（slow-start）^②的技术来逐步恢复传输：

^① 在发生拥塞的网络中，队列的长度一度会出现指数型增长。

^② 术语“慢开始”是 John Nagle 提出的，这个技术以前称为“软开始”（soft-start）。

慢开始（加法）恢复：在一条新连接上开始传输通信量时，或在一段时间的拥塞后增加通信量时，拥塞窗口的大小刚开始只有一个报文段，以后随着一个确认的到达，拥塞窗口的大小每次增加一个报文段。

下层的互联网在清除拥塞之后，或突然开始新连接时，采用慢开始技术可以使它不会立刻陷入新增通信量的沼泽中。

“慢开始”这个名字容易让人产生误解，因为在理想的情况下开始并不慢。TCP 把拥塞窗口的大小初始化为 1，发送一个初始的报文段之后，等待确认。当确认到达时，拥塞窗口的大小增加到 2，发送两个报文段之后又继续等待。这两个报文段的确认到达时，每个确认让窗口大小增加 1，于是拥塞窗口的大小就增加到 4，从而可以连续发送 4 个报文段。如果收到这四个报文段的确认，拥塞窗口的大小就加到 8 了。在经过四个往返时间之后，TCP 就可以连续发送 16 个报文段，通常这个值足以成为接收方窗口的最大值。即便接收方有更大的窗口，TCP 仅需经过 $\log_2 N$ 个往返时间就可以连续发送 N 个报文段。

为了避免窗口大小增长过快以至于造成更严重的拥塞，TCP 还额外附加了一个限制。当拥塞窗口到达拥塞发生前窗口大小的一半时，TCP 进入一个**拥塞避免**（congestion avoidance）的阶段，减慢窗口增长的速度。在拥塞避免期间，只有当窗口中的所有报文段都被确认之后，窗口大小才能增加 1。这两种技术合起来称为**加法增加乘法减小**（Additive Increase Multiplicative Decrease，简称 AIMD）。

把慢开始、乘法减小、加法增加、方差估计、计时器指数退避等技术结合在一起，就能显著地提高 TCP 的性能，同时不需要为协议软件增加大量的计算开销。使用了这些技术的 TCP 版本，比过去一些版本的性能提高了 2~10 倍。

12.20 快恢复和其他改进

早期版本的 TCP 有时称为 Tahoe，使用上面介绍的重传机制，在开始重传前必须等待计时器出现超时。1990 年出现了 Reno 版本的 TCP，它对旧版本做了几处改进，包括一个名为**快恢复**（fast recovery）或**快重传**（fast retransmit）的启发式方法，在偶尔发生丢失的环境中具有更高的吞吐率。

在快恢复中使用的技巧产生于 TCP 的累积确认机制：如果一个报文段丢失，当后续的报文段到达接收方时，要求接收方产生一个 ACK，该确认指明了丢失的报文段在流中的位置点。从发送方的角度来看，它会收到一连串的确认，每个确认中携带的是同一个序号。快重传启发式方法使用一连串的三个重复确认（即最初的确认加上三个完全相同的副本）来激活一次重传，而不必等到计时器超时后再重传。

在只有一个报文段丢失的情况下，等待重传的报文段被确认也可能令吞吐率减小。因此，为了维持更高的吞吐率，快重传启发式方法在等待重传报文段的确认时，还继续发送窗口中的数据。此外，拥塞窗口被人为地膨胀开：拥塞窗口因重传而被缩小一半，但伴随每个重复 ACK 在重传前后的到达，都会使拥塞窗口扩大一个最大长度的报文段。由此可见，在快重传期间，TCP 让很多报文段处于发送方和接收方之间的传输途中。

一个较新版本的 TCP 对快重传启发式方法做了进一步优化，该版本的名称是 NewReno。它针对同一个窗口中有两个报文段丢失的情况进行了优化。实质上，当发生快重传时，NewReno 记录了当前窗口的有关信息，并按照上述过程进行重传。对于重传的报文段，当 ACK 到达时，有两种可能的情况：ACK 指明的序号是窗口末端处的序号（在这种情况下重传的报文段是窗口中唯一丢失的报文段），或者 ACK 指明的序号比丢失报文段的位置高，但又比窗口末端处的序号小（在这种情况下窗口中还有第二个报文丢失了）。在后一种情况下，NewReno 继续重传第二个丢失的报文段。

对 AIMD 方案也有一些小的改进，用于较后期版本的 TCP 中。为有助于理解，可以设想 AIMD

如何^①改变发送方的拥塞窗口，对报文段丢失或确认到达做出响应：

$$w \leftarrow w - aw \text{ 当检测到丢失时}^{\circledast}$$

$$w \leftarrow w + b/w \text{ 当 ACK 到达时}$$

在原先的方案中， $a = 0.5$, $b = 1$ 。在考虑诸如 STCP 的协议时，研究人员建议把以设置为 0.125，把 b 设置为 0.01，防止拥塞窗口出现振荡，还可以稍微提高吞吐率。其他改进提议（如 HSTCP 协议）建议把 a 和 b 处理成 w 的函数，即 $a(w)$ 和 $b(w)$ 。最后，Vegas 和 FAST 之类的协议提出了 TCP 拥塞控制的改进方案，不再使用分组丢失率来度量拥塞，而是使用增长的 RTT 度量拥塞的程度，并把拥塞窗口的大小定义为估量 RTT 的一个函数。通常，这些改进只有在一些特殊情况下才能提高性能（如具有高带宽和低丢弃率的网络）；在其他情况下使用原先的（Reno）AIMD 拥塞控制。

关于 TCP 拥塞控制的最后一个提案与 UDP 有关。我们发现，虽然在发生拥塞时 TCP 减少了传输通信量，但 UDP 并没有减少，这意味着随着 TCP 流量继续回退，UDP 流量将占据更多的宽带。针对此问题提出了一种解决方案，名为 **TCP 友好速率控制**（TCP Friendly Rate Control，简称 TFRC）。由于 UDP 不使用滑动窗口，TFRC 让接收方向发送方报告分组丢失的情况，并让发送方使用被告知丢失的数据来计算 UDP 应该使用的发送速率，通过这些措施来尝试模仿 TCP 的行为。

12.21 显式反馈机制：SACK 和 ECN

大多数版本的 TCP 使用**隐式**（implicit）技术来检测丢失和拥塞。也就是说，TCP 使用超时和复制 ACK 来检测丢失，并通过改变往返时间来检测拥塞。研究人员注意到，如果 TCP 包括一些显式地提供这种信息的机制，就有可能做一些改进。下面两小节介绍了两种已有的显式技术。

12.21.1 选择确认

TCP 的累积确认机制有一种替代方案，称为**选择确认**（Selective ACKnowledgement，简称 SACK）机制。实质上，选择确认允许接收方明确指出已收到哪些数据，还缺少哪些数据。选择确认的主要优点在偶尔出现丢失的环境中才能体现出来：通过选择确认，可以使发送方清楚地知道要重传哪个报文段。

为 TCP 提出的选择确认机制没有完全取代累积确认机制，这样做也没有必要。事实上，TCP 包括两个 SACK 选项。第一个选项在连接建立时使用，它允许发送方来指定允许 SACK。第二个选项由接收方使用，在每个确认中包括已收到的特定数据块的有关信息。每个数据块的信息包括块中的第一个序号〔称为**左边界**（left edge）〕和块后的第一个序号〔称为**右边界**（right edge）〕。由于报文段首部的最大长度是固定的，确认可以包含至多 4 个 SACK 块。有意思的是，SACK 文档没有明确规定发送方如何响应 SACK；大多数实现重传所有丢失的块。

12.21.2 显式拥塞通知

为了处理网络中的拥塞，提出了第二种避免隐式测量的技术。这种称为**显式拥塞通知**（Explicit Congestion Notification，简称 ECN）的机制，需要遍布互联网中的路由器在出现拥塞时通知 TCP。此机制的概念很简单：当 TCP 报文段通过互联网传递时，路径上的路由器使用 IP 首部中的一对比特位来记录拥塞。这样，当报文段到达后，接收方知道报文段是否在某个位置点经历过拥塞。遗憾的是，需要了解拥塞发生情况的是发送方，而非接收方。因此，接收方使用下一个 ACK 通知发送方有拥塞发生。然后，发送方做出响应，把自己的拥塞窗口缩小。

ECN 使用 IP 首部中的两个比特位，以便让路由器记录拥塞，并使用 TCP 首部中的两个比特位

^① 参见 12.19 节。

^② 原书没有说明符号 w 的意思。根据上下文应当是表示拥塞窗口大小——译者注。

(取自保留区)，以便让发送方和接收方的 TCP 层相互通信。接收方使用其中一个 TCP 首部的比特位将拥塞信息送回发送方；发送方使用另一个比特位将自己收到拥塞通知的信息通知接收方。IP 首部中的两个比特位是从服务类型字段中取出来的^①。路由器可以选择设置其中任何一个比特，表示拥塞已经发生（使用两个比特是为了让本机制更稳健）。

12.22 拥塞、尾部丢弃和 TCP

我们讲过，通信协议划分成多个层，从而使设计人员能够每次只需要集中思考一个问题。把功能分散到各层中既非常必要，又十分有用，这意味着改变某一层的功能，不会对其他层造成影响，但这也意味着各层的操作是相互隔离的。例如，因为 TCP 的操作是端到端的操作，所以当两个端点之间的路径发生变化时（例如，路由改变或添加了新的网络路由器），TCP 保持不变。但是，各层的隔离限制了层间通信。例如，尽管最初源站上的 TCP 能与最终目的站上的 TCP 交互，但它不能与路径上的更低层元素交互。因此，发送端和接收端的 TCP 都不会接收到网络中有关情况的报告，任何一端在传输数据前都不会通知路径上的更低层。

研究人员已经发现，如果缺乏层间通信，在某一层上选择不同的策略或实现，可能会对更高层的性能有很大影响。对 TCP 而言，路由器用于处理数据报的策略既可能对单个 TCP 连接有影响，也可能对所有连接的总吞吐率有影响。例如，如果一个路由器造成一些数据报的时延比其他数据报更长^②，TCP 就会对重传计时器进行退避处理。如果时延超过了重传时限，TCP 就会以为有拥塞发生。这样，尽管每一层是独立定义的，但研究人员会尽力设计一些能与其他层的协议良好配合运行的机制和实现。

当路由器超载并丢弃数据报时，IP 实现策略和 TCP 之间会发生交互，这也是两者之间最重要的交互。因为路由器把每个传入的数据报先放入内存中的队列，数据报在队列里等待接受处理，所以这个策略的重点在于队列管理。当数据报到达的速度高于被转发的速度时，队列会增长；当数据报到达的速度低于被转发的速度时，队列会收缩。但是，由于内存是有限的，队列不能无限地增长。早期的路由器使用**尾部丢弃**（tail-drop）的策略来管理队列溢出问题：

路由器的尾部丢弃策略：如果数据报到达时输入队列已满，则丢弃该数据报。

“尾部丢弃”这个名字与该策略对到达数据报序列的影响有关。一旦队列装满了，路由器就开始丢弃额外的所有数据报。也就是说，路由器丢弃了序列的“尾部”。

尾部丢弃对 TCP 的影响很有意思。在简单的情况下，数据报携带单个 TCP 连接的报文段通过路由器，报文丢失使 TCP 进入慢开始阶段，它会减少吞吐率，直到 TCP 开始收到 ACK 报文并扩大了拥塞窗口。但是，当数据报携带许多 TCP 连接的报文段通过路由器时，由于尾部丢弃可能导致全局的同步，所以还可能发生更严重的问题。为了搞清楚这个问题，我们考虑数据报复用的一种典型情况：相继到达目的站的数据报分别来自不同的源站。因此，尾部丢弃策略很可能使路由器丢弃来自 N 个连接的 N 个报文段（每个连接丢失一个报文段），而不是来自一个连接的 N 个报文段。这种同时发生的丢失造成 TCP 的 N 个实例同时进入慢开始^③。

12.23 随机早期检测

路由器如何避免全局的同步呢？答案是一种尽可能避免尾部丢弃的方法。这种方法称为**随机早**

^① 为了让 ECN 与 DiffServ 兼容，ECN 使用了 DiffServ 没有用到的两个比特位。

^② 从技术上讲，时延中的方差称为抖动（jitter）。

^③ 有意思的是，如果共享一条链路的 TCP 连接数足够大（大于 500）且各自的 RTT 不同，就不会发生全局的同步。

期检测(Random Early Detection),或称**随机早期丢弃**(Random Early Discard 或 Random Early Drop)。它的简称 RED 最常用。实现 RED 的路由器使用两个阈值标记队列中的位置: T_{min} 和 T_{max} 。RED 的一般操作可以用三条规则来描述,这些规则决定了每个到达数据报的处理方式:

- 如果队列当前包含的数据报少于 T_{min} 个,则把新数据报添加到队列中。
- 如果队列当前包含的数据报多于 T_{max} 个,则丢弃新数据报。
- 如果队列当前包含的数据报在 T_{min} 和 T_{max} 之间,则随机地按照概率 p 丢弃新数据报。

RED 的随机性表明,路由器不会一直等到队列溢出后才驱使许多 TCP 连接进入慢开始,而是随着拥塞的增长而缓慢地随机丢弃数据报。总结如下:

路由器的 RED 策略: 当数据报到达时,如果输入队列已满,则丢弃这个数据报;如果输入队列未满,但大小超过了一个最小阈值,则按概率 p 丢弃数据报来避免同步。

RED 成功运作的关键在于合理地选择 T_{min} 和 T_{max} 阈值以及丢弃概率 p 。 T_{min} 必须足够大,以确保输出链路有较高的利用率。此外,由于在队列长度超过 T_{max} 时 RED 会进行类似尾部丢弃的操作,所以 T_{min} 与 T_{max} 之差必须大于在一个 TCP 往返时间内典型增加的队列长度(例如, T_{max} 至少设置为 T_{min} 的两倍)。否则,RED 会像尾部丢弃那样造成全局的同步。

丢弃概率 p 的计算是 RED 最复杂的部分。 p 的新值不使用常量,而是为每个数据报计算出一个新值,这个值取决于当前队列长度和阈值之间的关系。为了理解这一方法,我们可以把所有 RED 处理当成概率问题来对待。当队列长度小于 T_{min} 时,RED 不丢弃任何数据报,这样就使丢弃概率为 0。类似地,当队列长度大于 T_{max} 时,RED 丢弃所有的数据报,这样就使丢弃概率为 1。如果队列长度为 T_{min} 和 T_{max} 之间的值,丢弃概率应该在 0 和 1 之间呈线性变化。

尽管线性方案是 RED 概率计算的基础,但必须对它加以修改,以避免反应过度。由于网络通信量是突发性的,这可能造成路由器队列的长度剧烈波动,从而需要对线性方案进行修改。如果 RED 使用了一种简化的线性方法,在每次突发通信中,后到的一些数据报被丢弃的概率会很高(因为它们在到达时队列已经很长了)。但是,路由器不应该轻易地丢弃数据报,这样做会对 TCP 吞吐率有负面影响。所以,如果通信量突发时间短,丢弃数据报是不明智的,因为队列根本不会溢出。当然,RED 不能无限期地推迟丢弃,因为长期的突发通信量会使队列溢出,从而导致有可能造成全局同步问题的尾部丢弃。

RED 在队列满时怎样分配更高的丢弃概率,而不会丢弃每次突发产生的数据报呢?答案是一种从 TCP 借用来的技术:RED 没有使用实际的队列长度,而是计算一个加权平均的队列长度 avg ,然后使用这个平均长度来决定概率值。 avg 的值是指数加权平均,每当有数据报到达时,按照下面的公式来更新这个值:

$$avg = (1 - \gamma) * Old_avg + \gamma * Current_queue_size$$

其中, γ 表示 0~1 之间的值。如果 γ 足够小,平均值才会反映队列长度的长期变化动向,而不受短期突发通信量的影响^①。

除了决定 γ 值的公式,RED 还包含其他一些我们未曾注意的细节。例如,把常量选择为 2 的幂并使用整数运算,可以让 RED 的计算效率特别高。另一个重要细节与队列长度的度量有关,它既影响 RED 计算,也影响它对 TCP 的整体影响效果。特别地,由于转发数据报所需的时间与数据报长度成正比,用八位组来度量队列的长度比用数据报度量更有意义;这样做只需要将 p 和 γ 的公式做微小改动。用八位组度量队列长度对丢弃的通信量类型有影响,因为丢弃概率与发送方注入流中的数据量成正比,而不是与报文段数量成正比。与较长的数据报(例如,携带文件传送的通信量)相比,较短的数据报(例如,携带远程登录或服务器请求的通信量)被丢弃的概率较低。使用长度有一个好处,当确认报文通过一条拥塞的路径时,被丢弃的概率较低。因此,如果一个大数据段到达了接收方,发送方的 TCP 将收到 ACK 确认,从而避免了不必要的重传。

^① γ 的一个建议值为 0.002。

理论分析和模拟实验表明，使用 RED 的效果很好。它能够处理拥塞，避免尾部丢弃造成的同步，并使短期突发的通信量不会造成轻易丢弃数据报。IETF 现在推荐路由器实现 RED。

12.24 建立一个 TCP 连接

TCP 使用**三次握手**(three-way handshake)来建立连接。在最简单的情况下，握手过程如图 12.13 所示。

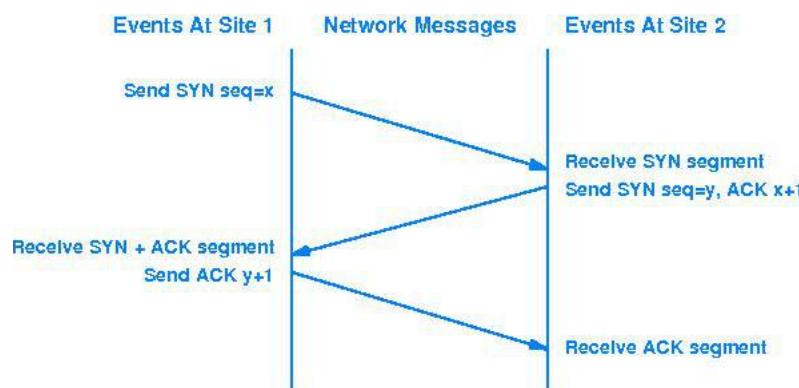


图 12.13 三次握手的报文序列。自上向下表示时间的推进，斜线表示网点之间报文段的传输。SYN 报文段携带了初始的序号信息

通过设置代码字段的 SYN^①位为 1 可以识别出握手的第一个报文段。第二个报文把 SYN 和 ACK 位均置为 1，表示这是对第一个 SYN 报文段的确认，并继续握手操作。最后一个握手报文仅仅是一个确认信息，通知目的主机双方都同意：已建立这个连接。

通常，在一台机器上运行的 TCP 软件被动地等待握手，而另一台机器上的 TCP 软件则主动地发起连接。但是，握手协议经过了精心设计，即使双方试图同时建立连接，协议也能正常地工作，因此连接可以由任一方或双方同时建立。一旦建立了连接，数据就可以双向对等地流动，而没有所谓的主从关系。

三次握手协议是连接的两端正确同步的充要条件。这是因为 TCP 建立在不可靠的分组交付服务之上，报文可能出现丢失、延迟、重复和无序交付的情况，因而协议必须使用超时机制，重传丢失的连接建立请求。如果重传和原先的请求在正建立连接时到达，或者当一个连接已经建立、正在使用和断开之后，被延迟的重传请求才到达，都会出现麻烦。三次握手协议只要添加一条规则，在已经建立连接后，TCP 忽略其他连接请求，就能解决这些问题。

12.25 初始序号

三次握手完成了两个重要的功能。它可以确保连接的双方都已做好数据传送的准备（而且彼此知道对方已准备好），而且使双方能就初始序号达成一致。在握手期间发送序号并获得确认。每台机器必须随机选择一个初始序号，用该序号来标志自己所发送的字节在流中的位置。不能总从同一个值开始选择序号。特别地，TCP 不能在每次创建连接时都只选择序号 1（否则就会出现本章的一道习题中谈及的问题）。当然，双方必须就初始序号达成一致，这一点很重要，因此确认中使用的

^① SYN 代表同步 (synchronization)，发音为 “sin”。

八位组编号与数据报文段中使用的八位组编号应该一致。

每个报文段包括了序号字段和确认字段，这就使两台机器仅用三个握手报文即可将两个方向的流序号协商好。发起握手的机器 A 把自己的初始序号 x 放到三次握手协议的第一个 SYN 报文段中，传到机器 B。机器 B 收到 SYN 后，记下这个序号，再把自己的初始序号 y 放到序号字段中，同时在确认字段中表明 B 等待接收第 $x+1$ 号八位组，然后把这个报文段发给 A。在最后一个握手报文中，A 知道了要从第 y 号八位组起接收 B 的流。在各种情况下，确认都约定使用期望收到的下一个八位组的编号。

前面讲过，TCP 通常通过交换包含最少量信息的报文段来实现三次握手。从协议的设计来讲，我们也可以把数据连同初始序号放到握手的报文段中传输。在这种情况下，TCP 软件在握手完成前都必须保留这些数据。在连接建立之后，TCP 软件可以把保留的这些数据释放出来，并迅速交付给等待数据的应用程序。详情参见该协议的规范。

12.26 关闭一个 TCP 连接

使用 TCP 通信的两个程序可以使用**关闭**（close）操作从容地结束会话。TCP 协议内部使用改进的三次握手来关闭连接。TCP 连接是全双工的，可以看成是两个独立的不同方向的流传送。当一个应用程序通知 TCP 自己再没有数据要发送时，TCP 将关闭**一个方向**（in one direction）的连接。发送方的 TCP 为了关闭它这一方的连接，把剩余的数据发送完后等待接收方对数据的确认，然后再发送一个代码字段的 FIN 位被置为 1 的报文段。接收方的 TCP 对 FIN 报文段进行确认，并通知本端的应用程序再也没有可用数据了（例如使用操作系统的文件结束符机制）。

一旦在某一方向上的连接已关闭，TCP 就拒绝接受来自该方向的更多数据。这时，在相反的方向上，数据还可以继续流动，直到发送方关闭连接。当然，尽管连接已经关闭，确认还是会继续反馈给发送方。当连接的两个方向都已关闭后，该连接的两个端点的 TCP 软件就会删除这个连接的记录。

由于 TCP 使用改进的三次握手协议来关闭连接，关闭连接的细节比上面所述的过程更复杂。图 12.14 描述了这个过程。

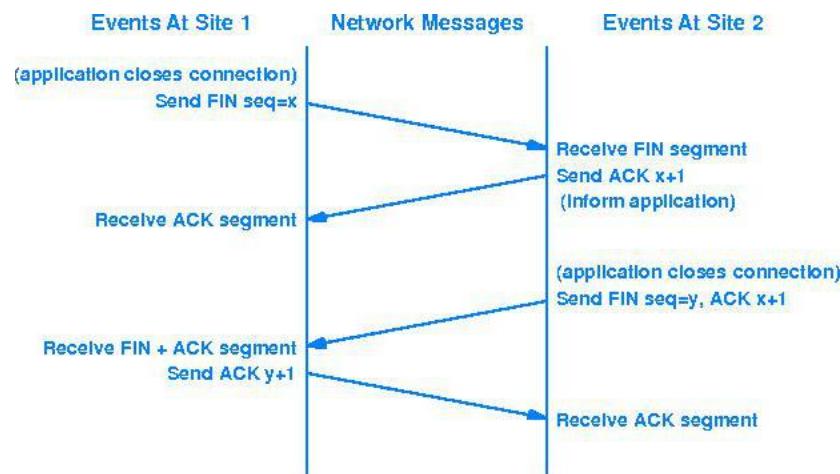


图 12.14 用于关闭连接的改进三次握手。接收到第一个 FIN 报文段的网点立即对其进行确认，并会在发送第二个 FIN 报文段之前延误一段时间（通知应用程序）

用于建立连接的三次握手和用于断开连接的三次握手，两者间的不同之处发生在机器收到第一个 FIN 报文段以后。TCP 不是立即发送第二个 FIN 报文段，而是先发送一个确认，然后通知相应的应用程序关闭连接。通知应用程序并获得响应可能会花费大量的时间（例如，可能涉及人机交互操作）。先发送确认是为了防止对方重传第一个 FIN 报文段。最后，当应用程序指示 TCP 软件彻底

关闭这个连接时，TCP 软件发送第二个 FIN 报文段，而原来发起关闭的网点回送第三个报文，即 ACK。

12.27 TCP 连接的复位

在正常情况下，应用程序使用完连接之后，使用**关闭**（close）操作来结束一个连接。因而，关闭连接的操作类似于关闭文件操作，可以看成是正常使用过程的一部分。有时会出现异常情况，迫使应用程序或网络软件中断这个连接。TCP 为这种异常的断开连接操作提供了复位措施。

要将连接复位时，发起端送出一个报文段，其代码字段的 RST 位被置为 1。另一端对复位报文段的反应是立即中断连接。TCP 还要通知应用程序出现了连接复位。复位就是即刻中止连接，即连接双方立即停止传送，并立即释放缓冲区之类的资源。

12.28 TCP 状态机

与大多数协议类似，使用一个称为**有限状态机**（finite state machine）的理论模型，可以更好地解释 TCP 协议的操作。图 12.15 显示了 TCP 的有限状态机，图中的圆圈表示状态，箭头表示状态之间的转换。每个状态转换箭头上的标记表示 TCP 收到哪种信息才进行转换，以及在响应中发送什么信息。例如，每个端点的 TCP 软件都是从 CLOSED（关闭）状态开始的。应用程序必须发出**被动打开**（passive open）指令（等待其他机器来建立连接），或者发出**主动打开**（active open）指令（发起连接）。主动打开指令使状态从 CLOSED 状态转换成 SYN SENT（同步发送）状态。如果 TCP 出现这样的状态转换，就发出一个 SYN 报文段。当另一端返回一个包括 SYN 和 ACK 的报文段后，TCP 转换到 ESTABLISHED（连接建立）状态，开始数据传送。^{anything / reset}

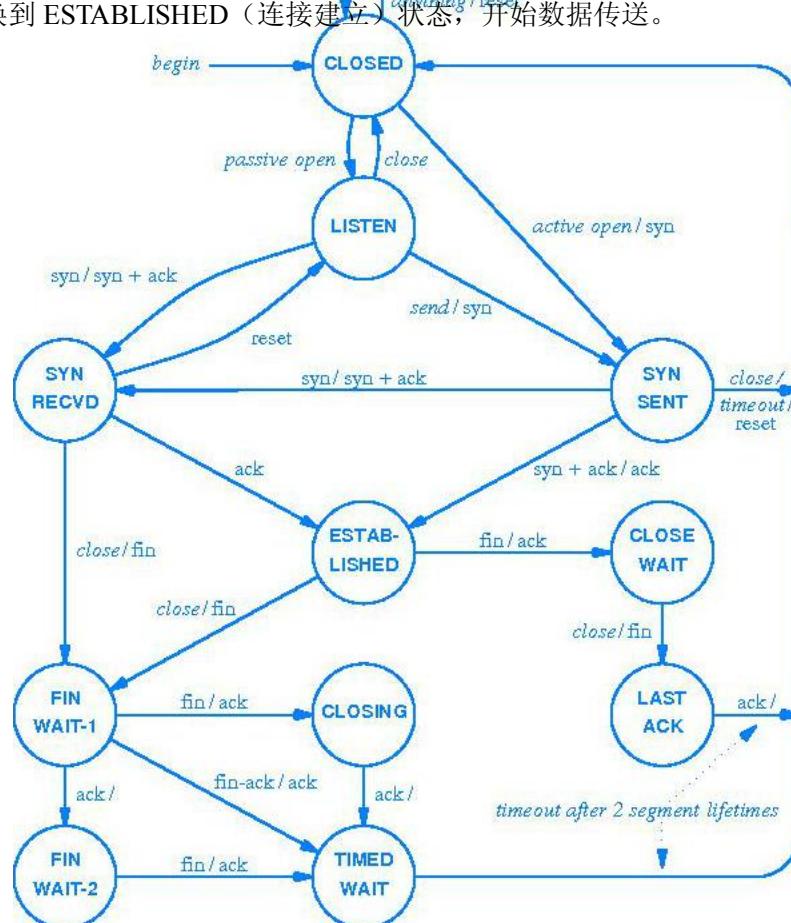


图 12.15 TCP 有限状态机。各端均从关闭状态开始。每个转换都标出了导致该转换的输入和该转换的输出（如果有输出时）

从 TIMED WAIT（超时等待）状态可以看出 TCP 如何处理不可靠交付导致的问题。TCP 有一个**最大报文段生存时间**（maximum segment lifetime，简称 MSL）的概念，表示一个旧的报文段在互联网中可能存活的最长时间。为了避免前面连接的报文段与当前连接的报文段混淆起来，TCP 在关闭一个连接之后就进入 TIMED WAIT 状态。在删除该连接的记录之前，它在该状态停留的时间长达最大报文段生存时间的两倍。如果在这个时限之内有该连接的重复报文段到达，TCP 就会拒绝接收它。然而，为了处理最后一个确认丢失的情况，TCP 对有效的报文段进行确认并重新设定计时器。由于这个计时器使 TCP 能够区分新旧连接，如果在计时器时限内另一端重传了一个连接的 FIN 请求，就可以防止 TCP 用一个 RST（复位）报文段做出响应。

12.29 强迫数据交付

前面讲过，TCP 能够自由地把流划分成若干报文段进行传输，而不受应用程序使用的传送单元大小的影响。允许 TCP 自由进行划分的主要优点是效率高。它可以在缓冲区中积累足够多的八位组，构造出长度适中的报文段，这样就降低了由于报文段仅携带少量数据而带来的高开销。

虽然缓冲技术提高了网络的吞吐率，但它可能妨碍一些应用程序。考虑使用 TCP 连接把字符从交互终端传到远端主机的情况。此时，用户希望每次敲键都能立即得到响应。如果发送方的 TCP 软件把数据放入了缓冲区，那么对这次敲键的响应可能被延迟到几百次敲键之后。类似地，由于接收方的 TCP 软件把数据提交给应用程序使用前，先要对数据进行缓冲，因此迫使发送方立即传输数据可能仍无法保证数据立即交付给接收方。

为了适应交互式用户的需求，TCP 提供了**推**（push）操作，应用程序可以使用这个操作强迫 TCP 递交当前在流中已有的八位组，而不必等到填满缓冲区。推操作除了强迫 TCP 立即发送一个报文段外，还要求 TCP 把报文段代码字段的 PSH 位置为 1，从而使数据能立即交付给接收端的应用程序。因此，从一个交互终端发送数据时，应用程序使用推操作来处理每一次敲键过程。类似地，在书写一个字符或一行字符之后，应用程序通过调用推操作，强迫输出立即被发送并显示在终端上。

12.30 保留的 TCP 端口号

与 UDP 类似，TCP 将静态和动态的端口绑定技术结合起来，它把一组**熟知端口**（well-known port）留给一些常用的程序（例如电子邮件）使用，同时又给操作系统留出了很多端口号，在应用程序需要时由操作系统来进行分配。许多熟知端口现在已经在使用。虽然从前分配的熟知端口号都比较小，但是现在已经给一些熟知端口分配了超过 1024 的端口号（在 UNIX 系统中，前 1023 个端口被当成只有特权程序才可以使用的特殊端口）。图 12.16 列出了一些当前已分配的 TCP 端口。需要指出的是，虽然 TCP 和 UDP 的端口号是独立的，但对于从 TCP 和 UDP 都能访问的服务，设计人员倾向于给它们指定同一个整数端口号。例如域名服务既可用 TCP，也可用 UDP 访问。在这两个协议中，53 号端口都被留给域名系统中的服务器使用。

Decimal	Keyword	Description
0	ECHO	Reserved
7	ECHO	Echo
9	DISCARD	Discard
11	USERS	Active Users
13	DAYTIME	Daytime
15	netstat	Network Status program
17	QUOTE	Quote of the day
19	CHARGEN	Character Generator
20	FTP-DATA	File Transfer Protocol (data)
21	FTP	File Transfer Protocol
22	SSH	Secure Shell
23	TELNET	Terminal connection
25	SMTP	Simple Mail Transport Protocol
37	TIME	Time
53	DOMAIN	Domain name server
67	BOOTPS	BOOTP or DHCP Server
79	FINGER	Finger
80	WWW	World Wide Web server
88	KERBEROS	Kerberos security service
110	POP3	Post Office Protocol vers. 3
111	SUNRPC	SUN Remote Procedure Call
119	NNTP	USENET News Transfer Protocol
123	NTP	Network Time Protocol
143	IMAP	Internet Message Access Protocol
161	SNMP	Simple Network Management Protocol
443	HTTPS	Secure HTTP
465	SMTPS	SMTP over SSL (TLS)
515	SPOOLER	LPR spooler
873	RSYNC	Rsync protocol
993	IMAPS	Secure IMAP
995	POP3S	Secure POP3
1080	SOCKS	Proxy server protocol

图 12.16 目前已分配的 TCP 端口号举例。TCP 协议尽可能地使用和 UDP 相同的端口号

12.31 TCP 的性能

TCP 是一个复杂的协议，它要处理基于多种不同的底层网络技术的通信。许多人认为，既然与其他运输层协议相比，TCP 能处理更复杂的任务，其代码必然是烦琐而低效的。但令人惊讶的是，我们讨论过的一般性问题并不妨碍 TCP 的性能。在伯克利曾经做过一个实验，在 10 Mbps 以太网上的两个工作站之间使用 TCP 来传输数据，这里用的 TCP 与全球因特网上高效运转的 TCP 是相同的，实验表明在该以太网上能达到稳定的 8 Mbps 用户数据吞吐率^①。在 Cray Research 公司，研究人员展示的 TCP 系统具有接近每秒吉比特的吞吐率。

12.32 糊涂窗口综合症与小分组

TCP 的开发人员发现，收发两端的应用程序以不同的速率工作时，可能产生严重的性能问题。我们知道，TCP 对进入的数据进行缓冲处理，让我们思考接收方应用程序每次仅读取一个八位组时可能发生什么情况。当第一次建立连接之后，接收方的 TCP 申请分配一个 K 字节的缓冲区，并在确认报文段中使用窗口（WINDOW）字段将可用缓冲区的大小通告给发送方。如果发送应用程序快速地产生数据，发送方 TCP 传输的报文段所携带的数据很快就会装满接收方的缓冲区。最终，发送方会收到确认信息，得知整个窗口已被填满，接收方的缓冲区已没有多余的空间。

当接收应用程序从饱和的缓冲区中读取 1 八位组的数据后，缓冲区就空出 1 八位组的空间。而当接收方的缓冲区中有可用空间时，接收机器上的 TCP 就会生成一个确认，并使用窗口字段将可用空间大小通知给发送方。在这个例子中，接收方会通告 1 八位组的窗口。发送方 TCP 得知有可用空间之后，会发送一个包含 1 八位组数据的报文段。

尽管仅有 1 八位组的窗口通告能够正常发挥作用，让接收方的缓冲区不断地被填满，但这样做会导致发送方产生一系列短的数据报文段。发送方 TCP 必须构造一个仅包含 1 八位组数据的报文

^① 以太网、IP 和 TCP 的首部以及分组之间的间隙占用了其余的带宽。

段，把报文段封装到 IP 数据报中，然后进行传输。当接收应用程序又读取一个八位组后，TCP 又生成一个确认，这样又使发送方传输一个仅包含 1 八位组数据的报文段。最终，发送方与接收方之间的这种交互可能进入一个稳定的状态，届时 TCP 为每一个八位组数据发送一个单独的报文段。

传输短的报文段浪费了网络带宽，带来了不必要的计算负载。由于每个报文段仅携带 1 八位组的数据，首部与数据的比率太大，从而造成了带宽的浪费。而由于收发双方的 TCP 都必须处理每个报文段，所以造成了计算负载的增加。发送方的 TCP 软件必须申请分配缓冲区空间，形成一个报文段首部，并为每个报文段计算检验和。类似地，发送机器上的 IP 软件必须把报文段封装到数据报中，计算首部检验和，转发数据报，并把它传送给相应的网络接口模块。在接收机器上，IP 软件必须检验 IP 首部检验和，并把报文段传递给 TCP 软件。TCP 必须检验报文段的检验和，检查序号，提取报文段中的数据，并把数据放入缓冲区中。

尽管在前面讲过接收方通告一个小的可用窗口时会产生怎样的报文段，但发送方也可以让每个报文段包含少量的数据。设想有一个 TCP 实现，只要有数据要发送，它就会尽快地发送数据，如果发送应用程序每次只生成 1 八位组的数据，想想看会发生什么情况。在应用程序产生 1 八位组数据后，TCP 要创建并传输一个报文段。如果应用程序每次产生固定大小为 B 八位组的数据块，而发送方的 TCP 每次从缓冲区中提取最大报文段长度为 M 的数据块，且 $M \neq B$ ，由于在缓冲区中提取的最后一块数据可能会很少，这时 TCP 也可能发送短的报文段。

上述问题称为**糊涂窗口综合症** (silly window syndrome，简称 SWS)，早期的 TCP 实现深受此问题的困扰。归纳如下：

早期的 TCP 实现存在糊涂窗口综合症的问题，即每个确认报文通告了少量的可用空间，而每个报文段仅仅携带少量的数据。

12.33 避免糊涂窗口综合症

现行的 TCP 规范用启发式策略来防止糊涂窗口综合症。在发送机器上使用一种启发式策略，可以避免在每个报文段中携带少量数据进行传输。在接收机器上使用另一种启发式策略，可以避免发送增量小的窗口通告，这种通告可能会造成小数据分组的产生。尽管这两种启发式策略合在一起使用很好，但让发送方和接收方都实现对糊涂窗口的避免，在某一方无法有效实现糊涂窗口避免的情况下，有助于确保 TCP 仍具有良好的性能。

在实际应用中，TCP 软件必须既包括发送方糊涂窗口避免的代码，也包括接收方糊涂窗口避免的代码。这是因为 TCP 的连接是一个全双工连接——数据可以在任一方向上流动。因此，TCP 的实现既要包括发送数据的代码，也要包括接收数据的代码。

12.33.1 接收方对糊涂窗口的避免

接收方用来避免糊涂窗口综合症的启发式策略简明易懂。通常，接收方保存了一份当前可用窗口的内部记录，并在窗口大小有显著提高之前，推迟发送那些窗口大小增量小的通告。此处所说的“显著”，其定义与接收方的缓冲区大小以及最大报文段长度有关。TCP 把“显著”的幅度定义为接收方缓冲区空间的一半，或者最长报文段中所含数据的八位组个数。

在接收应用程序低速提取数据八位组的情况下，接收方避免糊涂窗口的策略能够防止小窗口通告的发生。例如，当接收方的缓冲区完全装满后，会发送一个含有零窗口通告的确认。随着接收应用程序从缓冲区中提取八位组，接收方的 TCP 计算出缓冲区中新的可用空间，但并不立即发送窗口通告，而是等到可用空间达到缓冲区的一半或最长报文段所含数据量时，才发送新的窗口通告。因此，发送方总是收到当前窗口中空间大幅增加的通告，从而使它能够传送长的报文段。启发式策略可以概括如下：

接收方避免糊涂窗口的策略：接收方在通告一个零窗口之后，要等到缓冲区的可用空间至少达到总空间的一半，或等于一个最长报文段所含的数据量时，才发送更新后的窗口通告。

12.33.2 推迟确认

接收方可采用两种方法来实现糊涂窗口避免。在第一种方法中，TCP 对到达的每个报文段进行确认，但是要等到窗口空间达到糊涂窗口避免启发式策略所指定的限度之后，才让接收方发送窗口增大的通告。在第二种方法中，当糊涂窗口避免策略指出窗口不足以达到通告所需的大小时，TCP 将推迟确认的发送。TCP 标准推荐使用推迟确认的技术。

推迟确认 (delayed acknowledge) 的技术既有优点也有缺点。其主要优点是推迟确认能够减少通信量，从而提高吞吐率。如果在确认推迟期间有额外的数据到达，那么对收到的所有数据只需使用一个确认。而如果接收应用程序在数据到达之后立即产生响应（如远程登录会话中的回送字符），这么短的时延可以允许确认捎带在一个数据段中发送回来。此外，只有当接收应用程序从缓冲区中提取数据之后，TCP 才能移动自己的窗口。如果接收应用程序能够在数据到达之后尽快读取数据，这么短的时延允许 TCP 在确认数据的报文段中捎带一个更新后的窗口通告。如果没有推迟确认，TCP 就要在数据到达后立刻发送一个确认，随后再发送另一个确认，对窗口大小进行更新。

推迟确认技术的缺点也很明显。其主要缺点是当接收方把确认推迟时间太长了，发送方的 TCP 会重传报文段。这种不必要的重传浪费了网络带宽，从而降低了吞吐率。另外，重传还加重了收发双方的计算负载。此外，TCP 使用确认的到达时间来估计往返时间，推迟确认可能会干扰往返时间的估计，并使重传时间变得过长。

为了避免潜在的问题，TCP 标准规定了确认被推迟时间的限度。TCP 的实现对确认最多能推迟 500 ms。此外，为了使 TCP 能获得足够多的样本，以准确地估计往返时间，TCP 标准建议接收方至少应该每隔一个报文段进行一次确认。

12.33.3 发送方对糊涂窗口的避免

发送方用来避免糊涂窗口综合症的启发式策略非常巧妙，令人惊奇。前面讲过，它的目标是避免发送短的报文段，而发送应用程序可能产生任意小块的数据（如每次 1 八位组）。因此，为达到这个目标，发送方的 TCP 必须允许应用程序多次调用写 (write) 操作，还必须把每次调用中传送的数据收集起来，形成一个长的报文段进行传输。这就是说，发送方的 TCP 必须推迟报文段的发送，直到积累了适当数量的数据后再发送报文段。这种技术称为**组块** (clumping)。

那么 TCP 在传输数据之前应该等待多久呢？一方面，如果 TCP 等待太久，应用程序的时延就会过长。更重要的是，由于 TCP 不知道应用程序会不会在近期产生更多的数据，所以它无法决定是否该继续等待下去。另一方面，如果 TCP 等待的时间不够长，报文段就会较短，吞吐率就会较低。

在 TCP 之前设计的协议同样遇到过这样的问题，它们使用了组块技术把数据组装成更大的分组。例如，为了在网络上进行高效传输，早期的远程终端协议在每次敲键之后延迟几百毫秒，判断用户是否会继续敲键，然后才进行传输。但是，TCP 被设计成一个通用的协议，可以被各种应用程序调用。在 TCP 连接上传输的字符可能源于用户敲击键盘，也可能源于传送文件的程序。一个固定的时延值不可能适用于所有情况。

类似于 TCP 用于重传的算法和避免拥塞使用的慢开始算法，发送方 TCP 为避免发送短报文段而采用的技术也具有自适应能力，即时延值可以根据下层互联网的当前性能而定。类似于慢开始技术，发送端的糊涂窗口避免技术称为**自计时** (self clocking)，因为它并不计算时延，而是利用确认的到来来触发其余分组的传输。这个启发式方法可归纳如下：

发送方避免糊涂窗口的策略：在一个连接上已传输的数据还未被确认的情况下，发送应用

程序又产生了后续数据，并照常把数据放入输出缓冲区。但这时它并不发送后续报文段，而是等到数据足以填满一个最大长度的报文段之后，再把缓冲区中累积的数据发送出去。该策略适用于任何情况，包括推操作在内。

如果某个应用程序每次仅产生 1 八位组的数据，TCP 会立即发送最初的那个八位组，但是在确认到达之前，TCP 会把后续数据存入缓冲区。因此，当应用程序产生数据的速率比网络的速率快很多时（如传送一个文件），后续各个报文段将包括很多数据；而当应用程序产生数据的速率比网络速率更慢时（如用户敲键盘），不用经长时间延迟就可以发送较短的报文段。

这项技术根据其发明者的名字取名为 Nagle 算法，它只需少量的计算开销，设计得十分巧妙。主机不需要为每个连接设置独立的计时器，在应用程序生成数据时也不需要查看时钟。更为重要的是，这项技术能够适应不同的网络时延、最大报文段长度和应用程序速度的任意组合情况，而且在常规情况下不会降低网络的吞吐率。

为什么在常规情况下网络能保持高吞吐率呢？为提高吞吐率而优化过的应用程序不会每次只生成 1 八位组的数据（那样做会产生不必要的操作系统负载）。这种应用程序每次调用时都会产生大块的数据，因而 TCP 的发送缓冲区刚开始就有足够装满至少一个最大长度报文段的数据。另外，由于应用程序产生数据的速率比 TCP 传送数据的速率快，发送缓冲区几乎总是处于饱和状态，因而 TCP 不会推迟传输。这样，随着应用程序继续不断地填充缓冲区，TCP 就会持续地以下层互联网所能承受的最大速率来发送报文段。归纳如下：

现行的 TCP 要求收发双方实现避免糊涂窗口综合症的启发式方法。接收方要避免小窗口的通告，而发送方要使用自适应机制来推迟传输，以便将数据组块形成长的报文段。

12.34 小结

传输控制协议（TCP）定义了由互联网提供的一个关键服务，即可靠的流交付。TCP 提供两台机器之间的全双工连接，允许它们高效地交换大量数据。

TCP 使用滑动窗口协议来高效地使用网络。由于 TCP 几乎不给下层交付系统提要求，因此能够灵活地在各种交付系统上运行。由于 TCP 提供流量控制，所以能够使各种不同速率的系统相互通信。

报文段是 TCP 使用的基本传送单元。报文段用于递送数据或控制信息（如两台机器上的 TCP 软件建立或断开连接所用的控制信息）。报文段的格式允许一台机器在某个方向上流动的报文段首部中捎带对另一个方向上流动数据的确认信息。

TCP 通过让接收方通告其愿意接受的数据量来实现流量控制，还使用紧急数据措施和推操作来支持带外报文的传输。

当前的 TCP 标准说明了对重传计时器进行指数退避的算法和一些拥塞避免算法，如慢开始、乘法减小和加法增加。此外，TCP 使用启发式策略来避免传输短的报文段。最后，为避免 TCP 同步和提高吞吐率，IETF 建议路由器使用 RED 而不用尾部丢弃。

12.35 深入研究

TCP 的标准可在 Postel [RFC 793] 找到；Braden [RFC 1122] 更新了标准，澄清了几点问题。Clark [RFC 813] 描述了 TCP 的窗口管理，Postel [RFC 879] 讨论了 TCP 的最大报文段长度。Nagle [RFC 896] 论述了 TCP/IP 网络上的拥塞问题，并说明了发送端避免糊涂窗口所用的自计时技术的效果。Karn and Partridge [1987] 讨论了对往返时间的估计，提出了 Karn 算法。Jacobson [1988] 给出了拥塞控制算法，该算法现已成为标准的组成部分。Floyd and Jacobson [1993] 提出了 RED 方法，Clark and Fang

[1998]讨论了使用 RED 的一种分配框架。Tomlinson [1975]进一步研究了三次握手的细节。Mills [REC 889]给出了因特网往返时延的测量情况。Jain [1986]描述了滑动窗口环境中基于计时器的拥塞控制。Floyd et. al. [RFC 3782]介绍了对快恢复用到的 NewReno 算法的改进。Handly et. al. [RFC 3448]讨论了 TCP 友好速率控制 (TFRC)。Mathis et. al. [RFC 2018]定义了 SACK 选项，Ramakrishnan et. al. [RFC 3168]定义了 ECN 选项。

12.36 习题

1. TCP 使用有限长的字段来记录流序号。研究协议规范，找出在两台机器之间允许任意长度流传输的方法。
2. 本书指出，TCP 有一个选项允许接收方指明它愿意接受的最大报文段长度。在 TCP 已具有窗口通告机制的情况下，为什么还要支持这个选项来规定最大报文段长度？
3. 在什么样的时延、带宽、负载以及分组丢失率情况下，TCP 没有必要重传大量的数据？
4. 一个丢失的 TCP 确认并不一定会导致重传，试解释原因。
5. 用本地机器做实验，看看 TCP 如何处理机器重启动（复位）。建立一个连接（如远程登录），然后让其处于空闲状态。等到目的机器崩溃并重启动后，让本地主机发送一个 TCP 报文段（如在远程登录操作界面中键入字符）。
6. 设想有一种 TCP 实现会丢弃无序到达的报文段，即使这些分组位于接收窗口中。也就是说，此种 TCP 实现仅接受已收到字节流之后的报文段。这样的 TCP 实现能发挥作用吗？把它和标准的 TCP 实现进行比较。
7. 考虑检验和的计算。假设报文段中的检验和字段并未置零，而检验和计算结果却是零。你能从中得出什么结论？
8. 支持和反对“自动关闭空闲连接”各有什么理由？
9. 如果两个应用程序使用 TCP 来发送数据，但每次只发送含一个字符的报文段（例如，通过使用推操作），那么它们的数据最多有多大的网络带宽利用率？
10. 假定一个 TCP 实现在每次创建连接时使用 1 作为初始序号，那么当系统崩溃并重启动之后，会使远程系统误认为旧的连接现在依然是打开的。试解释这一现象。
11. 弄清楚 TCP 实现如何解决**重叠报文段**（overlapping segment）问题。这个问题是这样引起的：接收方只从数据流中接收所有字节的一份副本，哪怕发送方发送了有部分字节重叠的两个报文段（例如，前一报文段携带从 100 号到 200 号的字节，而后一报文段携带从 150 号到 250 号的字节）。
12. 两个网点分别执行主动打开和被动打开并逐步进行三次握手，跟踪这两个网点上 TCP 的有限状态机转换。
13. 阅读 TCP/IP 规范，找出在什么情况下 TCP 会从 FIN WAIT-1 状态转换到 TIMED WAIT 状态？
14. 两台机器都同意从容地关闭连接，试跟踪这两台机器上的 TCP 状态转换。
15. 假设 TCP 在一个无限带宽的通道上使用 64 KB 的最大窗口值发送报文段，报文段的平均往返时间为 20 ms，最大吞吐率是多少？如果平均往返时间增加到 40 ms（带宽不变），那么最大吞吐率又是多少？
16. 你能否推导出一个公式，把最大可能的 TCP 吞吐率表示成三个变量的函数，三个变量分别是网络带宽、网络时延、处理一个报文段并生成一个确认所需的时间。提示：考虑习题 15。
17. 描述一种可能让连接的一端无限期地停留在 FIN WAIT-2 状态的异常情况。提示：考虑数据报丢失和系统崩溃的情况。

-
18. 试说明，当路由器实现 RED 后，一个特定的 TCP 连接丢弃分组的可能性与该连接产生通信量的比例成正比。

第13章 路由选择：核心网络、对等网络与算法

13.1 引言

前面的章节着重介绍了 TCP/IP 提供的网络层服务，以及提供这些服务的主机和路由器中的协议细节。在讨论中，假设路由器总是包含了正确的路由，而且前面还讲过，路由器可以利用 ICMP 重定向机制通知与其直接相连的主机更改路由。

本章主要讨论两个问题：“每个路由表中应该包含什么数据值？”以及“如何获得这些数据值？”为回答前一个问题，要考虑互联网体系结构与路由选择之间的关系。特别地，将围绕主干网结构和对等主干网结构的互联网展开讨论，研究它们使用的路由选择体系结构。为回答后一个问题，将讨论两种基本类型的路由传播算法，了解它们如何自动提供路由信息。

本章首先讨论一般意义上的转发。后面几节将集中讨论互联网体系结构，并描述路由器交换路由信息所用的算法。第 14 章和第 15 章继续对路由选择做进一步讨论，探讨分属两个独立管理组的路由器用来交换信息的协议，以及在单个组内的所有路由器之间使用的协议。

13.2 路由表的产生

在第 3 章中讲过，IP 路由器能够提供网络间的互连。每个路由器与两个以上的物理网络相连，在这些网络之间转发数据报，它从自己的一个网络接口上接受到达的数据报，并通过另一个网络接口发送出去。除了那些目的站就在本机所在网络上的数据报，主机把所有 IP 通信量传递给路由器，路由器再把数据报朝着其目的站的方向转发出去。数据报在运送过程中将经过若干个路由器，直到最后到达一个直连到目的站所在网络的路由器。因此，路由器系统组成了互联网的结构基础，并可处理所有的通信量（主机之间的直接交付除外）。

第 7 章介绍了主机和路由器转发数据报所用的算法，还说明了算法如何使用一个表来做出转发决策。路由表中的每个表项都指明了一个目的地址的网络部分，并给出了从路由器到目的网络的路径上的下一个机器的地址。在实际应用中，每个表项还指明了到达下一跳所用的那个接口。

我们还没讲过主机和路由器如何获得路由表中的信息。这个问题涉及两个方面的内容：一是表中该放入什么数据值，二是如何获得这些数据值。答案与互联网体系结构的复杂性、网络规模以及管理策略有关。

一般来说，路由的建立包括路由的初始化和更新。每个路由器在启动时都必须建立一个初始的路由集合，还要随着路由的改变而更新该表（如某个特定网络中的硬件出故障时）。初始化过程与操作系统有关。在某些系统中，路由器启动时从辅存中（如硬盘）读取初始路由表，然后将其驻留在高速内存中。在另一些系统中，路由器刚开始只有一张空表，然后必须使用显式命令（例如，启动命令脚本中包含的命令）填入表项值。最后，还有一些系统在启动时，先从本机直接相连的网络所含有的地址集合推导出一个初始的路由集合，然后和相邻的机器联络，请求得到其他路由。

初始路由表构建出来之后，路由器还必须适应路由的变化。在变化缓慢的小型互联网中，管理员可以用手工方式建立和修改路由。但是，在变化迅速的大型互联网中，人工更新的速度慢得不可想象，而且容易导致人为的错误，因此需要自动更新路由的方法。在理解 IP 路由器所用的路由表自动更新协议之前，先要回顾几个基本概念。下一节的内容就是基本概念的回顾，它为路由选择提供了必要的概念基础。

13.3 利用部分信息转发

路由器与典型主机之间的基本区别是主机通常对它们所连接的互联网的结构知之甚少。主机没有掌握所有可能目的站的全部信息，甚至也不了解所有可能目的网络的信息。实际上，许多主机的路由表中仅有两个路由表项：一个是用于本地网络的路由，一个是用于相邻路由器的默认路由。主机可以把所有非本地的数据报发送给本地路由器去交付。这就是说：

即使只有部分转发信息，主机也能成功地转发数据报，因为它可以依靠一个路由器来完成转发。

如果路由器只有部分信息，还能转发数据报吗？答案是可以，但是仅在某些特定的情况下才可以。为了有助于理解，我们可以把互联网想象成一个泥泞道路纵横交错的外地乡村，在道路的交叉路口有方向标记指示。想象一下，你没有地图，并且不懂当地语言，所以既无法问路，也看不懂路标，而你需要去一个名叫 Sussex 的村子旅行。你开始了自己的行程，沿着唯一的一条通往镇外的路出发，然后开始寻找方向标记。第一个路标上写着：

到 Norfolk 向左，到 Hammond 向右，到其他地方径直向前^①。

因为路标上没有明确地列出你要找的目的地，所以你继续向前走。按照路由选择的术语来说，我们认为你选择了**默认路由**（default route）。途中经过若干个路标之后，你终于看到了如下文字的路标：

到 Essex 向左，到 Sussex 向右，到其他地方径直向前。

此刻你就向右转，再经过几个路标之后，就会到达一条通向 Sussex 的路。

我们所设想的旅行模拟了数据报在互联网上的传输过程，路标模拟的是路径沿线上路由器中的路由表。没有地图，也没有向导，旅行完全依赖于路标，正如互联网中数据报的转发完全依赖于路由表。显然，即使路标仅仅包含部分信息，也可以起到导航的作用。

这里的一个关键问题是正确性。作为一个旅行者，你也许会问：“我如何确保沿路标前进将会把我带到目的地？”你还会问：“我如何确保沿路标行进的路线就是带我到目的地的最短路线？”当你经过了许多路标之后，如果还没看到一个路标明确列出你的目的地，这些问题就会让你倍加困惑。当然，问题的答案与道路系统的拓扑结构和路标的內容都有关系，但有一个基本要求：如果从整体来看所有路标上的信息，它们应该是一致的和完备的。从另一个角度来看这个例子会发现，我们没有必要在每一个交叉路口标记每个目的地。只要所有明确的标记指出了通往目的地的最短路径，并且标出了通往各个目的地的最短路径的转向，这些路标上还可以标记出默认路径。后面的几个例子介绍了一些让路标内容具有一致性的方法。

作为一种极端情况，设想一个简单的星型拓扑结构的道路系统。在该系统中，每个村子仅有一条道路与之相连，而所有的路都交汇于一个中心点。为了确保一致性，中心交汇点的路标必须包括听有可能目的地的信息。而在另一种极端情况下，设想有一个任意道路的集合，在各个交叉路口的路标上都列出所有可能目的地的信息。为了确保一致性，必须做到：如果在某一个交叉路口的路标上指出到达目的地 D 要经过道路 R，从其他道路到 D 的路径都不会比从 R 到 D 的这条路更短。

这两种极端的结构都不太适用于互联网的路由器系统。一方面，使用中心点交汇的方法是行不通的，因为没有哪一台机器的速度足够快，以至于能作为中央交换机来处理所有经过的通信量。另一方面，在所有路由器中保存全部可能目的站信息是不切实际的，因为在网络发生变动或管理员需要检查一致性时，需要传播大量的信息。因此，我们寻求的解决方案允许由各个组来自主地管理本地的路由器，添加新的网络互连和路由信息，而不必变动远处的路由器。

第三种拓扑结构有助于理解后面讨论的互联网体系结构。设想在这个结构中半数的城市位于整

^① 幸运的是，路标是用你可以看懂的语言印刷的。

个国家的东部，而另一半城市位于西部，而在分隔东西两部分的河流上仅有一座桥。假如住在东部的人不喜欢西部的人，他们只想让路标上列出东部的目的城市，而不列出西部城市。住在西部的人则正好与之相反。如果东部的每个路标都明确地列出东部的所有目的城市，并指出一条过桥的默认路径通向西部城市；同时西部的每个路标都明确地列出西部的所有目的城市，并指出一条过桥的默认路径通向东部城市，这时的路由信息就是一致的。

13.4 最初的因特网体系结构与核心

我们有关转发和路由传播协议的知识很多是从因特网的试验中得来的。在 TCP/IP 发展早期，ARPANET 作为当时因特网的主干网，参与研究的一些网点都与它相连。在最初的试验中，每个网点自己管理路由表，采用手工方式来配置到其他目的站的路由。随着羽翼渐丰的因特网逐渐发展壮大，人工维护路由的技术变得越来越不切合实际，因而需要一种自动维护路由的机制。

因特网的设计人员所选的路由器体系结构由少量中心路由器和大量外围路由器组成，中心路由器保存了所有可能目的站的全部信息，而大量外围路由器仅保存部分信息。换用我们模拟的场景来说，这就像让小部分位于中心路口上的路标列出全部目的地，而让其他外围路口上的路标仅列出本地附近的目的地。只要每个外围路口的路标上的默认路径指向某个中心路口，旅行者就一定能到达最终的目的地。

保存全部信息的中心路由器集合称为因特网的**核心**（core），有时也称为**无默认区**（default-free zone）。把因特网的路由选择划分为两级系统，既有优点也有缺点。优点在于它允许本地管理员能够在外围路由器中处理局部的变化，而不会影响因特网的其他部分。缺点在于它带来了潜在的不一致性。在最坏的情况下，某个外围路由器的错误可能会造成远处的路由不可达。

我们可将上述思想归纳如下：

核心路由选择体系结构的优点是，因为非核心路由器使用部分信息，某个外围的网点可以自主地变更局部的路由。缺点是某个网点可能造成不一致性，使一些目的站不可达。

路由表之间的不一致性通常由以下几个原因造成：计算路由表的算法出错；给那些算法提供了不正确的数据；把结果传输到其他路由器的过程中出错。协议的设计人员设法控制这些错误造成的影响，目的是在任何时候保持所有路由的一致性。如果由于某种原因使路由变得不一致了，路由选择协议应该具有足够的健壮性来快速检测并纠正这些错误。最重要的是协议在设计上应该能抑制这些错误造成的影响。

如果仍记得早期的因特网是从一个广域的主干网 ARPANET 发展而来的，就容易理解早期因特网的体系结构了。提出核心路由器系统的主要动机来自于把本地网络连接到 ARPANET 的要求。图 13.1 展示了这样的思想。

要理解图 13.1 中的路由器为何不能使用部分信息，可以想象在某些路由器使用默认路由的情况下，数据报传输所经过的路径。在源站网点，本地路由器检查是否有到目的站的明确路由，如果没有，则把数据报发送到默认路由指明的路径上。如果数据报的目的站没有明确路由，无论它们的最终目的站是什么，都会沿着同一条默认路径来传输。在传输路径上的下一个路由器把那些目的站有明确路由的数据报转到其他路径，并按默认路由发送其余的数据报。为了确保全局一致性，一连串默认路由必须经过所有路由器，形成一个大环。因此，这种结构要求所有本地网点协调好它们的默认路由。此外，即使默认路由经过协调达成了一致，转发效率仍然不高，因为一个数据报在从源站传送到目的站时可能经过所有 n 个路由器。

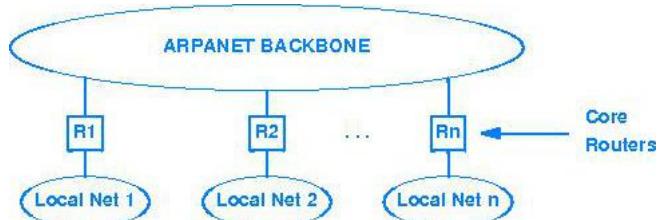


图 13.1 早期因特网的核心路由器系统可以看成是把本地局域网连接到 ARPANET 的一组路由器。本地网络上的主机把所有非本地的通信量传给最近的核心路由器

为了避免默认路由造成的低效率，早期的因特网避免使用默认路由。设计人员设法让路由器相互交换路由信息，从而使每个路由器都具有全部的路由。由于所有路由器都连接到一个主干网网络，所以这样做很容易。

13.5 从核心结构到对等主干网结构

在因特网中引入 NSFNET 主干网，增加了路由选择结构的复杂性，迫使设计人员发明出一种新的路由选择结构。实际上，因特网从一个中心的主干网发展成一组**对等主干网络**（peer backbone networks），或简称为**对等方**（peers）。图 13.2 中显示的是包括两个主干网的因特网拓扑结构。

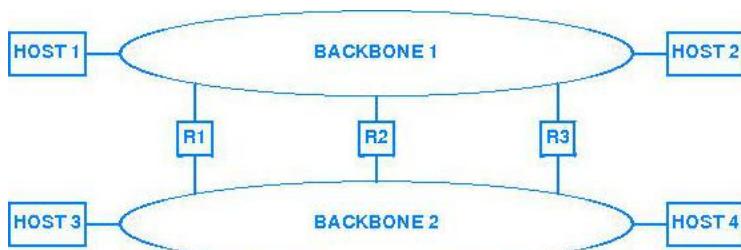


图 13.2 两个对等主干网通过多个路由器互连的例子。1989 年的因特网就具有这种结构

为了理解在多个对等主干网之间进行 IP 路由选择的复杂性，我们来看看图 13.2 中从主机 3 到主机 2 的路由。假设该图是当时根据地理方位绘制的：主机 3 在西海岸，连接到主干网 2；而主机 2 在东海岸，连接到主干网 1。在建立主机 3 和主机 2 之间的路由时，管理员必须从下列方案中进行选择：

- (a) 主机 3 的通信量选择的路由先经过西海岸的路由器 R₁，然后通过主干网 1 到达主机 2；
- (b) 主机 3 的通信量在主干网 2 上转发，经中西部的路由器 R₂，然后通过主干网 1 到达主机 2；
- (c) 主机 3 的通信量在主干网 2 上转发，经过东海岸的路由器 R₃，然后到达主机 2；

还有更迂回的路由：通信量可能从主机 3 经西海岸路由器 R₁，通过主干网 1 到达中西部路由器 R₂，又回到主干网 2 上，到达东海岸路由器 R₃，最后再通过主干网 1 到达主机 2。这种路由是否可取，取决于网络使用的策略以及不同路由器和主干网的容量。

对于大多数对等主干网的配置，一对地理位置接近的主机之间的通信量应该选择最短路径，与

那些在地理上跨越国家边界的通信量所选的路由是不同的。例如，从主机 3 到主机 1 的通信量应该通过西海岸路由器来传输，因为这个路由器到两个主干网的距离是最短的。

从表面上看，所有这些陈述很简单，但由于以下两个原因实现起来比较复杂。首先，虽然标准的 IP 转发算法使用 IP 地址中的网络部分来选择路由，但是在一个对等主干网的结构中，最优的转发需要为每台主机建立单独的路由。对于上面讨论的例子，即使主机 1 和主机 2 都与 ARPANET 主干网相连，主机 3 中的路由表还是需要为主机 1 和主机 2 建立不同的路由。其次，两个主干网的管理员必须同意确保路由在所有路由器之间的一致性，否则可能发生转发环路（forwarding loop）^①。当一组路由器中的路由构成一个环路时，就会发生转发环路。

13.6 自动路由传播

前面讲过，最初的因特网核心系统避免了使用默认路由，因为它把所有可能目的站的全部信息传播到了每个核心路由器。许多公司的互联网现在使用的就是与此类似的方法——公司内的路由器运行相应的程序来传递路由信息。下一节将讨论计算和传播路由信息的两种基本算法，在后面两章中将介绍使用这些算法的协议。

路由选择协议有两个重要功能。首先，负责计算一组最短的路径；其次，通过持续地更新路由信息，响应网络故障或拓扑的改变。因此，在考虑路由传播的问题时，研究协议和算法的动态行为是很必要的。

13.7 距离向量（Bellman-Ford）路由选择

距离向量（distance-vector）^②这个术语指的是用于传播路由信息的一类算法。距离向量算法的思想很简单。每个路由器在路由表中列出所有已知路由。路由器启动时对路由表进行初始化，为自己直接相连的每个网络生成一个表项。表中的每个表项指出了一个目的网络，并给出到那个网络的距离，通常用跳（hop）数来度量距离值（后面还将更精确地给出跳的定义）。如图 13.3 所示，这是一个路由表的初始内容，该路由器与两个网络相连。

Destination	Distance	Route
Net 1	0	direct
Net 2	0	direct

图 13.3 一个初始的距离向量路由表，与路由器直接相连的网络各自有一个表项。每个表项包含网络的 IP 地址和到该网络的整数距离值

每个路由器定期把自己的路由表副本发送给与其直接相连的其他路由器。当来自路由器 J 的报告到达路由器 K 之后，K 检查这个报告中列出的目的站以及到各个目的站的距离。如果 J 知道有一条更短的路由去某个目的站，或者 J 列出 K 的表中不曾有的目的站，或者 K 目前到某个目的站的路由经过 J，而 J 到该目的站的距离有所改变，K 就会替换自己的路由表中的相应表项。例如，图 13.4 给出了路由器 K 原有的路由表和从另一路由器 J 传来的更新报文。

^① 转发环路又称为**路由选择环路**（routing loop）。

^② 术语“距离向量”的几个同义词是：vector-distance, Ford-Fulkerson, Bellman-Ford 和 Bellman，后三个词是根据发表这个概念的研究人员名字命名的。

Destination	Distance	Route	Destination	Distance
Net 1	0	direct	Net 1	2
Net 2	0	direct	Net 4	3
Net 4	8	Router L	Net 17	6
Net 17	5	Router M	Net 21	4
Net 24	6	Router J	Net 24	5
Net 30	2	Router Q	Net 30	10
Net 42	2	Router J	Net 42	3

(a)

(b)

图 13.4 (a) 路由器 K 原有的路由表; (b) 从路由器 J 传入的路由更新报文。有箭头标记的表项用来更新原有的表项，或给 K 的路由表添加新表项

注意，如果 J 报告到某目的站的距离是 N，那么 K 中路由表更新后的表项中的距离就是 N + 1（从 J 到目的站的距离加上从 K 到 J 的距离）。当然，路由表项还有第三栏，指明路径上的下一跳。在初始的路由表中，各个表项中的下一跳的值都标记为**直接交付** (direct delivery)。当路由器 K 根据来自路由器 J 的报文添加或更新某个表项时，它把该表项中的下一跳赋值为路由器 J。

距离向量这个术语来源于定期发送报文中的信息。一个报文包含成对的 (D, V) 列表，其中 V 表示目的站，称为**向量** (vector)，而 D 是到该目的站的距离。注意，距离向量算法是以第一人称的语气来报告路由的，也就是说，我们可以把一个路由器的通告理解成：“我可以到达距离 D 处的目的站 V”。在这种设计中，所有路由器必须参与路由的距离向量信息交换，以使路由具有高效性和一致性。

虽然距离向量算法易于实现，但它们也有缺点。在一个完全静态的系统中，距离向量算法计算最短路径，并把路由正确地传播到所有目的站。但当路由迅速发生变化时，算法可能无法稳定下来。当一个路由发生变化时（即新建一个连接或旧连接出故障），相应的信息缓慢地从一个路由器传到另一个路由器。同时，一些路由器可能会有错误的路由信息。

13.8 可靠性和路由选择协议

大多数路由选择协议使用无连接运输服务。早期的协议直接把报文封装在 IP 数据报中，现在的路由选择协议通常把报文封装到 UDP 中^①。遗憾的是，IP 和 UDP 都提供了同样的语义：报文可能丢失、延迟、重复、破坏或无序交付。这样，使用它们的路由选择协议必须对出现的故障情况进行补救。

路由选择协议使用了几种技术来处理交付问题。校验和用于处理报文破坏的问题。报文丢失的问题通过**软状态** (soft state)^② 来处理，或通过确认和重传来处理。为了处理无序交付以及在旧报文到达时产生相应的回答，路由选择协议通常使用了**序号** (sequence number)。

13.9 链路状态 (SPF) 路由选择

距离向量算法最主要的缺点是其无法适应网络规模变大的情况。除了前面提到的对变化反应太慢的缺点之外，该算法还要进行大量的报文交换，由于每次路由更新都包含每个可能的网络对应的

^① 下一章讨论一个例外：使用 TCP 的路由选择协议。

^② 回顾前面的内容，软状态依靠超时机制来删除旧信息。

表项，报文的长度与互联网中的网络总数成正比。此外，由于距离向量协议要求每个路由器都参与进来，所以交换的信息量可能会十分庞大。

距离向量算法有一种替代算法，即所谓的**链路状态**（link state 或 link status）算法，或称**最短路径优先**（shortest Path First，简称 SPF）^①算法。SPF 算法要求每个参与的路由器都被给予或计算出拓扑结构的信息。我们可以想象每个路由器都有一张映射图，图中标出了其他所有路由器以及与路由器连接的网络，这种方法最易于我们思考拓扑结构。用抽象的术语表示，路由器对应于图中的结点，而连接路由器的网络对应于图中的边。当且仅当两个结点对应的路由器之间能直接通信时，这两个结点之间才有一条边。

参与 SPF 算法的路由器并不发送包含目的站列表的报文，而是要完成两项任务。首先，它要积极地检测所有相邻路由器的状态。用图论的术语来讲，如果两个路由器共享一条链接，则称它们是相邻的路由器；用网络的术语来讲，两个相邻的路由器连接到同一个网络中。其次，路由器要向所有其他路由器定期传播链路状态信息。

为了检测直接连接的相邻路由器的状态，两个相邻路由器交换短报文，确认对方是否可达且处于活跃状态。如果相邻的路由器回答了，则称这两者之间的链路是正常工作的（up），否则就称链路是有故障而不能工作的（down）^②。为了通知其他所有路由器，每个路由器定期广播一个报文，报文中列出该路由器的各个链路状态。这个状态报文并不指明路由，而是报告成对的路由器之间是否能够通信。路由器中的协议软件设法把各个链路状态的报文副本交付给所有相关的路由器（如果底层网络不支持广播，就只能通过点对点地转发各个报文副本实现交付）。

链路状态报文到达之后，路由器使用其中的信息更新自己的互联网映射图，把链路的状态标记为正常或故障。链路状态变化之后，路由器使用著名的**Dijkstra 最短路径算法**（Dijkstra shortest path algorithm），在相应的图中求最短路径^③。Dijkstra 算法从单个源点开始，计算它到其他所有目的地的最短路径。

SPF 算法的主要优点之一就是每个路由器使用相同的原始状态数据，独立地计算路由，而不需要依靠中间路由器的计算。由于链路状态报文在传输过程中不会改动，所以调试问题较为容易。由于路由器在本地完成路由计算，所以确保了计算的收敛性。最后，由于链路状态报文仅携带与单个路由器直接相连的链路信息，报文的长短与下层互联网中的网络数目无关。因此，SPF 算法比距离向量算法更适用于大规模的互联网。

13.10 小结

为了确保所有网络保持高可靠的可达状态，互联网必须提供全局一致的转发。主机和大多数路由器仅含有部分路由信息，依靠默认路由把数据报发送给远方的目的站。全球因特网最初使用核心路由器结构来解决路由选择问题，在这种结构中，一组核心路由器各自包含所有网络有关的全部信息。

当额外有一些主干网加入到因特网后，出现了一种新的路由选择结构来适应这种扩展的拓扑结构。目前仍存在一组独立管理的对等主干网，这些主干网通过多个地方的路由器互连起来。

当路由器交换路由信息时，通常会使用两种基本算法，即距离向量算法或 SPF 算法。距离向量算法的主要缺点是进行分布式的最短路径计算，当网络连接的状态频繁变化时，计算可能不收敛。因此，SPF 算法更适用于大型互联网或下层拓扑结构变化快的互联网。

^① “最短路径优先”这个名字的用法并不恰当，因为所有的路由选择算法都要计算最短路径。

^② 实际上，为了避免在正常状态和故障状态之间来回变化，许多协议使用 n 中取 k 的规则（k-out-of-n rule）来检测活跃状态，即在大多数请求得不到回答的情况下，链路状态才由正常转为故障，而在大多数请求都得到回答的情况下，链路状态由故障转为正常。

^③ “最短路径优先”这个词是由 Dijkstra 最先提出来的。

13.11 深入研究

本章中核心路由器系统的定义出自 Hinden and Sheltzer [RFC 823]。Braden and Postel [RFC 1812] 给出了因特网路由器的进一步规范。Braun [RFC 1093] 和 Rekhter [RFC 1092] 讨论了 NSFNET 主干网中的路由选择。SPF 路由选择的使用比因特网出现的时间早；最早的 SPF 协议样本之一出自 ARPANET，ARPANET 使用内部的路由选择协议来建立和维护分组交换机之间的路由。Clark [RFC 1102] 和 Braun [RFC 1104] 都讨论了基于策略的路由选择。

13.12 习题

1. 如果一个路由器发现自己将要把一个 IP 数据报转发回报文到达时通过的那个网络接口，它应该怎么办？理由是什么？
2. 阅读 RFC 823 和 RFC 1812 之后，说明一个因特网核心路由器（即含有完整路由信息的路由器）在前一个问题描述的情况下应该怎么做。
3. 在一个核心系统中，路由器怎么使用默认路由把所有非法数据报发送到某个特定机器？
4. 想象有一个管理员意外地错误配置了一个路由器，路由器通告它与六个特定的网络都有直接连接，但事实上并非如此。其他路由器应该如何保护自己免受无效通告的影响，同时还能接受从这种“不可信”的路由器传来的其他更新信息？
5. 路由器产生的什么类型的 ICMP 报文？
6. 假设一个路由器在使用不可靠的运输服务进行交付。路由器如何确定某个指定的相邻路由器是正常还是出了故障？提示：查阅 RFC 823，看最初的核心系统如何解决这个问题。
7. 如果两个路由器分别通告到达指定的网络 N 需要同样的开销 k 。试描述出某种情况，使通过其中一个路由器转发可能比通过另一个转发需要更少的总跳数。
8. 一个路由器怎样知道传入的数据报携带了路由更新报文？
9. 研究图 13.4 所示的距离向量更新。对于表中更新的每个表项，给出路由器执行更新的理由。
10. 研究在数据报重复、延迟或无序交付时，如何使用序号来确保两个路由器不会被搞糊涂。应该如何选择初始序号？为什么？

第14章 对等网络间的路由选择（BGP）

14.1 引言

前一章介绍了路由传播的思想。本章将继续深入讨论互联网路由选择的体系结构。本章讨论了自治系统的概念，并介绍了一组网络和路由器在管理机构控制下使用的协议，用于将本网络有关的路由信息传播给其他组。

14.2 路由更新协议的范围

没有哪种路由更新协议可允许因特网中的所有路由器都参与路由交换。实际上，路由器必须进行分组，其原因有三方面。第一，即使每个网点仅包含一个网络，也没有哪种路由选择协议能适用于任意数量的网点，因为增加新网点就会增加路由通信量，如果这组路由器的数目足够多，路由选择占用的带宽就会超出控制。第二，由于因特网路由器不在同一个网络上，它们无法直接通信。第三，在一个大型互联网中，网络和路由器不是由单个实体全权管理的，路由选择也不总是使用最短路径。实际上，由于网络是一些独立的商业群组所拥有和管理的，所以不同的组可以选择不同的策略。一个路由选择体系结构必须提供一种方法，使各组能够独立控制路由选择和访问。

限制路由器交互具有重要意义。这一思路被因特网中使用的大多数路由选择体系结构所采用，并且可以解释我们将要研究的一些机制。这个重要的思路归纳如下：

尽管路由器需要交换路由信息，但是在一个任意规模的互联网中让所有路由器都参与某个路由更新协议，这是不现实的。

14.3 确定组大小的实际限度

上面一段话引出了许多问题。例如，规模多大的互联网被认为是“大型”的？如果只让有限的一些路由器参与路由信息的交换，那些排除在外的路由器会怎么样？它们的功能正常吗？没有参与交换的路由器总是能够把数据报转发给参与交换的路由器吗？参与交换的路由器能够把数据报转发给没有参与交换的路由器吗？

关于网络规模的问题，答案与对所用算法的理解和路由器所连接网络的容量有关，还与路由选择协议的细节有关。需要考虑两方面的问题：时延和开销。时延很容易理解，例如，当路由器使用距离向量协议时，想一想某个变化通知到所有路由器所经历的最大时延。每个路由器都必须接收新信息，更新其路由表，然后把信息转发给相邻的路由器。在 N 个路由器按直线拓扑排列的互联网中，把某个变化通知给所有路由器需要 N 步。这样，必须对 N 进行限制，以保证更新信息的快速分发。

开销的问题也很容易理解。由于每个参与路由选择协议的路由器必须发送报文，参与的路由器越多，意味着路由通信量越多。此外，如果路由选择报文包含所有可能目的站的列表，每个报文的长度会随着路由器和网络的数量增多而增加。为了确保路由通信量只占底层网络整个通信量的一小部分，必须限制路由选择报文的大小。

实际上，大多数网络管理员没有足够的信息对时延或开销进行详细分析，他们通常会采用一个

简单的启发式指导原则：

在一个广域网上最多允许 12 个路由器参与单个路由信息协议就会很安全；在一组局域网上最多允许 60 个左右的路由器参与也会很安全。

当然，这一原则只给出了一般性的建议，还有很多种例外情况。例如，如果底层网络具有特别低的时延和高容量，参与的路由器数量可以更多一些。类似地，如果底层网络容量非常低或通信量很大，则参与的路由器数量必须更少一些，才能避免路由通信量造成网络超载。

因为互联网不是静态的，路由选择协议将产生多少通信量，或者路由通信量将占用底层带宽多少百分比，估计起来都会很困难。例如，随着网络上的主机数量不断增长，随之产生的通信量增长会占用更多的网络容量。另外，新的应用也可能导致通信量的增加。因此，网络管理员在选择一种路由选择体系结构时，不能仅仅依靠上述指导原则，而是通常会实现一种**通信量监视**（traffic monitoring）的方法。在实际应用中，通信量监视器被动地监听网络，并记录关于通信量的统计信息。需要特别指出的是，监视器既能够计算网络利用率（即底层带宽的使用率），也能够计算携带路由选择协议报文的分组占总分组数的百分比。管理员可以通过一段长时间（例如几周或几个月）的测量来观察通信量趋势，并且可以使用这个观察结果来判断是否有太多的路由器参与了一个路由选择协议。

14.4 一个基本思想：额外跳

虽然参与一个路由选择协议的路由器数量必须加以限制，然而这样做意味着有些路由器将被排除在组外，于是就会导致这样的结果：**组外的路由器**（“outsider”）似乎只能把组内的一个成员作为默认路由。在早期的因特网中，随着多个路由器被添加到多个网点，核心系统实际上具有中枢路由选择的功能，非核心路由器把要交付的数据报发送给核心系统。但是，非核心路由器的加入给研究人员带来一个问题：如果组外的路由器使用组内的一个成员作为默认路由，这种路由选择将不是最优的。更重要的是，在这种体系结构中，人们不需要大量的路由器或一个广域网。但是，当未参与的路由器使用一个参与路由器进行交付时，可能出现非最优路由的问题。为了弄清楚原因，可以看图 14.1 中的例子。

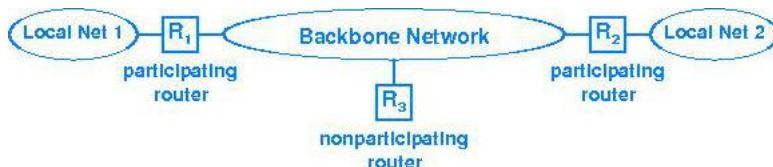


图 14.1 一个可能造成额外跳问题的体系结构。当连接到主干网的未参与路由器有一条到参与路由器的默认路由时，就会造成非最优的路由选择

在图 14.1 中，路由器 R₁ 和 R₂ 分别连接到局域网 1 和局域网 2。由于它们参与了路由选择协议，所以各自都知道怎样到达局域网 1 和局域网 2。假设未参与的路由器 R₃ 选择其中一个参与路由器 R₁ 作为默认路由。也就是说，R₃ 把所有发往未直接连通的网络上的数据报都发给 R₁。例如，R₃ 把发往网络 2 的数据报通过主干网发送到所选的参与路由器 R₁，然后 R₁ 必须把该数据报通过主干网转发给 R₂。当然，最优的路由选择要求 R₃ 把发往网络 2 的数据报直接传输给 R₂。注意，选择哪个参与路由器都没有什么区别。只有当数据报的目的站位于所选的参与路由器直连的网络时，才能得

到最优路径；其他情况则需通过其他主干网路由器转发数据报，从而让数据报在主干网上进行第二次不必要的传输。还要注意到，参与路由器不能使用 ICMP 重定向报文来通知 R_3 其路由不是最优的，因为 ICMP 重定向报文只能发送到初始源站，而不能发送到中间路由器。

我们把图 14.1 所示的路由选择异常现象称为**额外跳** (extra hop) 问题。这个问题隐藏得很深，因为表面上看起来一切都正常——数据报的确到达了它们的目的地。但是，由于路由选择不是最优的，系统效率很低。每个有额外跳的数据报都耗费了中间路由器的资源，并且占用的主干网带宽是它本应占用的两倍。要解决这个问题，需要修改一些原有的观点：

把参与一个路由更新协议的一组路由器处理为一个默认交付系统，可能会引起数据报通信量的额外跳问题；需要建立一种机制，允许未参与的路由器从参与路由器那里了解路由信息，以便选择最优的路由。

14.5 自治系统的概念

我们应该如何为因特网中的路由器分组，让每个组各自运行一个路由更新协议？回答这个问题的关键在于意识到因特网不是由一些独立的网络组成的。实际情况是，因特网中的许多网络和路由器分别属于一些单位和个人。某个指定的实体拥有的网络和路由器在单独一个管理机构的管辖范围内，这个管理机构能够确保其内部路由信息的一致性和可用性。另外，管理机构可以选择其中一个路由器充当这样的机器：它将把单位内的网络情况通知给外部环境，并了解单位外部的网络情况。

出于路由选择的目的，由单个管理机构控制的一组网络和路由器称为一个**自治系统** (autonomous system，简称 AS)。在一个自治系统内的路由器可以自由地选择相应的机制，以发现路由、传播路由、确认路由以及检测路由的一致性（下一章研究自治系统用来在内部传播路由信息的一些协议）。

虽然自治系统的定义可能有些含糊，但必须精确地定义自治系统之间的边界，从而允许一些自动化的算法做出路由选择的决策。例如，一个自治系统可能更愿意禁止路由选择分组通过竞争对手的自治系统，即使的确存在那样的一条路径。为了让自动化的路由选择算法有可能把不同的自治系统区分开，一个负责分配所有因特网号码的中央机构集中为每个自治系统分配**自治系统号码** (autonomous system number)。当两个自治系统中的路由器交换路由信息时，每个路由器可通过协议了解另一个自治系统的号码。

我们可以把以上思想总结如下：

因特网被划分成多个自治系统，每个系统各归一个管理机构所有。一个自治系统可以自由地选择内部的路由选择体系结构和协议。

在实际应用中，虽然一些大单位已经获得了自治系统号码，允许它们连接到多个 ISP，但我们可以把一个自治系统想象成与一个大型 ISP 对应，即：

在目前的因特网中，一个大型 ISP 是一个自治系统。在非正式的讨论中，工程师提到自治系统之间的路由选择时，通常是指一些主要 ISP 间的路由选择。

14.6 外部网关协议和可达性

每个自治系统需要配置一个或多个路由器，与其他自治系统进行通信。路由器被配置用来了解和搜集自治系统内部网络相关的信息，并把这些信息传播出去；另外路由器还要接受其他自治系统中的网络有关信息，并把那些信息散布到自治系统内部。从技术上讲，我们称自治系统向外部通告

网络可达性 (network reachability), 并使用**外部网关协议** (Exterior Gateway Protocol, 简称 EGP)^① 的术语来表示两个自治系统之间传递网络可达性信息所用的协议。严格地讲, EGP 并不是一个路由选择协议, 因为通告网络可达性与传播路由信息是不同的。但是, 在实际应用中, 大多数网络专家没有对此进行严格区分, 我们很可能会听到外部网关协议被称为路由选择协议。

目前, 在因特网中使用了一种 EGP 来交换可达性信息。这种 EGP 称为**边界网关协议** (Border Gateway Protocol, 简称 BGP), 该协议的发展过程中有四个 (不同的) 版本。每个版本都被编了号, 当前版本的正式名字是 BGP-4。遵照互联网业界的 standard 惯例, 我们用术语 BGP 来代替 BGP-4。

当一对自治系统都同意交换路由信息时, 每个自治系统必须为自己指派一个路由器来运行 BGP^②; 可以称这两个路由器相互成为 BGP 的**对等端** (peers)。由于使用 BGP 的路由器必须和另一个自治系统中的 BGP 对等端通信, 在自治系统的“边缘”附近来选择机器的做法很明智。因此, BGP 术语称这个机器为**边界网关** (border gateway) 或**边界路由器** (border router)。图 14.2 举例说明了这种思想。

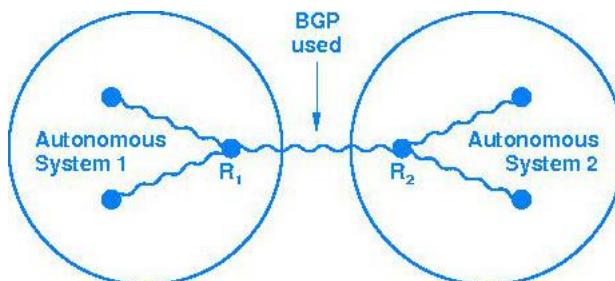


图 14.2 概念示意图: 两个路由器 R_1 和 R_2 各自收集自治系统内部其他路由器的信息后, 使用 BGP 通告其自治系统内的网络。使用 BGP 的单位通常会选择一个靠近自治系统外“边缘”的路由器

在图 14.2 中, 路由器 R_1 收集自治系统 1 中的网络有关信息, 并使用 BGP 把这些信息报告给路由器 R_2 , 与此同时, 路由器 R_2 把自治系统 2 的信息报告给路由器 R_1 。

14.7 BGP 的特征

BGP 在几个方面都非同寻常。最重要的是, 由于 BGP 通告的是可达性信息而不是路由信息, 因此 BGP 既没有使用距离向量算法, 也没有使用链路状态算法。然而, 它可用以下几个特征来刻画:

- **自治系统间通信** (Inter-Autonomous System Communication)。因为 BGP 被设计成外部网关协议, 它担任的主要任务就是允许一个自治系统与另一个自治系统进行通信。
- **多个 BGP 路由器之间的协调** (Coordination Among Multiple BGP Speakers)。如果一个自治系统有多个路由器分别与外部自治系统中的对等端进行通信, 可以使用一种形式的 BGP (称为 iBGP) 来协调这一组路由器, 确保它们都能够传播一致的信息。
- **可达性信息的传播** (Propagation of Reachability Information)。BGP 允许一个自治系统通告在其内部可达的目的站, 或者通过它可达的目的站, 并能够从另一个自治系统了解这类信息。
- **下一跳范例** (Next-Hop Paradigm)。与距离向量路由选择协议类似, BGP 为每个目的站提

^① 这个术语是在用“网关”的名称代替“路由器”的时期提出来的, 术语沿用了过去这种说法。

^② 自治系统通常在连接其他自治系统的路由器上运行 BGP。

供了下一跳信息。

- **策略支持** (Policy Support)。大多数距离向量协议通告的就是本地路由表中的路由，与此不同的是，BGP 可以实现一些本地管理员选择的策略。例如，运行 BGP 的路由器经过配置，能够把自治系统内计算机可达的目的站和通告给其他自治系统的目的站区分开。
- **可靠运输** (Reliable Transport)。BGP 在传递路由信息的协议中显得不太寻常，因为它采用的是可靠的运输服务，因此 BGP 使用 TCP 来进行通信。
- **路径信息** (Path Information)。BGP 通告不是单纯地指出可达目的站和这些目的站各自的下一跳，而是指出了路径的有关信息，允许接收方了解到目的站的路径上的一系列自治系统，并避免循环。
- **增量更新** (Incremental Updates)。为了节约网络带宽，BGP 没有在每个更新报文中传送完整的信息，而是只交换一次完整信息，然后让后续的报文只携带增加部分的变化，这部分变化称为**增量** (delta)；
- **支持无分类编址** (Support For Classless Addressing)。BGP 支持 CIDR 地址。也就是说，BGP 把前缀长度和各个地址一起发送。
- **路由聚合** (Route Aggregation)。BGP 允许发送方把路由信息聚合在一起，发送单个条目来表示多个相关的目的站，通过这种方式来节约网络带宽。
- **鉴别** (Authentication)。BGP 允许接收方对报文进行鉴别（即确认发送方的身份）。

14.8 BGP 的功能和报文类型

BGP 对等端完成三项基本功能。第一项功能包括初始的对等端获取和鉴别。两个对等端建立一个 TCP 连接并进行报文交换，确保双方都同意进行通信。第二项功能是协议的要点：每一方发送肯定或否定的可达性信息。也就是说，发送方通过给出各个目的站的下一跳信息，可以通告一个或多个目的站是否可达，或者发送方可以声明以前通告的一个或多个目的站已经不再可达。第三项提供了验证功能，在通信过程中定期检验两个对等端及其相互之间的网络连接是否一切正常。

为了处理上述的这三项功能，BGP 定义了五种基本报文类型，如图 14.3 所示。

Type Code	Message Type	Description
1	OPEN	Initialize communication
2	UPDATE	Advertise or withdraw routes
3	NOTIFICATION	Response to an incorrect message
4	KEEPALIVE	Actively test peer connectivity
5	REFRESH	Request readvertisement from peer

图 14.3 BGP 中的五种基本报文类型

14.9 BGP 报文首部

每个 BGP 报文的开头都是一个标识报文类型的固定首部。图 14.4 显示了首部的格式。16 八位组的标记 (MARKER) 字段包含了一个标记值，双方都同意使用这个值来标记报文的开始。2 八位组的长度 (LENGTH) 字段指明了以八位组为计量单位的报文总长度。最小的报文长度为 19 八位组（这种报文在首部后没有携带数据），允许的最大报文长度是 4096 八位组。最后，1 八位组的类型 (TYPE) 字段含有的值为图 14.3 中列出的报文类型之一。

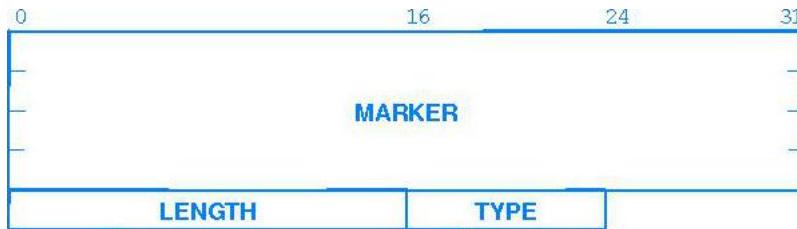


图 14.4 每个 BGP 报文前面的首部的格式

标记字段似乎与众不同。在初始的报文中，该标记的值为全“1”；如果两个对等端都同意使用鉴别机制，标记就可以包含鉴别信息。在任何情况下，双方必须就这个值达成一致，这样才能将它用于**同步**（synchronization）。要知道为什么需要同步，可以回忆前面讲过的内容，所有 BGP 报文是通过流运输（即 TCP）进行交换的，这种传输方式无法识别相邻两个报文之间的边界。在这样的环境下，任何一端的一个简单差错都会产生难以预料的后果。例如，如果发送方或接收方算错了报文中的八位组数目，就会造成**同步差错**（synchronization error）。更重要的是，由于运输协议不指明报文的边界，所以运输协议不会把差错通知给接收方。因此，为了确保发送方和接收方保持同步，BGP 在每个报文的开头放置一个双方都知道的序列，并且要求接收方在处理报文前先确认这个值是否完好无损。

14.10 BGP OPEN 报文

两个 BGP 对等端一旦建立了 TCP 连接，就分别发送一个 OPEN（打开）报文，报文中声明了各自的自治系统号码并建立其他操作参数。除了标准首部，OPEN 报文还包含一个**保持计时器**（hold timer）的值，它指明了两个相继到达的报文在接收时可能间隔的最大秒数。图 14.5 显示了 OPEN 报文的格式。

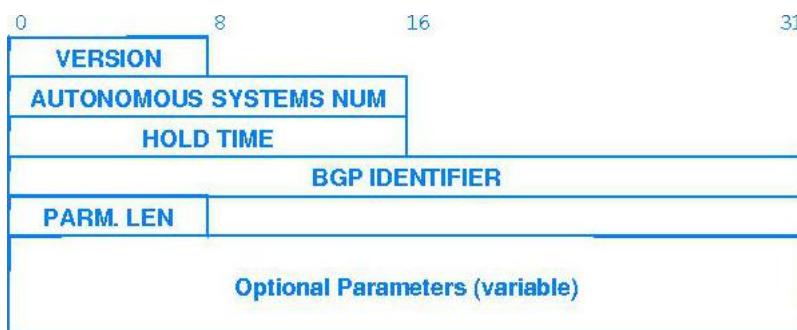


图 14.5 在启动时发送的 BGP OPEN 报文的格式。这些八位组跟在标准的报文首部之后

大多数字段都很直观。版本（VERSION）字段标识使用的协议版本（这个格式是版本 4 的）。前面讲过，每个自治系统都被分配了唯一的编号。自治系统号码（AUTONOMOUS SYSTEMS NUM）字段给出了发送方的系统的自治系统号码。保持时间（HOLD TIME）字段指明了接收方等待来自发送方的报文的最长时间。接收方需要用这个值来设置一个计时器。每当有一个报文到达时，这个计时器被重置；如果计时器超时，接收方则推断发送方不再可用（并停止沿着从发送方那里获得的

路径转发数据报)。

BGP 标识符 (BGP IDENTIFIER) 字段包含一个 32 位整数值，它可以唯一地标识发送方。如果一台机器有多个对等端 (例如，多个对等端可能位于多个自治系统中)，它必须在所有通信中使用同一个标识符。协议规定这个标识符是一个 IP 地址。因此，路由器必须选择自己的其中一个 IP 地址，用在与所有 BGP 对等端的通信中。

OPEN 报文的最后一个字段是可选的。如果有，则这个名为可选参数 (Optional Parameter) 的字段中包含一个参数列表，而参数长度 (PARM.LEN) 字段会给出以八位组为计量单位的长度值。在字段名后有一个“**可变**” (variable) 标注，表示在不同报文中这个字段的长度可变。当报文中有参数时，参数列表中的每个参数前面有 2 八位组的首部，第一个八位组指明参数类型，第二个八位组指明参数长度。如果没有参数，PARM.LEN 的值为 0，报文在此处结束，后面不会再有其他数据。

现有的参数被用于鉴别和容量协商；研究人员还提出了一个允许更大 AS 号的参数^①。鉴别参数的首部标识鉴别的类型，后接对应该类型的数据。报文中的鉴别参数是为了允许 BGP 对等端选择一种鉴别机制，而不用让鉴别机制的选择成为 BGP 标准的一部分。

当接受传入的 OPEN 报文时，使用 BGP 的机器通过发送一个 KEEPALIVE 报文做出响应 (下面将会讨论到)。在两个 BGP 对等端能够交换路由信息之前，每一方都必须发送一个 OPEN 报文并接收一个 KEEPALIVE 报文，因此 KEEPALIVE 报文具有确认 OPEN 报文的作用。

14.11 BGP UPDATE 报文

一旦两个 BGP 对等端创建了一条 TCP 连接，发送 OPEN 报文并得到了确认，对等端就使用 UPDATE (更新) 报文来通告可达的新目的站，或者当目的站变得不可达时撤销原先的通告。图 14.6 显示了 UPDATE 报文的格式。

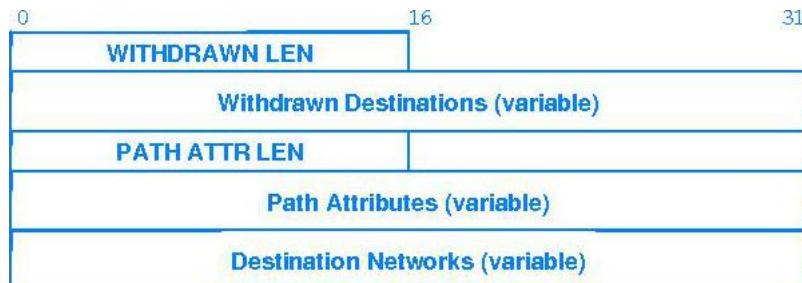


图 14.6 BGP UPDATE 报文的格式，其中长度可变的区域可省略。这些八位组跟在标准报文的首部之后

如图所示，每个 UPDATE 报文分成两个部分：第一部分列出以前通告过，而现在正准备撤销的目的站，第二部分给出将要通告的新目的站。同样，标注“可变”的字段没有固定的长度；如果对于特定的 UPDATE 报文不需要该字段信息，则可从报文中省略这个字段。撤销长度 (WITHDRAWN LEN) 字段占 2 八位组，它指出后面的撤销目的站 (Withdrawn Destination) 字段的长度。如果没有要撤销的目的站，撤销长度字段的值为零。类似地，路径属性长度 (PATH ATTR LEN) 字段指出要通告的新目的站相关的路径属性 (Path Attribute) 的长度。如果没有新目的站要通告，路径属性长度字段值为零。

^① 现已在分配了超过 30,000 个 AS 号 (将近占可用自治系统号码的一半)，因此这种方案是把 AS 号由 16 位扩展到 32 位。

14.12 压缩的掩码地址对

撤销目的站字段和目的网络字段都包含一个 IP 网络地址的列表。为了适用于无分类编址，BGP 必须把每个 IP 地址与一个地址掩码一起发送。然而，BGP 不是把地址和掩码分别作为独立的 32 比特单元来发送，而是使用了一种压缩表示方法，以缩减报文的长度：图 14.7 显示了这种格式。



图 14.7 BGP 用于存储目的地址和相关掩码的压缩格式

由图可知，BGP 实际上并没有发送 1 比特掩码，而是把掩码信息编码成 1 八位组，放在地址的前面。这个掩码八位组包含一个二进制整数，指出掩码中“1”的比特数（假设掩码中的比特是连续相邻的）。掩码八位组后的地址也进行了压缩，只包含 IP 地址中那些被掩码覆盖的八位组。因此，掩码值为 8 或更小则后面的地址只有 1 八位组，掩码值为 9~16 则后面的地址有 2 八位组，掩码值为 17~24 则后面的地址有 3 八位组，掩码值为 25~32 则后面的地址有 4 八位组。有趣的是，这个标准还允许掩码八位组包含零值（这种情况下后面没有地址八位组）。零长度的掩码很有用，因为它对应于一个默认路由。

14.13 BGP 路径属性

我们知道，BGP 不是一个纯粹的距离向量协议，因为它通告的内容不只有下一跳信息。在更新报文的路径属性字段中包含了通告的其他一些信息。发送方可以使用路径属性来指出所通告目的站的下一跳，指出到目的站路径沿线上的自治系统的列表，还可以指出路径信息是从另一个自治系统得到的，还是从发送方的自治系统内部得到的。

路径属性被当成公共属性，应用于报文中通告的所有目的站，以缩减 UPDATE 报文的长度，注意到这一点很重要。因此，如果不同的属性应用到不同的目的站，它们必须在单独的 UPDATE 报文中通告。

在 BGP 中路径属性非常重要，其原因有三个方面。第一，接收方可以使用路径信息检查转发环路。发送方可以明确指出一条通过所有自治系统到达目的站的路径。如果目的站的自治系统出现在路径上的自治系统列表中，则必须拒绝该通告，否则就会出现转发环路；第二，接收方可以使用路径信息实现策略约束（例如，若路径上的自治系统列表包含竞争对手的自治系统，则拒绝该路径）。第三，接收方可以使用路径信息了解所有路由的源站。路径属性除了允许发送方指明信息来自自治系统内部还是来自另一个系统，还允许发送方声明信息是利用 BGP 之类的外部网关协议收集的，还是利用内部网关协议^①收集的。因此，每个接收方可以决定是否接受或拒绝从对等端以外的自治系统中发起的路由。

从概念上讲，路径属性字段包含路径属性项的列表，列表中的每个项目由一个三元组构成：
(类型, 长度, 值)

设计人员没有选择用固定长度的字段，而是选择了一种灵活的编码方法，可以使每个表项所占的空间最小化。如图 14.8 所示，类型信息总要占用 2 八位组，但其他字段的长度是可变的。

^① 下一章将讨论内部网关协议。



Flag Bits	Description
0	0 for required attribute, 1 if optional
1	1 for transitive, 0 for nontransitive
2	0 for complete, 1 for partial
3	0 if length field is one octet; 1 if two octets
5-7	unused (must be zero)

(b)

图 14.8 (a) 出现在每个 BGP 属性路径项前的 2 八位组的类型字段; (b) 各个标志位的含义

路径属性字段中每个表项包含的类型码可能是 8 个可能的类型码之一。图 14.9 列举了这 8 个类型码。

Type Code	Meaning
1	ID of the origin of the path information
2	List of autonomous systems on path to destination
3	Next hop to use for destination
4	Discriminator used for multiple AS exit points
5	Preference used within an autonomous system
6	Indication that routes have been aggregated
7	ID of autonomous system that aggregated routes
8	ID of community for advertised destinations

图 14.9 BGP 的属性类型码及其含义

对于路径属性列表中的每一项，2 八位组的类型字段后面是一个长度字段，字段大小为 1 八位组或 2 八位组。如图所示，标志位 3 指明了长度字段的大小。接收方使用类型字段来确定长度字段的大小，然后使用长度字段的内容来确定属性值字段的大小。

14.14 BGP KEEPALIVE 报文

两个 BGP 对等端定期交换 KEEPALIVE（保活）报文，以测试网络连通性，并确认两个对等端持续工作正常。KEEPALIVE 报文只有标准报文首部，没有其他数据。因此，整个报文长度是 19 八位组（最小的 BGP 报文长度）。

BGP 使用 KEEPALIVE 报文出于两个原因。首先，由于 BGP 使用 TCP 传输报文，TCP 没有一种机制能持续测试连接点是否可达，因而需要定期交换 KEEPALIVE 报文。但是，如果不能交付应用程序发送的数据，TCP 会向应用程序报错。因此，只要双方定期发送一个 KEEPALIVE 报文，就会知道 TCP 连接是否有故障。其次，KEEPALIVE 报文比其他报文占用的带宽少。许多早期的路由选择协议定期交换路由信息来测试连通性。但是，由于路由信息变化并不频繁，报文内容很少改变。此外，由于路由选择报文通常很大，重发同样的报文会浪费网络带宽。为了提高效率，BGP 把路由更新和连通性测试的功能区分开，允许 BGP 频繁地发送小的 KEEPALIVE 报文，而在可达性信息有变化的情况下使用更大的 UPDATE 报文。

前面讲过，使用 BGP 的路由器在它打开一个连接时设定了一个**保持计时器** (hold timer)，该计

时器定义了 BGP 在接收一个报文前能够等待的最长时间。作为一种特殊情况，保持计时器可以设定为零，表明不使用 KEEPALIVE 报文。如果保持计时器的值大于零，标准建议把 KEEPALIVE 间隔时间设置为保持计时器的三分之一。在任何情况下，使用 BGP 的路由器都不能让 KEEPALIVE 间隔时间小于 1 s（这样就可与非零的保持计时器不能小于 3 s 的需求相一致）。

14.15 从接收方的角度来看信息

与大多数传播路由信息的协议不同，外部网关协议不只是报告可达的那些目的站，还必须提供从自治系统外部路由器的角度认为正确的信息。这涉及两个问题：策略和最优路由。策略问题很显然：自治系统内部的路由器被允许到达指定的目的站，而外部的路由器被禁止到达同一个目的站。路由问题意味着路由器必须通告从自治系统外部来看的最优的下一跳。图 14.10 说明了这一思路。

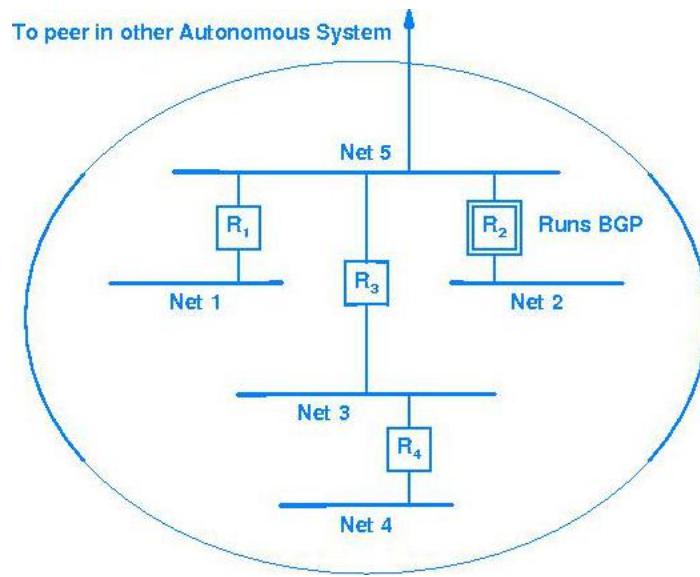


图 14.10 一个自治系统的例子。路由器 R₂运行 BGP 并从自治系统外部路由器的角度来报告信息，而不是报告自己的路由表中的信息

在图 14.10 中，指定路由器 R₂代表这个自治系统来运行 BGP。它必须报告到网络 1 至网络 4 的可达性。但是，当给出下一跳的信息时，它报告网络 1 可通过 R₁到达，网络 3 和网络 4 可通过 R₃到达，而网络 2 可通过 R₂到达。

14.16 外部网关协议的关键约束

前面已讲过，由于外部网关协议支持策略约束，它们通告的网络可能是其可达网络的子集。但是，还有一个更基本的限制施加于外部路由：

外部网关协议对**距离度量** (distance metric) 既不传递也不做任何解释，即使距离度量是可用的。

类似 BGP 的协议的确允许运行协议的路由器声明某个目的站变得不可达，或者给出到该目的站的路径上的自治系统列表，但不能传输或比较两条路由的开销，除非这两条路由来自同一个自治系统。实质上，BGP 只能指出是否存在一条路径到达目的站，既不能传输也不能计算出哪条路径更短。

现在我们就可以理解为什么 BGP 要认真标明它所发送信息的来源。实际情况是：当路由器从

两个不同自治系统中的对等端接收到关于某个指定目的站的通告时，它不能比较两者的开销。因此，使用 BGP 通告可达性就等于在说“我的自治系统提供了一条到达该网络的路径”，路由器无法说“我的自治系统能比另一个自治系统提供一条到达该网络的更好路径”。

查看有关距离的解释，就会意识到 BGP 不能用来作为路由选择算法。特别地，即使一个路由器知道有两条路径到达同一网络，也无法知道哪条路径更短，因为它不知道在中间的自治系统上的路由开销：例如，假设一个路由器使用 BGP 与两个自治系统 p 和 f 中的对等端进行通信。如果自治系统 p 中的对等端通告了一条到指定目的站的路径，此路径中途经过了自治系统 p, q 和 r；自治系统 f 中的对等端也通告了一条到同一目的站的路径，该路径中途经过了自治系统 f 和 g，接收方无法比较这两条路径的长度。通过三个自治系统的路径，可能在每个自治系统中只包含一个局域网；而通过两个自治系统的路径，可能在每个自治系统中需要几跳。接收方没有获得完整的路由信息，因此无法进行比较。

由于自治系统不包含距离度量，所以必须非常小心，只能通告通信量传输将会选取的路由。从技术角度讲，我们把外部网关协议称为**可达性协议** (reachability protocol)，而不是路由选择协议。归纳如下：

由于类似 BGP 的外部网关协议只传播可达性信息，接收方可以实现策略约束，但不能选择开销最少的路由。发送方必须只通告那些通信量传输将会选取的路径。

此处的要点是，任何使用 BGP 来提供外部路由信息的互联网，或者必须依靠策略来选择路由，或者假设经过的每个自治系统具有同等的开销。尽管这样的限制看起来好像没什么副作用，但还是会造出一些出人意料的后果：

1. 尽管 BGP 能够通告多条到某指定网络的路径，但它不允许多条路径同时使用。也就是说，在任一时刻，从一个自治系统中的一台计算机到另一个自治系统中的一个网络，即使两者之间有多条物理连接存在，所有通信量也都会沿着同一路经传输。还要注意，即使源站系统把通信量分到两条或更多路径上发送出去，外部的自治系统也只使用一条返回路径。这样会造成一对机器之间的时延和吞吐量并不对称，使互联网难以监测和调试。
2. BGP 不支持任意自治系统之间的路由器分担负载。如果有多个路由器把两个自治系统连接起来，人们可能希望把通信量平均分配给这些路由器。BGP 允许自治系统按网络来划分负载（例如，把自治系统自身划分为多个子集，并让多个路由器通告划分的各个部分），但它不提供更一般的负载分担。
3. 作为第 2 点的特例，在由多点互连起来的两个或多个广域网结构中，单独使用 BGP 不能提供最优路由选择，而是必须由管理员人工配置各个外部路由器通告哪些网络。
4. 为了进行合理化的路由选择，一个互联网中的所有自治系统都必须就通告可达性的一致性方案达成协议。也就是说，单独使用 BGP 无法保证全局的一致性。

14.17 因特网的路由选择体系结构

为了让互联网正确无误地运转，路由信息必须是全局一致的。单个协议，诸如处理一对路由器之间信息交换的 BGP，无法保证全局一致性，因此需要进一步努力，使路由信息全局地合理化。在最初的因特网路由选择体系结构中，核心系统保证了路由信息的全局一致性，因为在任何时候核心到每个目的站都只有唯一的一条路径。但是，核心系统和后来出现的**路由选择仲裁系统** (routing arbiter system) 都已被淘汰。然而，人们一直没有设计出一种单一的机制作为替代方案来处理路由选择合理化的任务，目前的因特网仍没有一种集中的机制来验证路由和确保全局一致性。

为了理解目前因特网的路由选择体系结构，需要仔细研究因特网的物理拓扑结构。一对 ISP 可以私自互连起来（例如，双方都同意在两个路由器间租用一条线路），或者在**因特网交换点** (Internet

eXchange Point, 简称 IXP) 互连起来, 因特网交换点又称为**网络接入点** (Network Access Point, 简称 NAP)。我们称这两个 ISP 加入了**专用对等连接** (private peering), 或称它们进入了**对等商定** (peering agreement)。用路由选择的术语来说, 一个专用对等连接表示了两个自治系统之间的边界。两个 ISP 定义了它们之间的三种关系, 分别是**上游** (upstream)、**下游** (downstream) 和**转接** (transit)。上游关系用于表示一个大型 ISP 同意从一个较小的 ISP 处获取通信量, 下游关系用于表示一个大型 ISP 把通信量传递给一个较小的 ISP, 转接关系用于表示一个 ISP 同意接受来自其他 ISP 的通信量, 并同意把通信量转发给其他 ISP。

为了有助于确认路由是有效的, ISP 使用称为**路由选择登记** (Routing Registry) 的服务。实质上, 一个路由登记处维护着有关哪个 ISP 拥有哪些地址块的信息。因此, 如果 ISP A 向 ISP B 发送一个声明网络 N 可达的通告, ISP B 可以使用路由登记处的信息来验证地址 N 已指派给 ISP A。遗憾的是, 存在很多路由登记处, 没有哪一种机制可用来确认一个登记处中的数据。由此就会产生一些暂时性的路由选择问题, 例如**黑洞** (black hole) 问题, 在黑洞中包含的一些特定地址对于因特网的任何部分都不可达。当然, ISP 和大多数路由登记处都试图快速找到并修复这种问题, 但没有一个集中授权的登记处, 因特网的路由选择就不可能完全做到没有缺陷。

14.18 BGP NOTIFICATION 报文

除了上述 OPEN 和 UPDATE 报文类型, BGP 还支持一种 NOTIFICATION (通知) 报文类型, 用于进行控制, 或在差错发生时使用。一旦检测到永久性的差错问题, BGP 就会发送一个通知报文, 然后关闭 TCP 连接。图 14.11 给出了这种报文的格式。

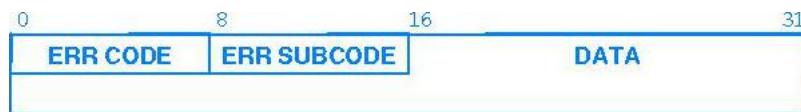


图 14.11 BGP NOTIFICATION 报文的格式。这些八位组在标准报文首部之后

标注为差错码 (ERR CODE) 的 8 位字段指明差错发生的原因, 图 14.12 列举了几种可能的原因。

ERR CODE	Meaning
1	Error in message header
2	Error in OPEN message
3	Error in UPDATE message
4	Hold timer expired
5	Finite state machine error
6	Cease (terminate connection)

图 14.12 BGP NOTIFICATION 报文中差错码字段的可能取值

对于每个可能的差错码, 差错子码 (ERR SUBCODE) 字段给出了进一步的解释说明。图 14.13 列举了差错子码字段的可能取值。

Subcodes For Message Header Errors

- 1 Connection not synchronized
- 2 Incorrect message length
- 3 Incorrect message type

Subcodes For OPEN Message Errors

- 1 Version number unsupported
- 2 Peer AS invalid
- 3 BGP identifier invalid
- 4 Unsupported optional parameter
- 5 Authentication failure
- 6 Hold time unacceptable

Subcodes For UPDATE Message Errors

- 1 Attribute list malformed
- 2 Unrecognized attribute
- 3 Missing attribute
- 4 Attribute flags error
- 5 Attribute length error
- 6 Invalid ORIGIN attribute
- 7 AS routing loop
- 8 Next hop invalid
- 9 Error in optional attribute
- 10 Invalid network field
- 11 Malformed AS path

图 14.13 BGP NOTIFICATION 报文中差错子码字段的含义

14.19 小结

路由器必须划分成多个组，否则大量的路由通信量将会令网络无法承受。因特网是由一组互连的自治系统组成的，其中每个自治系统由处于一个管理机构控制下的路由器和网络组成。自治系统使用外部网关协议（EGP）给其他自治系统通告路由。特别要指出，一个自治系统必须先把自己的网络可达性通告给另一个系统，从其他系统中的源站才能到达这个自治系统的网络。

边界网关协议（BGP）是应用最广泛的外部网关协议。BGP 包含五种报文类型，用于发起通信（OPEN）、发送可达性信息（UPDATE）、报告出错情况（NOTIFICATION）、重新确认信息（REFRESH）和确信对等端仍在通信中（KEEPALIVE）。每个报文的开头都是一个标准首部，首部中包括可选的鉴别信息。BGP 使用 TCP 进行通信。

在全球因特网中，每个大型 ISP 都是一个独立的自治系统，而自治系统之间的边界由两个 ISP 之间的对等商定组成。在物理拓扑结构上，对等连接可以出现在因特网交换点中，或在一条专用的租用线路上。一个 ISP 使用 BGP 与其对等端通信，双方都通告自己可达的网络（即网络前缀），并通过把可达性信息转发给对等端，让对方能够了解自己可达的网络。虽然存在路由选择登记的服务帮助 ISP 来确认通告，但由于因特网目前还没有一个集中授权的登记处，因而还可能会发生问题。

14.20 深入研究

关于早期因特网路由选择的背景知识可从[RFCs 827, 888, 904, 975]中找到。Rekhter and Li [RFC 1771]^①描述了边界网关协议的版本 4（BGP-4）。BGP 已经历了三次重要的修订；以前的版本参见[RFCs 1163, 1267, 1654]。Traina[RFC 1773]报告了 BGP-4 的使用体验，Traina[RFC 1774]分

^① 在本书翻译期间已发现 2006 年 1 月发表的 RFC 4271 已经把 RFC 1771 划归陈旧的——译者注。

析了产生的路由通信量。最后，Villamizar et. al. [RFC 2439]讨论了路由摆动（即路由定期变动）的问题。更多有关 BGP-4 的细节参见[RFCs 1997, 2918, 3392, 2796, 3065]。

14.21 习题

1. 如果你的网点运行了 BGP 之类的外部网关协议，你通告了多少路由？从一个 ISP 导入的路由有多少？
2. 一些 BGP 实现软件使用了“**抑制**”(hold down) 机制，使协议在从对等端接收到一个“**停止请求**”(cease request) 报文之后，延迟一段固定的时间后再接受那个对等端的 OPEN 报文。试分析“抑制”机制有助于解决什么问题。
3. BGP 的正式规范中包括一个解释 BGP 工作过程的有限状态机。绘制状态机的图并标出状态之间的转换。
4. 如果一个自治系统中的路由器发送 BGP 路由更新报文给其他自治系统中的路由器，在报文中，它宣称自己到互联网中的所有可能目的站都可达，此时会发生什么情况？
5. 两个自治系统通过相互发送 BGP 更新报文，可能会建立一个路由选择环路吗？为什么。
6. 使用 BGP 通告路由的路由器，是否应该把被通告的路由与本地路由表中的路由区别对待？例如，如果路由器还没有在本地路由表中装入到某个网络的路由，它是否应该通告那个网络的可达性信息？为什么？提示：阅读 RFC 1771。
7. 关于前一个问题，仔细研究 BGP-4 规范。如果通告了到一个目的站的可达性，但这个目的站还没有列入本地路由表，这种行为是否合法？
8. 如果你在为一个大公司工作，查看该公司是否包括多个自治系统。如果是，它们之间怎样交换路由信息？
9. 把一个大型的跨国公司划分为多个自治系统，这样做的主要优点是什么？主要缺点是什么？
10. A 公司和 B 公司使用 BGP 交换路由信息。为了不让 B 公司的计算机访问 A 公司其中一个网络 N 上的机器，A 公司的网络管理员配置 BGP，把 N 从发给 B 的通告中省略掉。这样做网络 N 就会是安全的吗？为什么？
11. 由于 BGP 使用了可靠的运输协议，所以 KEEPALIVE 报文不会丢失。把 KEEPALIVE 的间隔时间指定为保持计时器值的三分之一，这样做有意义吗？为什么？
12. 查阅 RFC 文档，了解路径属性字段的具体内容。BGP UPDATE 报文的最小长度是多少？

第15章 自治系统内的路由选择（RIP 和 OSPF）

15.1 引言

前一章介绍了自治系统的概念并讲解了 BGP 协议。BGP 是一种外部网关协议，路由器使用该协议把自己所在的自治系统内的网络通告给其他自治系统。本章将进一步研究自治系统内的路由器如何获得本系统内部其他网络的信息，完成对互联网路由选择问题的概述。

15.2 静态内部路由与动态内部路由的对比

在一个自治系统内的两个路由器彼此互为**内部** (interior) 路由器。例如，如果一个大学校园里的机器都属于同一个自治系统，则可以认为其中的两个路由器互为内部路由器。

自治系统中的路由器如何获得关于本自治系统内部网络的信息呢？在变化缓慢的小型互联网中，管理员可以手工建立和修改路由。管理员保留一张关于网络的表格，并在有新网络加入该自治系统或从该自治系统中删除一个网络时，更新该表格。例如，图 15.1 显示的是一个小型公司的互联网。

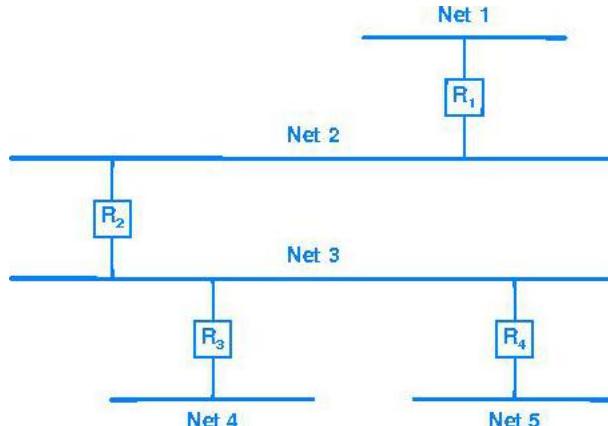


图 15.1 在一个网点中包括了 5 个以太网和 4 个路由器的小型互联网。在这个互联网中，任意两台主机之间仅存在一条可能的路由

在图 15.1 所示的互联网中，选择路由是件很容易的事情，因为任何两点之间仅存在一条路径。管理员可以手工配置所有主机和路由器中的路由。如果互联网发生变动（例如，添加一个新网络），管理员必须重新配置所有机器中的路由。

手工配置路由的系统存在明显的缺点，它不能适应网络的迅速增长，并且在网络发生故障时，需要依靠人来改变路由。对大多数互联网来说，人类对情况变化的反应速度不够快，来不及对问题进行处理，因此必须采用自动的方法。自动选择路由还有利于提高可靠性，为了理解这一点，我们假设给图 15.1 所示的互联网添加一个路由器，使之变为图 15.2 所示的结构。

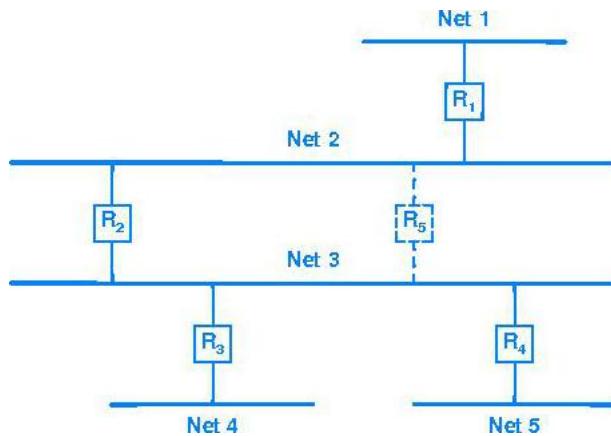


图 15.2 增加了路由器 R_5 后，网络 2 和网络 3 之间多了一条备用路径。当原有路由出现故障时，路由选择软件能够迅速地把路由切换到备用路径

如图 15.2 所示，一些主机之间存在多条路径，在这种情况下，管理员通常选择其中一条作为**基本路径**（primary path），即所有通信量都使用该路径。如果该基本路径上的路由器或网络出现故障，就必须改动路由，沿着备用路径发送通信量。自动改变路由有两点好处：一方面，由于计算机响应故障的速度比人快得多，因而自动改变路由耗时少；另一方面，由于人在输入网络地址时可能会犯点小错误，而自动路由选择不太容易出错。因此，即使在小型互联网中也应该使用自动系统来迅速而可靠地改变路由。

为了把保持路由信息准确的任务自动化，内部路由器之间要定期进行通信，相互交换路由信息。BGP 为外部路由器的通信提供了一个已被广泛接受的标准，与此不同的是，目前在自治系统内部或网点内部使用的协议并非只有单独一个。造成协议多样性的原因，一部分是因为各个自治系统的拓扑结构各式各样，采用的技术五花八门。另一部分原因是需要简单性与强大功能的折中——易于安装和配置的协议往往不能提供强大的功能。因此，目前有不少协议被大家广泛使用。大多数小型自治系统选择单一的一种协议，只用它在内部传播路由信息，而大型自治系统常常选择使用几种协议。

由于没有单一的标准，我们把内部路由器交换路由信息所用的任何协议统称为**内部网关协议**（Interior Gateway Protocol，简称 IGP）。图 15.3 显示了这个通用的概念：两个自治系统，每个系统使用一种特定的 IGP 在内部路由器之间传播路由信息，然后使用 BGP 与外部的自治系统进行通信。

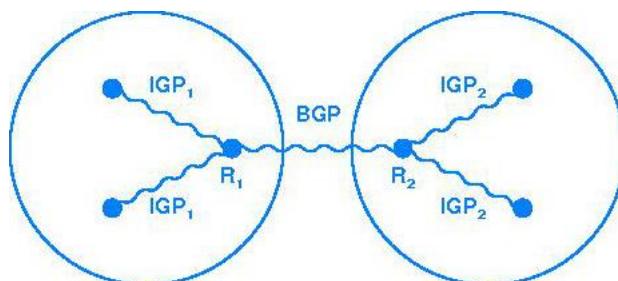


图 15.3 两个自治系统在内部各自使用自己的 IGP，但是在外部路由器与另一个自治系统之间使用 BGP 进行通信

在图 15.3 中, IGP_1 和 IGP_2 分别表示自治系统 1 和自治系统 2 所用的内部网关协议。此图还展示了这样一个重要概念:

一个路由器可以同时使用两个不同的路由选择协议, 一个用于到自治系统之外的通信, 另一个用于自治系统内部的通信。

具体来讲, 运行 BGP 通告可达性的路由器通常还需要运行某种 IGP, 以获得其自治系统内部的信息。

15.3 路由信息协议

15.3.1 RIP 的发展史

使用最广泛的一种 IGP 是路由信息协议 (Routing Information Protocol, 简称 RIP), 它最初是用一个实现该协议的程序 `routed` 来命名的^①。`routed` 软件最初是在加利福尼亚大学伯克利分校设计出来的, 用于为其局域网上的机器提供一致的路由信息。该软件依靠物理网络广播来迅速交换路由信息。它并不是设计用于大型广域网的 (尽管现在厂商也在销售适用于广域网的 RIP 版本)。

基于施乐 (Xerox) 公司 Palo Alto 研究中心 (Palo Alto Research Center, 简称 PARC) 早期所做的关于网络互连的研究, `routed` 实现了源于 Xerox NS RIP 的一个新协议, 它更为通用化, 能够适应多种网络。

尽管 RIP 在其前辈的基础上做了一些小改动, 但 RIP 成为一种流行的 IGP 并非由于它的技术有过人之处, 而是由于伯克利分校把 `routed` 软件和流行的 4 BSD UNIX 系统一起分发, 从而使许多 TCP/IP 网点根本没考虑其技术上的优劣就采用了 `routed`, 并开始使用 RIP。一旦安装并使用了这个软件, 它就成为本地路由选择的基础, 厂商也开始提供与 RIP 兼容的产品。

15.3.2 RIP 操作

底层 RIP 协议是距离向量路由选择算法在本地网络上的一种直接而简单的实现, 它把路由选择的参与者分为**主动** (active) 机器和**被动** (passive) 机器。被动机器即为静默 (silent) 机器。主动路由器向其他路由器通告其路由, 而被动路由器接收通告并在此基础上更新其路由, 但它们自己并不通告路由。只有路由器能以主动模式运行 RIP, 而主机必须使用被动模式。

以主动模式运行 RIP 的路由器每隔 30 s 广播一个路由更新报文, 该报文包含了取自路由器当前路由数据库的信息。每个更新报文由一组**序偶** (pair) 构成, 每个序偶由一个 IP 网络地址和到该网络的整数距离构成。RIP 使用**跳计数度量** (hop count metric) 来衡量距离。在 RIP 度量方法中, 路由器与它直接相连的网络之间的距离被定义为 1 跳^②, 路由器与通过另一个路由器可到达的网络之间的距离为 2 跳, 其余以此类推。因此, 从一个给定源站到目的站的一条路径上的**跳计数** (number of hops 或 hop count), 指的是数据报沿该路径传输时所经过的网络数。显然, 使用跳计数来衡量最短路径并不一定会得到最佳结果。例如, 一条经过三个以太网的跳计数为 3 的路径, 可能比经过两条卫星连线的跳计数为 2 的路径快得多。为了补偿传输技术上的差距, 许多 RIP 实现允许管理员在通告低速网络路由时人工配置高的跳计数。

运行 RIP 的主动机器和被动机器都要监听所有的广播报文, 并根据前面所说的距离向量算法来更新其路由表。例如, 在图 15.2 所示的互联网中, 路由器 R_1 在网络 2 上广播的路由信息报文中包含了序偶 (1, 1), 意思是它能够以开销 1 到达网络 1。路由器 R_2 和 R_5 收到这个广播报文之后, 建立一条通过 R_1 到达网络 1 的路由 (开销为 2)。然后, 路由器 R_2 和 R_5 在网络 3 上广播各自的 RIP

^① 这个名字当初是按照 UNIX 的命名习惯, 把字母 “d” 附加在守护进程名之后而形成的, 发音为 “route-d”。

^② 其他路由选择协议定义直接连接为零跳。

报文时就会包含序偶 (1, 2)。最终，所有的路由器和主机都会建立到网络 1 的路由。

RIP 规定了提高性能和可靠性的几个规则。例如，当路由器收到另一个路由器传来的路由时，必须运用滞后 (hysteresis) 规则，意味着它不会用开销相同的路由来替换原有的路由。在我们的例子中，如果路由器 R₂ 和 R₅ 都发出以开销 2 到达网络 1 的通告，那么 R₁ 和 R₄ 只会建立一条到达网络 1 的路由，而该路由通过了碰巧最先发出通告的那个路由器。总结如下：

为了防止路由在开销相同的路径之间摇摆不定，RIP 规定在获得开销更小的路由之前保留原有路由不变。

如果最先通告路由的路由器出现故障（如发生崩溃），会有什么后果呢？RIP 规定所有监听方必须为通过 RIP 获得的路由设置计时器。当路由器在路由表中添加新路由时，它也为新路由启动一个计时器。当该路由器又收到通告该路由的另一个 RIP 报文后，必须重启计时器。如果经过 180 s 后还没有收到通告该路由的报文，该路由就变为无效的路由。

RIP 必须处理底层算法造成的三类错误。第一，由于算法不能明确地检测出路由选择环路，RIP 或者假定参与者是可信任的，或者采取一定的预防措施。第二，为了防止出现不稳定的现象，RIP 必须把最大可能的距离设定为一个较小的值（RIP 使用的值是 16）。因此，对于那些实际跳计数接近 16 的互联网，管理员必须把它划分为若干部分，或者采用其他协议^①。第三，路由更新报文在网络上传播的速度慢，RIP 使用的距离向量算法会产生慢收敛 (slow convergence) 或计数到无穷大 (count to infinity) 的问题，从而产生不一致性。选择一个小的无穷大 (16)，有助于限制慢收敛，但不能彻底解决问题。

15.4 慢收敛问题

路由表的不一致问题和慢收敛问题并非仅在 RIP 中出现，这是任何距离向量协议中都可能发生的一个基本问题。在此类协议中，更新报文中仅携带由目的网络和到该网络的距离构成的序偶。为了理解这个问题，我们考虑图 15.4 所示的一组路由器，图中描绘出在图 15.2 的互联网中从各处到网络 1 的路由。

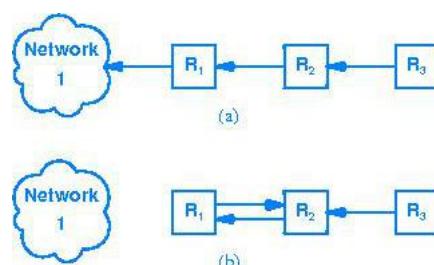


图 15.4 慢收敛问题：(a) 中的三个路由器都有到网络 1 的路由；(b) 中到网络 1 的路由已经消失了，R₂ 对网络 1 的路由通告造成了路由选择环路

如图 15.4(a)所示，路由器 R₁ 直接与网络 1 相连，所以在它的路由表中有一条距离为 1 的路由，在周期性的路由广播中将会包括这个路由。R₂ 从 R₁ 处得知了这个路由，在自己的路由表中建立相应的路由，并以距离 2 通告该路由。最后，R₃ 从 R₂ 处得知该路由并以距离 3 通告它。

现在假设 R₁ 到网络 1 的连接失效了。那么 R₁ 立即更新它的路由表，把它到网络 1 的距离设为

^① 注意，跳计数衡量的是网络的跨距（即两个路由器之间的最长距离），而不是路由器的数量。大多数公司的互联网具有小的跨距。

16（无穷大）。在下一次广播时， R_1 将报告这一更开销的路由。但是，除非协议包含了额外的机制预防此类情况，否则可能有其他路由器在 R_1 之前先广播了其路由。具体来说，假设 R_2 正好在 R_1 与网络 1 的连接失效后通告其路由。如果是这样， R_1 就会收到 R_2 的报文，并对此使用通常的距离向量算法：它注意到 R_2 通告了到网络 1 的开销更低的路由，于是通过计算得知现在到达网络 1 需要 3 跳（ R_2 到网络 1 的开销是 2 跳，再加上到 R_2 的 1 跳）。然后，在路由表中装入新的路由，其下一跳是 R_2 。图 15.4(b) 描绘出这一结果。此刻，如果 R_1 和 R_2 中的任何一个收到发往网络 1 的数据报，就会彼此来回传递该数据报，直到其生存时间的计数器超时。

这两个路由器后续的 RIP 广播不能迅速解决这个问题。在下一轮交换路由信息的过程中， R_1 通告它的路由表中的各个表项。而 R_2 得知 R_1 到网络 1 的距离是 3 之后，计算出自己路由的新距离值为 4。到第三轮时， R_1 从 R_2 收到包含增大距离值的报告，然后把自己的路由表中的相应距离值增加到 5。如此循环往复，直至距离值到达 RIP 无穷大。

15.5 解决慢收敛问题

对于图 15.4 中的例子，可以使用**水平分割更新**（split horizon update）技术来解决慢收敛问题。在使用水平分割技术时，路由器不会把关于某路由的信息通过获得该路由的接口传送回去（即从一个接口学到的路由不再从这个接口发送回去）。在这个例子中，水平分割不允许路由器 R_2 把它到网络 1 的路由再通告给 R_1 ，因此如果 R_1 与网络 1 的连接失效， R_1 必须停止通告它到网络 1 的路由。使用水平分割方法，不会在本例的网络中出现路由选择环路，而是会在几轮路由更新之后，使所有路由器都知道网络 1 是不可达的。但是，水平分割更新技术并不能解决所有拓扑结构中的路由选择环路确问题，本章习题中就提供了一个这样的拓扑结构。

考虑慢收敛问题的另一种方法是使用信息流的概念。如果路由器通告了一条到某个网络的短路由，所有接收路由器将迅速地把该路由添加到自己的表中。如果路由器停止通告某条路由，协议在定该路由不可达之前，会采用某种超时机制，一旦发生超时，路由器寻找一条替代路由并开始传播该信息。遗憾的是，路由器并不知道这条替代路由是否依赖于刚刚消失的路由。因此，否定信息通常不会很迅速地传播。有一句话很适合描述这种现象：

好消息传得快，坏消息传得慢。

解决慢收敛问题的另一种技术是采用**抑制**（hold down）法。抑制法迫使参与协议交互的路由器在收到关于某网络不可达的报文后，在一段固定时间内忽略关于该网络的信息。抑制期的典型时长为 60 s。该技术的想法是，等待足够长的时间，以确保所有机器都收到坏消息，而不会错误地接受过时的报文。需要指出的是，所有参与 RIP 的机器都要遵循抑制策略，否则仍然会发生路由选择环路现象。抑制技术的缺点是：如果出现了路由选择环路，那么在抑制期内这些路由选择环路仍然会维持下去。更严重的是，即便存在替代路由，抑制期间的所有不正确路由都会保留下来。

解决慢收敛问题的最后一一种技术是**毒性逆转**（poison reverse）。当一条连接消失后，通告该连接的路由器在几个更新周期内会保留相应的路由表项，并在广播该连接对应的路由时将路由开销指定为无限长。为提高毒性逆转法的效率，我们应该把它与**触发更新**（triggered update）技术结合使用。触发更新技术使得路由器在收到坏消息之后立即进行广播，而不必等待下一个广播周期。通过立即发送更新信息，路由器尽最大可能减少了因为相信好消息而受到错误影响的时间。

遗憾的是，虽然触发更新技术、毒性逆转技术、抑制技术和水平分割技术能够解决一些问题，但它们又带来了一些新问题。例如，当许多路由器共享一个公共网络时，如果采用触发更新技术，一个广播就能改变这些路由器的路由表，从而引发一轮新的广播。如果第二轮广播改变了路由表，

又会引起更多的广播。由此可能产生广播雪崩^①。

使用广播技术（这样可能产生路由选择环路）和抑制技术防止慢收敛问题，都会使 RIP 在广域网上的工作效率极低。广播要耗费大量宝贵的带宽。即使不出现广播雪崩现象，所有机器定期进行广播也意味着通信量随着路由器数量的增多而增加，而在线路容量有限的情况下，路由选择环路可能就是致命的问题。路由器需要交换一些路由选择报文，以打破路由选择环路，但当环路中循环转发的分组造成线路饱和之后，就很难做到这一点，甚至不可能做到。同样，在广域网中，抑制期如果太长，就会使更高层协议使用的计时器超时，从而导致连接中断。

15.6 RIP1 报文格式

RIP 报文大致可分为两类：路由信息报文和用于请求信息的报文。它们都使用同样的格式，由固定的首部和后面可选的网络和距离序偶列表组成。图 15.5 给出了 RIP1 所用的报文的格式。

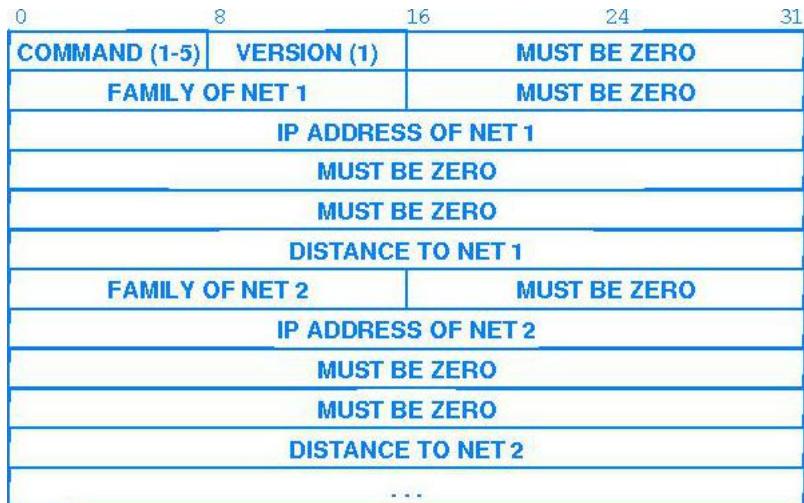


图 15.5 RIP 版本 1 的报文格式。在 32 位的首部之后，报文包含了一组序偶，每个序偶由一个网络 IP 地址和一个到该网络的整数距离值构成

在图 15.5 中，命令（COMMAND）字段指明一种操作；现在使用的只有几个命令：

路由器或主机可以通过发送请求命令向另一个路由器询问路由信息。路由器使用响应命回答请求。但是，大多数情况下路由器不经请求就会定期广播响应报文。版本（VERSION）字段包含了协议的版本号（本例中是 1），接收方检测该字段，以确保自己能够对报文做出正确解释。

^① 为了有助于避免底层网络上发生碰撞，RIP 要求每个路由器在发送触发更新报文前，等待一小段随机长度的时间。

15.7 RIP2 地址约定

RIP 的通用性也体现在它传输网络地址的方式上。它的地址格式并不局限于供 TCP/IP 用户使用，还能与多个网络协议族一起使用。如图 15.5 所示，RIP 报告的每个网络地址可以长达 14 八位组。当然，IP 地址仅需 4 八位组，RIP 定义其余八位组必须为零^①。网络协议族 i (FAMILY OF NET i) 字段指明了解释网络地址应采用的协议族。RIP 使用了 4 BSD UNIX 操作系统中给各个地址协议族指派的值（给 IP 地址指派的值是 2）。

除了正常的 IP 地址之外，RIP 规定地址 0.0.0.0 为默认路由。RIP 在通告的每个路由后都附加了距离度量值，包括默认路由。因此可以让两个路由器以不同的距离值来通告默认路由（例如到互联网其余部分的路由），选择其中的一条作为基本路径，另一条作为备用路径。

RIP 报文中每个条目的最后一个字段是到网络 i 的距离 (DISTANCE TO NET i) 字段，它含有一个到达指定网络距离的整数值。距离值是用路由器的跳数来衡量的，但是它的取值范围限制为 1~16，其中距离 16 代表无限远（即不存在路由）。

15.8 RIP 路由的解释和聚合

因为 RIP 最初设计用于分类地址，版本 1 中没有包含关于子网掩码的任何内容。当子网编址加入 IP 协议以后，RIP 版本 1 也进行了扩展，以允许路由器交换子网编址的地址。但是，由于 RIP1 更新报文不包含明确的掩码信息，所以添加了一个重要的限制：在路由更新报文中，路由器可以包含特定主机或特定子网的地址，只要所有接收方能够无二义性地解释其地址即可。具体来说，子网路由只能包含在通过特定网络发送的更新报文中，而这个网络是被划分子网的前缀的一部分，并且这个网络使用的子网掩码和地址使用的子网掩码必须一样。从本质上讲，这个限制表明，RIP1 不能用于传播变长的子网地址或无分类地址。可归纳如下：

因为 RIP1 没有包括明确的子网信息，如果接收方能够按照本地提供的子网掩码无二义性地对地址进行解释，RIP1 才允许路由器发送子网路由。因此，RIP1 只能用于分类地址或定长的子网地址。

当运行 RIP1 的路由器既连接到一个或多个属于网络前缀 N 的子网的网络，又连接到一个或多个不属于 N 的子网的网络，会发生什么情况呢？路由器必须为两种不同类型的接口准备不同的更新报文。通过属于 N 的子网的接口发送的更新报文，能够包含子网路由，但通过其他接口发送的报文则不能包含子网路由，而是需要由路由器 **聚合** (aggregate) 子网信息，并以单个路由的形式通告给网络 N。

15.9 RIP2 扩展和报文格式

对地址解释的限制表明，RIP 版本 1 不能用于传播变长的子网地址或 CIDR 所使用的无分类地址。当定义了 RIP 的版本 2 (RIP2) 之后，协议进行了扩展，每个地址都相应地包含一个明确的子网掩码，这就使版本 2 能够传播任意形式的路由信息。另外，RIP2 的更新报文还包含了明确的下一跳信息，也就是说，RIP2 报告了每个路由的来源，从而可以防止出现路由选择环路和慢收敛问题。

为了防止 RIP 不必要地增加主机 CPU 的负载，RIP2 的设计者还允许 RIP2 使用多播而不是用广播来更新。而且，他们还给 RIP2 分配了固定的多播地址 224.0.0.9，这意味着使用 RIP2 的机器不

^① 设计人员选择把 IP 地址放在第 3 个至第 6 个八位组位置，以确保地址按 32 位对齐。

需要运行 IGMP^①。最后，RIP2 多播仅限于在单个网络中使用。

综合以上特性，RIP 的第二版大大提高了协议的性能，不但在功能上明显增强，而且提高了传输效率和对错误的抵抗能力。

RIP2 使用的报文格式是 RIP1 格式的扩展形式，一些附加的信息占据了原地址字段中未用的八位组。具体来说，每个地址明确包含了下一跳和子网掩码的信息，如图 15.6 所示。

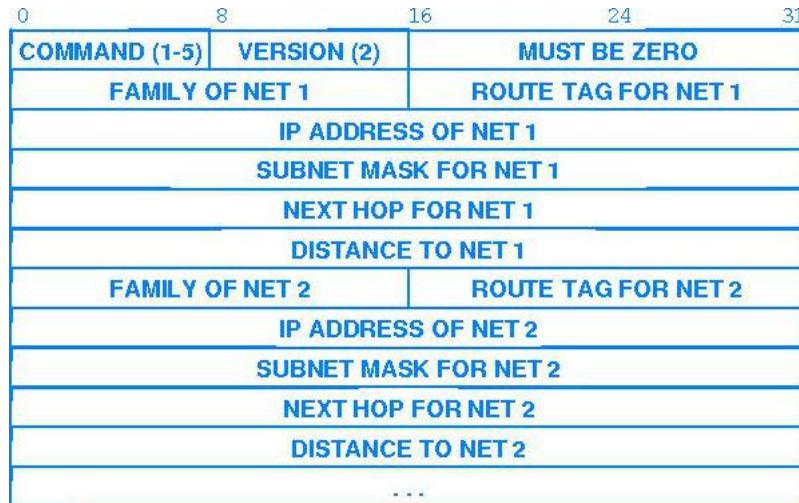


图 15.6 RIP2 报文格式。除了网络 IP 地址和到该网络的整数距离值构成的序偶以外，报文还包含每个地址的子网掩码以及明确的下一跳信息

RIP2 还给每个条目附加了一个 16 位的路由标记（ROUTE TAG）字段。在传输路由时，路由器发送和接收的标记必须相同。因此，该标记提供了传播路由来源之类的额外信息的方式。例如，如果 RIP2 从另一个自治系统得知一个路由，可以使用路由标记字段传播那个自治系统的编号。

因为 RIP2 中的版本号所占位置与 RIP1 中的相同，两种版本的协议可同时在某个给定的路由器上使用，不会相互干涉。在处理传入的报文之前，RIP 软件先检查版本号。

RIP 报文中并没有包含一个明确的长度字段或条目数量，而是假设下层交付机制能告诉接收方传入的报文长度。具体来说，在 TCP/IP 系统中，RIP 报文依赖于 UDP 把报文长度告诉接收方。RIP 使用了 UDP 端口 520。虽然可以在其他 UDP 端口发起 RIP 请求，但请求报文的 UDP 目的端口总是 520，同时这也是 RIP 广播报文发送的源端口。

15.10 RIP 跳计数的缺点

用 RIP 作为内部网关协议，在两个方面限制了路由选择：第一，RIP 要求路由选择使用跳计数值。第二，由于用一个小的跳计数值来表示无穷大，RIP 限制了使用它的互联网的规模。具体来说，RIP 把互联网的跨距（即跨越互联网的最大距离）限制为 16。也就是说，使用 RIP 的互联网在任何两台主机之间最多有 15 个路由器。

注意，对于网络跨距的限制，既不是对路由器总数的限制，也不是对路由器密度的限制。实际上，大多数校园网的跨距较小，即使此类网络有许多路由器，但由于采用分层拓扑结构而使网络跨距不大。例如，考虑一个典型的公司内联网。公司内联网大多采用层次结构，由高速主干网络和多个路由器组成，每个路由器把一个工作组连接到主干网络上，而每个工作组占据一个局域网。尽管公司可以包含几十个工作组，但整个内联网的跨距只是 2。即使对每个工作组加以扩展，使它再包含一个路由器，与一个或多个另外的局域网连接，最大跨距也只会增加到 4。类似地，把层次再扩

^① 第 16 章将介绍**网际组管理协议**（Internet Group Management Protocol，简称 IGMP）。

展一级也只会把跨距增加到 6。因此，RIP 施加的限制只会影响大型的自治系统，或影响不按层次结构组织的自治系统。

然而，即使在最佳情况下，跳计数也仅仅提供对网络响应能力和容量的粗略估计。因此，使用跳计数并不总能得到最小时延或最高容量的路由。而且，基于最小跳计数来计算路由有严重的缺点，它会使路由选择相对固定不变，因为路由无法对网络负载的变化做出响应。后面的几节将讲解另一种距离度量，并将解释跳计数度量尽管存在一些限制，但依然保持流行的原因。

15.11 时延度量 (HELLO 协议)

虽然 HELLO 协议现在已废弃不用，但它曾经是在因特网上使用的一个 IGP 实例，它使用的路由度量不是跳计数。HELLO 提供两个功能：对一组路由器的时钟进行同步，并使每台机器都能计算出到目的地的最短时延路径。因此，HELLO 报文携带路由信息以及时间戳信息。

HELLO 的基本原理很简单：采用距离向量算法传播路由信息，不是让路由器报告跳计数，而是报告到目的站的时延的估计值。由于时钟已经同步，路由器就可以估算每个到达分组的时延：在传输一个分组之前，发送方把当前时钟值复制到分组，接收方则用当前时钟值减去收到分组中的时钟值，从而计算出分组的时延。

HELLO 协议使用标准的距离向量方法进行更新。当报文从机器 X 到达时，接收方检查报文中的每个条目，如果有一条通过 X 的路由比当前路由开销更低（即从接收方到 X 的时延加上从 X 到目的站的时延小于当前接收方到目的站的时延），则把下一跳改为 X。

15.12 时延度量和振荡

从表面上看，使用时延作为路由度量比用跳计数会得到更好的路由。事实上，HELLO 在早期因特网的主干网上确实很好地发挥了作用。但是，大多数协议中没有使用时延作为距离度量有一个重要原因：不稳定性。

即使两条路径有同样的特征，任何快速改变路由的协议都可能变得不稳定。与跳计数不同，时延不是固定的，由此产生了不稳定性。硬件时钟偏差、测量期间的 CPU 负载加大、链路层同步造成的比特延迟，这些都会在时延测量方面产生轻微差异。因此，如果路由选择协议对于时延上的微小差异迅速做出反应，就会产生一种“**二级振荡**”(two-stage oscillation) 的效应，通信量在两条可互相替代的路径之间来回切换。在第一阶段，路由器发现路径 A 上的时延稍微小一些，就突然把通信量切换到这条路径上。在下一轮，路由器发现路径 B 上的时延稍微小一些，又把通信量切换回路径 B。

为了有助于避免振荡，使用时延的协议实现了几种启发式方法。第一，使用前面讨论过的抑制技术来防止迅速改变路由。第二，协议不是尽可能精确地测量并直接比较时钟值，而是把测量值的后几位数字四舍五入掉，或者设置一个最小的**阈值**(threshold)，小于该阈值的差异则忽略不计。第三，协议不对每次测量的时延进行比较，而是不断地计算和比较最近测量值的**平均值**(average)，或者应用“N 中取 K”(K-out-of-N) 的规则，最近 N 次时延测量中至少有 K 次小于当前时延，才能改变路由。

即使采用了启发式方法，在对特征不同的路径上的时延进行比较时，使用时延的协议也会变得不稳定。为了理解其原因，需要知道通信量对时延的巨大影响。如果没有通信量，网络时延很简单，就是硬件把比特从一点传输到另一点所需的时间。但是，随着网络上通信量负载的增加，时延开始增大，因为系统中的路由器需要把等待传输的分组排队。如果负载稍稍超过了 100% 的网络容量，队列就会变得超长，意味着有效时延变成了无限大。归纳如下：

通过网络的有效时延取决于通信量：随着负载增加到 100% 的网络容量，时延也会迅速增长。

因为时延对于负载的变化极其敏感，使用时延作为距离度量的协议会很容易地落入一个**正反馈循环** (positive feedback cycle)，负载中的一个小小的外部变化（例如，一台计算机突发了额外的通信量）就会触发这个循环。增加的通信量会加大时延，从而使协议改变路由。但是，因为路由变化影响了负载，会使时延产生更大的变化，这又意味着协议将再次重新计算路由。由此可见，使用时延的协议必须包含减弱振荡的机制。

前面描述的启发式方法可以解决一种简单情况的路由振荡，即路径具有相同的吞吐率特征，并且负载不会过多。但是，当替代路径的时延和吞吐率特征不同时，启发式方法的效率就会很低。例如，考虑两条路径上的时延：一条通过卫星，另一条通过低容量的串行线（例如 9600 波特的串行线）。在协议的第一阶段，两条路径都是空闲的，看起来好像串行线的时延比卫星的低得多，因而选择串行线传输通信量。因为串行线容量低，它会迅速变得超载，使时延急速上升。在第二阶段，串行线上的时延会比卫星的高得多，因此协议会使通信量离开超载的路径。因为卫星通道具有很大容量，造成串行线路超载的通信量施加在卫星通道上不算太多，这意味着卫星通道上的时延不会随着通信量而变化。在下一轮通信中，没有负载的串行线路上的时延看上去又比卫星路径上的小得多。协议会使路由选择再次发生变化，循环往复。在实际应用中，的确发生过这样的振荡。正如在这个例子中所看到的，这种振荡问题很难进行管理，因为对一个网络影响微小的通信量可能使另一个网络超载。

15.13 将 RIP 和 HELLO 以及 BGP 组合起来

我们已经发现，同一个路由器既可以使用内部网关协议获取本自治系统内部的路由信息，也可以使用外部网关协议向其他自治系统通告路由信息。在理论上，编写一个软件将这两种协议组合在一起，使之具有无需人工干预就能收集路由和通告路由的能力，这应该是很容易的事。但由于技术和政策上的阻碍，实践起来十分困难。

从技术角度讲，内部网关协议，如 RIP 和 HELLO，都是路由选择协议。路由器使用这样的协议，根据从本自治系统内其他路由器获得的信息，更新其路由表。因此，类似 `routed` 这样的实现 RIP 的 UNIX 程序，能够通告从本地路由表中得到的信息，并在收到更新信息时更改本地路由表。RIP 相信同一自治系统中的路由器传来的数据是正确的。

与内部协议不同，BGP 之类的外部协议不信任其他自治系统中的路由器。因此，外部协议并不通告本地路由表中的所有可能路由，而是保持一个网络可达性的数据库，并在发送或接收信息时应用一定的策略约束。忽略这种策略约束会在更大程度上影响路由选择，使互联网的某些部分变得不可达。例如，如果运行 RIP 的自治系统内的一个路由器偶尔广播了一条实际不存在的到普度大学网络的低开销路由，运行 RIP 的其他路由器会接受该路由，并把它添加到自己的路由表中，然后它们会把到普度大学的通信量转给错误的路由器。这样，该自治系统内的主机都不可能到达普度大学。如果 BGP 没有实现策略约束，问题就会更严重。例如，如果自治系统内的一个边界路由器使用 BGP 把这个错误路由传播到其他自治系统，那么对于因特网上的某些部分来说，普度大学就是不可达的了。

15.14 gated：自治系统之间的通信

有一种机制能够提供自治系统之间的接口，即 `gated`^① 程序，它能够理解多种协议（既包括 IGP

^① `gated` 是 “gate daemon”的缩写，发音为 “gate-d”。

也包括 BGP)，并确保能实现策略约束。例如，gated 可以像 routed 程序那样接收 RIP 报文并修改本地计算机的路由表，也可以使用 BGP 通告那些从本自治系统可达的路由。gated 所遵循的规则允许管理员明确指出 gated 可以通告和不能通告哪些网络，以及应该如何报告到那些网络的距离。因此，尽管 gated 程序不是 IGP，但在路由选择问题上仍起着重要作用，因为它表明，建立自动机制来链接 IGP 和 BGP 且不牺牲协议的安全防护是可行的。

gated 的历史很有趣，它最初是由康奈尔大学的 Mark Fedor 创建的，被 MERIT 用于 NSFNET 骨干网。很多科研人员给 gated 贡献了许多新想法，形成了一个行业联盟，最终 MERIT 把 gated 卖给了 Nexthop。

gated 程序通过实现度量变换完成了另一项有用的任务。于是，我们可以很方便地在两个自治系统之间使用 gated，而且还可以很方便地在各自参与一种 IGP 的两组路由器之间的边界上使用 gated。

15.15 开放 SPF 协议

第 13 章讨论过链路状态路由选择算法，该算法使用 SPF 计算最短路径，当网络变大时该算法比距离向量算法更好。为了鼓励采用链路状态技术，IETF 的一个工作组设计了使用链路状态算法的内部网关协议，称为**开放 SPF 协议**（Open SPF，简称 OSPF），这个新协议取得了几项突出成就。

- 该协议名副其实，其规范可在公开发表的文献中找到。该协议成为开放标准，任何人无需支付许可证费用即可将其实现，这就鼓励了众多的软件厂商支持 OSPF。因此，该协议非常流行，广泛替代了以前的各种协议。
- OSPF 包含了**服务类型路由选择**（type of service routing）。管理员可以设置到某个目的站的多条路由，分别对应于某种服务类型或优先级。当为数据报选择路由时，运行 OSPF 的路由器使用 IP 首部中目的站地址和服务类型字段的内容进行路由选择。OSPF 是首批提供按照服务类型选择路由的 TCP/IP 协议之一。
- OSPF 提供了**负载均衡**（load balancing）功能。如果管理员对于某个目的站规定了若干条开销相同的路由，OSPF 会把通信量均匀地分配给这几条路由。OSPF 也是首批提供负载均衡功能的开放式 IGP 协议之一，RIP 之类的协议只给每个目的站计算一条路由。
- 为了允许一个网点上的网络增长并易于管理，OSPF 允许网点把网络和路由器划分为若干称为**区域**（area）的子集。每个区域是**自包含的**（self-contained）；一个区域的拓扑结构对其他区域来说是隐藏的。因此，给定网点上的多个组就能在使用 OSPF 选择路由的过程中相互协作，同时每个群组保留了独立改变其内部网络拓扑结构的能力。
- OSPF 协议规定，路由器之间交换的任何信息都能被**鉴别**（authenticated）。OSPF 支持各种鉴别机制，而且允许各个区域之间的鉴别机制互不相同。支持鉴别是为了保证只有可信的路由器才能传播路由信息。要理解为什么这会成为一个问题，不妨设想使用没有鉴别的 RIP1 会出现什么情况。如果一个怀有恶意的人使用 PC 机传播 RIP 报文，通告了一条开销很低的路由，那么各个运行 RIP 的路由器和主机就会更改它们的路由，把数据报送到这台 PC 机。
- OSPF 支持**特定主机**（host-specific）的路由、**特定子网**（subnet-specific）的路由以及**无分类路由**（classless routes）和**特定分类网络**（classful network-specific）的路由。在大型互联网中可能会需要这几种类型的路由。
- 为适应可**多点接入**（multi-access）的网络（如以太网），OSPF 扩展了第 13 章中提到的 SPF 算法。我们使用一个点对点的图来描述该算法，运行该算法的路由器定期广播与所有可达邻站之间的链路状态报文。如果有 K 个路由器与以太网相连，它们就会广播 K^2 个可达性报文。为了减少广播的次数，OSPF 使用更复杂的拓扑结构图，图中每个结点代表一个路

由器或一个网络。于是，OSPF 允许每个多点接入的网络都有一个**指定网关** (designated gateway)。即一个指定的路由器，它代表该网络相连的所有路由器发送链路状态报文，该报文报告了从该网络到其所连的全部路由器之间的所有链路状态。

- 为减少**未参与系统** (nonparticipating system) 上的负载，OSPF 还尽可能地利用已有的硬件广播功能来交付链路状态报文。OSPF 通过 IP 多播发送报文，并让 IP 多播机制把多播映射到下层网络。而且，为了消除对 IGMP 的依赖，协议预设了两个 IP 多播地址：224.0.0.5 用于所有路由器；224.0.0.6 用于所有“指定路由器”。为了避免 OSPF 报文送出局域网范围，还要对路由器进行配置，防止它将发送给上述两地址的报文转发出去。
- 为了获得最大的灵活性，OSPF 允许管理员描绘一个虚拟的网络拓扑结构，它是通过摒弃物理连接的细节而抽象出来的。例如，管理员可以在路由图上为两个路由器之间配置一条虚链路，即使这两个路由器之间的物理连接需要通过一个转接网络进行通信。
- OSPF 允许路由器之间交换从其他（外部）网点获得的路由信息。一个或多个连接到其他网点的路由器获得这些网点的信息后，在发送更新报文时可把这些信息包含进来。在报文格式中区分了从外部获得的信息和从本网点内部路由器得到的信息，因此不会混淆路由的来源或可靠性。

15.15.1 OSPF 的报文格式

每个 OSPF 报文的开始部分是长度固定为 24 八位组的首部，如图 15.7 所示。

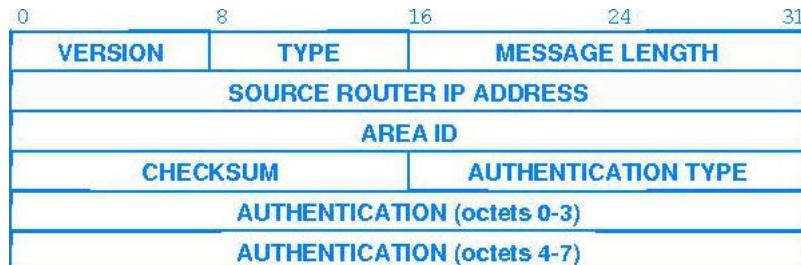


图 15.7 固定长度的 24 八位组 OSPF 报文首部

版本 (VERSION) 字段指出了协议的版本号。类型 (TYPE) 字段指出了报文的类型，可能的报文类型见下表：

源路由器 IP 地址 (SOURCE ROUTER IP ADDRESS) 字段给出了发送方的地址，而区域标识符 (AREA ID) 字段给出了一个 32 位的区域标识号。

由于每个报文可以包含鉴别信息，鉴别类型 (AUTHENTICATION TYPE) 字段指明所使用的鉴别机制 (目前，0 代表没有鉴别，1 代表简单地使用口令进行鉴别)。

15.15.2 OSPF 的 HELLO 报文格式

OSPF 定期在每个链路上发送 HELLO 报文，以便建立并测试邻站的可达性。图 15.8 给出了该报文的格式。网络掩码 (NETWORK MASK) 字段包含了发送报文所用网络的掩码 (关于掩码的细节参见第 9 章)。路由器失效间隔 (ROUTER DEAD INTERVAL) 字段的内容是以秒计的时限值，超过该时间还未反应的邻站就被认为已经失效 (dead)。HELLO 间隔 (HELLO INTERVAL) 字段是以秒计的 HELLO 报文发送的间隔时间。路由器优先级 (GWAY PRIO) 字段是一个整数，表示该路由器的优先级数，可用于选择一个后备的指定路由器。指定路由器 (DESIGNATED ROUTER) 字段和后备指定路由器 (BACKUP DESIGNATED ROUTER) 字段包含 IP 地址，它们给出了发送方为传送报文的网络所指定的这两种路由器。邻站 i 的 IP 地址 (NEIGHBOR $_i$ IP ADDRESS) 字段给出了发送方最近收到的 HELLO 报文中所有邻站的 IP 地址。

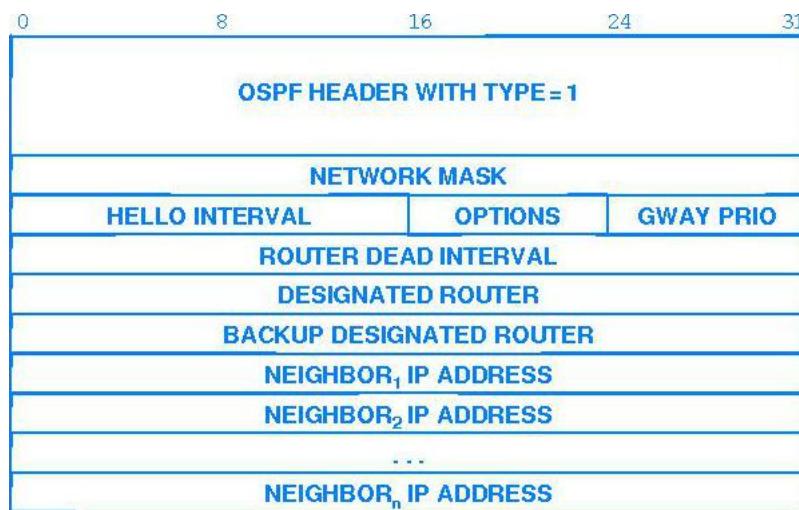


图 15.8 OSPF 的 HELLO 报文格式。一对相邻的路由器定期交换这样的报文来测试可达性

15.15.3 OSPF 的数据库描述报文格式

路由器通过交换 OSPF 数据库描述 (database description) 报文来初始化它们的网络拓扑数据库。在交换过程中，一个路由器作为 **主方** (master)，另一个作为 **从方** (slave)。从方通过发送响应报文对收到的数据库描述报文进行确认。图 15.9 给出了报文的格式。

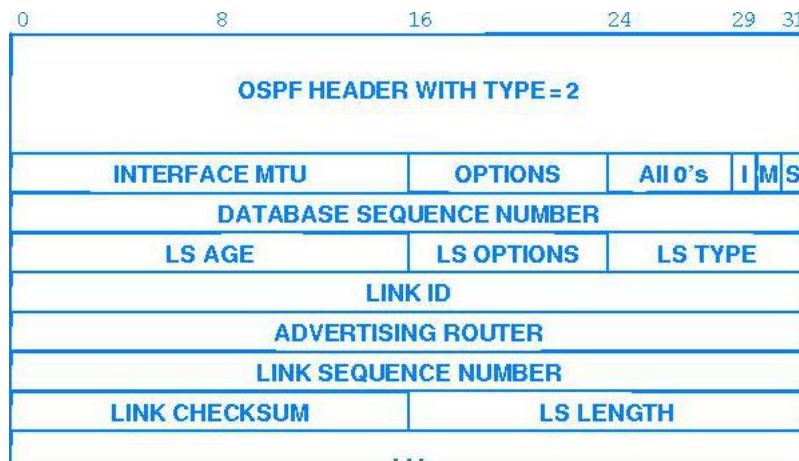


图 15.9 OSPF 数据库描述报文的格式。对各个指定的链路来说，从链路年龄 (LS AGE) 字段开始的各字段格式一样，重复使用

拓扑数据库可能会很大，所以要通过使用 I 比特和 M 比特把它划分成若干报文。在初始报文

中，I比特置1，如果存在后续报文，则将M比特置1。S比特指出该报文是由主方发送的（置1）还是由从方发送的（置0）。数据库序号（DATABASE SEQUENCE NUMBER）字段对报文序列编号，使接收方能够分辨出是否有丢失的报文。在初始报文中，该序号值是一个随机整数R，而后续报文的序号就从R开始递增。

INTERFACE MTU字段给出了不用分段就可以在网络接口上传送的IP数据报的最大长度值。从链路年龄（LS AGE）到链路长度（LS LENGTH）之间的一些字段描述了网络拓扑结构中的一条链路，这些字段格式对每个链路都是一样的。链路类型（LS TYPE）字段采用下表中的值来描述链路。

链路标识符（LINK ID）字段给出了链路的标识值（可以是路由器或网络的IP地址，根据链路类型而定）。

链路年龄（LS AGE）字段用于将报文排序，它给出了该链路建立后所经过的以秒为单位的时间值。通告路由器（ADVERTISING ROUTER）字段指出了通告该链路的路由器的地址，链路序号（LINK SEQUENCE NUMBER）字段包含了由发送该报文的路由器生成的整数编号，以确保报文不会丢失且不会无序交付。链路检验和（LINK CHECKSUM）字段为链路信息是否被破坏提供了进一步的验证措施。

15.15.4 OSPF 的链路状态请求报文格式

路由器与邻站交换了数据库描述报文之后，可能发现自己的数据库中某部分信息已经过时了。为了请求邻站提供更新信息，路由器发送**链路状态请求**（link status request）报文。这个报文列出了请求查询特定链路的信息，如图15.10所示。邻站以这些链路的最新信息作为响应。图中显示出的三个字段对于每个请求交换状态的链路是重复使用的。如果请求的列表太长，就需要发送多个请求报文。

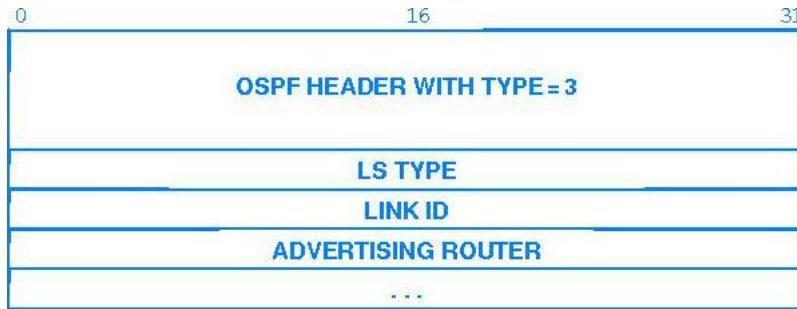


图15.10 OSPF的链路状态请求报文的格式。路由器把这样的报文发送到邻站，请求一组特定链路的当前信息

15.15.5 OSPF 的链路状态更新报文格式

路由器使用**链路状态更新**（link status update）报文来广播链路的状态。每个更新由一系列通告

组成，如图 15.11 所示。

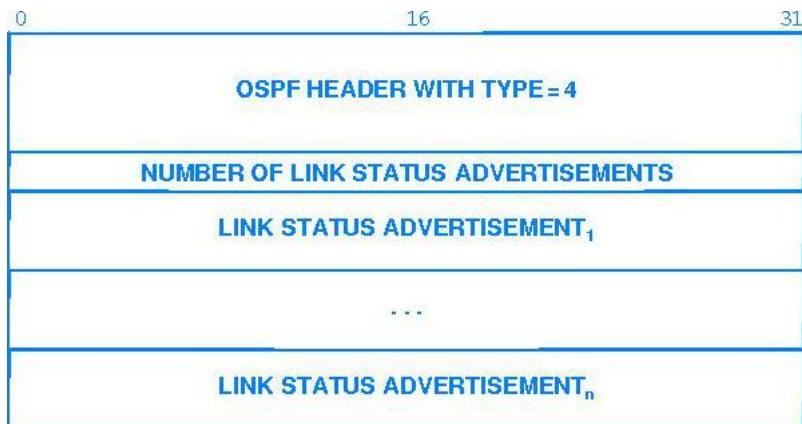


图 15.11 OSPF 的链路更新报文格式。路由器发送这样的报文来向其他路由器广播与其直接相连的链路的信息

每个链路状态通告的首部格式如图 15.12 所示。各字段的值与数据库描述报文中的相同。

在链路状态首部之后有四种可能的格式，分别用来描述：从一个路由器到某个给定区域的链路，从一个路由器到某个特定网络的链路，从一个路由器到属于单个 IP 网络所划分子网的物理网络的链路（参见第 9 章），以及从一个路由器到位于其他网点中的网络的链路。在所有的情况下，链路状态首部中的 LS TYPE 字段指出了使用的是哪种格式。由此，收到链路状态更新报文的路由器，清楚地知道所描述的目的站是在本网点的内部还是外部。

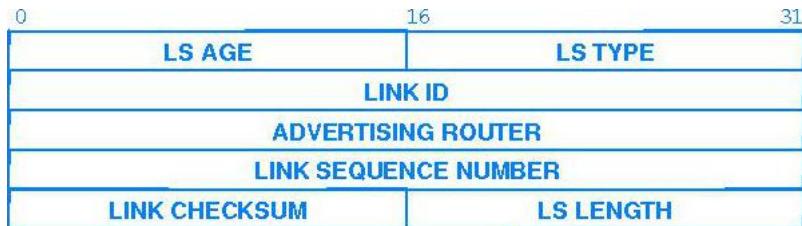


图 15.12 链路状态通告的首部格式

15.16 利用部分信息选择路由

我们在讨论互联网路由器体系结构和路由选择时，是从讨论**部分信息** (partial information) 的概念入手的。主机可以只使用部分信息来选择路由，因为它可以依靠路由器来选择路由。很显然，并不是所有路由器都具有完整的信息。大多数自治系统使用单个路由器与其他自治系统相连。例如，如果一个网点与因特网相连，则至少需要一个路由器把该网点连接到 ISP。自治系统内的路由器知道自治系统内部的目的站，但要使用默认路由把其他所有通信量都发送给 ISP。

对路由器的路由表加以研究，就很容易理解如何使用部分信息来进行路由选择。处于因特网核心位置的路由器拥有到所有可能目的站的路由信息全集，这样的路由器不使用默认路由。除了位于因特网核心的 ISP 中的路由器之外，其他路由器通常没有完整的路由信息，它们依靠默认路由来处理那些自己不知道的网络地址。

大多数路由器使用默认路由会导致两个结果。一个结果是可能无法检测本地路由选择的错误。例如，如果自治系统内的某台机器错误地把一个分组转发到了该系统之外，而不是转发给内部的路由器，这个外部系统就会把分组转发回来（也许会发送到一个不同的入口点）。因此，即使路由选择不正确，由于分组在外部系统转了一圈又回来了，通信看起来好像保持了连通。这个问题对于使

用高速局域网的小型自治系统并不太严重，但在使用线路速率相对较低的广域网时，错误的路由就会造成灾难性的后果。另一个结果具有正面的意义，尽可能地使用默认路由，使大多数路由器交换的路由更新报文比包含完整信息的路由更新报文短得多。

15.17 小结

管理员必须选择自治系统内部的本地路由器之间传输路由信息的方式。手工维护路由信息的方法仅仅适用于缓慢变化、互连较少的小规模互联网；多数互联网要求使用自动过程来自动发现并更新路由。在一个管理者控制下的两个路由器，通过运行内部网关协议（IGP）来交换路由信息。

IGP 或者实现距离向量算法，或者实现链路状态算法（又称最短路径优先，即 SPF 算法）。我们讨论了 3 个具体的 IGP 协议：RIP 协议、HELLO 协议和 OSPF 协议。RIP 是一个距离向量协议，由 UNIX 程序 `routed` 来实现，是最流行的 IGP 之一。它使用水平分割技术、抑制技术和毒性逆转技术来消除路由选择环路和计数到无穷大的问题。HELLO 协议虽已废弃不用，但是仍然很有意义，因为它是采用时延而不是跳计数作为距离度量的距离向量协议的实例。我们讨论了时延作为路由度量的缺点，并指出：尽管在路径具有相同吞吐率特征时使用启发式方法可防止不稳定现象的产生，但在路径特征不同时仍会产生长期不稳定的现象。最后讲到，OSPF 是一种实现链路状态算法的协议。

我们还讲到，`gated` 程序提供了一个协议开发平台，它提供 RIP 之类的内部网关协议与 BGP 之类的外部网关协议之间的接口，并能使自治系统内部的路由信息采集过程和向其他自治系统通告路由的过程自动化。

15.18 深入研究

Hedrick [RFC 1058] 讨论了交换路由信息的一般算法，并包含 RIP1 的标准规范。Malkin [RFC 2453] 给出了 RIP2 的标准。Mills [RFC 891] 提供了 HELLO 协议的文档。Mills and Braun [1987] 考虑了时延与跳计数度量之间的转换问题。Moy [RFC 1583] 对 OSPF 规范及其设计动机进行了详细描述。Moy [RFC 2328] 提供了 OSPF 版本 2 的详细文档。Fedor [June 1988] 详细介绍了 `gated` 程序。

15.19 习题

1. RIP 支持哪些网络协议族？提示：阅读 4.3 BSD UNIX 程序员手册中关于网络的章节。
2. 考虑一个大型自治系统使用 HELLO 之类的基于时延的内部网关协议的情况。如果其中某个子组（subgroup）决定在其路由器上使用 RIP，该自治系统会有什么困难？
3. 在 RIP 报文中，每个 IP 地址在 32 位边界上对齐。如果携带 RIP 报文的 IP 数据报从一个 32 位边界开始，那么这些地址还会在 32 位边界上对齐吗？
4. 自治系统可以小到仅有一个局域网，也可以大到由多个远程网络构成。网络大小的差异为什么会使定义标准的 IGP 变得很困难呢？
5. 说明在什么样的环境下水平分割技术能够防止慢收敛问题。
6. 考虑由许多运行 RIP 的局域网所构成的互联网。举出一个例子，说明在收到某网络不可达信息后，即使软件使用“抑制”技术也会造成路由选择环路。
7. 主机能够以主动模式来运行 RIP 吗？为什么？
8. 在什么情况下，使用跳计数作为度量比使用时延作为度量能产生更好的路由？
9. 能否设想出一种情况，在该情况下自治系统选择不通告其内部所有网络的信息？提示：考

虑一个大学的情况。

10. 泛泛地讲, RIP 负责分发本地路由表, 而 BGP 负责分发一张包含若干网络和到达这些网络所用路由器的表格(即路由器可以发送一个 BGP 通告, 它与自己本地路由表中的表项并非完全匹配)。这两种方法的优点各是什么?
11. 思考时延度量与跳计数度量相互转换所用的函数。能充分防止转发环路的函数应该具有哪些特性? 你能给出其必要条件吗? 提示: 参考 Mills and Braun [1987]。
12. 在什么样的环境下 SPF 协议会造成路由选择环路? 提示: 考虑尽最大努力交付的情况。
13. 编写一个应用程序, 把请求发送给运行 RIP 的路由器, 然后显示返回的路由。
14. 仔细阅读 RIP 的规范。在查询的响应报文中报告的路由是否会与路由更新报文中报告的路由不同? 如果不同, 区别是什么?
15. 假设有一个网点使用变长的子网, 说明为什么需要使用 RIP2 才能正确传播路由(即找出一个 RIP2 能正确工作, 但 RIP1 不能正确工作的例子)。
16. 仔细阅读 OSPF 的规范。管理员如何使用虚链路功能?
17. OSPF 允许管理员自己分配许多标识符, 从而在多个网点中可能会出现重复的标识符。如果合并两个运行 OSPF 的网点, 那么应该修改哪些标识符呢?
18. 试比较 4 BSD UNIX 所提供的各个版本的 OSPF 和 RIP 软件。在源代码大小、目标代码大小和数据存储区大小方面有什么不同? 你能得出什么结论?
19. 能否在内部路由器之间使用 ICMP 重定向报文传递路由信息? 为什么?
20. 编写一个程序, 根据你所在单位的互联网情况, 使用 RIP 查询机制从各路由器处获得路由, 并报告任何不一致性的情况。
21. 如果你所在的单位运行着 Zebra 之类的管理多种 TCP/IP 路由选择协议的软件, 请你找到一份配置文件的副本, 并解释其中每项的含义。

第16章 因特网多播

16.1 引言

前面几章定义了 IP 数据报单播时使用的交付机制。本章将探讨 IP 的另一特性：数据报的多点交付。首先需要简单回顾底层的硬件支持，然后详述多点交付的 IP 编址以及路由器在传播必要的路由选择信息时所用的协议。

16.2 硬件广播

许多硬件技术都包含了同时（或近似同时）向多个目的站发送分组的机制。在第 2 章中曾回顾了几种底层网络技术，并讨论了多点交付的最普通方式：**广播**（broadcasting）。广播交付意味着网络向每个目的站交付一个分组的副本。在类似于以太网的总线技术中，广播交付可以由单次分组传输来完成。在由交换机和点到点连接构成的网络中，软件必须有广播的能力，即通过各个独立的连接转发分组的副本，直到所有交换机都收到一份副本为止。

在大多数硬件技术中，计算机通过把分组发送给一个特殊的称为**广播地址**（broadcast address）的保留目的地址，来指定广播交付。例如，以太网硬件地址由 48 比特标识符构成，每一比特均为“1”的地址表示广播地址。每台机器上的硬件都可以识别本机的硬件地址以及广播地址，对目的地址为这两者之一的传入分组都接收。

广播的主要缺点是对资源的消耗，除了使用网络带宽外，每个广播都会让网络上的所有计算机动用计算资源。例如，似乎可以设计另外一种互联网协议族，它在本地网上就使用广播来交付数据报，而依靠 IP 软件来丢弃并非发往本机的数据报。问题是这种方法效率太低，网络上的所有计算机都必须接收和处理每一个数据报，即使它们将会丢弃到达的大多数数据报。因此，TCP/IP 的设计者使用单播编址和类似 ARP 这样的地址绑定机制来消除广播交付。

16.3 多播的硬件起源

有些硬件技术支持另一种不如广播那么普遍应用的多点交付方式，称为**多播**（multicasting）。与广播方式不同，多播方式允许每个系统自己选择是否参与到某个特定的多播群组中。典型情况下，硬件技术会保留大量的用于多播的地址，当一组机器之间要进行通信时，它们就选择一个特定的**多播地址**（multicast address）。在配置好用以识别该多播地址的网络接口硬件后，该组中的所有机器都会收到传送给这个多播地址的任何分组的副本。

从概念上讲，其他任何形式的编址都可以看成是一种特殊的多播编址。例如，我们可以认为常规的**单播地址**（unicast address）就是每组中仅含有一台计算机的一种特殊多播编址形式。类似地，我们可以把定向广播地址看成是特定网络上的所有计算机都是多播群组成员的多播编址形式，而其他多播地址则可以对应任意的机器群组。

话虽这么说，但是多播并不能替代其他常规编址形式，因为其转发和交付的底层机制实现上是有本质区别的。单播和广播地址识别的是连到某个物理网段的一台或一组计算机，因此转发所依靠的是网络拓扑结构。多播地址标志了任意一组接收方，因此转发机制必须把分组传播到所有网段。例如，考虑由自适应网桥连接两个局域网段，且该网桥已经掌握了各主机地址。如果在网段 1 上的

一台主机给同样位于网段 1 的另一台主机发送了一个单播帧，网桥不会把该帧转发给网段 2。但是，如果主机使用了多播地址，网桥就会转发该帧。因此，可以得出结论：

尽管把多播编址看成是包含了单播和广播地址的一般化形式好像会有帮助，但底层转发和交付机制会使多播效率更低。

16.4 以太网多播

以太网提供了硬件多播的最好例子。以太网有一半的地址都是为多播而保留的：使用高位八位组的低位比特来区分常规单播地址(0)和多播地址(1)。使用**线分十六进制记法**(dashed hexadecimal notation)^①来描述，多播地址为：

01-00-00-00-00-00₁₆

在以太网接口板初始化后，就能够接受发往该计算机的硬件地址或者以太网广播地址的分组。而使用设备的驱动程序软件可以对该设备重新进行配置，使它还能识别一个或多个多播地址。例如，假设驱动程序配置了下面的以太网多播地址：

01-5E-00-00-00-01₁₆

配置完成后，接口就能够接受发往该计算机的单播地址、广播地址或这个多播地址的分组（硬件会忽略发往其他多播地址的分组）。下面几节会讲解 IP 如何利用多播硬件以及具有特殊含义的多播地址。

16.5 IP 多播

IP 多播(IP multicasting)是对硬件多播的互联网抽象。它仍然表示允许到达一个主机子集的传输，但它的概念更广泛，允许该子集跨越整个互联网上的任意物理网络。在 IP 术语中，这个子集称为**多播群组**(multicast group)。IP 多播具有下列一般特征：

- **群组地址**。每个多播群组有唯一的 D 类地址。少数几个 IP 多播地址是由因特网管理机构永久分配的，分别对应几个永久存在的群组，即使这些群组当前没有成员。其他地址是临时性的，可供公开使用。
- **群组数量**。IP 最多可同时提供 2^{28} 个多播群组的地址，因此实际群组数量受路由选择表的大小限制，而不是受编址的限制。
- **动态群组成员关系**。一台主机可在任何时候加入或退出一个 IP 多播群组。另外，一台主机可以参加任意数目的多播群组。
- **硬件的使用**。如果底层网络硬件支持多播，IP 就使用硬件多播来发送 IP 多播。如果硬件不支持多播，IP 则使用广播或单播来交付 IP 多播。
- **网络间转发**。因为 IP 多播群组中的成员可以连到多个物理网络上，所以需要特殊的**多播路由器**(multicast router)来转发 IP 多播，通常这种能力是附加在常规路由器上的。
- **交付语义**。IP 多播与其他 IP 数据报交付一样使用尽最大努力交付机制，这意味着多播数据报可以丢失、延迟、重复或无序到达。
- **成员关系和传输**。任意主机都可以向任意一个多播群组发送数据报，群组的成员关系只用于确定主机是否接收发给某个群组的数据报。

^① 线分十六进制记法用两个十六进制数位代表每个八位组，八位组和八位组之间用短破折线分隔。只有在没有二义性的情况下，下标 16 才可以省略。

16.6 概念性组成部分

一般目的的互联网多播系统需要有三个概念性组成部分：

1. 多播编址方法
2. 有效的通知和交付机制
3. 有效的网络间转发工具

对于整体设计，需要解决许多目标、细节和约束问题。例如，除了给群组提供足够的地址之外，多播**编址方法**（addressing scheme）必须达到两个相互冲突的目标：允许分配地址时的本地自治，同时又能定义具有全局含义的地址。类似地，主机需要一种**通知机制**（notification mechanism），把自己参与的多播群组通知给路由器，路由器需要一种**交付机制**（delivery mechanism），把多播分组传输给主机。这里又有两种可能性：如果硬件支持多播，我们则希望这个系统尽量有效利用硬件多播，但也允许 IP 多播在不支持硬件多播的网络上交付。最后，多播**转发工具**（forwarding facility）才是设计上的最大挑战。我们的目标是一种既有效且具有动态性的方法，它应该能够沿最短路径转发多播分组，而不应该向一个不可达群组成员的路径发送数据报副本，并且还要允许主机在任何时候参加或退出群组。

IP 多播技术包括了所有这三个方面。它定义了 IP 多播编址，指定了主机如何发送和接收多播数据报，描述了路由器用于确定网络上的多播群组成员的协议。本章其余部分更详细地讨论了各个方面，首先从编址开始。

16.7 IP 多播地址

前面讲过，IP 多播地址划分为两类：永久分配的和可供临时使用的。永久地址称为**熟知的**（well-known），用于因特网上的主要服务以及基础结构维护（例如，多播路由选择协议）。其他多播地址相应于**过渡多播群组**（transient multicast group），需要使用时则创建，群组成员为 0 时则丢弃。

与硬件多播类似，IP 多播使用了数据报的目的地址来规定数据报必须通过多播进行交付。IP 保留了 D 类地址用于多播，其形式如图 16.1 所示。



图 16.1 用于多播的 D 类 IP 地址的格式。比特 4~31 标志了特定的多播群组

地址的前 4 比特的内容是 1110，指出这是一个多播地址。其余的 28 比特标识了特定的多播群组，其中不再有结构层次。具体来说，群组字段并未划分哪些比特用来标识群组的来源或所有者，也没有包含管理信息，例如是否所有成员都处于同一个物理网络。

以点分十进制记法表示时，多播地址的范围是：

224.0.0.0 至 239.255.255.255

许多地址空间部分已经分配了特殊含义。例如，最低地址 224.0.0.0 是保留未用的，不能分配给任何群组。224.0.0.1 永久地分配给了**全系统群组**（all systems group），224.0.0.2 永久地分配给了**全路由器群组**（all routers group）。全系统群组包含参与 IP 多播的一个网络上的所有主机和路由器，而全路由器群组只包含参与的路由器。一般说来，这两个群组都用于控制协议，并且发往这些地址的数据报只到达与发送方处于同一本地网络的机器。不存在对应于互联网中所有系统或路由器的 IP 多播地址。不仅如此，此后一直到 224.0.0.255 的地址都限制在单个网络上使用（即禁止路由器转

发发往该范围地址的数据报，且发送方应当将 TTL 置为 1)。一般情况下只有多播路由协议会用到这些地址。图 16.2 给出了几个永久分配的地址的例子^①。

Address	Meaning
224.0.0.0	Base Address (Reserved)
224.0.0.1	All Systems on this Subnet
224.0.0.2	All Routers on this Subnet
224.0.0.3	Unassigned
224.0.0.4	DVMRP Routers
224.0.0.5	OSPFIGP All Routers
224.0.0.6	OSPFIGP Designated Routers
224.0.0.7	ST Routers
224.0.0.8	ST Hosts
224.0.0.9	RIP2 Routers
224.0.0.10	IGRP Routers
224.0.0.11	Mobile-Agents
224.0.0.12	DHCP Server / Relay Agent
224.0.0.13	All PIM Routers
224.0.0.14	RSVP-Encapsulation
224.0.0.15	All-CBT-Routers
224.0.0.16	Designated-Sbm
224.0.0.17	All-Sbms
224.0.0.18	VRRP
through	
224.0.255	Other Link Local Addresses
224.0.1.0	
through	Globally Scoped Addresses
238.255.255.255	
239.0.0.0	Scope restricted to one organization
through	
239.255.255.255	

图 16.2 几个永久 IP 多播地址分配的例子。其他许多地址都有具体含义

16.8 多播地址语义

IP 对待多播地址的方式与对待单播地址不同。例如，多播地址只可作为目的地址。因此，多播地址从不会出现在数据报的源地址字段，也不会出现在源路由或记录路由选项中。而且，不会生成关于多播数据报的 ICMP 差错报文（例如，目的不可达、源抑制、回送回答或超时）。因此，发往多播地址的 ping 命令是得不到回答的。

禁止 ICMP 差错报文的规则令人惊讶，因为 IP 路由器的确也遵守多播数据报首部中的生存时间字段规则。和平时一样，每个路由器将该计数减 1，如果计数到 0 则丢弃数据报（但不会发送 ICMP 报文）。我们将会看到，有一些协议使用生存时间计数作为一种限制数据报传播的方式。

16.9 把 IP 多播映射到以太网多播

虽然 IP 多播标准没有覆盖所有类型的网络硬件，但它的确指定了如何将 IP 多播地址映射到以太网多播地址。这个映射是有效且容易理解的：

^① 除了这里的例子之外，233.0.0.0/8 之间的多播地址也被建议为自治系统内部多播保留。

要把 IP 多播地址映射为相应的以太网多播地址，只需将 IP 多播地址的低位 23 比特放到特殊的以太网多播地址 $01\text{-}00\text{-}5E\text{-}00\text{-}00\text{-}01_{16}$ 的低位 23 比特上。

例如，IP 多播地址 224.0.0.2 变成以太网多播地址 $01\text{-}00\text{-}5E\text{-}00\text{-}00\text{-}02_{16}$ 。

有趣的是，这种映射并不是唯一的。因为 IP 多播地址中有 28 个有效比特用来标识多播群组，可能同时有多个群组映射到一个以太网多播地址。设计者采用这种方法作为折中。一方面，使用 28 比特中的 23 比特作为硬件地址，已经包括了绝大部分多播地址。这个地址集合已经足够大了，因此两个群组所选地址的低位 23 比特完全相同的概率很小。另一方面，安排 IP 使用以太网多播地址中的固定部分会使调试更容易，并且还能消除 IP 与其他共享以太网的协议之间的干扰。这种设计方法的后果是，主机有可能会收到终点不是本机的某些多播数据报，因此 IP 软件必须仔细检查所有传入数据报的地址，以便丢弃那些不需要的数据报。

16.10 主机和多播交付

前面讲过，IP 多播既可用在单个物理网络上，也可用在互联网上。对于前一种情况，主机只要直接把数据报放在一帧中，并使用接收方正在监听的一个硬件多播地址，就可以把它发给目的主机。对于后一种情况，特殊的**多播路由器**（multicast router）负责在网络间转发多播数据报，因此主机必须把数据报发给多播路由器。令人惊讶的是，主机并不需要给多播路由器添加一个路由，也不需要指定主机的默认路由。事实上，主机把多播数据报转发给路由器时所用的技术与单播和广播数据报时所用的路由选择查找技术不同，主机只不过使用本地网络硬件的多播能力来传送数据报。多播路由器监听所有 IP 多播传输，如果网络上有多播路由器，它将会接收该数据报，如果需要，则将其转发到另一个网络。因此，本地多播和非本地多播的主要区别在于多播路由器，而不在于主机。

16.11 多播作用域

多播群组的**作用域**（scope）指的是群组成员的范围。如果所有成员都在同一个物理网络上，我们就认为该群组的作用域限制在一个网络内。类似地，如果群组的所有成员都在一个机构内，则认为该群组的作用域限制在一个机构内。

除了群组的作用域，每个多播数据报也有一个作用域，定义为该多播数据报将传播通过的网络集合。非正式地讲，将数据报的作用域称为它的**范围**（range）。

IP 使用两种技术来控制多播作用域。第一种技术依靠数据报的**生存时间**（Time-To-Live，简称 TTL）字段控制范围。通过把 TTL 设置成一个较小的值，主机就能限制该数据报转发的距离。例如，对于同一网络上的主机和路由器之间通信的控制报文，标准规定其 TTL 必须为 1。这样做的结果是，路由器永远不会转发携带控制信息的任何报文，因为 TTL 超时会让路由器丢弃该数据报。类似地，如果一台主机上运行着的两个应用程序要使用 IP 多播进行处理器间的通信（例如为了测试软件），就可以设 TTL 为 0，防止数据报离开本主机。可以使用较大的 TTL 连续值扩展作用域的概念。例如，一些路由器厂商建议把一个网点的路由器配置为限制多播数据报离开本网点，除非数据报的 TTL 值大于 15。我们认为，在数据报首部中使用 TTL 字段可以对数据报的作用域提供非常大的控制作用。

第二种控制作用域的技术称为**确定管理范围**（administrative scoping），它由一些保留的地址空间组成，这些地址空间仅用于某网点或某机构内部的本地群组。按照标准，因特网上的路由器禁止转发其地址在受限空间中的数据报。因此，为了防止群组成员之间的多播通信意外地到达组外，机构可以为群组分配一个具有本地作用域的地址。

16.12 扩展主机软件以处理多播

如图 16.3 所示，主机可以按三种级别参与 IP 多播。

Level	Meaning
0	Host can neither send nor receive IP multicast
1	Host can send but not receive IP multicast
2	Host can both send and receive IP multicast

图 16.3 参与 IP 多播的三种级别

为了使主机能够发送 P 多播，所需进行的修改并不复杂，但是要把主机软件改进成能够接收 IP 多播数据报则较为复杂。要发送一个多播数据报，应用程序必须能够提供一个多播地址作为其目的 IP 地址。而要接收多播数据报，应用程序则必须能够声明自己要加入或退出某个多播群组，并且协议软件必须向每个参与的应用程序转发一份发往该群组的已收到的数据报。此外，下一节将会看到，主机必须运行一个协议，把自己的群组成员状态通知给本地多播路由器。这些复杂性大多来源于一个基本的思想：

主机加入特定网络上的特定 IP 多播群组。

也就是说，拥有多个网络连接的主机可以加入某个特定网络的多播群组，而不是其他网络上的多播群组。若要理解为何将群组成员关系与网络联系起来，就要想到在本地机器组合之间可以使用 IP 多播。主机可能要使用多播应用程序与某个网络上的机器而不是另一个网络上的机器进行通信。

由于群组成员关系与具体的网络相关，所以软件必须为与该机器相连的每个网络独立地保留多播地址列表。此外，应用程序请求加入或退出某个多播群组时，必须指定具体的网络。

16.13 网际组管理协议

为了参与本地网络上的 IP 多播，主机必须使用允许收发多播数据报的软件。为了参与跨越多个网络的多播，主机必须通知本地多播路由器。本地多播路由器与其他多播路由器联系，传递成员信息并建立路由。我们将在后面看到，这个概念与互联网路由器之间的常规路由传播十分相似。

在多播路由器能够传播多播成员信息之前，必须确定本地网络上有一台或多台主机已经决定要加入某个多播群组。为此，多播路由器和实现多播的主机必须使用**网际组管理协议**（Internet Group Management Protocol，简称 IGMP）来进行群组成员信息的通信。因其目前版本为 3，所以本书所讲的这个协议的正式名称为 IGMPv3。

IGMP 类似于 ICMP^①。与 ICMP 一样，它使用 IP 数据报来携带报文，同样也提供了 IP 使用的服务。因此：

虽然 IGMP 使用 IP 数据报来携带报文，但我们认为 IGMP 是 IP 协议整体的一部分，而不是一个独立的协议。

此外，IGMP 是 TCP/IP 的标准之一，所有接收 IP 多播的机器（例如，参与级别 2 的所有主机和路由器）都需要 IGMP。

从概念上讲，IGMP 的工作分为两个阶段。第一阶段：当主机加入一个新的多播群组时，它向

^① 第 8 章讨论了 ICMP，即网际控制报文协议。

该群组的多播地址发送一个 IGMP 报文，以声明其成员关系。本地多播路由器接收到这个报文之后，向互联网上的其他多播路由器传播这个群组成员信息，以建立必要的路由。第二阶段：因为成员是动态的，本地多播路由器周期性探询本地网络上的主机，以便确定各个群组中是否仍然有成员存在。只要有任何一台主机为某个群组做出了响应，路由器就保持该群组的活跃性。如果经过若干次探询后，某个群组中始终没有成员，多播路由器就认为该群组中不再有本网络中的主机，于是停止向其他多播路由器通告该群组的成员信息。

此外，IGMP 允许主机上的应用程序安装一个**源地址过滤器**（source address filter），它的作用是指定主机是否应当容纳或排除来自某个源地址的多播通信量。因此，即使加入了某个多播群组，也可以拒绝来自特定源的发往这个群组的数据报。过滤器的存在有其重要意义，因为 IGMP 允许主机在向本地路由器发送群组成员信息的同时也递交过滤规则设置。如果出现了两个应用程序互相矛盾的情况（即一个应用程序排除某个源，而另一个应用程序却容纳这个源），主机上的软件必须合理地解释这两种过滤规则，然后在本地决定哪些应用程序可以接收特定的数据报。

16.14 IGMP 的实现

IGMP 经过了仔细的设计，以避免添加会造成本地网络拥塞的开销。而且，因为一个网络可以包含多个多播路由器，也可以包含参与多播的多个主机，所以 IGMP 必须避免让这些参与者产生不必要的控制通信量。IGMP 通过如下几种方式使得对网络的影响最小：

1. 主机与多播路由器之间的所有通信都使用 IP 多播。也就是说，当 IGMP 报文封装到 IP 数据报中进行传输时，IP 目的地址就是多播地址（路由器把通用 IGMP 查询发送到全系统群组地址，主机也会发送某些 IGMP 报文给全路由器群组地址，主机和路由器都会把某个特定群组的 IGMP 报文发往该群组的地址）。因此，携带 IGMP 报文的数据报在传输过程中尽量利用硬件的多播能力。其结果是，在支持硬件多播的网络上，不参与 IP 多播的主机就收不到 IGMP 报文。
2. 当探询确定群组成员时，多播路由器不会为每个群组发送单独报文，而是发送一个探询，请求得到关于所有群组的信息^①。默认的探询间隔是 125 s，这意味着 IGMP 不会产生太多通信量。
3. 如果有多个多播路由器连接到同一个网络，它们会迅速而有效地选出一个路由器来负责探询主机成员。因此，当网络中添加更多的多播路由器时，网络上的 IGMP 通信量总量不会增加。
4. 主机并不会同时响应路由器的 IGMP 查询，事实上，每个查询中包含一个值 N，它指定了最大响应时间（默认值是 10 s）。当查询到达时，主机选择 0 至 N 之间的一个随机时延，并在这个时延之后发送响应报文。实际上，如果某主机是多个群组的成员，它就会为每个群组选择不同的随机数，这样主机对路由器的响应就会在 10 s 内随机分布。
5. 多个群组成员的报告可以通过一个分组发送，以节省带宽。

IGMP 之前的版本 IGMPv2 使用了另一种节省带宽的方法。在 IGMPv2 中，网络上的主机会监听这些报告，一旦网络上有某一台主机报告了一个群组中的成员关系，网络上的其他主机就会抑制自己的报告（这些报告是不必要的，因为网络上的路由器已经知道这个组中至少还有一个成员）。在 IGMPv3 中，主机不为每个报告发送一个分组，而是将许多报告集中在一个分组中。因为我们认为一台主机在同一时间只可能是为数不多的几个群组的成员，所以来自一台主机的所有报告完全有可能集中在同一个分组中。因此，IGMPv3 没有对抑制进行管理，而是对传输进行管理，这表示路由器必须掌握每台主机的群组成员关系和过滤器。

^① 这个协议实际上也包括一种允许路由器查询某个指定群组的报文类型。

16.15 群组成员状态的转换

主机上的 IGMP 必须记住所属的每个多播群组的状态以及与每个群组相关的源过滤器^①。可以认为主机保存着一个记录群组成员信息的表。最初该表的各项都是空的，当主机上的某个应用程序加入一个新的群组时，IGMP 软件就为其分配一项，并填入关于该群组的信息，包括该应用程序指定的地址过滤器。当某个应用程序要退出一个群组时，相应的表项就会从系统中删除。在生成报告时，IGMP 软件要咨询这个表，并整理每个群组的所有过滤器，然后形成一个报告。

图 16.4 所示的状态转换图非常清楚地描述了 IGMP 软件响应各种事件的动作。

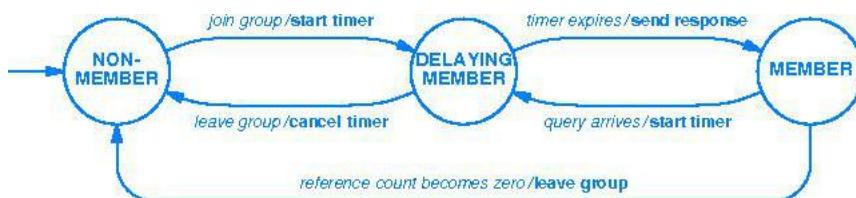


图 16.4 主机的多播群组表中每个表项的三种可能状态以及各状态之间的转换，每次转换都标有一个事件和一个动作。这个状态转换并未显示加入和退出一个群组时所发送的报文

主机为自己当前所属的每个群组维护着一个独立的表项。如图 16.4 所示，当主机首次加入该群组时，或者来自多播路由器的查询到达时，主机将表项转换到延迟成员（DELAYING MEMBER）状态，并选择一个随机的时延。如果群组中的另一台主机在定时器超时之前响应了路由器的查询，该主机就会取消定时器并转换到成员（MEMBER）状态。如果定时器超时，则主机在转换到成员状态之前先要发送一个响应报文。因为路由器每隔 125 s 才产生一次探询，所以可以想象大多数时间主机都保持在成员状态。

图 16.4 中省略了几处细节。例如，如果主机处于延迟成员状态时有查询到达，则协议要求主机复位自己的定时器。更重要的是，为了保持与 IGMPv1 和 IGMPv2 的后向兼容性，版本 3 也能处理版本 1 和版本 2 的报文，因此在同一个网络上可以同时使用 IGMP 的三个版本。

16.16 IGMP 成员关系查询报文格式

IGMPv3 定义了两种报文类型：成员关系查询（membership query）报文和成员关系报告（membership report）报文。成员关系查询报文是由路由器发送给组成员的探询报文。成员关系报告报文是由主机生成的，用来报告主机上的应用程序目前正在使用哪些群组。图 16.5 所示为成员关系查询报文的格式。

^① 全系统群组 224.0.0.1 是一个例外，主机永远不会报告说自己是这个群组中的成员。

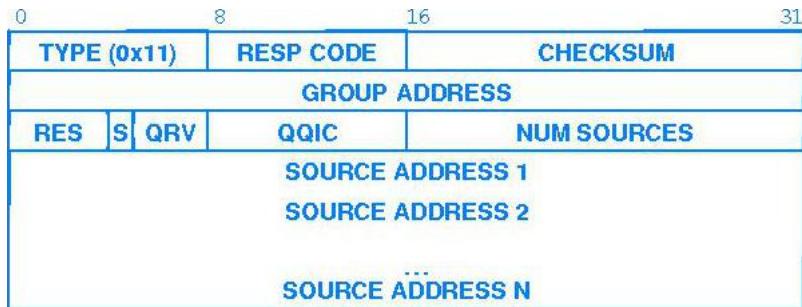


图 16.5 IGMP 成员关系查询报文的格式

如图 16.5 所示，成员关系查询报文首先是固定长度的 12 八位组首部。类型 (TYPE) 字段标识了报文的类型，图 16.6 所示为几个不同版本的 IGMP 的类型列表。

Type	Protocol Vers.	Meaning
0x11	3	Membership query
0x22	3	Membership report
0x12	1	Membership report
0x16	2	Membership report
0x17	2	Leave group

图 16.6 IGMP 报文类型。为了保持后向兼容，版本 3 的实现必须能够识别版本 1 和版本 2 的报文类型

当路由器探询群组成员时，标记为 RESP CODE 的字段指定了群组成员在计算随机时延时所用的最大间隔。如果这个字段的首位是比特 0，则表示这个值是以 0.1 s 为单位的一个整数值。如果这个字段的首位是比特 1，则表示这个值是一个浮点数，其指数长度为 3 位，尾数长度为 4 位。群组中的每台主机在响应之前都要先等待一个 0 至指定值之间的随机时间长度。我们也曾经说过默认值是 10 s。IGMP 允许路由器在每个查询报文中设置一个最大值，是为了让管理员能够控制 IGMP 通信量。如果网络中包含多台主机，较大的时延值会使响应时间更分散，从而降低了多台主机响应同一个查询的可能性。检验和 (CHECKSUM) 字段包含了该报文的检验和 (IGMP 检验和只用 IGMP 报文部分进行计算，所用算法与 TCP 和 IP 的一样)。群组地址 (GROUP ADDRESS) 字段或者用于指定特定的群组，或者包含 0 来代表一个通用查询。也就是说，如果路由器向指定群组或者指定群组加源的组合来发送一个查询，它要填写群组地址字段。S 字段指示路由器是否要抑制当一个更新到达时计数器应当执行的正常更新动作。这个比特对主机不起作用。QRV 字段允许 IGMP 在损耗较大的网络上多次发送同一个分组，从而控制了其健壮性。它的默认值是 2，发送方发送该报文 QRV - 1 次。QQIC 字段指明了查询报文的**查询间隔** (Querier's Query Interval)，即成员关系查询之间的间隔时间。QQIC 字段的表达方式与 RESP CODE 字段一致。

IGMP 报文的最后一部分由 0 或多个源组成。源地址数 (NUM SOURCES) 字段指定了随后的表项数目。每个源地址 (SOURCE ADDRESS) 包含一个 32 位 IP 地址。在通用查询 (即路由器需要知道网络中所有在用的多播群组的信息) 和特定群组查询 (即路由器需要知道某个指定的多播群组的信息) 报文中，源的数目为 0。对于**特定群组和源的查询** (group and source specific query)，报文中包含了一个或多个源，路由器用这样的报文查询多播群组和任何指定源的组合的接收状态。

16.17 IGMP 成员关系报告报文格式

IGMPv3 中使用的第二种类型的报文是**成员关系报告** (membership report) 报文，它是主机用来向路由器传递其参与状态的报文。图 16.7 描绘了它的格式。

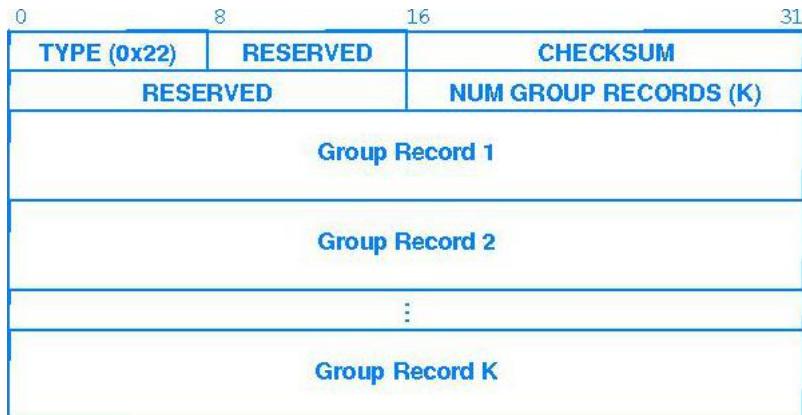


图 16.7 IGMP 成员关系报告报文的格式

如图 16.7 所示，成员关系报告报文由指定了报文类型以及群组记录数 K 的 8 八位组首部和 K 个**群组记录** (Group Records) 组成。图 16.8 所示为每个群组记录的格式。整个格式简单明了。起始字段为记录类型 (REC TYPE) 字段，允许发送方指定记录中的源列表对应的是**包含过滤器** (inclusive filter)、**排除过滤器** (exclusive filter) 还是对上一个报告的更改 (例如，还需要包含或排除一个源)。多播地址 (MULTICAST ADDRESS) 字段指定了该群组记录所指的多播地址，源地址数 (NUM OF SOURCES) 字段指定了群组记录中包含的源地址的数目。

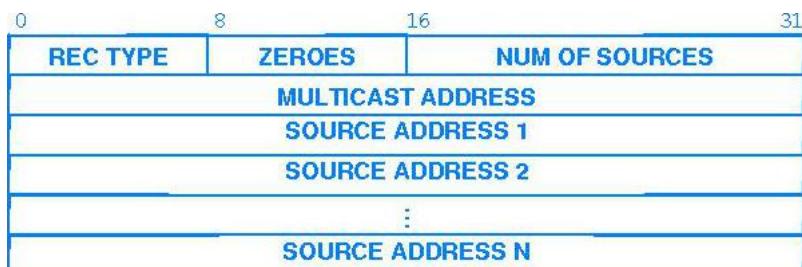


图 16.8 IGMP 成员关系报告报文中的每个群组记录的格式

需要注意的是，IGMP 并没有提供所有可能的报文或工具。例如，IGMP 不包括主机用来查找一个群组的 IP 地址的机制。也就是说，应用软件在使用 IGMP 加入到某个群组之前，必须先知道这个群组的地址。因此，有些应用程序使用永久分配的群组地址，而有些应用程序则允许管理员在安装软件时配置这个地址，还有一些应用程序则动态获取这个地址 (如从一台服务器上获取)。类似地，版本 3 中也没有为主机提供能够主动退出一个群组的显式报文，或监听某个群组的所有通信的显式报文。事实上，为了退出一个群组，主机会发送一个成员关系报告报文，它定义了一个具有空 IP 源地址列表的包含过滤器。而为了监听所有的源，主机会发送一个定义了具有空 IP 源地址列表的排除过滤器的成员关系报告报文。

16.18 多播转发和路由选择信息

前面描述的 IGMP 和多播编址方法指定了主机怎样和本地路由器交互，以及怎样通过单个网络传送多播数据报，但是没有说明路由器之间怎样交换群组成员信息，或者怎样确保每个数据报的副本能够到达所有群组成员。更重要的是，尽管已经提出了许多协议，但仍没有形成单一的标准来传播多播路由选择信息。实际上，虽然已经做了很多努力，但业界还未就整体计划达成一致，现有的协议在目标和基本方法上存在着差异。

为什么多播路由选择如此困难呢？为什么不把传统的路由选择方法扩展成处理多播呢？答案是，多播路由选择和常规路由选择在基本方法上是不同的，因为多播转发与常规转发不同。为了展示其差别，可考虑在图 16.9 所示的结构上实现多播转发。

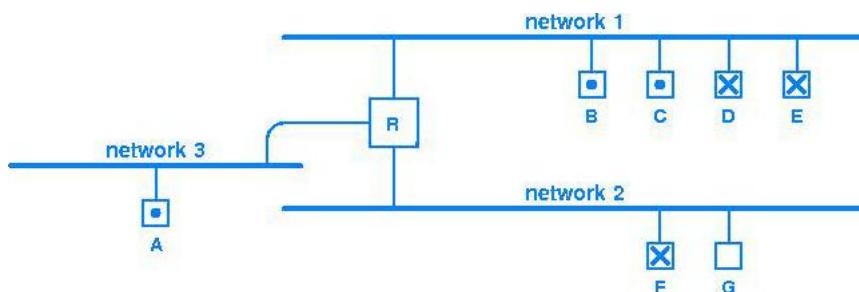


图 16.9 用一个路由器连接三个网络的简单互联网来描绘多播转发。有“.”标记的主机参与一个多播群组，有“×”标记的主机参与另一个多播群组

16.18.1 对动态路由选择的需求

即使对于图 16.9 所示的简单拓扑结构，多播转发也和单播转发不同。例如，该图显示了两个多播群组：以“.”标记的群组有成员 A, B 和 C，以“×”标记的群组有成员 D, E 和 F。“.”群组没有网络 2 中的成员。为了避免不必要的浪费带宽，路由器根本不应该让发往“.”群组的分组通过网络 2，但是主机可以在任何时候参加任何群组。如果该主机是所在网络上第一个加入某群组的，必须改变多播转发过程，将这个网络包括进来。这样，就得出了常规路由选择和多播路由选择的一个重要区别：

在单播路由选择中，只有当拓扑结构改变或设备出故障时才会发生路由改变，而多播路由选择则不同，一个应用程序加入或退出多播群组就会造成多播路由的变化。

16.18.2 目的地地址转发的不足之处

图 16.9 中的例子展示了多播路由选择的另一方面。如果主机 F 和主机 E 各自给“×”群组发送一个数据报，路由器 R 将接收并转发这些数据报。因为两个数据报发往同一个群组，目的地址是相同的，但正确的转发动作不同：R 把 E 的数据报发送到网络 2，而把 F 的数据报发送到网络 1。当路由器接收到主机 A 发往“×”群组的数据报时，采取了第三种动作：转发两个副本，一个给网络 1，另一个给网络 2。因此，我们得出了常规转发和多播转发的另一个重要差别：

多播转发时路由器要检查的不只是目的地址。

16.18.3 任意的发送方

图 16.9 还体现了多播转发的最后一个特性，因为 IP 允许任意主机（不一定是群组的成员）向

群组发送数据报。在图中，虽然主机 G 不是任何群组的成员，并且 G 所在的网络没有“.”群组的成员，但是 G 可以给“.”群组发送数据报。更重要的是，该数据报在通过互联网时，很可能穿过其他没有群组成员的网络：归纳如下：

多播数据报可以从非多播群组成员的计算机上发起，并且可能转发经过没有任何群组成员的网络。

16.19 基本的多播转发范例

从前面的例子中可以看出，多播路由器在处理数据报时必须用到不止一个目的地址。因此产生了一个问题，“决定如何转发数据报时，多播路由器到底需要使用哪些信息呢？”为了回答这个问题，需要理解这一点：由于多播目的地址代表的是一个计算机集合，所以最佳的转发系统将使数据报到达集合中的所有成员，并且一个数据报不会两次通过同一个同络。尽管图 16.9 所示的单个多播路由器可以简单地避免把数据报从它到来的接口再发回去，但是只依靠接口无法防止数据报在排成环状的一系列路由器之间进行转发。为了避免此类路由选择环路，多播路由器必须依靠数据报的源地址。

最先出现的多播转发的思路是以前描述过的一种广播形式。这种方法称为**反向路径转发**(Reverse Path Forwarding，简称 RPF)^①，这种机制用数据报的源地址来避免数据报反复通过一个环路。为了使用 RPF，多播路由器必须有一个常规路由选择表，其中有到每个目的站的最短路径。当数据报到达时，路由器提取出源地址，并在本地路由选择表中进行查找，找到 I，也就是通往源的接口。如果数据报是通过接口 I 到达的，路由器就给其他接口转发一份副本，否则就丢弃这个副本。

因为确保了互联网中的所有网络都发送过每个多播数据报的某个副本，所以基本的 RPF 方法能够保证多播群组中的每台主机都能接收到发往该群组的每个数据报的副本。但是，RPF 无法单独用于多播路由选择，因为对于既没有群组成员也不通向群组成员的网络，在这样的网络上传输多播数据报就是对带宽的浪费。

为了避免传播不需要的多播数据报，引入了 RPF 的修改版本，称为**截尾反向路径转发**(Truncated Reverse Path Forwarding，简称 TRPF) 或**截尾反向路径广播**(Truncated Reverse Path Broadcasting，简称 TRPB)。这一方法遵循 RPF 算法，但是通过避免不通向群组成员的路径，进一步对传播进行限制。为了使用 TRPF，多播路由器需要两项信息：常规路由选择表和通过每个网络接口可达的多播群组列表。当多播数据报到达时，路由器首先应用 RPF 规则。如果 RPF 指定丢弃该副本，路由器就这样做。然而，如果 RPF 要求通过特定接口传输该数据报，路由器首先会进行额外的检查，以便验证数据报目的地址中指定的一个或多个群组成员通过该接口是可达的。如果通过该接口没有群组成员可达，路由器就会跳过该接口并继续检查下一个接口。实际上，现在就可以理解截尾的来源了：某路径上不再有群组成员时路由器就会使转发截止。

总结如下：

在进行转发决策时，多播路由器使用了数据报的源地址和目的地址。基本的转发机制称为截尾反向路径转发。

16.20 TRPF 的后果

尽管 TRPF 保证了多播群组的每个成员都能收到发给群组的每个数据报的副本，但它也会造成

^① 反向路径转发有时也称为**反向路径广播**(Reverse Path Broadcasting，简称 RPB)。

两个后果。首先，因为 TRPF 依靠 RPF 来避免环路，所以 TRPF 也会像常规 RPF 那样给某些网络带来额外的数据报副本，图 16.10 所示为产生重复的情况。

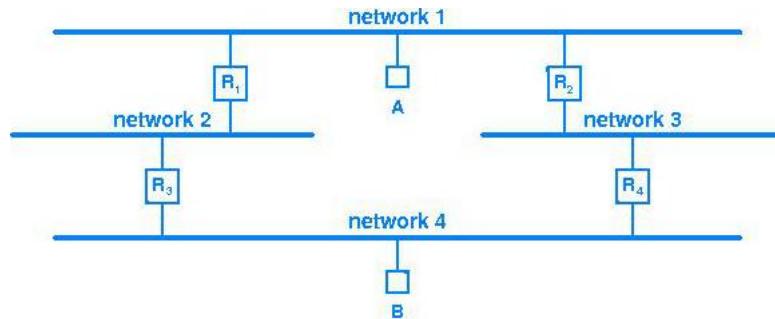


图 16.10 这种拓扑结构造成 RPF 方法给某些目的站交付了一个数据报的多份副本

在图 16.10 中，当主机 A 发送数据报时，路由器 R₁ 和 R₂ 各收到一份副本。因为数据报是通过到 A 的最短路径上的接口到达的，R₁ 给网络 2 转发一份副本，R₂ 给网络 3 转发一份副本。当 R₃ 从网络 2 收到一份副本时（到 A 的最短路径），把它转发给网络 4。遗憾的是，R₄ 也给网络 4 转发了一份副本。这样，尽管 R₃ 和 R₄ 丢弃了通过网络 4 到达的副本，从而防止了环路，但主机 B 还是收到了数据报的两份副本。

因为 TRPF 在转发数据报时使用了源地址和目的地址，所以产生了另一种后果：交付取决于数据报的源地址。例如，图 16.11 显示了多播路由器如何通过固定的拓扑结构，从两个不同的源站转发数据报。

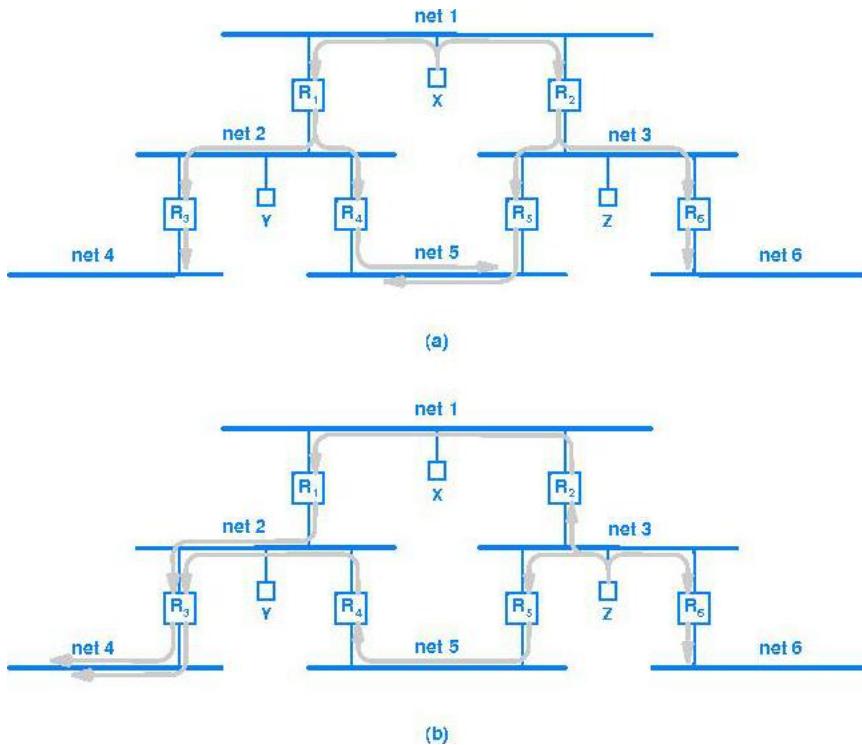


图 16.11 假设源站是图(a) 主机 X 和图(b) 主机 Z，并且各个网络上分别有群组的一个成员，使用 TRPF 时多播数据报所经过的路径。收到的副本数目取决于源站

如图 16.11 所示，源站影响了数据报到某网络所沿的路径，还影响了交付细节。例如，在图 16.11(a)

中，主机 X 的传输使 TRPF 给网络 5 发送了数据报的两份副本。而在图 16.11(b)中，主机 Z 只有一个副本到达网络 5，但有两个副本到达网络 2 和网络 4。

16.21 多播树

研究人员使用图论术语来描述从特定源站到多播群组的所有成员的一组路径，他们认为这些路径定义了图论中的**树**（tree）^①，有时也称为**转发树**（forwarding tree）或**交付树**（delivery tree）。每个多播路由器对应于树中的一个**节点**（node），连接两个路由器的网络则对应于树中的一条**边**（edge）。数据报的源站是树的**根**（root）或**根节点**（root node）。最后，从源站出发沿一条路径到达的最后一个路由器称为**叶**（leaf）路由器。这种术语表示方法有时也应用于网络，研究人员把连接在叶路由器上的网络称为**叶网络**（leaf network）。

以图 16.11 为例，图 16.11(a)是以 X 为根的树，R₃, R₄, R₅ 和 R₆ 是叶路由器。从技术上讲，图 16.11(b)不是树，因为路由器 R₃ 在两条路径上。在非正式场合下，研究人员通常会忽略这个细节，把这样的图也认为是树。

使用图论术语可以表达一个重要的原则：

多播转发树被定义为一组通过多播路由器的路径，通过这些路径可以从源站到达多播群组的所有成员。对于某个多播群组，每个可能的数据报源都能确定一个不同的转发树。

在得出这个基本原则后，紧接着需要考虑的就是用于转发多播的路由表的大小。与常规路由选择表不同，多播路由选择表中的每个表项由一对地址标识：

(多播群组, 源站)

从概念上讲，**源站**（source）标识了能够向群组发送数据报的一台主机（即互联网中的任何主机）。在实践中，为每台主机单独保留一个表项肯定是不明智的，因为位于单个网络上的所有主机定义的转发树是相同的。因此，为了节省空间，转发协议使用网络前缀作为源站。也就是说，每个路由器定义一个转发表项，同一物理网络上的所有主机都使用这个转发表项。

通过使用网络前缀代替主机地址，可以明显减小多播路由选择表，但还是比常规路由选择表大得多。常规表的大小与底层互联网的网络数量成正比，而多播表则不同，其大小与互联网的网络数和多播群组数的乘积成正比。

16.22 多播路由选择的实质

细心的读者可能已经注意到 IP 多播和 TRPF 特性之间有不一致性。前面讲过，TRPF 代替 RPF，用以避免不必要的通信量。也就是说，除非通过某个网络能够到达群组的一个成员，否则 TRPF 不会向该网络转发数据报。因此，多播路由器必须了解群组成员的情况。前面还讲过，IP 允许任何主机在任何时候加入或退出一个多播群组，这就导致了成员关系的迅速变化。更重要的是，成员关系并不遵循本地作用域，加入群组的主机可能和向该群组转发数据报的某些路由器距离很远，所以群组成员信息必须通过底层互联网进行传播。

成员问题是路由选择的核心：所有多播路由选择方法都提供了一种传播成员信息的机制，同时还提供了转发数据报时使用该信息的方式。一般来讲，因为成员关系会迅速改变，从某个路由器得到的信息并不是完全正确的，因此路由选择可能滞后于变化。所以，多播的设计表现出路由选择通信量的额外开销与数据传输效率之间的一种折中。一方面，如果群组成员信息没有迅速传播，多播路由器将不能做出最佳决策（例如，可能会不必要地通过某些网络转发数据报，也可能没有把数据

^① 树就是不含任何回路的图（即一个路由器不会出现在多条路径上）。

报发给所有群组成员)。另一方面，把每个成员关系变化都通知给每个路由器的多播路由选择方法对于互联网来说就是场灾难，因为产生的通信量会让互联网无法承受。每种设计方法都在这两个极端之间做出了某种折中。

16.23 反向路径多播

多播路由选择的一种最早形式是从 TRPF 派生而来的，称为**反向路径多播**(Reverse Path Multicast，简称 RPM)，该方法对 TRPF 进行了扩展，使其更具动态性。这种设计基于三个假设。第一，与消除不必要的传输相比，确保多播数据报到达要发往的群组的每个成员更重要。第二，每个多播路由器都包含一个具有正确信息的常规路由选择表。第三，多播路由选择应该尽可能提高效率(例如，消除不必要的传输)。

RPM 采取两步处理过程。开始时，RPM 使用 RPF 广播方法通过互联网中的所有网络发送每个数据报的副本。这样做可以确保所有群组成员都接收到一个副本。同时，通过 RPM 让多播路由器互相通知关于不能通往群组成员的路径的情况。一旦了解到某路径上没有群组成员，路由器就会停止沿该路径的转发。

路由器怎样了解群组成员的位置呢？与大多数多播路由选择方法一样，RPM 自底向上传播成员信息。信息先从选择加入或退出群组的主机开始。主机使用 IGMP 与本地路由器之间传递成员信息，这样，虽然多播路由器不知道远端群组成员的情况，但能知道所有本地成员的情况(例如，每个直接相连的网络上的成员)。因此，连到叶网络上的路由器可以决定是否向叶网络转发(如果叶网络不包含某群组的成员，将叶网络连接到互联网的路由器就不会向叶网络转发)。叶路由器除了要采取本地行动之外，还要通知回溯到源站的路径上的下一个路由器。一旦这个路由器了解到某个网络接口之后不存在群组成员，它就会停止通过该网络转发群组的数据报。当一个路由器发现其后不存在群组成员时，就会通知通往根的路径上的下一个路由器。

用图论的术语来讲，当路由器了解到沿某条路径没有成员并停止转发时，就从转发树中剪除(即去掉)这条路径。实际上，RPM 可称为**广播并剪除**(broadcast and prune)策略，因为路由器在收到允许剪除路径的信息之前，会使用 RPF 进行广播。对于 RPM 算法，研究人员还使用了另一个术语，他们把系统称为**数据驱动**(data-driven)的，因为路由器不会向其他任何路由器发送群组成员信息，除非有该群组的数据报到达。

在数据驱动模式中，路由器还必须处理这样的情况：主机在路由器已经剪除了某个群组的路径之后，又决定要加入该群组。RPM 自底向上处理这种加入情况：当主机通知本地路由器，说它已经加入某个群组时，路由器就会查阅自己的群组记录，获得以前向其发送过剪除请求的路由器的地址。路由器会发送一个新报文，撤销前一次剪除的效果，并使数据报能够再次流动起来。这种报文称为**嫁接请求**(graft request)，该算法被认为是把以前剪除的分支接回树上。

16.24 多播路由选择协议

IETF 已经调查了不少多播协议，包括**距离向量多播路由选择协议**(Distance Vector Multicast Routing Protocol，简称 DVMRP)、**核心基干树**(Core Based Trees，简称 CBT)、**协议无关多播**(Protocol Independent Multicast，简称 PIM)以及**OSPF 多播扩展**(Multicast extensions to OSPF，简称 MOSPF)。这些协议的形式略有不同。虽然这些协议都已经实现，并且其中的几个协议也有运营商的支持，但还没有哪一种是必备的标准。接下来要详细解释这些协议。

16.24.1 距离向量多播路由选择协议

在多播协议中，最早的是距离向量多播路由选择协议（DVMRP），该协议允许多播路由器在相互之间传递群组成员关系和路由选择信息。DVMRP 类似于第 15 章中描述的 RIP 协议，但为多播进行了扩展。实质上，该协议传递的信息既包括当前的多播群组成员关系，也包括路由器之间传送数据报的费用。对于每个可能的（群组，源站）序偶，路由器在物理互连基础上拼接了一个转发树。当路由器收到发往一个 IP 多播群组的数据报时，通过相应于转发树中各分支的网络链路^①，发送数据报的副本。DVMRP 由一个称为 mrouted 的 UNIX 程序实现，它使用了特殊的**多播核**（multicast kernel）。

mrouted 使用了多播隧道，使网点能够通过因特网转发多播。在每个网点上，管理员可以配置一个通往其他网点的 mrouted 隧道。这个隧道使用 IP-in-IP 封装来发送多播。也就是说，当 mrouted 接收到一个由本地主机生成的多播数据报时，将数据报封装在一个常规的单播数据报中，并向其他每个网点的 mrouted 转发一个副本。当它接收到来自某一条隧道的单播数据报时，mrouted 会取出多播数据报，然后根据多播路由选择表进行转发。

16.24.2 核心基干树

核心基干树（CBT）避免进行广播，并允许所有源站尽可能地共享同一个转发树。为了避免进行广播，除非某条路径上有一台或多台主机加入到多播群组中，CBT 才会沿路径转发多播数据报。因此说，CBT 与 DVMRP 使用的**洪泛并剪除**（flood and prune）方法是相反的，它不是先转发数据报，直到传播了否定信息才停止转发，而是直到收到肯定信息后才沿该路径转发。这种方法不是数据驱动的，而是使用了**需求驱动**（demand-driven）形式。

CBT 中的需求驱动形式意味着，当主机使用 IGMP 加入一个特定群组时，本地路由器在转发数据报之前必须通知其他路由器。应该通知哪个（些）路由器呢？在所有需求驱动的多播路由选择方法中，这个问题都是至关重要的。在前面讲过的数据驱动方法中，路由器通过数据通信量的到达而得知路由选择报文应该发往何处（把路由选择信息回传到通信量来自的网络）。但是，在需求驱动的方法中，在群组的成员信息传播开之前，不会有该群组的通信量到达。

CBT 将静态算法和动态算法结合起来构建多播转发树。为了使该方法具有规模可伸缩性，CBT 把互联网划分成**区段**（region），每个区段的大小由网络管理员确定。在每个区段内，有一个路由器被指定为**核心路由器**（core router），区段内的其他路由器或者必须通过配置掌握自己所在区段的核心，或者必须使用动态的**发现机制**（discovery mechanism）来查找这个核心。

了解核心的位置很重要，因为只有这样，区段内的多播路由器才能形成该区段的**共享树**（shared tree）。当一台主机加入一个多播群组时，本地路由器 L 接收到该主机的请求，路由器 L 生成一个 CBT **加入请求**（join request），并使用常规单播路由选择把该请求发给核心。通往核心的路径上的每个中间路由器都对这个请求进行检查。一旦该请求到达了已成为 CBT 共享树一部分的路由器 R，R 就会返回一个确认，把群组成员信息传给自己的上层路由器，并开始转发该群组的通信量。在确认报文重新传回叶路由器的过程中，中间路由器检查该报文，并配置其多播路由选择表，以转发该群组的数据报。因此，路由器 L 在路由器 R 处链接到转发树。

总结如下：

因为 CBT 使用了需求驱动形式，所以它把互联网划分成区段，并为每个区段指定了一个核心路由器，区段中的其他路由器通过给核心发送加入请求，动态地建立转发树。

^① DVMRP 的版本 2 和版本 3 之间的变化比较大，因为它结合了上面描述的 RPM 算法。

16.24.3 协议无关多播

实际上，协议无关多播（PIM）由两个独立的协议组成，除了名称和基本报文首部格式之外，这两个协议没有什么类似之处，它们分别是：**PIM-密集模式**（PIM-Dense Mode，简称 PIM-DM）和**PIM-稀疏模式**（PIM-Sparse Mode，简称 PIM-SM）。产生这种差别是由于没有哪一种协议可独自适用于所有的可能情况。实际上，PIM 密集模式设计用于局域网环境，在这种环境下，几乎所有网络中都有主机在监听每个多播群组。而 PIM 稀疏模式设计用于适应广域网环境，在这种环境下，某多播群组的成员只占所有可能网络的很小一个子集。

之所以采用术语**协议无关**（protocol independence），是因为 PIM 假定有一个包含到达每个目的地的最短路径的传统单播路由选择表。既然 PIM 没有指明应当如何建立这个表，那么可以使用任意的路由选择协议。因此，PLM 的操作与路由器采用的单播路由选择协议“无关”。

为了适应为数众多的监听方，PIM-DM 使用广播并剪除方法，首先使用 RPF 把每个数据报都转发到所有路由器，直到收到明确的剪除请求时才停止发送。相反，PIM 稀疏模式可看成是对 CBT 基本概念的扩展。稀疏模式指定了一个称为**汇聚点**（Rendezvous Point，简称 RP）的路由器，其功能等价于一个 CBT 核心路由器。

16.24.4 OSPF 的多播扩展

类似 PIM 这样的多播路由选择协议使用的是单播路由选择表中的信息，除此之外，研究人员已经在研究这样一个更广范围的问题：“多播路由选择如何从常规路由选择协议收集到的额外信息获益？”实际上，OSPF 之类的链路状态协议给每个路由器提供了互联网拓扑结构的一个副本。更具体地说，OSPF 给路由器提供了它自己的 OSPF 区域（area）的拓扑结构。

提供了这样的信息之后，多播协议就可以利用该信息计算转发树了。这一思路体现在 OSPF 多播扩展协议（MOSPF）中，该协议使用 OSPF 的拓扑数据库为每个源站形成一个转发树。MOSPF 具有需求驱动形式的优点，也就是说，特定群组的通信量直到需要时（例如，由于主机加入或退出群组）才进行传播。需求驱动方法的缺点在于传播路由选择信息的费用：一个区域中的所有路由器必须维护关于每个群组的成员信息。而且，这些信息必须是同步的，以确保每个路由器的数据库完全一致。最后的结果是，MOSPF 发送的通信量较少，但比数据驱动的协议发送了更多的路由选择信息。

16.25 可靠多播和 ACK 内爆

术语**可靠多播**（reliable multicast）指的是这样的系统：使用多播交付并能够保证所有群组成员收到按序到达、无丢失、无重复且未遭破坏的数据。从理论上讲，可靠多播既具有转发方案比广播更高效的优势，又具有让所有数据完好到达的优势，因此可靠多播具有很大的潜在利益和可用性（例如，股票交易所可使用可靠多播把股票价格交付到许多目的站）。

在实践中，可靠多播并不像听起来那么具有普遍性，也并不简单。第一，如果多播群组有多个发送方，“按序”交付数据报的说法就变得毫无意义；第二，我们已经看到，广泛使用的多播转发方法，如 RPF，即使在小型互联网上也会产生重复的数据报；第三，除了保证所有数据最终到达之外，音频或视频之类的应用程序希望用可靠的系统来约束时延或抖动；第四，因为可靠性要求确认，且一个多播群组可以有任意数目的成员，传统的可靠协议需要发送方处理任意数目的确认。遗憾的是，没有哪种计算机有足够的计算能力来做到这一点。我们把这个问题称为**ACK 内爆**（ACK implosion），它已经成为许多研究的主要焦点。

为了克服 ACK 内爆问题，可靠多播协议采用了一种分层方法，这种方法把多播限制到一个源

站^①。在发送数据之前，从该源站要建立到所有群组成员的转发树，还要标志出**确认点**（acknowledgement point）。

确认点又称为**确认聚集器**（acknowledgement aggregator）或**指定路由器**（designated router，简称 DR），它由转发树中的一个路由器构成，该路由器同意缓存数据的副本，并处理来自树中下层路由器或主机的确认。如果需要重传，确认点从缓存中取出一个副本。

大多数可靠多播方法使用了负面确认，而不是正面确认，也就是说，主机直到数据报丢失后才做出响应。为了让主机检测到丢失，必须给每个数据报分配唯一的序号。当检测到丢失时，主机就会发送 NACK 请求重传。NACK 沿转发树向源站方向传播，直到到达一个确认点。确认点处理 NACK，并沿转发树重传丢失数据报的一个副本。

确认点如何确保具有序列中所有数据报的副本呢？它使用的方法与主机所用的相同。当数据报到达时，确认点检查序号，在内存中放一个副本，然后继续沿转发树向下传播该数据报。如果确认点发现丢失了一个数据报，就会沿树向源站的方向发送一个 NACK。NACK 或者到达另一个具有数据报副本的确认点（在此情况下由确认点传送第二个副本），或者到达源站，由源站重传丢失的数据报。

分支拓扑结构和确认点的选择对于可靠多播方法的成功与否至关重要。没有足够的确认点，一个丢失的数据报就会造成 ACK 内爆。实际上，如果给定路由器有许多下层机器，丢失的数据报可能造成该路由器由于重传请求过多而超载。遗憾的是，自动选择确认点很不容易实现。结果是，许多可靠多播协议需要进行手工配置。因此，多播最适合用于长期使用的服务，不会迅速变化的拓扑结构，并且中间路由器同意作为确认点的情况。

还有别的可替代方法是可靠的吗？一些研究人员正在试验一些加入冗余信息的协议，以减少或消除重传。其中一种方法是发送冗余的数据报，对于每个数据报，源站不只发送一份副本，而是发送 N 份副本（典型的值是 2 或 3）。在路由器实现**随机提前丢弃**（Random Early Discard，简称 RED）策略时，冗余数据报方法十分有效，因为多个副本被丢弃的可能性非常小。

另一种冗余方法涉及到**前向纠错码**（forward error-correcting code）。与音频 CD 使用的纠错码类似，该方法需要发送方在数据流中的每个数据报上加入纠错码信息。如果丢失了一个数据报，纠错码包含了足够的冗余信息，使接收方能够重构出丢失的数据报，而不需要请求重发。

16.26 小结

IP 多播是对硬件多播的抽象化。它允许将一份数据报交付到多个目的站。IP 使用 D 类地址来定义多播交付，实际传输会尽可能使用硬件多播。

IP 多播群组是动态变化的：主机可以在任何时候加入或退出群组。对于本地多播，主机只需要有接收和发送多播数据报的能力。但是，IP 多播并不只限于单个物理网络，多播路由器传播群组成员信息并安排适当的路由，使群组的每个成员都能收到送往该群组的每一份数据报的副本。

主机使用 IGMP 与多播路由器通信，报告自己的群组成员关系。IGMP 被设计成高效率的，并避免占用网络资源。

对于通过互联网传播多播路由选择信息，已经设计了多种协议。两种基本的方法就是数据驱动和需求驱动。在任何一种情况下，多播转发表中的信息量比单播路由选择表中的大得多，因为对应每个（群组，源站）序偶都要有多播表项。

在因特网中，并不是所有路由器都能够传播多播路由或转发多播通信量。对于分布在两个或更多网点上的群组，当中间由一个并不支持多播路由选择的互联网隔开时，可使用 IP 隧道来传输多播数据报。使用隧道时，程序把多播数据报封装在常规的单播数据报中，接收方必须提取并处理这

^① 注意，一个源站并不会限制其功能性，因为该源站同意转发通过单播到达的任何报文，这样，任意主机可以给该源站发送一个分组，然后由源站把该分组多播到群组。

个多播数据报。

可靠多播指的是一种方法，使用的是多播转发，同时提供了可靠交付机制。为了避免 ACK 内爆问题，可靠多播方法或者使用分层结构的确认点，或者通过发送冗余信息实现。

16.27 深入研究

Cain et. al. [RFC 3376] 定义了本章描述的 IP 多播的标准，包括 IGMP 版本 3。Waitzman, Partridge, and Deering [RFC 1075] 描述了 DVMRP 技术，Estrin et. al. [RFC 2362] 描述了 PIM 稀疏模式，Ballardie [RFCs 2189, 2201] 描述了 CBT，Moy [RFC 1585] 描述了 MOSPF。

Eriksson [1994] 讲解了多播主干网。Casner and Deering [July 1992] 报告了因特网工程部 IETF 会议首次使用多播的情况。

16.28 习题

1. 标准建议用 IP 多播地址的 23 比特形成硬件多播地址。在这样的方案中，一个硬件多播地址上可能会映射多少个 IP 多播地址？
2. 讨论 IP 多播地址为什么应该从 28 比特中选用 23 比特。提示：一台主机实际可以属于多少个群组？一个网络上实际可能有多少台主机？
3. IP 必须检查所有传入的多播数据报的目的地址，如果该主机不在指定的多播群组内，则丢弃该数据报。主机并不属于某个群组，为什么会收到发往该群组的数据报？试解释原因。
4. 多播路由器需要知道在给定网络上是否有某个群组的成员。那么，有没有必要让多播路由器了解网络上属于多播群组的具体主机集合？
5. 从你周围的环境中找出三个可能受益于 IP 多播技术的应用程序。
6. 标准认为，IP 软件必须负责向主机上的那些属于多播群组的应用程序交付所有外发多播数据报的副本。这样的设计使编程更容易还是更困难呢？试解释原因。
7. 当底层硬件不支持多播时，IP 多播使用硬件广播技术进行交付。这样做时会造成问题？在这样的网络上使用 IP 多播有什么好处？
8. DVMRP 是从 RIP 产生的。阅读 RFC 1075 中关于 DVMRP 的内容，并比较这两种协议。与 RIP 相比，DVMRP 的复杂性体现在哪里？
9. 第三版的 IGMP 包括一种增加健壮性的方法，就是试图通过允许传输一个报文的多个副本的方法解决分组丢失问题。这种协议如何才能达到特定网络对健壮值的要求？
10. 解释多地址主机为什么可能需要加入一个网络的某个多播群组，而不是另一个网络上的多播群组。提示：考虑音频电子会议的情况。
11. 如果每个无线电台有 1000 万听众随机分布在世界各地，估计处理 100 个电台的音频多播所需的多播转发表的大小。
12. 试论证，在因特网中只有两类多播是可行的：一类是静态配置的针对大量订户的商用服务的多播，另一类是动态配置的包括少数参与者的服务（例如，分属三户的家庭成员们参加的一次电话会议）。
13. 考虑通过冗余传送获得的可靠多播。如果某条链路损坏的可能性很高，发送数据报的冗余副本更好，还是发送使用转发纠错码的副本更好？试解释原因。
14. 数据驱动的多播路由选择方法在低时延、大容量的本地网络上十分适用，而需求驱动的方法则适用于容量有限且高时延的广域环境下。是否有必要设计一种把这两种方法结合起来的协议？试解释原因。提示：考虑 MOSPF。
15. 设计一种定量度量方法，用于确定 PIM-SM 何时应该从共享树切换到最短路径树。

-
16. 阅读协议规范，找出 PIM-SM 中使用的“稀疏”概念。举一个群组成员稀疏分布的互联网例子，对于这种情况，更适合使用 DVMRP 作为多播路由选择协议。

第17章 IP 交换和 MPLS

17.1 引言

前几章介绍了 IP 编址以及几种数据报转发的算法，这些算法利用路由选择表和最长前缀匹配来查找下一跳。本章要考虑的是另一种方法，它避免了使用最长前缀匹配方法带来的额外开销。本章会解释其基本技术并介绍它在流量工程中的应用。

17.2 交换技术

在 20 世纪 80 年代，随着因特网的迅速普及，研究人员开始探索如何提高分组处理系统性能的方法。这时出现了一种想法，并且这种想法最终被商业运营商所采纳：用面向连接的方式取代 IP 的无连接分组交换方式，这样就可以利用更快捷的查找算法，而不必使用表的最长前缀匹配查找。这个基本概念就是**交换** (switching)，人们经常把它与**异步传递方式** (Asynchronous Transfer Mode, 简称 ATM) 联系起来，仅仅是因为这两个概念是同时出现的。但是，我们将会了解到，在传统的路由器上也可以实现交换，而不是必须依赖面向连接的硬件支持。交换的关键在于查找算法的基本思路：一个典型的处理器在索引一个数组时需要用常量时间，而在有 N 个元素的集合中进行查找时则需要 $\log_2 N$ 的时间。另外，索引只需要简单的计算，而查找通常涉及到多个存储地址的引用。

交换技术利用索引实现了速度上的提高。为了做到这一点，每个分组都要携带一个小整数，称为**标记** (label)。当分组到达交换机时，交换机读取分组的标记并使用这个值作为索引，检索一个规定了相应动作的表。图 17.1 描绘的就是这样一个概念。

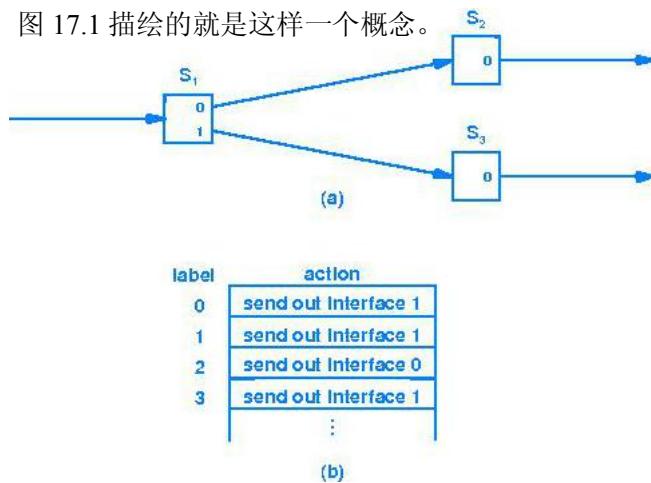


图 17.1 通过图(a) 有三个互相连接的交换机组成了网络和图(b) 交换机 S₁ 中的一个表来描绘基本交换概念

在图 17.1 中，每个表项指定的是交换机应当如何转发带有 0~3 标记的分组。根据这张表，带有标记 2 的分组将从接口 0 转发出去，而这个接口通往交换机 S₂。具有标记 0, 1 或 3 的分组将通过接口 1 转发，并通往交换机 S₃。

17.3 大规模网络、标记交换和路径

因为每个标记都是由一个小整数构成的，这就是最简单的交换机制带来的主要缺点。怎样才能

对这种方法进行扩展，将其应用到具有大量目的地的网络中呢？答案有两个。首先，只有当流（flow）处于活跃状态时才需要标记，而在特定时间同时处于活跃状态的流的数目并不多。交换使用的是面向连接（connection-oriented）的方式，它只给活跃的流分配一个标记，一旦流终止就恢复原样。第二，为了避免在使用标记时必须取得全体的一致认可，交换系统应用了标记交换（label switching 或 label switching）技术。标记交换的含义是，分组的标记在从一个路由器到达另一个路由器时可以发生改变。也就是说，交换机执行的动作也包括重写标记。图 17.2 描绘了这个概念。

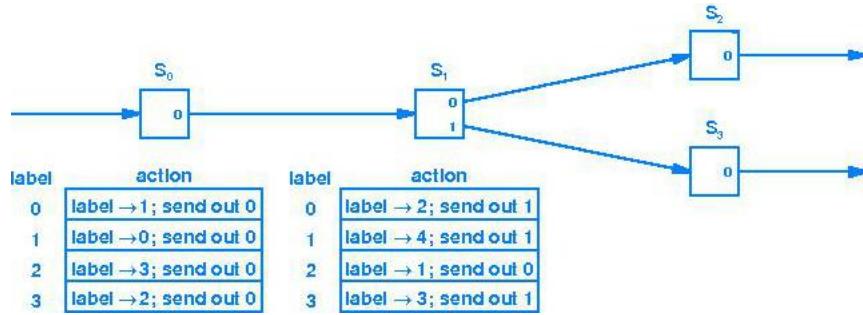


图 17.2 描绘了标记交换技术，分组中的标记可以被路由器重写

图中标记为 0 的分组到达 S_0 ，在分组转发前，它的标记被重写。因此，当交换机 S_1 接收到这个分组时，分组的标记已经是 1。同样， S_1 在将分组发往 S_3 之前会将标记 1 换成 4。标记交换使得交换网络更容易配置，因为它允许管理员在定义一条网络中的路径时，不用强求该路径上的每个点都必须使用相同含义的标记。事实上，一个标记只要在一跳上有效就可以了，也就是说，共享一条物理连接的两个交换机必须对如何分配通过这条连接的每个流的标记达成一致。

这里的重点在于：

交换使用面向连接的方式。为了避免在使用标记时必须取得全体的一致认可，交换技术允许由管理员定义一条由交换机组织的路径，且不要求分组用相同的标记经过整个路径。

17.4 IP 的交换应用

虽然交换所具有的面向连接的含义与 IP 的无连接方式存在冲突，但两者还是结合起来了。这样做有三个动机：

- 更快的转发
- 聚合的路由信息
- 管理聚合流的能力

更快的转发。这一点很容易理解：采取面向连接的方式可以允许路由器的转发动作更迅速，因为路由器可以使用索引来查找路由选择表。当然，速度上的提高只对那些既有高速连接（如 OC-192），又有高通信量速率的路由器来说有必要。这些路由器通常出现在因特网的内核（例如由某大型 ISP 拥有）。

聚合的路由信息。拥有因特网内核的大型 ISP 通过使用交换技术，避免在其路由器上出现复杂的路由选择表。事实上，IP 路由选择表的查找动作只会执行一次（例如，当分组第一次到达 ISP 时），然后给分组分配一个标记，而 ISP 的所有路由器都使用这个标记来转发分组。更进一步，因为标记只是用来指定分组下一步应当发往哪个 ISP，而不是指定最终的目的站，所以可以给多个分组分配相同的标记。

管理聚合流的能力。我们曾说过，大型 ISP 经常会签署关于通信量在对等结点之间传输的服务

等级协定 (Service Level Agreements, 简称 SLA)。通常，这个 SLA 指的就是聚合通信量（例如，在两个 ISP 之间转发的所有通信量或所有 VoIP 通信量）。如果给每个聚合通信量分配一个标记，就可以使测量和强制实施 SLA 的机制更容易实现。

17.5 IP 交换技术和 MPLS

到目前为止，我们所讨论的交换是一种通用的、面向连接的网络技术。Ipsilon 公司是最早生产将 IP 和硬件交换机结合在一起的产品的公司之一。该公司使用了 ATM 交换机，称其技术为 **IP 交换** (IP switching)，所生产的设备称为 **IP 交换机** (IP switch)。继 Ipsilon 公司之后，其他公司也有了系列设计和名字，包括**标志交换** (tag switching)、**第三层交换** (layer 3 switching) 和**标记交换** (label switching)。其中的一些思路已经由 IETF 统一成一个标准，称为**多协议标记交换** (Multi-Protocol Label Switching，简称 MPLS)^①。

MPLS 是如何工作的？其基本思想简单明确：连接到用户计算机系统的路由器依旧使用传统的转发方式，而网络内部的路由器则需要掌握 MPLS，并应用交换技术而不是常规的 IP 路由选择表查找。例如，对于一家大公司来说，在每幢建筑内的各网点可能会选择使用常规的转发技术，而对于建筑和建筑之间起互联作用的网络则要使用 MPLS 技术。

在数据报从一个常规网络到达一个 **MPLS 核** (MPLS core) (即应用了 MPLS 的一组相邻路由器) 之前，它必须分配得到一个标记。MPLS 核中的每个路由器都要用这个标记来判决如何转发这个数据报。最终，该数据报或者到达 MPLS 核的边缘 (此时会除去标记)，或者直接递交到另一个 MPLS 核 (如从一个大型 ISP 的 MPLS 核到另一个大型 ISP 的 MPLS 核)。当然，在数据报从一个 MPLS 核传递到另一个 MPLS 核之前，两个 ISP 的管理员必须先就每个标记的分配和含义达成一致。

17.6 分类、流和高层交换

当数据报第一次到达某个 MPLS 核时，必须为它分配一个初始标记 (即必须匹配到某个流上)。我们用术语**分类** (classification) 来表示这个分配标记的过程，而术语**分类器** (classifier) 指的是执行分类动作的硬件或软件。这种匹配可以表示为数学函数：

$$l = c(\text{datagram})$$

其中 l 就是标志了某个流的标记。

在实际应用中，函数 c 不检查整个数据报，而是只使用首部的字段。严格的**第三层分类** (layer 3 classification) 仅仅使用了 IP 首部中的字段，如源站和目的站的 IP 地址以及服务类别。大多数运营商实现了**第四层分类** (layer 4 classification)^②，而有一些运营商则实现了**第五层分类** (layer 5 classification)。第四层分类机制除了要检查 IP 首部中的字段之外，还要检查 TCP 或 UDP 首部中的协议端口号。第五层分类更进一步地检查了数据报的内部并考虑其有效载荷。

17.7 MPLS 的分层应用

前面讨论过，在组织内部使用的 MPLS 构造成两层结构：使用常规 IP 转发的外部区域和使用 MPLS 的内部区域。我们将会看到，也可以再多分几层。例如，假设一个公司有三个厂区，每个厂区又有多幢建筑。公司可以在建筑内利用常规转发，在厂区内的建筑之间的相互连接使用第一层 MPLS，而在不同厂区之间使用第二层 MPLS 连接。两层结构使公司可以为网点之间的通信量和建

^① 尽管在名字中含有“多协议”，但 MPLS 的焦点几乎完全集中在 IP 上。

^② 有些运营商用术语**第四层交换** (layer 4 switching) 来形象地说明其产品实现的是第四层分类。

筑之间的通信量选择不同的策略（例如，来往于建筑之间的通信量所经过的路径由通信量的类型决定，而每对网点之间的所有通信量走的是相同的路径）。

为了提供多层结构，MPLS 需要一个**标记栈**（label stack）。也就是说，MPLS 允许一个分组附带多个标记，而不是只能带一个标记。并且，在任何时候只能使用顶层的标记，一旦除去上层的标记，接下来的处理过程就使用下一个标记。

在前面举的例子中，该公司可以构建两个 MPLS 区：一个用于建筑之间的通信量传输，而另一个用于两个厂区之间的通信量传输。因此，如果数据报是在某厂区的两幢建筑之间进行传输，那么这个数据报就只附带一个标记（当数据报到达正确的建筑后，就除去该标记）。如果数据报必须首先在厂区之间传输，然后再到达目的厂区中的某幢建筑，那么数据报将会附带两个标记。顶层的标记用于数据报在厂区之间的传输，并在数据报到达正确的厂区时被除去。当顶层的标记除去以后，第二个标记用于将该数据报转发到正确的建筑中。

17.8 MPLS 封装

MPLS 不要求使用面向连接的网络技术。也就是说，一对 MPLS 路由器之间的物理连接可以由一个专用电路组成，如 OC-48 线路或以太网之类的传统网络。但是，传统网络并不提供在分组上附加标记的手段，而 IPv4 数据报首部也没有多余的空间存放标记。这样就带来一个问题：“MPLS 标记怎样才能和数据报一起传输？”答案在于一种封装技术：封装数据报时，在插入数据报之前首先要插入一个 MPLS 首部。图 17.3 所示的就是在以太网情况下的封装。

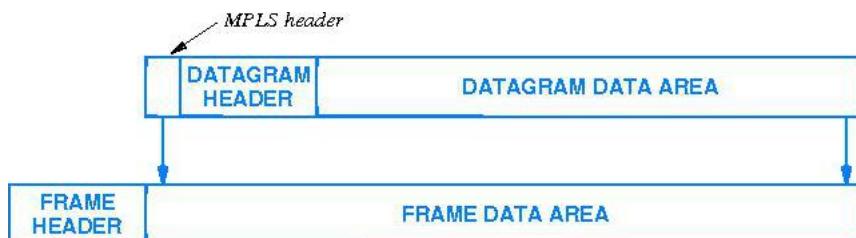


图 17.3 描绘了用于在以太网上发送 IP 数据报的 MPLS 封装。一个 MPLS 首部包含一个或多个表项，每个表项长 32 位

在发送使用 MPLS 封装的一个帧时，以太网的类型字段在单播的情况下设置为 8847_{16} ，而在多播的情况下设置为 8848_{16} ^①。这样，帧的内容就没有疑义了：接收方可以用帧的类型来判决这个帧是携带了 MPLS 还是一个常规的数据报。

MPLS 首部由一个或多个表项组成，每个表项长 32 位，指定了一个标记以及用于控制标记处理的信息。图 17.4 描绘的是首部中的一个表项的格式。



图 17.4 MPLS 首部表项中的字段。MPLS 首部由一个或多个这样的表项组成。其中并没有字段用来定义首部的整体长度

表项的第一个字段是标记（LABEL）字段；当这个表项到达堆栈的顶部时（即首部中的第一个表项），MPLS 就会使用这个标记来转发分组^②。标记为 EXP 的字段保留为实验用。S 位为 1 时表

^① 虽然 MPLS 为多播也分配了一个类型，但是它对多播的处理还有很多问题。

^② 虽然通常 MPLS 的每个标记对应于一个数组的索引，但 MPLS 并没有要求必须这样做，也可以在中间增加一

示这是堆栈的底部（即首部里的最后一个表项），否则 S 置 0。最后是生存时间（TTL）字段，它和 IP 数据报首部中的 TTL 字段类似：沿途使用标记来转发分组的每个交换机都会递减 TTL 字段的值。如果 TTL 达到 0，MPLS 则丢弃该分组。用这种方法可以防止分组循环不止，即使管理员在设置交换机时因失误而意外地创建了一个转发环路。

注意，一个 MPLS 标记的宽度为 20 位。从理论上讲，在设置 MPLS 时可以使用标记的所有 20 位，并同时容纳高达 2^{20} 个流（即 1,048,576 个流）。但是，实际上几乎没有哪个 MPLS 实例会使用大数目的流，因为通常需要管理员手工管理和设置每条交换路径。

17.9 标记交换路由器

实现了 MPLS 的路由器称为**标记交换路由器**（Label Switching Router，简称 LSR）。通常 LSR 由增补了 MPLS 处理能力的常规路由器形成。因此，除了能够交换 MPLS 通信量之外，LSR 通常也可以作为非 MPLS 的互联网和 MPLS 核心之间的接口。也就是说，LSR 可以接受来自常规网络的数据报，给该数据报分类，并分配一个初始的 MPLS 标记。它还能通过删除标记来终止一条 MPLS 路径，然后将数据报通过常规网络转发。这些功能称为**MPLS 入口**（MPLS ingress）和**MPLS 出口**（MPLS egress）功能。

LSR 内部的表用来定义每个标记的动作，称为**下一跳标记转发表**（Next Hop Label Forwarding Table），表中的每个表项称为**下一跳标记转发表项**（Next Hop Label Forwarding Entry，简称 NHLFE）。每个 NHLFE 至少定义两项内容，也可能定义三项以上的内容：

1. 下一跳信息（如外出接口）
2. 应当执行的操作
3. 所使用的封装（可选）
4. 如何加密处理标记（可选）
5. 处理该分组所需的其他信息（可选）

NHLFE 中的操作指明了分组是转换自或转换往分层 MPLS 的另一层，还是仅仅在同一层上沿路径交换。存在以下几种可能性：

1. 用指定的新标记取代栈顶部的标记，然后在当前层上继续转发。
2. 通过弹出栈中的顶层标记来退出多层 MPLS 中的一层。这时会使用栈中的下一个标记来转发该数据报，或者在栈中没有标记的情况下使用常规转发。
3. 用指定的新标记取代栈顶部的标记，然后在栈中压入一个或多个新标记，以增加多层 MPLS 的层次。

17.10 控制过程和标记分发

前面的讨论都集中在**数据路径**（data path）的处理问题上（即如何转发分组）。除此之外，定义 MPLS 的工程师还考虑到了用于**控制路径**（control path）的机制。控制路径处理指的是配置和管理：控制路径协议使管理员更容易创建或管理**标记交换路径**（Label Switched Path，简称 LSP）。

控制路径协议提供的主要功能是对标记的自动选择。也就是说，协议允许管理员在不用手工配置沿路径每个 LSR 的情况下建立 MPLS 路径。协议允许一条路径上的两个 LSR 为它们之间的链路选择一个空闲未用的标记，并为每个流填写 NHLFE 信息，这样在每一跳都可以交换标记。

为一条路径选择标记的过程称为**标记分发**（label distribution）。为了让 MPLS 执行标记分发而建立的两个协议是**标记分发协议**（Label Distribution Protocol，简称 LDP）和**基于约束的路由选择**

次计算，使标记间接地映射到一个索引上。

LDP (Constraint-Based Routing LDP, 简称 CR-LDP), 前者有时也称为 MPLS-LDP。除此之外, OSPF 和 BGP 以及 RSVP 之类的协议都可以经扩展后提供标记分发能力。虽然 IETF 已经认识到标记分发协议的必要性,但是开发了 MPLS 的 IETF 工作组目前还没有指定哪一种协议为必备的标准。

17.11 MPLS 和分片

MPLS 和 IP 分片在两个方面互相影响。首先, 我们曾讲过 MPLS 在通过 LSP 发送数据报时会添加有 4 个或更多八位组的 MPLS 首部。如果所有底层网络具有相同的帧最大尺寸, 那么这样做会带来意外的后果。例如, 考虑连接多个以太网的一个入口 LSR。如果数据报来自一条非 MPLS 路径, 且其大小正好等于以太网的 MTU, 那么再加上 32 位的 MPLS 首部就会使得到的有效载荷超出以太网的 MTU。因此, 入口 LSR 必须对原数据报进行分片, 为每片添加其 MPLS 首部, 并用两个分组而不是一个分组来传送。

MPLS 和分片间的第二个相互影响发生在入口 LSR 接收到的是 IP 分片而不是完整数据报时。在大多数情况下, 除了检查 IP 首部中的地址以外, 在分类时还要检查 TCP 或 UDP 的端口号 (即大多数 MPLS 系统在第四层或更高层执行分类动作)。遗憾的是, 只有数据报的第一个分片中含有运输协议首部。因此, 入口 LSR 只有两个选择, 或者收集分片并重组还原数据报, 或者使用传统的 IP 转发来处理这些分片。在网络内核只允许 MPLS 通信量的情况下, 入口 LSR 如果不重组数据报, 就必须丢弃这些分片。

17.12 网孔拓扑和流量工程

有趣的是, 大多应用 MPLS 的 ISP 沿用一种简单直接的方法: 定义一种**完全网孔** (full mesh) 的 MPLS 路径。也就是说, 如果 ISP 有 K 个网点, 并且与 J 个其他 ISP 对等, 那么 ISP 会为每一对可能的点之间定义一条 MPLS 路径。其结果是在任何一对网点之间移动的通信量都会通过这两个网点之间的同一条 MPLS 路径传输。定义独立的路径带来的好处是能够对从一个网点传到另一个网点的通信量进行度量 (或控制)。例如, 通过监视某一条 MPLS 连接, ISP 可以判断从自己的一个网点通过对等连接到达另一个 ISP 的通信量的大小。

有些网点对完全网孔的想法进一步扩展, 为每对网点之间定义多条路径, 以容纳不同类型的服务。例如, 具有最小跳数的路径可以为话音通信量保留, 因为它需要最小的时延, 而较长的路径用于其他通信量, 如电子邮件和万维网通信量。作为一种选择, MPLS 允许 ISP 在两个不相连的路径之间平衡通信量。MPLS 分类使得为数据选择路径时可以有多种度量方式, 包括 IP 源地址以及运输协议端口号。重点在于:

因为 MPLS 分类可以使用数据报中的任何字段, 包括 IP 源地址, 因此数据报接受的服务可以由发送数据报的消费者决定, 也可以由运载的数据类型决定。

我们用术语**流量工程** (traffic engineering) 来形象地表达创建 MPLS 路径并将数据报分配到每条路径的过程。除了为每条路径定义一组 LSR 之外, 流量工程还允许管理员使用第 28 章将详细讲解的**服务质量** (Quality of Service, 简称 QoS) 技术来控制每条路径上的通信量速率。因此, 可以在一条物理连接上定义两个 MPLS 流, 并可以使用 QoS 技术确保以下情况: 如果每个流都有通信量发送, 可以让底层带宽的 75% 用于某个流量, 而另一个流量只能用 25% 的带宽。

17.13 小结

为了获得更快的速度，交换技术使用索引而非最长前缀匹配查找法，因而交换要承袭面向连接的方式。因为路径沿途上的交换机都能重写标记，所以分配给某个流的标记可以沿途不断变化。

IP 数据报交换的标准称为 MPLS，是由 IETF 创立的。在沿着某条 LSP 发送数据报时，MPLS 先插入一个首部，创建具有一个或多个标记的栈，因此沿路径的其他 LSR 可以使用这些标记来转发数据报，而不需要执行路由选择表的查找工作。入口 LSR 会为来自非 MPLS 的主机或路由器的每个数据报分类，而出口 LSR 则能将来自 MPLS 路径的数据报传递给非 MPLS 主机或路由器。

17.14 深入研究

许多 RFC 都包含了对 MPLS 的建议，有的考虑如何扩展路由选择协议以处理标记交换，有的指出在具体的网络技术下如何使用 MPLS，如 ATM 和帧中继。Rosen et. al. [RFC 3031] 定义了整体体系结构，而 Rosen et. al. [RFC 3032] 则给出了 MPLS 首部和标记栈的细节。Andersson et. al. [RFC 3036] 定义了 LDP，而 Jamoussi [RFC 3212] 定义了基于约束的 LDP。最后，Awdueh et. al. [RFC 2702] 讨论了 MPLS 上的流量工程。

17.15 习题

1. 研究第 31 章中描述的 IPv6。其中哪个机制是直接与交换技术相关的？
2. 阅读有关 MPLS 的资料。MPLS 是否应当有第二层转发（即网桥）以及优化的 IP 转发？试解释原因。
3. 如果来自主机 X 的所有通信量都将通过一个两层 MPLS，为了保证不出现分片，需要采取什么动作？
4. 阅读更多有关基于约束的 LDP 的资料。它所考虑的可能的约束有哪些？
5. 如果你所在的网点的路由器支持 MPLS，就打开 MPLS 交换功能，然后测一测与常规路由选择表查找方法相比，它的速度提高了多少？提示：一定要度量到给定目的站的许多分组，以避免度量受到处理最初分组的影响。
6. 有没有可能做一个实验来判断你的 ISP 是否采用了 MPLS？假设它能传送任意分组。
7. 思科系统公司提供一种称为多层交换（Multi-Layer Switching，简称 MLS）的交换技术。阅读有关 MLS 的资料。MLS 和 MPLS 之间有哪些不同的地方？
8. 如果你所在的网点有一个提供 MLS 服务的 VLAN 交换机，打开这项服务并测试，如果一方发送了一个有效的以太网帧，但这个帧中所含的 IP 数据报是错误的，会发生什么情况？一个第二层交换机应该检查 IP 首部吗？试说明理由。

第18章 移动 IP

18.1 引言

前几章讨论的都是台式计算机的 IP 编址和转发机制。本章要考虑的是如何对 IP 技术进行扩展，设计出允许便携式计算机从一个网络移动到另一个网络的 IP 技术。

18.2 移动性、路由选择和编址

从广义上讲，术语**移动计算**（mobile computing）指的是允许计算机从一个位置移动到另一个位置的系统。移动性通常与无线技术相关，因为用无线技术来实现移动是很方便的。由于 IP 的编址机制是针对固定环境而设计和优化的，它在实现可移动性方面有些困难，原因在于主机地址前缀所标识的就是与主机相连接的网络。因此，把主机移到一个新的网络上就意味着：

- 主机地址必须改变
- 路由器必须在整个因特网上传播特定主机的路由

这两个选择都是不可行的。一方面，更改地址会打断所有现有运输层连接，甚至可能需要重新启动某些网络服务。此外，如果主机联系的是一个使用反向 DNS 查找法实现鉴别的服务器，那么可能还要对 DNS 进行修改。另一方面，特定主机路由选择方式的规模之巨大不可估量，因为如果每个主机都具有一条路由，那么在通信和保存时将需要超大的带宽和存储器。

18.3 移动 IP 特性

为了实现 IP 移动性，IETF 设计了一种技术。其正式名称是 **IP 移动性支持**（IP mobility support），而人们一般都称其为**移动 IP**（mobile IP）。它的一般特征包括：

透明性：对应用和运输层协议以及变动涉及不到的路由器来说，移动性是透明的。也就是说，如果在转换期间有一条 TCP 连接没有被使用过，则变化过后该连接依然有效。

与 IPv4 的互操作性：使用移动 IP 的主机既可以与运行常规 IPv4 软件的台式主机互操作，也可以与其他移动主机互操作，而且分配给移动主机的地址就是常规 IP 地址。

物理广泛性：解决方案允许在整个因特网范围内的移动。

安全性：移动 IP 提供了可用于确保所有报文都经过鉴别的安全设施（也就是说，防止其他计算机假冒移动主机）。

宏观移动性：移动 IP 不是试图处理迅速且正在进行的网络变换，而是重点关注持续时间较长的移动问题（例如，带着便携式计算机进行商业旅行的用户）。

18.4 移动 IP 操作概述

移动性的最大挑战在于既要允许主机保留其地址，又不要求路由器掌握特定主机的路由信息。移动 IP 解决这个问题的方法是允许一台计算机同时拥有两个地址：一个是应用程序使用的长期且固定的**主地址**（primary address），另一个是临时的**次地址**（secondary address）。这个临时地址只在计算机访问特定位置时有效。

移动主机的主地址是主机在**归属网** (home network) 上分配得到的。当它移动到**外地网** (foreign network) 并获得一个次地址后，移动主机必须将这个次地址发往**归属代理** (home agent)，通常这个代理是归属网中的一个路由器。这个代理同意截获发送给移动主机主地址的数据报，并使用 IP-in-IP 封装方式，把每个数据报以隧道方式传递到次地址。

如果移动主机再次变换地点，它会获得一个新的次地址，并将其新位置通知归属代理。当移动主机返回归属网后，它必须联系归属代理以便**撤销登记** (deregister)，也就是说，让代理停止截获数据报。类似地，移动主机可以选择在任何时候撤销登记（例如动身离开外地时）。

我们曾经讲过，移动 IP 是为宏观移动性设计的，而不是为快速连续的移动设计的。原因很明显，那就是额外开销。具体来说，在主机移动后，它必须检测到自己已经移动了，并通过与外地网的通信获得一个次地址，然后通过因特网与其归属网中的代理进行通信，并交待转发事宜。重点是：

由于每次移动后都需要相当大的额外开销，所以移动 IP 是为主机移动并不频繁，并且在给定位置停留相对较长的一段时期的情况而设计的。

18.5 移动编址细节

移动主机的主地址，或称**归属地址** (home address) 就是常规的 IP 地址，它的分配和管理与常规 IP 地址相同。在主机访问某个外地网时就会获得一个次地址，又称为**转交地址** (care-of address)，只有移动主机以及归属网或外地网中的代理上的 IP 软件才会使用这个地址。应用程序只使用主地址。

实际上有两种类型的转交地址。至于在某个外地网上可以使用哪种类型的转交地址，则由网络管理员决定。这两种类型的区别在于地址的获取方式以及负责转发的实体不同。第一种称为**同址转交地址** (co-located care-of address)，它要求移动主机自己来处理所有转发和隧道动作。从本质上说就是一台移动主机获取一个当地的地址（如通过第 22 章讨论的 DHCP 协议），然后处理联络归属代理进行注册登记以及如何接受发送到临时地址上的数据报等细节。

同址转交地址的主要优点是它的移动性，因为由主机自己处理所有通信事宜，外地网不需要有任何特殊的设施。它的主要缺点则是移动主机上需要有附加的软件。

移动 IP 的第二种形式称为**外地代理转交地址** (foreign agent care-of address)，它要求在远程网络上有一台活动的称为**外地代理** (foreign agent) 的参与者。当移动主机到达外地网后，主机必须首先找到具有代理身份的机器，然后与该代理通信，以获得一个转交地址。令人惊讶的是，外地代理并不需要为每台移动主机分配一个唯一的地址。事实上，该代理可以把自己的 IP 地址分配给所有到访的移动主机。

18.6 外地代理的发现

外地代理发现 (foreign agent discovery) 过程使用的是 **ICMP 路由器发现** (ICMP router advertisement) 机制，也就是每个路由器定期发送一个 **ICMP 路由器通知** (ICMP router advertisement) 报文，同时也允许主机发送 **ICMP 路由器恳求** (ICMP router solicitation) 报文，以索求通知报文^①。如果某个路由器具有外地代理的身份，那么它会在通知报文的尾部附加一个**移动代理扩展** (mobile agent extension)^②。移动扩展信息并没有使用独立的 ICMP 报文类型。事实上，如果 IP 首部中指明的数据报长度大于 ICMP 路由器发现报文中指明的长度，则说明报文中存在移动扩展。图 18.1 所

^① 移动主机也可以向**全代理群组** (all agents group) (224.0.0.11) 发送多播。

^② 移动代理还要给每个报文附加一个**前缀扩展** (prefix extension)，这个扩展指明的是该网络的前缀，移动主机通过它来确定自己已经移动到了一个新的网络上。

示为移动扩展的格式。

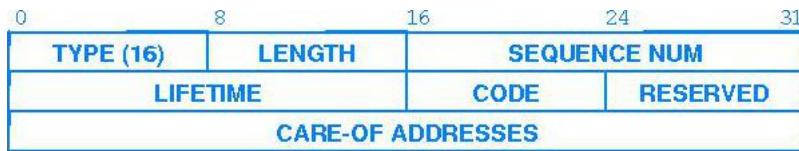


图 18.1 移动代理通知扩展报文的格式。这个扩展将附加在 ICMP 路由器通知的尾部

报文由一个八位组的类型 (TYPE) 字段开始，后面跟着一个八位组的长度 (LENGTH) 字段。长度字段指明了以八位组计的扩展报文长度，其中没有将类型和长度八位组计算在内。生存时间 (LIFETIME) 字段指明了代理接受登记请求所允许的以秒计的最长时间，全 1 表示无穷大。序号 (SEQUENCE NUM) 字段为报文指明一个序列号，以允许接收方判断报文的丢失。代码 (CODE) 字段中的每一位定义了一种特殊的代理特性，如图 18.2 所示。^{①②}

Bit	Meaning
7	Registration with an agent is required even when using a co-located care-of address
6	The agent is busy and is not accepting registrations
5	Agent functions as a home agent
4	Agent functions as a foreign agent
3	Agent uses minimal encapsulation [†]
2	Agent uses GRE-style encapsulation [‡]
1	Unused (must be zero)
0	Agent supports reversed tunneling

图 18.2 移动代理通知报文中代码字段的各位的含义，其中 0 位是八位组的最低有效位

18.7 代理登记

移动主机必须首先进行登记，才能够在外地接收数据报。登记 (registration) 过程允许主机：

- 在外地网的一台代理上进行登记
- 直接在它的归属代理上进行登记，以请求转发
- 重建过期的登记
- 在返回归属网后撤销登记

如果移动主机获得的是同址转交地址，则由移动主机直接登记，即移动主机通过这个转交地址直接与其归属代理通信。如果移动主机得到的是来自外地代理的转交地址，则移动主机应允许外地代理作为自己的全权代表完成登记。

18.8 登记报文格式

所有登记报文都是通过 UDP 发送的。登记报文由一组固定长度的字段开始，后面跟着可变长度的扩展部分。所有请求都要包含一个**移动—归属鉴别扩展** (mobile-home authentication extension)，以允许归属代理证实主机的身份。图 18.3 所示为登记报文的格式。

^① 最小封装 (Minimal encapsulation) 是 IP-in-IP 隧道的一个标准，它将原始首部进行简化，以节省空间。

^② 通用路由选择封装 (Generic Routing Encapsulation, 简称 GRE) 是允许对任意协议进行封装的一种标准，IP-in-IP 是其中的一个特例。

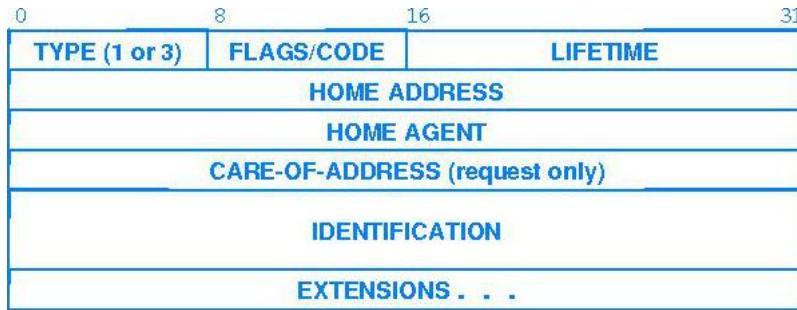


图 18.3 移动 IP 登记请求或回答报文的格式

类型 (TYPE) 字段指明了该报文是登记请求 (1) 还是登记回答 (3)。生存时间 (LIFETIME) 字段指明了以秒计的登记有效时间 (0 表示请求立即撤销登记; 全 1 表示无穷大)。接下来的归属地址 (HOME ADDRESS) 和归属代理 (HOME AGENT) 以及转交地址 (CARE-OF ADDRESS) 字段指明的是移动主机的两个 IP 地址及其归属代理的地址。标识 (IDENTIFICATION) 字段包含的是一个 64 位数字, 该数字由移动主机生成, 用于匹配发出的请求和收到的回答, 同时也防止移动主机收到过时的老报文。标志 / 代码 (FLAGS/CODE) 字段在登记回答报文中表示结果代码, 在登记请求报文中用于指明转发细节, 诸如登记是否对应于另一个 (即新的) 地址请求, 以及代理在向移动主机转发数据报时应当使用的封装。

在使用同址转交地址的情况下, 移动主机直接向其归属代理发送登记请求。否则, 移动主机将请求发送到外地代理, 然后由外地代理将该请求转发到归属代理。只要用到了外地代理, 外地代理和归属代理都要批准才能完成登记。

18.9 与外地代理之间的通信

我们曾提及外地代理可以将自己的一个 IP 地址分配为转交地址。其结果必然导致移动主机在外地网上的地址并不唯一。带来的问题是: 如果移动主机在网络中没有一个合法的 IP 地址, 它怎样与外地代理相互通信呢? 它们之间的通信需要放宽 IP 编址的规则, 并使用另一种地址绑定机制: 当移动主机向外地代理发送数据报时, 允许移动主机使用自己的归属地址作为 IP 源地址。反之, 当外地代理向移动主机发送数据报时, 允许代理使用移动主机的归属地址作为 IP 目的地址。为了避免发送无效的 ARP 请求, 在第一个请求到达时, 外地代理要记录移动主机的硬件地址, 并使用硬件地址发送回答。因此, 虽然没有使用 ARP, 但代理可以通过硬件单播向移动主机发送数据报。总结如下:

如果移动主机没有唯一的外地地址, 外地代理必须使用移动主机的归属地址进行通信。此时不依靠 ARP 进行地址绑定, 而是由代理在请求到达时记录移动主机的硬件地址, 并将记录的信息用于绑定。

18.10 数据报的传输和接收

一旦登记完成, 外地网中的移动主机就能与任何一台计算机进行通信。要做到这一点, 移动主机创建的数据报的目的地址字段中的地址就是目的计算机的地址, 而源地址字段中的地址是移动主

机的归属地址^①。数据报沿着最短路径从外地网到达目的计算机，但是返回的数据报不会沿着最短路径直接到达移动主机，而是首先到达移动主机的归属网。因为归属代理在登记时已经掌握了移动主机的位置，所以它可以截获数据报，并使用 IP-in-IP 封装通过隧道把数据报送到转交地址。总结如下：

因为移动主机在与任何终点之间进行通信时都会使用自己的归属地址作为源地址，所以返回的数据报将会转发到移动主机的归属网上，此时有一个代理截获这个数据报，将其封装在另一个数据报中，然后，或者直接转发给移动主机，或者转发给移动主机正在使用的外地代理。

18.11 两次穿越问题

上面的描述展示了移动 IP 的主要缺点：转发效率低。因为移动主机使用的是自己的归属地址，发送给移动主机的数据报首先会被转发给移动主机的归属网，然后才会到达该主机。因为计算机通信常常表现出**引用空间局部性** (spatial locality of reference)，即访问外地网的移动主机趋向于和当地网络上的计算机进行通信，所以这个问题尤其严重。为了理解为什么移动 IP 在处理空间局部性时的能力较差，可考虑图 18.4。

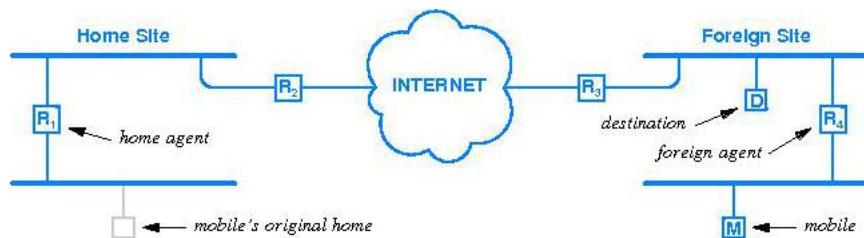


图 18.4 移动 IP 路由选择效率很低的一种拓扑结构。当移动主机 M 与当地的目的主机 D 进行通信时，来自 D 的数据报要通过因特网到达移动主机的归属代理，然后再返回到移动主机

在图 18.4 中，移动主机 M 从它的归属网移动到了一个外地网。假设该移动主机已经在自己的归属代理（路由器 R₁）上进行了登记，并且归属代理也已经同意转发数据报。现在考虑移动主机与当地的目的主机 D 之间的通信。从 M 到 D 的数据报经过路由器 R₄交付到达 D。但是，因为从 D 发送给 M 的数据报中包含的是 M 的归属地址，这些数据报会经路由器 R₃，通过因特网到达移动主机的归属网。当数据报到达 R₁（移动主机的归属代理）时，它们又要通过因特网以隧道方式返回外地网（或者直接交付给 M，或者转发给外地代理）。因为通过因特网的传输比本地交付代价高得多，所以我们把上面描述的情况称为**两次穿越问题**(two-crossing problem)，有时也称为**2X 问题**(2X problem)^②。

如果某个网点预计到访的移动主机会频繁地与当地的计算机进行交互，那么这个网点可以通过向网点内所有成员传播一个特定主机路由的方法来实现优化。然后，在移动主机离开时删除这个特定主机路由。当然，对位于该外地网之外的目的站来说，2X 问题仍然存在。

总结如下：

移动 IP 带来了路由选择效率低的问题，称为**2X 问题**。当移动主机与外地网中或邻近的一

^① 外地网和将其连接到因特网的 ISP 必须同意传输使用任意源地址的数据报。

^② 如果目的主机 D 并非紧邻移动主机，那么所发生的问题的严重性也比较轻，这样的问题称为**三角转发**(triangle forwarding) 或**狗腿转发** (dog-leg forwarding)。

台计算机之间进行通信时，就会发生这种问题。发往移动主机的每个数据报都要通过因特网先到达该移动主机的归属代理，再由归属代理将数据报转发回外地网。要消除这个低效率的问题，就需要传播一条特定主机的路由。

18.12 与归属网上的计算机之间的通信

前面讲过，当移动主机访问外地网时，移动主机的归属代理必须截获所有发往该移动主机的数据报。因为通常这个代理运行在连接归属网到因特网的路由器上，所以归属代理能够轻松截获所有从外部进入的数据报，但是有一种特殊情况：当移动主机的归属网中有一台主机向移动主机发送数据报时。因为 IP 指明的是通过本地网络直接交付，所以发送方不会将数据报转发到路由器上。事实上，发送方将试图通过 ARP 得到移动主机的硬件地址，封装数据报并传输数据报。

如果移动主机已经移动到了一个外地网上，归属代理必须安排好如何转发所有数据报，包括来自本地主机的。因此，归属代理使用一种**代理 ARP** (proxy ARP) 的方式来处理本地传输：只要归属网中有一台主机对移动主机的地址进行解析，归属代理就要响应该请求并提供自己的硬件地址。也就是说，本地主机被归属代理蒙骗而将所有目的为移动主机的数据报都转发到归属代理上，然后由归属代理将这些数据报转发给移动主机。

18.13 小结

移动 IP 允许一台计算机从一个网络移动到另一个网络，而不用更改其 IP 地址，也不需要所有路由器都传播特定主机的路由。当移动主机从自己原本所在的归属网移动到一个外地网时，它必须获得另一个临时地址，这个地址称为转交地址。应用程序使用的是移动主机本身的归属地址，而转交地址仅由下层的网络软件使用，以便经过外地网进行转发和交付。

一旦检测到自己已经移动了，移动主机或者获得一个同址转交地址，或者发现一个外地移动代理，并请求该代理分配一个转交地址。获得转交地址以后，移动主机向它的归属代理进行登记（或者直接登记，或者通过外地代理间接登记），并请求该代理转发数据报。

一旦完成登记，移动主机就可以和因特网上的任意计算机进行通信。从移动主机发送出的数据报直接转发给指定的地址。但是，返回给移动主机的所有数据报都将沿着路由到达移动主机的归属网，由归属代理截获，并将其封装在 IP 中，然后利用隧道传输给移动主机。

18.14 深入研究

Perkins [RFC 3344] 描述了 IP 移动性支持 (IP Mobility Support)，同时定义了相关报文的格式。Perkins [RFC 2003] 和 Perkins [RFC 2004] 以及 Hanks et. al. [RFC 1701] 详细描述了三种 IP-in-IP 封装机制。Montenegro [RFC 3024] 描述的是移动 IP 的反向隧道机制。最后，Johnson et. al. [RFC 3775] 讨论了 IPv6 的移动性支持。

18.15 习题

1. 比较 RFC 2003 和 RFC 2004 中的封装机制，它们各自的优缺点是什么？
2. 仔细阅读移动 IP 规范。路由器必须以怎样的频率发送移动代理通知？为什么？
3. 查阅移动 IP 规范。当外地代理把一个登记请求转发给移动主机的归属代理时，使用的是哪个协议端口？为什么？

-
4. 移动 IP 规范允许一个路由器既作为一个网络的归属代理，又作为外地代理，用于支持访问该网络的计算机。使用一个路由器完成这两种功能的优点和缺点各是什么？
 5. 移动 IP 规范从概念上定义了三种独立的鉴别方式：移动主机到归属代理、移动主机到外地代理以及外地代理到归属代理。这样划分的优点是什么？缺点是什么？
 6. 阅读移动 IP 规范，判断移动主机是如何加入一个多播组的？多播数据报将如何进行路由选择以到达移动主机？其优化方案是什么？

第19章 专用网络互连（NAT 和 VPN）

19.1 引言

前面的章节将互联网视为由路由器连接起来的许多网络构成的单层抽象结构。本章将探讨的是另一种结构：双层互联网体系结构。在这种结构中，每个机构有一个专用互联网，另外由一个中央互联网来连接这些专用互联网。

本章将讨论双层体系结构所采用的一些技术。其中一种技术用于解决受限的地址空间这个实际问题，另一种技术在**保密**（privacy）方面提供了一些增强的功能，以防止不相关的外人查看数据。

19.2 专用的和混合的网络

单层互联网体系结构的一个主要缺陷是缺乏保密性。如果一个机构由多个网点构成，那么经过互联网在这些网点之间往返的数据报的内容有可能被外人窥视，因为那些因特网服务提供者拥有这些数据报沿途经过的网络。而双层体系结构会区别对待**内部的**（internal）和**外部的**（external）数据报，也就是说，是同属一个机构的两台计算机之间发送的数据报，还是分属不同机构的两台计算机之间发送的数据报。这样做的目的是保持内部数据报**保密**（private），同时仍然允许其外部通信。

要确保机构内的计算机之间通信的保密性，最简单的方法是建立一个完全隔离的**专用互联网**（private internet），通常称为**专用网**（private network）或**专用内联网**（private intranet）。专用网由租用数字线路来连接各个网点，并且电信公司保证不让外人访问这些线路，因此所有数据都是保密的。

但是，完全隔离的内联网也不是人们通常所希望的，原因有两个。首先，大多数机构需要接入因特网，以便联络其消费者和供应商。其次，租用线路成本过高。因此，许多机构都在寻找其他成本较低的选择。例如，就相同的容量而言，公共电信公司的帧中继连接比租用线路便宜得多。另一种降低成本的办法是使用少数租用线路来连接网点，并依靠这些中间网点转发数据报。

成本最低的选择是使用因特网作为网点之间的连接。由此带来的问题是：

用因特网连接其网点的机构怎样才能做到数据保密？

解决这个问题的技术是允许该机构配置其**虚拟专用网**（Virtual Private Network，简称 VPN）^①。一方面 VPN 是**专用的**（private），因为这种技术保证任意一对计算机之间的通信对于外人来说是隐蔽的。另一方面 VPN 是**虚拟的**（virtual），因为它不需要使用租用线路连接各个网点。

使 VPN 得以实现的基本技术有两种：**隧道传输**（tunneling）技术和**加密**（encryption）技术。在第 16 章和第 18 章中已经涉及了隧道传输技术。VPN 的基本思路也一样：VPN 定义的是从某网点的一个路由器到另一个网点的路由器之间的通过因特网的一条隧道，同时它使用 IP-in-IP 封装经过隧道转发数据报。

尽管使用了相同的基本概念，但是 VPN 隧道与以前描述的隧道差别很大。特别是为了保证其保密性，VPN 在将外发数据报封装到另一个数据报中进行传输之前，先要将每个数据报加密^②。图 19.1 描绘了这个概念。

^① 这个名称有点使用不当，因为这种技术实际上是一个虚拟专用内联网。

^② 在第 30 章中将涉及 IP 的安全性，并且会讨论 IPsec 使用的封装。



图 19.1 VPN 使用 IP-in-IP 封装的说明。为了确保保密性，发送前先要加密内层数据报

如图 19.1 所示，整个内层数据报（包括首部）在封装前先要进行加密。当数据报通过隧道到达接收路由器时，路由器将数据报解密，还原出内层数据报，然后将其转发。尽管外层数据报在穿过隧道时可能经过任意网络，但是外人无法将内容解密，因为他们没有密钥。甚至连源站和目的站的标识也是隐藏的，因为内层数据报的首部也加了密。这样，只有外层数据报首部中的地址是可见的，其中源地址是隧道一端的路由器的 IP 地址，目的地址是隧道另一端的路由器的 IP 地址。

总结如下：

虚拟专用网通过因特网发送数据，但是会对网点间的传输进行加密，以保证其保密性。

19.3 VPN 编址技术和路由选择

要了解 VPN 的编址和路由选择技术，最简单的办法是将每条 VPN 隧道视为专用网中的一条租用线路。与专用网的情况相同，路由器包含了到达机构内每个目的站的明确路由。但是，VPN 不是为数据选择通过租用线路的路由，而是选择通过隧道的路由。例如，图 19.2 所示为一个 VPN 及其中一个处理隧道传输的路由器所用的路由表。

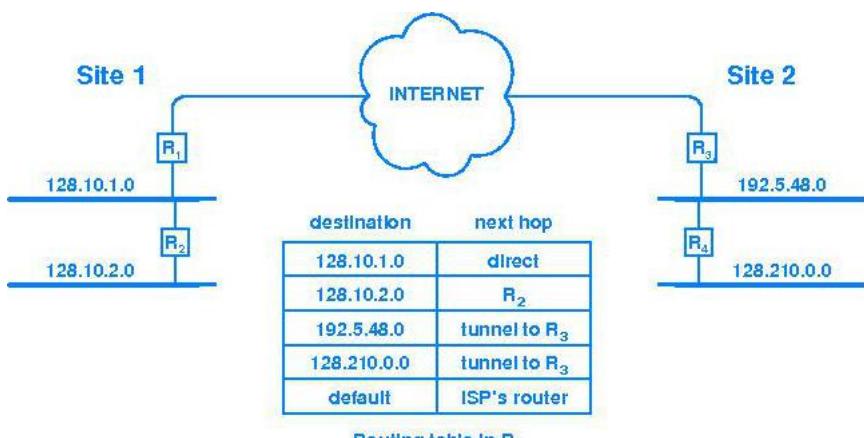


图 19.2 跨越两个网点的 VPN，以及 R₁ 的路由表。从 R₁ 到 R₃ 的隧道被配置为类似点到点的租用线路

作为 VPN 转发的一个例子，我们来考虑这样的情况：数据报从网络 128.10.2.0 中的计算机发送到网络 128.210.0.0 中的计算机上。发送主机把数据报转发给 R₂，R₂ 再把数据报转发给 R₁。按照 R₁ 的路由表，数据报必须通过隧道发送给 R₃。因此，R₁ 对数据报进行加密，把它封装在外层数据报的数据区中，目的地址是 R₃。然后，R₁ 通过本地 ISP 经因特网转发外层数据报。数据报到达 R₃ 后，R₃ 识别出它是以隧道方式来自 R₁，此时 R₃ 对数据区进行解码，还原出原始数据报，在路由选择表中查找它的目的地址，然后把数据报转发给 R₄ 进行最后的交付。

19.4 扩展 VPN 技术应用到个人主机

像这种能够为机构网点之间提供的成本不高且又能保密的通信技术，很容易扩展成个人计算机可利用的技术。一个典型的例子是，许多公司向其雇员提供 VPN 软件，以便每个雇员都能利用传统的因特网连接安全地访问本公司的网络。从本质上讲，VPN 软件将计算机中的协议栈进行重新配置，使它只能与公司的 VPN 路由器通信。外发数据报加密后发送到公司进行处理，而来自公司的传入数据报被接受并解密。因此，读者应当记住，在本书有关 VPN 的讨论中，都可以用单个计算机来代替一个网点。

19.5 使用专用地址的 VPN

VPN 提供给一个机构的编址方案与专用网所提供的一样。如果 VPN 中的主机不需要使用到因特网的连接，那么 VPN 可以配置使用任何 IP 地址。如果主机需要访问因特网，则使用混合编址方案。它们之间的细微差别在于，当使用专用编址方案时，每个网点只需要一个全球有效的 IP 地址，用来进行隧道传输。图 19.3 描绘了这一概念。

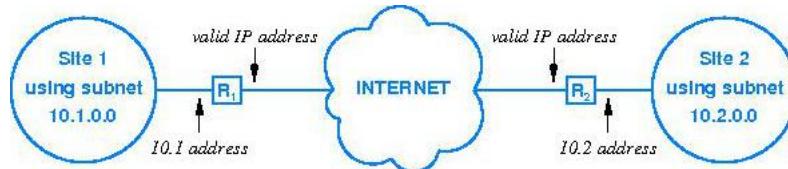


图 19.3 通过因特网连接两个完全专用网点的 VPN 编址示例，其中每个网点的计算机都使用专用地址

如图 19.3 所示，网点 1 使用子网 10.1.0.0/16，网点 2 使用子网 10.2.0.0/16。只需要两个全球有效的 IP 地址，其中一个分配给从路由器 R₁ 到因特网的连接，另一个分配给从路由器 R₂ 到因特网的连接。这两个网点中的路由选择表为专用地址指明路由，只有 VPN 隧道传输软件需要知道或使用全球有效的 IP 地址。

VPN 使用的编址结构与专用网的相同。完全隔离的 VPN 的主机可以使用任何地址，但是具有全球有效 IP 地址的混合结构必须向主机提供到因特网的访问。还是那个老问题：“在没有为每个主机都分配全球有效 IP 地址的情况下，网点怎样才能向其提供到因特网的访问？”通常有两种解决方案。

第一种解决方案称为**应用网关**（application gateway），它向主机提供到因特网服务的访问，但不提供 IP 层接入。每个网点中有一个多地址主机，既可以连接因特网（有一个全球有效的 IP 地址），又可以连接内部网络（使用一个专用 IP 地址）。在这个多地址主机上运行着一组称为应用网关的应用程序，其中每个程序处理一种服务。网点中的其他主机并不是把数据报直接发往因特网，而是把每个请求都发送到多地址主机上的相应应用网关，由它来访问因特网上的服务，然后再将信息通过内部网络返回到请求主机。例如，第 26 章描述了可以在外部主机和内部主机之间传递电子邮件报文的电子邮件转发器。

应用网关方法的主要优势在于不需要改变底层基础设施和编址方案就能正常工作。主要的缺点在于缺乏通用性，可总结如下：

每个应用网关只处理一种具体的服务，对于多种服务则需要多个网关。

结果是，尽管应用网关在特殊环境下十分有用，但它不能以通用的方式解决问题。因此，产生了另一种解决方案。

19.6 网络地址转换。

已经有一种技术解决了在不需要为网点中的每台主机都分配全球有效 IP 地址的情况下，在网点中的主机和其他因特网部分之间提供 IP 层接入的基本问题。这种技术称为**网络地址转换**（Network Address Translation，简称 NAT）技术，它要求每个网点具有一条到因特网的连接，并且至少有一个全球有效 IP 地址 G。将地址 G 分配给一台计算机（一台多地址主机或路由器），由它来连接网点和因特网，并且运行 NAT 软件。在非正式场合下，我们把运行 NAT 软件的计算机称为**NAT 盒**（NAT box），所有数据报在从网点到因特网或从因特网到网点的传输过程中都要通过 NAT 盒。

NAT 对传入数据报和外发数据报中的地址进行转换，用 G 替换每个外发数据报中的源地址，而用正确主机的专用地址替换每个传入数据报的目的地址。这样，从外部主机的角度来看，所有数据报都来自 NAT 盒，而所有响应也返回到 NAT 盒。从内部主机的角度来看，NAT 盒看上去是一个可达因特网的路由器。

NAT 的主要优势在于结合了通用性和透明性。NAT 比应用网关更通用，因为它允许任意内部主机访问因特网上任一台计算机中的任何服务。NAT 是透明的，因为它允许内部主机使用专用（即不可路由的）地址发送和接收数据报。

总结如下：

网络地址转换技术提供了从使用专用地址的主机到因特网之间的透明 IP 层访问。

19.7 NAT 转换表的创建

在 NAT 概述中省略了一个重要细节，没有指出 NAT 怎样知道应该由哪台内部主机接收来自因特网的数据报。实际上，NAT 维护着一个转换表，该表用于实施映射。表中的每个条目指明了两项：因特网中的主机的 IP 地址，网点中的主机的内部 IP 地址。当传入数据报从因特网到达时，NAT 在转换表中查找数据报的目的地址，提取相应的内部主机地址，用主机的地址替换数据报的目的地址，并通过本地网络把该数据报转发给主机^①。

在数据报从因特网到达之前，必须已经存在 NAT 转换表了，否则 NAT 无法识别数据报应该转发到哪一台内部主机。那么，这个表是何时又是怎样初始化的呢？有下面这几种可能性：

- **人工初始化**。在进行任何通信之前，由管理员手工配置转换表。
- **外发数据报**。这个表的建立是内部主机向外发送数据报时的副产品。NAT 利用外发数据报，在转换表中创建了一个表项，记录其源地址和目的地址。
- **传入名字查找**。作为进行域名查找的副产品而建立转换表。当因特网上的主机在查找一台内部主机的域名时^②，DNS 软件答复的地址是 G，同时它会在 NAT 转换表中创建一个条目，以便向正确的内部主机转发传入数据报。

每种初始化技术都有各自的优缺点。人工初始化提供了永久的映射关系，并允许 IP 数据报在任何时候的双向传输。使用外发数据报初始化转换表的优势在于自动化，但它不允许从外部发起的通信。使用传入域名查找则需要修改域名软件。它允许从网点外部发起的通信，但只有发送方在发送数据报之前先进行域名查找时才有效。

大多数 NAT 实现方式都使用外发数据报来初始化转换表，这种方法在各 ISP 中特别流行。为了了解其原因，可以想象一个为拨号客户提供服务的小型 ISP。图 19.4 所示的正是这种结构。

^① 在替换数据报首部中的地址时，NAT 必须重新计算首部的检验和。

^② 第 23 章描述了**域名系统**（Domain Name System，简称 DNS）的工作方式。

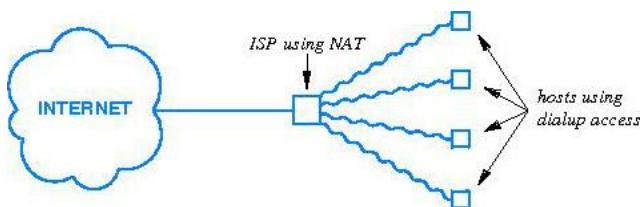


图 19.4 向拨号客户提供服务的小型 ISP 所使用的 NAT。NAT 转换允许 ISP 为每个拨号客户分配一个专用的地址

ISP 必须在客户拨入时为其分配一个 IP 地址。NAT 允许 ISP 分配专用地址（例如，给第一个客户分配 10.0.0.1，给第二个客户分配 10.0.0.2，以此类推）。当客户向因特网上的目的站发送数据报时，NAT 利用外发数据报对转换表进行初始化。

19.8 多地址 NAT

到目前为止，我们已经描述了在外部地址和内部地址之间实现一对一地址映射的一种简化 NAT 实现。也就是说，一对一映射允许网点上任何时候最多只有一台计算机访问因特网上的一台给定计算机。在实践中，使用了更复杂的 NAT 形式，允许一个网点中的多台主机并发访问给定的某个外部地址。

NAT 的一种允许并发访问的变体虽然保留了一对一映射，但允许 NAT 盒持有多个因特网地址。这种机制称为**多地址 NAT**(multi-address NAT)，它给 NAT 盒分配一组 K 个全球有效地址 G_1, G_2, \dots, G_K 。当第一台内部主机访问某个给定目的站时，NAT 盒选择地址 G_1 ，在转换表中添加一个条目，并发送数据报。如果另一台主机联系同一个目的站，NAT 盒选择地址 G_2 ，以此类推。这样，多地址 NAT 最多允许 K 个内部主机访问某个给定目的站。

19.9 端口映射 NAT

另一种流行的 NAT 变体通过转换 TCP 或 UDP 协议端口号以及地址来提供并发性。这种方案有时称为**网络地址端口转换**（Network Address Port Translation，简称 NAPT），把 NAT 转换表扩展为包含更多的字段。除了一对源和目的 IP 地址以外，这个表还包含了一对源和目的协议端口号，以及 NAT 盒使用的一个协议端口号。图 19.5 所示为这个表的内容。

Private Address	Private Port	External Address	External Port	NAT Port	Protocol Used
10.0.0.5	21023	128.10.19.20	80	14003	tcp
10.0.0.1	386	128.10.19.20	80	14010	tcp
10.0.2.6	26600	207.200.75.200	21	14012	tcp
10.0.0.3	1274	128.210.1.5	80	14007	tcp

图 19.5 NAPT 使用的转换表的例子。这个表中除了有 IP 地址以外还有端口号

图 19.5 中的表的各条目分别对应于 4 台内部计算机，这些计算机当前正在访问全球因特网上的目的站。所有通信都使用 TCP，值得注意的是，有两台内部主机，10.0.0.5 和 10.0.0.1，都在访问计算机 128.10.19.20 上的协议端口 80（Web 服务器）。在这个例子中，两个连接使用的两个源端口恰巧不同，但实际上源端口的唯一性无法保证，两台内部主机完全有可能选择相同的源端口号。因此，为了避免潜在的冲突，NAT 给用于因特网的每个通信分配了一个唯一的端口号。前面提到过，TCP 用一个四元组来标识每个连接，这个四元组代表了两端各自的 IP 地址和协议端口号。表中的前两项对应的是两台内部主机各自的 TCP 连接，它们用以下四元组来标识：

(10.0.0.5, 21023, 128.10.19.20, 80)

(10.0.0.1, 386, 128.10.19.20, 80)

但是，同样是这两条 TCP 连接，在执行了 NAPT 转换后，因特网中接收数据报的计算机在识别它们时用的是四元组：

(G, 14003, 128.10.19.20, 80)

(G, 14010, 128.10.19.20, 80)

其中，G 是 NAT 盒的全球有效地址。

NAPT 的主要优势在于，能够仅用一个全球有效 IP 地址获得通用性。主要缺点在于其通信仅限于 TCP 或 UDP。只要所有通信都采用 TCP 或 UDP，NAPT 就允许一台内部计算机访问多台外部计算机，并允许多台内部计算机访问同一台外部计算机，相互之间不会发生冲突。16 比特的端口空间最多允许 2^{16} 对应用程序同时通信。总结如下：

目前存在几种 NAT 变体，其中包括最流行的 NAPT 方式，它除了要转换 IP 地址，还要转换协议口号。

19.10 NAT 和 ICMP 之间的交互

对于 IP 地址，哪怕是一点点的改变，也可能会在高层协议中产生不可预料的副作用。特别是 NAT 必须处理好 ICMP，以维持其透明性效果。例如，假设一台内部主机使用 ping 命令来测试因特网中某个目的站的可达性。主机希望对于发出的每个 ICMP 回送请求 (echo request) 报文都能接收到一个 ICMP 回送回答 (echo reply)。因此，NAT 必须把传入的回送回答报文转发到正确的主机上。但是，NAT 并不转发所有来自因特网的 ICMP 报文。例如，如果 NAT 盒中的路由不正确，它必须就地生成 ICMP 重定向 (redirect) 报文。这样，当一个 ICMP 报文从因特网到达时，NAT 必须首先确定报文是应该就地处理，还是应该发送给某台内部主机。在转发给内部主机之前，NAT 还要转换 ICMP 报文。

为了理解 ICMP 转换的必要性，可以考虑一个 ICMP 目的不可达 (destination unreachable) 报文，在这个报文中包含了造成错误的数据报 D 的首部。遗憾的是，NAT 在发送 D 之前进行了地址转换，所以源地址并不是内部主机使用的地址。因此，在转发报文之前，NAT 必须打开 ICMP 报文并转换 D 中的地址，以使这些地址看起来与内部主机使用的地址完全一致。改变以后，NAT 必须重新计算 D 中的检验和、ICMP 首部中的检验和以及外层数据报首部的检验和。

19.11 NAT 和应用之间的交互

虽然 ICMP 会使 NAT 变得有些复杂，但是应用协议的影响效果则更加明显。一般来讲，NAT 不能用于把 IP 地址或协议端口作为数据来发送的任何应用。例如，当两个程序使用第 25 章描述的 **文件传送协议** (File Transfer Protocol，简称 FTP) 时，在这两个程序之间存在一个 TCP 连接。作为协议的一部分，其中一个程序要获取本机上的一个协议端口，把这个端口号转化成 ASCII 码，然后通过 TCP 连接把结果发送给另一个程序。如果两个程序之间的连接经过 NAPT 从一台内部主机到一台因特网上的主机，那么必须把包含在数据流中的端口号改为和 NAPT 选择的端口号一致，而不用内部主机所用的端口。实际上，如果 NAT 不能打开数据流并改变端口号，则协议将会失败。已经有了一些 NAT 的具体实现方式，能够识别 FTP 之类的流行协议，并在数据流中进行了必要的改动。人们还开发出一些应用的变体，以避免在反方向构造连接^①。但是，还存在一些不能使用 NAT 的应用。总而言之：

^① 不要求服务器反向连接回客户的 FTP 版本称为**被动 FTP** (passive FTP)。

NAT 影响了 ICMP 和更高层的协议。除了类似 FTP 的少数几个标准应用，把 IP 地址或协议端口号作为数据传送的应用协议将无法利用 NAT 正常操作。

改变数据流中的内容在两个方面增加了 NAPT 的复杂度。第一，这意味着 NAPT 必须详细了解传输此类信息的每个应用的情况。第二，如果端口号是以 ASCII 表示的，正如 FTP 这种情况，改变值就会改变传输的八位组的数目。在 TCP 连接中额外多插入一个八位组都很困难，因为流中的每个八位组有一个序号。因为发送方不知道已经插入了额外的数据，它会继续分配序号，没有考虑额外的数据。当接收方收到额外的数据时，会产生对该数据的确认。这样，当插入额外数据时，NAT 必须转换每个外发数据段的序号以及每个传入确认中的序号。

19.12 数据报分片情况下的 NAT

前面对 NAT 的描述都做了一个有关 IP 的重要假设：NAT 系统接收的是完整的 IP 数据报。没有分片处理。那么如果数据报是分片的，又会怎样呢？存在两种情况。对于基本的 NAT，分片不会带来问题，因为每个分片都携带着 IP 源地址和目的地址。而在 NAPT（使用最广的 NAT 的变体）的情况下，表的查找除了要使用来自 IP 首部中的 IP 地址外，还要用到来自运输协议首部中的端口号。遗憾的是，只有数据报的第一个分片才携带运输协议首部。因此，在执行查找动作之前，NAPT 系统必须接收并检查数据报的第一个分片。

正因为如此，NAPT 系统只能遵从以下两种设计之一：系统可以保存各个分片并试图重装数据报，或者可以丢弃分片而只处理完整的数据报。这两种选择都不够理想。重装需要状态信息，也就是说，系统不能跟上高速率或大流量的节奏。丢弃分片意味着系统将无法处理任意的流量。实际上，只有那些为低速网络设计的 NAPT 系统才会选择重装，更多的系统则会拒绝分片的数据报。

19.13 概念性地址域

我们把 NAT 描述为一种可用于将专用网连接到因特网上的技术。实际上，NAT 可用于互连任意两个**地址域**（address domain）。这样，即使两个机构都使用了相同地址 10.0.0.0 的专用网，它们之间还是可以使用 NAT。更重要的是，NAT 可用在两个层次上：不仅可用于 ISP 地址域和全球因特网之间，也可用于客户的专用地址域和 ISP 的专用地址域之间。最后，NAT 可以和 VPN 技术结合使用，形成一个混合体系结构。在这种结构中，专用地址用在机构内，NAT 用于提供各个网点和全球因特网之间的连接。

作为多层 NAT 的示例，考虑一个在家办公的人，他使用着几台连接到局域网的计算机。这个人可以给家里的计算机分配专用地址，并在家网和公司内联网之间使用 NAT。公司也可以分配专用地址，并在自己的内联网和全球因特网之间使用 NAT。

19.14 slirp 和 iptables 程序

曾经有两种网络地址转换的具体实现在不同时期特别流行，它们都是为 UNIX 操作系统设计的。从 4.4 BSD 衍生而来的 slirp 程序，是为图 19.4 所示的拨号结构设计的。slirp 把 PPP 和 NAT 结合在一个程序中。它的宿主计算机必须具有一个有效 IP 地址、一个永久的因特网连接以及一个或多个拨号调制解调器。slirp 的主要优势是能将 UNIX 操作系统上的一个普通用户账号用于一般的因特网访问。具有专用地址的计算机拨入并运行 slirp。一旦 slirp 开始运行，拨号线路从 ASCII 命令方式切换到 PPP 方式，拨号计算机启动 PPP 并获得因特网的访问（例如访问一个 Web 站点）。

slirp 实现了 NAPT，它使用协议端口号来分用连接，并能够重写协议端口号和 IP 地址。可以

有多台计算机（例如一个局域网上的计算机）通过运行 UNIX 系统上的一个 slirp 同时访问因特网。

另一个流行的 NAT 实现是为 Linux 操作系统设计的，名为 iptables，这个软件结合了用于分组重写和防火墙的工具以及内核。因为 iptables 提供了全状态的分组检查，所以可以用一组特殊的 iptables 规则构成 NAT 或 NAPT。

19.15 小结

尽管专用网络确保了保密性，但是成本很高。虚拟专用网（VPN）技术提供了一种低成本的替代方法，允许机构使用因特网互连多个网点，并用加密来保证网点之间的通信量的保密性。类似于传统专用网络，虚拟专用网可以是完全隔离的（给所有主机分配专用地址的情况），也可以是一种混合结构，允许主机与全球因特网上的目的主机进行通信。

有两种技术可提供不同地址域之间的通信：应用网关和网络地址转换（NAT）。应用网关的作用相当于一个代理，它接收来自某个地址域中的主机发出的请求，并将请求发送到另一个地址域中的目的站，然后将结果返回给最初发出请求的主机。对于每一种服务来说，都要有一个相应的独立应用网关。

网络地址转换提供了从某个具有专用地址的主机到全球因特网之间的透明 IP 层访问。NAT 特别受 ISP 的欢迎，因为它允许消费者在使用专用 IP 地址的同时也可以访问任意因特网服务。但是，需要在数据流中传递地址或端口信息的应用无法使用 NAT，除非 NAT 经过程序处理可以识别这个应用，并对数据进行一些必要的修改。而绝大多数的 NAT 实现只能识别很少的几个（标准）服务。

另外还有一些 NAT 的变体。大多数 NAT 的实现都要执行**网络地址和端口转换**(Network Address and Port Translation，简称 NAPT)。除了重写 IP 地址之外，NAPT 还要重写运输层协议口号，这是为了提供完全的通用性，并允许某机构中的一台计算机上运行的任何应用程序都能同时访问任何一个因特网服务。

19.16 深入研究

有许多路由器和软件运营商都在销售虚拟专用网技术，通常可以选择加密方案和编址结构。查阅运营商的文字资料可了解更多信息。

几个版本的 NAT 都可以在市场上购买到，包括一些将 NAT 和安全防火墙合并在一个盒子中的专用硬件设备。Srisuresh and Holdrege [RFC 2663] 以及 Srisuresh and Egevang [RFC 3022] 定义了 NAT 的术语，包括 NAPT。

有关 iptables 工具的更多细节可以在 Linux 文档中找到。从下面这个 URL 可以找到一个资源页面：

<http://www.netfilter.org>

有关 slirp 的更多信息可以在该程序的文档中找到。slirp 的一个资源页面在：

<http://slirp.sourceforge.net>

19.17 习题

1. 在什么环境下，当通过因特网发送相同的数据时，使用 VPN 实际上会比传统 IP 传送更多的分组？提示：考虑封装。
2. 阅读 slirp 文档，找出关于**端口重定向**（port redirection）的内容。为什么需要使用端口重定向？

-
3. 通过两个 NAT 盒连接三个地址域时，会有什么潜在问题？
 4. 在前一个问题中，目的地址会转换几次？源地址会转换几次？
 5. 考虑在两个 NAT 盒互连三个地址域的条件下发送一个 ICMP 主机不可达报文，将发生几次地址转换？将发生几次端口号转换？
 6. 假设我们决定创建一个与现有因特网平行的新的因特网，并且它们用来分配地址的地址空间也一样。NAT 技术是否能够连接这两个使用相同地址空间的任意大的因特网？如果能，应该怎样做。如果不能，试解释原因。
 7. NAT 对于主机是完全透明的吗？为了回答这个问题，需要找出一个分组序号，主机通过传送这个分组序号就可以确定它是否位于一个 NAT 盒之后。
 8. 将 NAT 技术与 VPN 技术合并在一起的优点是什么？缺点呢？
 9. 找到 slirp 程序，并对它进行测试，以衡量其性能。slirp 所产生的额外开销会对数据报造成延迟吗？试解释原因。
 10. 在一个 Linux 系统上配置专用地址域和因特网之间的 NAT。哪个熟知服务能够正常使用，哪个不能？
 11. 阅读了解一种称为**两次 NAT** (twice NAT) 的 NAT 变体，它允许任何时候从 NAT 盒的任何一端发起通信。两次 NAT 怎样确保转换的一致性？如果两次 NAT 的两个实例用于互连三个地址域，结果对于所有主机是完全透明的吗？

第20章 客户—服务器交互模型

20.1 引言

前面的章节阐述了 TCP/IP 的技术细节，包括提供基本服务的协议和提供所需路由信息的路由器体系结构。现在我们已经了解了基本技术，就可以进一步考察那些与 TCP/IP 互联网协作并从中受益的应用程序。虽然具体的应用实例既实用又有趣，但它们不是讨论的重点。重点是相互通信的应用程序之间的交互模式。相互协作的应用程序之间的主要交互模式是**客户—服务器**(client-server)模型^①。客户—服务器交互构成了基本的网络通信，并且为应用服务提供了基础。本章讨论其基本模型，后面几章将描述它在各种应用程序中的使用。

20.2 客户—服务器模型

服务器(server)指的是能在网络上提供服务的一个计算机程序。服务器接受收到的请求，产生响应，并将结果返回给请求方。对于最简单的服务，到达的每个数据报就是一个请求，并用另一个数据报返回响应。

当一个可执行程序向服务器发出请求并等待响应时，它就成为**客户**(client)。由于客户—服务器模型是对同一台机器上进程间相互通信的自然合理的延伸，因此对于程序员而言，很容易编制利用此模型进行交互的程序。

服务器能完成简单的或复杂的任务。例如，一个**日期—时间服务器**(time-of-day server)一旦收到客户发出的分组，立即返回当前的时间。一个**万维网服务器**(web server)接收来自浏览器的读取某个网页的请求，并将所要求的网页返回给浏览器。

通常，服务器是以应用程序来实现的^②。用应用程序实现服务器的优点是可以在任何一个支持 TCP/IP 通信的计算系统上执行。例如，如果某个服务器上的负载不断增加，就可以把这个服务器移到 CPU 速度更快的机器上。有一些技术允许一个服务器程序在多台物理上相互独立的计算机上复制运行，以提高可靠性和改善性能。如果一台计算机的主要任务是支持某个服务器程序，那么“服务器”这一名称不但指服务器程序，也指这台计算机。因此，常出现一些类似“机器 A 是我们的 Web 服务器”的说法。

20.3 一个简单的例子：UDP 回送服务器

最简单的客户—服务器交互形式是，利用数据报的交付机制将报文从客户传送到服务器，然后再返回客户。例如，图 20.1 所示的**UDP 回送服务器**(echo server)中的交互。首先，UDP 回送服务器启动。服务器指明它将使用为 UDP 回送服务器保留的端口，7 号端口。然后，服务器进程进入一个具有三个步骤的无限循环：(1) 等待到达回送端口的数据报；(2) 对调源地址和目的地址^③(包括源和目的 IP 地址和 UDP 端口号)；(3) 将数据报返回原发送方。而在其他某个网点，当一个程序

^① 虽然近来有些媒体上的文章宣称客户—服务器计算体系结构已经被**对等通信**(peer-to-peer，简称 P2P)所取代，但是其底层软件仍然依据这里所描述的客户—服务器模型。

^② 许多操作系统把一个正在运行的应用程序称为**进程**(process)、**用户进程**(user process)或者**任务**(task)。

^③ 有一道习题建议更详细地讨论这一步骤。

分配得到一个未用的 UDP 协议端口，向 UDP 回送服务器发出一个 UDP 报文，并等待响应时，这个程序即成为一个 UDP **回送客户** (echo client)。客户期待收到与发出的数据完全相同的数据。

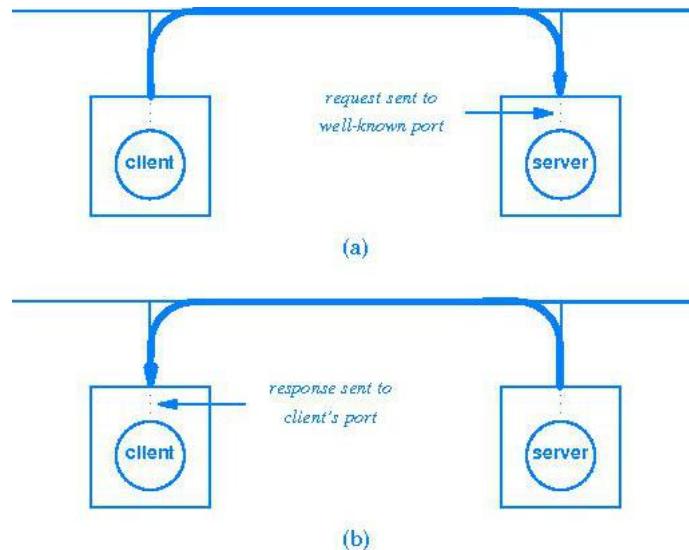


图 20.1 客户一服务器模型举例：UDP 回送

UDP 回送服务体现了通常适用于客户一服务器交互的两个重要特点。第一点是服务器和客户在生存时间上的不同：

服务器在交互开始之前已运行，并且通常持续不停地接收请求并发送响应。客户则是发出请求并等待响应的任何程序，它（通常）在有限次使用服务器后终止。

第二点有关技术细节，涉及保留端口和未保留端口标识的使用：

服务器在为回送服务保留的一个熟知端口上等待请求。客户则为其通信任意分配一个非保留的未使用端口。

在客户一服务器交互的过程中，两个端口只有一个需要保留。为每个服务分配一个唯一的端口标识符，可以使构造客户和服务器都变得很简单。

谁会使用回送服务？这并不是普通用户会感兴趣的服务。但是，设计、实现、测试或修改网络协议软件的编程人员，或者测验路由和调试通信问题的网络管理人员，都会经常将回送服务器用于测试。例如，回送服务可用于检验能否连通远端机器。

20.4 时间和日期服务

回送服务器实在太简单了，所以服务器或客户端的实现所需的代码都不多（前提是操作系统提供了访问底层 UDP/IP 协议的有效途径）。我们的第二个例子是时间服务器，该例子表明，即使是简单的客户一服务器交互，也能提供有用的服务。时间服务器解决的是自动设置计算机的日历时钟的问题。一旦设置好时间，该时钟就能像手表一样保持精确的时间。

大多数笔记本电脑使用电池来维持时钟运行。但是，与时间服务器的交互可用于机器自动将系统时钟设置为正确的时间，而不需要用电池来维护时间。如果每个网络上至少有一台机器运行着时间服务器，那么笔记本电脑只要发送一个请求就能获得当前时间。

20.4.1 日期和时间的表示

应当如何表示时间呢？一种表示方法是将时间和日期保存为某个起始日期后的秒数。TCP/IP

协议定义的起始日期为 1900 年 1 月 1 日，并用 32 位整数来保存时间。这种表示方法可以容纳可见未来的所有日期。

20.4.2 时间服务器的交互

客户和提供时间服务的服务器之间的交互方式与回送服务类似。首先，启动服务器应用程序，然后等待客户与之联络。服务器假设每个到达的数据报都是一个对当前时间的请求，并发送一个 UDP 报文做出响应，该报文包含用 32 位整数表示的当前时间。归纳如下：

向时间服务器发送一个数据报相当于发出得到当前时间的请求，服务器返回一个包含当前时间的 UDP 报文作为响应。

20.5 服务器的复杂性

前面提到的两个例子展现的是基本的顺序服务器（即服务器一次只处理一个请求）。顺序服务器在收到请求后，先发送回答，然后再查看是否有下一个请求到达。在服务器很忙的时候，操作系统的协议软件必须将到达的请求进行排队。

在实践中，服务器通常比客户更难构造，因为作为产品的服务器是并发的（concurrent），也就是说，服务器要同步处理多个请求。为了更好地理解并发的重要性，可以想象一个正在通过拨号连接向用户发送大图片的万维网服务器。它一边使缓慢的传送继续不断，一边响应其他请求。要做到这一点，服务器采用图 20.2 所示的工作步骤。

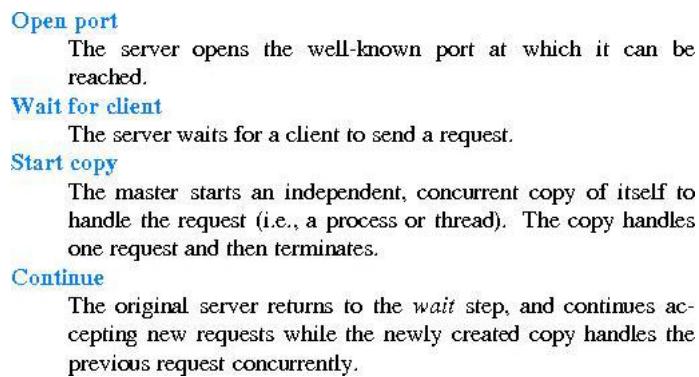


图 20.2 并发服务器采用的步骤，以允许服务器同时处理多个请求

并发服务器的主要优点在于速度：后面到达的请求不需要等待以前启动的请求完成；而主要的缺点是复杂：并发服务器的构造很复杂。

除了服务器由于处理并发请求而造成的复杂性之外，服务器还必须强制实施授权和保护规则，这也导致了复杂性。服务器程序通常需要以最高优先级执行，因为它们必须读系统文件、维护日志文件并访问受保护的数据。操作系统通常不会限制服务器访问用户文件。因此，服务器不能盲目地认可来自其他网点的请求。每个服务器要负责实施系统访问和保护策略。

最后，服务器必须防止不正常的请求和引起服务器程序异常终止的请求。服务器经常难以预见潜在的问题。例如，普度大学的一个项目设计了一个文件服务器，允许学生的操作系统访问 UNIX 分时系统上的文件。学生发现请求服务器打开名为/dev/tty 的文件时会引起服务器的异常终止，因为在 UNIX 系统中这个名字指的是程序正在使用的控制终端，而在系统启动时创建的服务器并没有这个终端。

我们对服务器的讨论可以归纳如下：

服务器通常比客户更难构造，因为即使服务器可用应用程序实现，也必须在运行它的计算机系统上实施所有访问和保护措施，而且必须自我保护，预防发生各种可能的错误。

20.6 广播一个请求

迄今为止，我们所有的客户—服务器交互的例子都需要客户事先知道服务器的完整地址。但是在有些情况下，客户并不知道服务器的地址。例如，机器在启动时可以使用 DHCP 获得一个 IP 地址，但是客户并不知道服务器的地址^①。事实上，客户会广播其请求。

重点是：

对于客户不知道服务器所在位置的协议来说，客户—服务器模型允许客户程序广播请求。

20.7 客户—服务器模型的替代方案

是否存在客户—服务器交互的替代方案呢？有一种特殊情况非常重要。前面讲过，当一个应用程序需要信息时就变成了一个客户，但有时尽量减少这种交互也很重要。第 5 章的 ARP 协议使用的就是客户—服务器交互模型的一种变形：高速缓存。运行 ARP 的计算机必须维护一个地址绑定的高速缓存，以提高后面查询的效率。当近期查询历史对今后的查询具有很好的指导作用时，高速缓存提高了客户—服务器交互的性能。

虽然高速缓存提高了性能，但它并没有改变客户—服务器交互的本质。这个本质就是我们假设进程必须由需求驱动。我们假定程序只有在需要信息时才会执行，并作为一个客户获取所需的信息。采用需求驱动的观点是很自然的，它来源于经验。高速缓存降低了除第一个请求之外的所有请求的查询开销，有助于降低获取信息的全部费用。

怎样才能降低第一个请求的信息查询开销呢？在一个分布式系统中，可能有并发的后台任务在收到任何请求之前就在不停地收集和传播信息，这样甚至可以使第一个请求的查询费用降低。更重要的是，预收集信息允许某个系统持续运行，即使其他机器或连接它们的网络出现故障。

一个称为 *ruptime* 的早期 UNIX 程序就体现了这种思想。一旦 *ruptime* 被唤醒，它就会报告本地网络上的每台计算机自从系统启动后的 CPU 的负载及时间。因为每台计算机上都运行着一个后台程序，它使用 UDP 周期性地广播有关机器的信息，并收集从其他计算机传来的信息，所以这个报告是即时发生的。这样，这些信息随时都可以无延迟地从本地获取（即不需要网络通信）。

预收集有一个主要的缺点：即使没有人关心收集来的数据，机器仍要占用处理器时间和网络带宽。例如，*ruptime* 广播和信息收集在整个晚上都要持续进行，即使那时根本没人登录并读取信息。如果只有几台机器连接在某个网络上，预收集的开销还无关紧要。但是，对于包含多台机器的网络，预收集所产生的巨大广播通信量使之代价太高。因此预收集不是客户—服务器交互的最受欢迎的替代方案。

20.8 小结

分布式程序需要网络通信。这些程序通常采用客户—服务器交互模式。一个服务器进程等候请求，并完成基于请求的动作。这些动作通常包括发送响应。客户程序形成请求，将其发送给服务器，然后等待回答。

^① 第 22 章将讨论 DHCP。

我们已经看到了一些客户和服务器的例子，并发现一些客户直接发送请求，而另一些客户则广播请求。当客户不知道本地网络上的服务器地址时，广播特别有用。

我们也注意到，如果服务器使用诸如 UDP 的互联网协议，就可以接收并响应来自互联网的请求。如果使用物理帧和物理硬件地址通信，这种通信将局限于一个物理网络。

最后，我们考虑的是客户—服务器模型的替代方案，它使用预收集信息来避免延迟。预收集的一个例子来源于一个机器状态服务。

20.9 深入研究

UDP 回送服务在 Postel [RFC 862] 中定义，UNIX 程序员手册描述了 `ruptime` 命令（也可见 `rwho` 命令的相关描述）。有各种 RFC 定义了许多本章未讨论的服务器协议，包括丢弃、字符产生、日期和时间、活跃的用户以及日期的引用。在以后的章节中会讨论其他一些协议。

20.10 习题

1. 构造一个 UDP 回送客户，它把数据报发送到一个指定的回送服务器，然后等待回答，并将回答与所发出的信息进行比较。
2. 仔细考虑 UDP 回送服务器中对 IP 地址的处理。在什么情况下通过颠倒源和目的 IP 地址来创建新 IP 地址是不正确的？
3. 正如我们所见，服务器可用独立的应用程序实现，也可以通过在操作系统的协议软件中构造服务器代码来实现。那么，每个服务器都有一个应用程序（或用户进程）的优点和缺点各是什么？
4. 假设你并不知道运行 UDP 回送服务器的本地机器的 IP 地址，但却知道它响应从端口 7 收到的请求。能否找到一个与之联系的 IP 地址？
5. 构造一个用于 UDP 时间服务的客户。
6. 在什么情况下服务器可以和它的客户不在同一个物理网络上，试归纳其特点并解释原因。
7. 所有机器都周期性地广播其状态信息的主要缺点是什么？
8. 查看实现 4BSD UNIX 的 `ruptime` 命令的服务器所广播的数据格式。客户除了能获取机器的状态信息以外，还能得到其他什么信息？
9. 你所在的网点上的计算机运行了哪些服务器？如果你不能访问给定计算机启动时的服务器列表的系统配置文件，看看系统有没有一个命令能打印出已打开的 TCP 和 UDP 端口的列表（如 UNIX 的 `netstat` 命令）。
10. 一些服务器允许管理员从容（gracefully）关闭它们或重启动。服务器从容关闭的优点是什么？

第21章 套接字接口

21.1 引言

至此，我们着重讨论了 TCP/IP 协议底层的原理和概念，但还没有描述协议软件与应用程序之间的接口。本章将介绍一个已成为事实上的标准的**应用程序接口**（Application Program Interface，简称 API），并描述一些用于访问 TCP/IP 协议的范例应用程序。

推迟讨论 API 有两个原因。首先，原则上我们必须区分接口和协议，因为协议标准不会定义一个明确的接口。其次，在实际应用中，接口的确切细节依赖于具体的操作系统和编程语言。

我们将讨论的是**套接字**（socket）API。套接字最初是作为 BSD UNIX 操作系统的一部分出现的，其后的 Linux 以及新的 BSD 系统中都有套接字。微软在其系统中采纳套接字后称之为 Windows Sockets^①。读者还应记住，这里列举的操作并非 TCP/IP 标准的一部分。

21.2 UNIX I/O 范式与网络 I/O

UNIX 发展于 20 世纪 60 年代末和 70 年代初，它原本是为单处理器计算机上的分时系统设计的。它是一个面向进程的操作系统，其中的应用程序作为用户级进程运行。应用程序通过**系统调用**（system call）与操作系统进行交互。从编程人员的角度来看，系统调用看上去、用起来都与其他过程调用类似。它们带有参数并返回一个或多个结果。参数可以是值（如一个整数值）或应用程序中指向对象的指针（如用来填入字符的缓冲区）。

UNIX 输入和输出（I/O）原语来源于 Multics 和早期系统，遵从有时称为**打开一读一写一关闭**（open-read-write-close）的范式。用户进程在进行 I/O 操作之前，首先调用 open，指明使用的文件或设备，并获取使用许可。open 调用返回一个短整型的文件描述符^②，进程将使用它完成对已打开文件或设备的 I/O 操作。一旦打开了一个对象，用户进程就可以激活一个或多个 read 或 write 操作，进行数据传送。read 调用将数据传给用户进程；write 调用将数据从用户进程传给文件或设备。read 和 write 调用都有三个参数，分别指明使用的文件描述符、缓冲区的地址和传输字节数。在所有的传送操作完成以后，用户进程调用 close 通知操作系统，它已完成对对象的使用（如果进程终止前未调用 close，操作系统会自动关闭所有打开的描述符）。

21.3 在 UNIX 中添加网络 I/O

在 BSD UNIX 中添加网络协议的设计小组认为，由于网络协议比传统 I/O 设备更复杂，用户进程和网络协议之间的交互肯定比用户进程与传统 I/O 设施之间的交互更复杂。具体说来，协议接口必须允许程序员既能创建被动等待连接的服务器代码，又能创建主动请求连接的客户代码。此外，应用程序发送数据报时，希望给每个数据报指定目的地址，而不是在“打开”时就绑定目的地址。为了处理所有这些情况，设计人员决定不完全照搬传统的 UNIX “打开一读一写一关闭” 范式，并添加了几个新的操作系统调用和库例程。给 UNIX 添加网络协议从根本上增加了 I/O 接口的复杂性。

由于设计人员为适应多种协议而尝试构造一种通用机制，这就进一步增加了 UNIX 协议接口的

^① 通常程序员会用术语 WINSOCK 代替 Windows Sockets。

^② 术语“文件描述符”的出现与 UNIX 系统中设备被映射到文件系统中有关。

复杂性。例如，通用性使操作系统除包含 TCP/IP 协议以外还可以包含其他协议族的软件，并允许一个应用程序同时使用一种或多种协议。结果是，应用程序不能只靠提供一个 32 位地址就希望操作系统能够正确地解释它。应用程序必须明确指出这 32 位数字代表一个 IP 地址。

21.4 套接字的抽象化

在套接字 API 中，网络 I/O 的基础在于一种称为**套接字**（socket）的抽象^①。我们认为套接字是为通信提供端点的机制。应用程序在需要时请求操作系统创建一个套接字，而系统则返回一个短整数，由应用程序用来引用新创建的套接字。如果它对应的是 TCP 连接，那么这条连接的两个端点都必须指明。如果是 UDP 使用的套接字，那么对方端点可以不用指明，此时对方端点必须在每次发送报文时以参数的形式传递。

21.5 创建一个套接字

socket 函数根据需要创建套接字。它带有三个整数参数并返回一个整数结果：

```
result = socket(pf, type, protocol)
```

参数 pf 指明套接字使用的协议族（即指明如何解释地址）。当前的协议族包括 IPv4（PF_INET），IPv6（PF_INET6），Xerox 公司的 PUP 互联网（PF_PUP），Apple 计算机公司的 AppleTalk 网络（PF_APPLETALK）和 UNIX 文件系统（PF_UNIX）^②。

参数 type 指明了要求的通信类型。可能的类型包括可靠数据流交付服务（SOCK_STREAM）和无连接数据报交付服务（SOCK_DGRAM），以及允许有特权的程序访问底层协议或网络接口的原始类型（SOCK_RAW）。

因为一个协议族可能有多个协议提供相同类型的通信，所以套接字就有了第三个参数，用于选择一个具体的协议。如果在协议族中只存在一种能提供某种 type 通信的协议（例如 IPv4 中只有 TCP 支持 SOCK_STREAM 服务），这时可把第三个参数设置为 0。

21.6 套接字的继承与终止

操作系统要提供用于创建新进程或线程的机制。例如，UNIX 系统使用 fork 和 exec 系统调用来自创建一个进程，以运行某个指定的应用程序。在大多数系统中，当创建新进程时，该进程会继承对所有打开套接字的访问。我们将看到，服务器会使用套接字继承来允许该服务器的多个副本并发处理多个请求。操作系统在内部保存了对每个套接字的引用计数，因此它知道有多少个应用程序（进程）访问了套接字。

无论是老进程还是新进程，对现有的描述符都有同样的访问权，并且都能访问套接字。因此，应当由程序员负责确保两个进程能有效地共享使用套接字。

当进程完成了对套接字的使用时就调用 close。close 具有如下形式：

```
close(socket)
```

此处，socket 参数指明了要关闭的套接字描述符。无论是什么原因，只要进程终止，系统就会关闭所有仍然打开的套接字。在系统内部，close 调用将递减套接字的引用计数，并在计数为零时删除套接字。

^① 如今已经有充分的理由将套接字看成是操作系统核心的一部分，但是也可以由库例程提供套接字 API。

^② 为了使程序更可读，程序员用类似 PF_INET 的符号常量来代替数字值。套接字 API 为每个常量定义了一个数值。

21.7 指明本地地址

一个套接字在初始创建时并未与任何本地或目的站地址关联。对于 TCP/IP 协议，这意味着新的套接字并不是从本地或远端的 IP 地址或协议端口号开始的。应用程序并不关心它们使用的本地地址，而是很愿意让协议软件来填写计算机的 IP 地址，并选择一个端口号。但是，在某个熟知端口上操作的服务器进程必须为系统指明端口。一旦创建了套接字，服务器就使用 bind 功能为套接字建立一个本地地址^①。bind 具有如下形式：

```
bind(socket, localaddr, addrlen)
```

参数 socket 是要绑定的套接字描述符。参数 localaddr 是一个指定套接字要绑定的本地地址的结构，参数 addrlen 是一个指定地址长度（字节数）的整数。设计人员选用图 21.1 所示的结构表示地址，而不是将地址简单表示成一个字节序列。

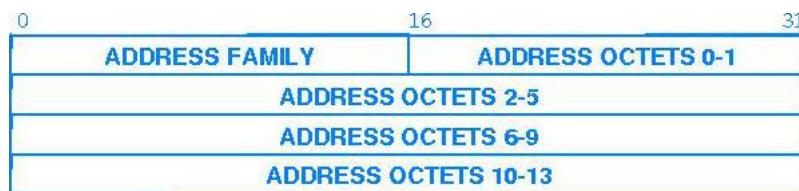


图 21.1 给套接字接口传递 TCP/IP 地址时使用的 sockaddr 结构

这个结构通常称为 sockaddr，以一个识别地址所属协议族的 16 位地址族 (ADDRESS FAMILY) 字段开始，随后是不超过 14 八位组的地址。注意，当某个特殊的地址结构作为参数传递到套接字函数时，这个结构必须伪装成通用结构 sockaddr。

地址族字段中的值决定其余地址八位组的格式。例如，地址族字段中值为 2 则意味着其余八位组中包含的是一个 TCP/IP 地址。每个协议族都规定了自己如何使用地址字段中的八位组。对于 TCP/IP 地址，套接字地址称为 sockaddr_in。它包括一个 IP 地址和一个协议端口号（即，一个互联网套接字地址结构可以包含一个 IP 地址以及该地址的一个协议端口）。图 21.2 所示为一个 TCP/IP 套接字地址的明确结构。



图 21.2 使用 TCP/IP 地址时一个套接字地址结构 (sockaddr_in) 的格式。此结构包括一个 IP 地址以及此地址上的端口号

虽然可以在调用 bind 时在地址结构中指定任意值，但不是所有可能的绑定都有效。例如，调用方可能申请了一个已被另一程序使用的本地协议端口，或者可能申请了一个无效 IP 地址。在这样的情况下，bind 调用失败并返回一个错误码。

21.8 将套接字连接到目的地址

初始创建的套接字处于一种**未连接的状态** (unconnected state)，也就是说，套接字并未与任何远端目的地址关联。函数 connect 为套接字绑定一个永久目的地址，并将它置于**连接状态** (connected

^① 如果一个客户没有调用 bind，操作系统则会自动分配一个端口号；一般情况下，端口号是顺序分配的。

state)。应用程序在通过可靠数据流套接字传输数据之前，必须调用 connect 建立一个连接。用于无连接数据报服务的套接字不需要在使用前建立连接，但如果建立了连接，传输数据时不必每次都指定目的地址。

connect 函数具有如下形式：

```
connect(socket, destaddr, addrlen)
```

参数 socket 是要连接的套接字的整数描述符。参数 destaddr 是一个套接字地址结构，它指明要与套接字绑定的目的地址。参数 addrlen 指明按字节计算的目的地址长度。

connect 的语法与底层协议有关。选择 PF_INET 协议族中的可靠数据流交付服务，就意味着选择了 TCP。在此情况下，connect 建立起与目的站的 TCP 连接，如果失败就返回一个错误码。在无连接服务的情况下，connect 只是在本地存储目的地址，没有更多的作用。

21.9 通过套接字发送数据

一旦应用程序建立起套接字，就可以使用套接字传输数据。一共有五个系统调用可供选择：send, sendto, sendmsg, write 和 writev。send 和 write 以及 writev 只用于已建立连接的套接字，因为它们不允许调用方指定目的地址。这三个调用之间的区别很小。write 带有三个参数：

```
write(socket, buffer, length)
```

参数 socket 包含一个整数套接字描述符（write 也可以与其他类型的描述符一起使用）。参数 buffer 包含要发送的数据的地址。参数 length 指明要发送的数据的长度。write 调用在数据传出去之前是阻塞的（即，如果套接字使用的内部系统缓冲区已满，则发生阻塞）。与大多数系统调用类似，write 给调用它的应用程序返回一个错误码，以便让程序员知道操作是否成功。

writev 系统调用与 write 类似，只不过 writev 使用“收集写”（gather write）的形式，使应用程序在写报文时可以不将报文复制到内存的连续字节区域。writev 具有如下格式：

```
writev(socket, iovector, vectorlen)
```

参数 iovector 给出一个 iovec 类型的数组地址，它含有一个指向构成报文的各字节块的指针序列。如图 21.3 所示，每个指针都伴有一个块长度。参数 vectorlen 指明 iovector 中的指针项的数目。

send 函数具有如下形式：

```
send(socket, message, length, flags)
```

此处的 socket 参数指明所用套接字的描述符，参数 message 给出要发送数据的地址，参数 length 说明要发送的数据的字节数，参数 flags 控制传输方式。有一个 flags 值允许发送方指明报文必须在支持带外传输的套接字上带外发送。例如，回忆第 12 章中曾讲过，带外报文在 TCP 表示法中对应的是紧急数据。flags 的另一个值允许调用方请求不使用本地路由选择表发送报文。其意图是允许调用方来控制路由选择，这样就可以编写网络调试软件。当然，并不是所有的套接字都支持来自任意程序的所有请求。一些请求需要调用程序必须具有特殊权限，而另一些请求则在所有套接字上都未得到支持。



图 21.3 用于 writev 和 readv 的 iovector 格式

函数 `sendto` 和 `sendmsg` 允许调用方在无连接的套接字上发送报文，因为它们都需要调用方指定目的地址。`sendto` 将目的地址作为参数，它具有如下格式：

```
sendto(socket, message, length, flags, destaddr, addrlen)
```

前 4 个参数与 `send` 函数使用的完全相同。最后两个参数指定一个目的地址，并给出地址长度。参数 `destaddr` 使用图 21.2 中定义的 `sockaddr_in` 结构来指定目的地址。

程序员通常喜欢使用 `sendmsg` 函数，以免 `sendto` 所需的一长串参数使程序效率不高，而且难以读懂。`sendmsg` 具有如下格式：

```
sendmsg(socket, messagestruct, flags)
```

此处的参数 `messagestruct` 是如图 21.4 所示的结构。该结构包含要发送报文的信息、长度、目的地址和地址长度。这一调用尤其有用，因为有一个相应的输入操作（下面将描述）将产生格式完全相同的报文结构。



图 21.4 `sendmsg` 使用的 `messagestruct` 报文结构的格式

21.10 通过套接字接收数据

与 5 种不同的输出操作类似，套接字 API 提供了 5 个相应的输入函数 `read`, `readv`, `recv`, `recvfrom` 和 `recvmsg`，进程可以使用这些函数通过套接字接收数据。传统的输入操作 `read` 只能在已连接的套接字上使用。它具有如下形式：

```
read(descriptor, buffer, length)
```

此处，参数 `descriptor` 给出了从中读取数据的套接字的整数描述符或文件描述符，`buffer` 指定了存储数据的存储区地址，参数 `length` 指定了要读取的最大字节数。

`read` 的一种替代形式是 `readv`，它允许调用方使用一种“分散读取”(scatter read) 类型的接口，这种接口会把要读的数据放在不相邻的位置。`readv` 的格式如下：

```
readv(descriptor, iovector, vectorlen)
```

参数 `iovect` 给出了 `iovec` 类型结构的地址（参见图 21.3），它包含一组指向存储待读数据的内存块的指针。参数 `vectorlen` 指定 `iovect` 中的数据项的数目。

除了传统的输入操作，还有三种函数用于网络报文的输入。进程调用 `recv` 从已建立连接的套接字接收数据，其格式如下：

```
recv(socket, buffer, length, flags)
```

参数 `socket` 指明从中接收数据的套接字的描述符。参数 `buffer` 指定存放报文的内存缓冲区地址。参数 `length` 指明缓冲区的长度。最后，参数 `flags` 允许调用方控制接收方式。它有几个可能的值，其中之一允许调用方通过提取下一个传入报文的副本来自行预览数据，但不从套接字中删除此报文。

函数 `recvfrom` 允许调用方指明从一个无连接的套接字上输入数据。它包括更多的参数，以允许调用方指明在何处记录发送方地址。格式如下：

```
recvfrom(socket, buffer, length, flags, fromaddr, addrlen)
```

两个额外参数 `fromaddr` 和 `addrlen` 分别是一个套接字地址结构的指针和一个整数。操作系统使用

fromaddr 来记录报文发送方的地址，并使用 addrlen 记录发送方地址的长度。注意，上面讨论过的输出操作 sendto，其地址格式与 recvfrom 完全一样，因此很容易发送回答。

用于输入的最后一个系统调用是 recvmsg，与 sendmsg 输出操作类似。recvmsg 的操作方式类似 recvfrom，但需要的参数更少。它的格式为：

```
recvmsg(socket, messagestruct, flags)
```

此处，参数 messagestruct 给出一个结构的地址，该结构存放传入报文的地址以及发送方地址的位置。recvmsg 生成的结构与 sendmsg 使用的结构完全一样，这样就使它们能够很好地成对操作。

21.11 获取本地的和远程的套接字地址

刚才讲过，新创建的进程从创建它们的进程处继承打开的套接字。有时，新创建的进程需要判断套接字连到了哪个目的地址。进程也可能希望判断套接字的本地地址。有两个函数可以提供这些信息：getpeername 和 getsockname。尽管名字如此，其实这两个函数处理的都是我们认为的“地址”。

函数 getpeername 判断套接字连接的对端（也就是远端）地址，格式如下：

```
getpeername(socket, destaddr, addrlen)
```

参数 socket 指明需要判定对端地址的套接字。参数 destaddr 是一个指针，指向接收套接字地址的 sockaddr 类型的结构（见图 21.1）。最后，参数 addrlen 是指向接收套接字地址长度的一个整数指针。getpeername 只用于已连接的套接字。

函数 getsockname 返回与套接字相关联的本地地址，格式如下：

```
getsockname(socket, localaddr, addrlen)
```

与想象中的一样，参数 socket 指明需要获取本地地址的套接字。参数 localaddr 是用于保存套接字地址的 sockaddr 类型的结构指针。最后，参数 addrlen 也是一个整数指针，指向套接字地址长度。

21.12 获取和设置套接字选项

除了将套接字绑定到本地地址或连接到目的地址之外，还需要有一种允许应用程序控制套接字的机制。例如，当使用有超时和重传机制的协议时，应用程序可能要读取或设置超时参数，也可能想控制缓冲区空间的分配，判断是否允许发送广播，或控制带外数据的处理。设计人员决定，与其给每个新控制操作增加一个新函数，还不如构造一个控制机制。该机制有两个操作：getsockopt 和 setsockopt。

函数 getsockopt 允许应用程序请求获取关于套接字的信息。调用方指明套接字、感兴趣的选项和存放请求信息的位置。操作系统检查套接字使用的内部数据结构，并将被请求的信息传给调用方。调用格式如下：

```
getsockopt(socket, level, optionid, optionval, length)
```

参数 socket 指明需要信息的套接字。参数 level 标识出操作是用于套接字本身还是其底层协议。参数 optionid 指明申请的是哪一个选项。optionval 和 length 这对参数定义两个指针。第一个指针给出系统存放请求值的缓冲区地址，第二个指针给出系统存放选项值长度的一个整数地址。

利用函数 setsockopt，应用程序可以使用通过 getsockopt 获取的一组值来设置套接字选项。调用方指明一个要设置选项的套接字、要改变的选项和选项的值。setsockopt 的调用格式如下：

```
setsockopt(socket, level, optionid, optionval, length)
```

此处的参数与 getsockopt 的非常类似，只是参数 length 包含的是传给系统的选项长度。调用方必须提供选项的一个合理值和该值的正确长度。当然不是所有选项都可用于所有套接字。各个请求的正确性和语义与套接字的当前状态和使用的底层协议有关。

21.13 指明服务器的队列长度

在套接字的各个选项中，有一个选项的使用率很高，以至于专门为它构造了一个独立的函数。为理解其原因，我们来考虑一个服务器。服务器创建一个套接字，将它绑定到一个熟知协议端口，并等候请求。如果一个服务器使用可靠数据流交付，或者如果计算一个响应所花费的时间不算少，在服务器完成对前一请求的响应之前可能又到达了一个新请求。为了避免协议拒绝或丢弃传入的请求，服务器必须告知底层协议软件，它希望把来不及处理的请求排队。

函数 `listen` 允许服务器为传入的请求准备一个套接字。根据不同的底层协议，`listen` 将套接字置于被动模式，准备接受连接。当服务器唤醒 `listen` 时，它也通知操作系统协议软件应将同时到达套接字的请求排队。格式是：

```
listen(socket, qlength)
```

参数 `socket` 给出服务器准备使用的套接字的描述符，参数 `qlength` 指定套接字上的请求队列的长度。调用 `listen` 以后，系统至多将 `qlength` 个连接请求排队。如果当一个请求到达时队列已满，操作系统将通过丢弃此请求拒绝连接。`listen` 只用于已选择可靠数据流交付服务的套接字。

21.14 服务器如何接受连接

正如我们所见，服务器进程使用函数 `socket` 和 `bind` 以及 `listen` 创建一个套接字，将它绑定到熟知端口，并指明连接请求的队列长度。注意，当调用 `bind` 将套接字与一个熟知协议端口绑定时，套接字并没有连接到外部目的地址。实际上，外部目的地址必须指定一个通配符，从而允许套接字可接收来自任意客户的连接请求。

一旦建立了套接字，服务器需要等待连接。它用函数 `accept` 来做此工作。调用 `accept` 将发生阻塞，直到到达一个连接请求。它的格式如下：

```
newsock = accept(socket, addr, addrlen)
```

参数 `socket` 指明等候连接的套接字描述符。参数 `addr` 是指向 `sockaddr` 结构类型的指针，参数 `addrlen` 是指向一个整数的指针。当一个请求到达时，系统将发出请求的客户地址填入参数 `addr`，将 `addrlen` 设为地址的长度。最后，系统创建一个新套接字，其目的地址已连接到发出请求的客户，并给调用方返回一个新的套接字描述符。原来那个套接字的目的地址仍为通配符，并保持打开状态，因此主服务器可继续在原套接字上接受其他连接。

在一个连接请求到达时，`accept` 调用将返回。服务器可交互地或并发地处理请求。在交互方式中，服务器自己处理请求，关闭新套接字，然后调用 `accept` 获取新连接请求。在并发方式中，`accept` 调用返回后，主服务器创建一个从服务器来处理请求（用 UNIX 术语表达就是创建一个子进程来处理请求）。这个从进程继承了新套接字的副本，因而它可以继续服务于请求。当从进程完成任务后，关闭套接字并终止服务。原（主）服务器进程在启动从进程后关闭新套接字的副本，然后调用 `accept` 获取下一个连接请求。

服务器的并发设计似乎令人困惑，因为多个进程使用了同一个本地协议端口。理解这一机制的关键在于底层协议处理协议端口的方式。前面讲过，在 TCP 中，一对端点之间定义一个连接，因此有多少个进程在使用同一个本地协议端口是无关紧要的，只要它们连接到了不同的目的地。在并发服务器的情况下，每个客户对应一个进程，另外还有一个额外进程用来接受连接。主服务器进程使用的套接字将通配符作为外部目的地址，这样就允许它连接到任意外部网点。每个正在工作的进程都有一个外部目的地址。一旦 TCP 报文段到达，就被送到与报文来源连接的套接字。如果不存在此套接字，将报文段送到用通配符作为外部目的地址的套接字。然而，由于通配符作为外部目的地址的套接字没有一个打开的连接，它只能处理请求新连接的 TCP 报文段。

21.15 处理多重服务的服务器

套接字 API 提供另一种可能令人感兴趣的服务器设计,因为它允许单个进程在多个套接字上等待连接。使这一设计成为可能的系统调用是 `select`, 它通常用于 I/O, 而不只是用于套接字上的通信^①。`select` 具有如下格式:

```
nready = select(ndesc, indesc, outdesc, excdesc, timeout)
```

一般情况下, `select` 调用会发生阻塞, 等待一组文件描述符中的某一个做好准备。参数 `ndesc` 指明应该检查多少个描述符(被检查的描述符总是在 0 到 `ndesc-1` 之间)。参数 `indesc` 是指向一个屏蔽位的指针, 该屏蔽位指明要进行输入检查的文件描述符。参数 `outdesc` 是指向要进行输出检查的文件描述符的屏蔽位的指针。参数 `excdesc` 是指向要进行异常条件检查的文件描述符的屏蔽位的指针。最后, 如果参数 `timeout` 非零, 它就是一个指定在返回调用前等待连接的最长时间值的整数指针。若 `timeout` 为零, 就使得调用阻塞, 直到有一个描述符准备好。由于 `timeout` 参数包含超时值的指针, 而本身并不是整数, 所以进程可通过传递值为零的整数地址申请零时延(即一个进程可轮询查看 I/O 是否准备好)。

`select` 调用返回指定组中 I/O 已准备好的描述符数目。它也可以通过由 `indesc` 和 `outdesc` 以及 `excdesc` 指定的屏蔽位, 通知应用程序所选择的哪一个文件描述符已准备好。因此, 在调用 `select` 之前, 调用方必须打开与要检查的描述符对应的那些屏蔽位。在调用过程中, 其余所有设置为 1 的位对应于 I/O 准备好的文件描述符。

为了在多个套接字上进行通信, 进程首先创建它所需的所有套接字, 然后使用 `select` 判断哪一个套接字最先准备好进行 I/O 操作。一旦发现有一个套接字已准备好, 进程就使用上面定义的输入或输出过程进行通信。

21.16 获取与设置主机名字

大多数操作系统维护着一个内部主机名。对于因特网上的机器, 内部主机名经常被选为该机器主要网络接口的域名。函数 `gethostname` 允许用户进程访问主机名, 函数 `sethostname` 允许有特权的进程设置主机名。`gethostname` 具有如下格式:

```
gethostname(name, length)
```

参数 `name` 给出存放名字的字节数组的地址, 参数 `length` 是一个指定名字数组长度的整数。为了设置主机名, 有特权的进程发出的调用格式是:

```
sethostname(name, length)
```

参数 `name` 给出存放名字的字节数组的地址, 参数 `length` 是一个给出名字数组长度的整数。

21.17 获取与设置内部主域

操作系统维护了一个指定机器所属命名域的字符串。当一个网点获得部分域名空间的权限时, 它生成一个识别所在空间部分的字符串, 并使用这个字符串作为域名。例如, 在域

```
cs.purdue.edu
```

中的机器的名字取自**亚瑟王传奇**(Arthurian legend)。因此, 你可以找到名为 `merlin.arthur.guenevere` 和 `lancelot` 的机器。域本身命名为 `camelot`, 因此必须通知此组中每台主机上的操作系统, 让它知道自己位于 `camelot` 域中。为了完成这个功能, 有特权的进程使用函数 `setdomainname`, 其格式如下:

```
setdomainname(name, length)
```

^① Windows Sockets 中的 `select` 版本仅用于套接字描述符。

参数 name 给出包含域名的字节数组的地址，参数 length 是给出名字长度的一个整数。

用户进程调用 getdomainname 向系统查询域名。它的格式如下：

```
getdomainname(name, length)
```

其中参数 name 给出存放域名的字节数组的地址，参数 length 是给出数组长度的一个整数。

21.18 套接字库的调用

除了上面描述的函数之外，套接字 API 还提供了一组库例程，以执行一些与网络有关的有用的函数。图 21.5 显示了系统调用和库例程之间的区别。系统调用将控制传给计算机操作系统，而库例程与其他过程类似，程序员将它们绑定到一个程序中。

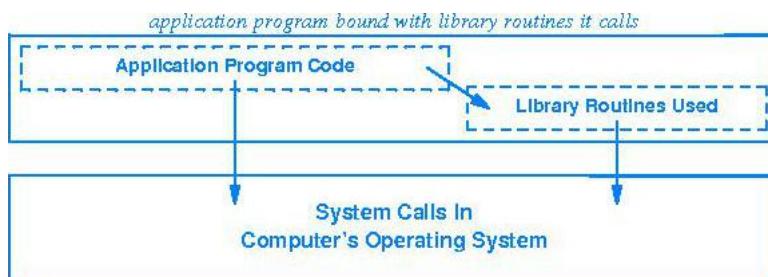


图 21.5 库例程被绑定到应用程序中，而系统调用是操作系统的一部分，这是两者之间的区别。程序可以任意调用这两者，库例程还可以调用其他库例程或系统调用

许多库例程提供数据库服务，从而允许一个进程判断机器名和网络服务、协议端口号以及其他相关信息。例如，一组库例程提供对网络服务的数据库访问。我们将服务数据库中的每一项看成一个三元组，每个三元组包括（人们可读的）网络服务、支持服务的协议和服务所用的协议端口。有各种各样的库例程，能够让进程获取任何一项信息。

以下几节将考察几组库例程，解释其使用目的并提供有关如何使用这些例程的信息。正如我们所见，各组库例程按一种模式提供对顺序数据库的访问。每组库例程允许应用程序：建立与数据库（可能是本地的或远程计算机上的一个文件）之间的通信，一次获取一项内容，最后关闭连接。用于这项操作的例程命名为 setXent，getXent 和 endXent，此处 X 表示数据库的名字。例如，用于主机数据库的库例程命名为 sethostent，gethostent 和 endhostent。描写这些例程的章节只概述其调用，不再重复其使用细节。

21.19 网络字节顺序转换例程

前面讲过，各个机器在整数值存储方式上各不相同，TCP/IP 协议为字节顺序定义了与机器无关的标准。套接字 API 提供了 4 个宏，用于本地主机字节顺序和网络标准字节顺序之间的转换。为了使程序可移植，当编写程序时，每次将一个整数值从本地机器复制到网络分组时，或从网络分组中把值复制到本地机器中时，都应该调用转换例程。

这 4 个转换例程都是以一个值作为参数的函数，并返回字节重排后的一个新值。例如，为了把一个短整数（2 字节）从网络字节顺序转换为本地主机字节顺序，程序员可以调用 ntohs（network to host short，网络到主机短整数转换），其格式为：

```
localshort = ntohs(netshort)
```

参数 netshort 是网络标准字节顺序的 2 字节（16 位）整数，结果 localshort 是本地主机字节顺序的短整数。

在 C 编程语言中称 4 字节（32 位）的整数为**长整数**（long）。函数 ntohs（network to host long，网络到主机长整数转换）将 4 字节长整数从网络标准字节顺序转换到本地主机字节顺序。程序以函数的形式调用 ntohs，提供一个网络字节顺序的长整数作为参数：

```
locallong = ntohs(netlong)
```

还有两个相似的函数，允许程序员完成从本地主机字节顺序到网络字节顺序的转换。函数 htons 将一个本地主机字节顺序的短整数（2 字节）转换为网络标准字节顺序的短整数。程序以函数的形式调用 htons：

```
netshort = htons(localshort)
```

最后一个转换例程是 htonl，它把长整数转换成网络标准字节顺序。与其他例程一样，htonl 也是一个函数：

```
netlong = htonl(locallong)
```

很明显，转换例程能保持如下数学关系：

```
netshort = htons(ntohs(netshort))
```

和

```
localshort = ntohs(htons(localshort))
```

类似的关系对于长整数转换例程也成立。

21.20 IP 地址操作例程

由于许多程序要在 32 位 IP 地址和对应的点分十进制表示方式之间相互转换，因此套接字库包括完成此转换的实用例程。过程 inet_aton 和 inet_network 都将点分十进制格式转换为网络字节顺序的 32 位 IP 地址。inet_aton 形成一个 32 位的主机 IP 地址；inet_network 形成的是主机地址部分为零的网络地址^①。它们的格式如下：

```
error_code = inet_aton(string, address)
```

和

```
address = inet_network(string)
```

此处的参数 string 给出一个 ASCII 字符串的地址，其中包含以点分十进制格式表示的数字，参数 address 是一个指向存放了二进制值的长整数的指针。点分十进制格式有 1~4 个数字段，中间用点隔开。如果 4 个数字段都出现，那么每个数字段对应的是最后得到的 32 位整数中的一个字节。如果少于 4 个数字段，就扩展最后一个数字段，以填充其余字节。

过程 inet_ntoa 执行的是 inet_aton 的逆向操作，它完成从 32 位整数到点分十进制格式的 ASCII 字符串的映射，格式如下：

```
str = inet_ntoa(internetaddr)
```

此处的参数 internetaddr 是一个网络字节顺序的 32 位 IP 地址，str 是结果得到的 ASCII 字符串地址。

通常操作 IP 地址的程序必须将网络地址与主机在网络上的本地地址相结合。过程 inet_makeaddr 完成这种结合，其格式如下：

```
internetaddr = inet_makeaddr(net, local)
```

参数 net 是主机字节顺序的 32 位网络 IP 地址，参数 local 是代表网络上本地主机地址的整数，同样也是主机字节顺序的。

过程 inet_ntof 和 inet_lnaof 提供 inet_makeaddr 的反向功能，将一个 IP 地址的网络部分和本地

^① 由于这些函数都假设地址是有类别的，因此 inet_network 和 inet_lnaof 不处理 CIDR 地址。

主机部分分开。它们的格式如下：

```
net = inet_ntof(internetaddr)
```

和

```
local = inet_lnaof(internetaddr)
```

此处的参数 `internetaddr` 是网络字节顺序的 32 位 IP 地址，返回结果为主机字节顺序。

21.21 访问域名系统^①

与 TCP/IP 域名系统之间的接口由 5 个库例程组成。调用这些例程的应用程序成为一个域名系统的客户，它发送一个或多个域名服务器请求并接收响应。

一般的思路是，由程序形成查询请求，发送到一个服务器并等待响应。由于存在许多选项，而例程只有几个基本的参数，所以使用了一个全局结构 `res` 来存放其他参数。例如，`res` 中的一个字段允许调试报文，而另一个字段控制查询代码使用 UDP 还是 TCP。`res` 中的大多数字段有合理的默认值，因此可以在不改变 `res` 的情况下使用例程。

程序在使用其他过程之前先调用 `res_init`。该调用没有参数：

```
res_init()
```

`res_init` 读取一个包含运行域名服务器的机器名之类信息的文件，并将结果存入全局结构 `res` 中。

过程 `res_mkquery` 形成域名查询，并将查询放到一个内存缓冲区中。这个调用的格式是：

```
res_mkquery(op, dname, class, type, data, datalen, newrr, buffer, buflen)
```

前 7 个参数直接与域名查询中的字段对应。参数 `op` 指明请求的操作，`dname` 给出包含域名的字符数组的地址，`class` 是给出查询种类的整数，`type` 是给出查询类型的整数，`data` 给出包含在查询中的数据数组的地址，`datalen` 是给出数据长度的整数。除了库过程，套接字 API 还给应用程序提供了为重要值定义符号常量的能力。因此，程序员不必了解协议细节就能使用域名系统。最后的两个参数 `buffer` 和 `buflen`，分别指明存放查询的缓冲区地址和长度。最后，在当前实现中参数 `newrr` 尚未使用。

一旦程序形成一个查询，它就调用 `res_send` 将其发送给一个名字服务器并获取响应。格式为：

```
res_send(buffer, buflen, answer, anslen)
```

参数 `buffer` 是保存待发送报文的存储区指针(可以推测，应用程序调用过程 `res_mkquery` 形成报文)。参数 `buflen` 是指明报文长度的整数。参数 `answer` 给出将写入响应的存储区的地址。整型参数 `anslen` 指定响应区域的长度。

除了形成和发送查询的例程之外，套接字库例程还包含两个例程，在传统 ASCII 串与查询中使用的压缩格式之间转换域名。过程 `dn_expand` 将一个压缩域名扩充为完整的 ASCII 串，格式如下：

```
dn_expand(msg, eom, compressed, full, fulllen)
```

参数 `msg` 给出包含要扩充名字的域名报文的地址，`eom` 指定报文结束的界限，超过此限就不能再扩充。参数 `compressed` 是指向压缩后的名字首字节的指针。参数 `full` 是将写入扩充名的数组的指针。参数 `fulllen` 是指明数组长度的整数。

产生压缩的名字比扩充压缩的名字更复杂，因为压缩涉及到删除公共的后缀。在压缩名字时，客户必须保存以前出现过的后缀的记录。过程 `dn_comp` 通过将后缀与以前用过的后缀列表进行比较，并删除可能的最长后缀，来压缩一个完整的域名。调用格式为：

```
dn_comp(full, compressed, cmprlen, prevptrs, lastptr)
```

参数 `full` 给出一个完整域名的地址。参数 `compressed` 指向将要存放压缩名的字节数组，参数 `cmprlen` 指明数组的长度。参数 `prevptrs` 是一个数组地址，数组中是指向当前报文中之前压缩的后缀的指针，

^① 第 23 章将详细讨论域名系统。

`lastptr` 指向此指针数组末尾。通常，如果使用了一个新后缀才会调用 `dn_comp` 压缩域名并更新 `prevptrs`。

也可以使用过程 `dn_comp` 将域名从 ASCII 字符串转换为非压缩的内部格式（也就是说，不删除后缀）。为此，进程唤醒 `dn_comp` 时只需将参数 `prevptrs` 设为 NULL（即为 0）。

21.22 获取主机信息

有些库过程允许进程通过给出主机的域名或 IP 地址来查询有关此主机的信息。当在已访问过域名服务器的机器上使用这些例程时，库过程通过发出一个到服务器的请求并等待响应，使进程成为一个客户。当这些例程用在还没有访问域名系统的系统上时（例如不在因特网上的主机），例程就从存放于辅助存储器的数据库中获取所需的信息。

函数 `gethostbyname` 用一个主机域名作为参数，并返回指向该主机信息的结构指针。调用格式如下：

```
ptr = gethostbyname(namestr)
```

参数 `namestr` 是指向包含主机域名的字符串的指针。返回值 `ptr` 指向包含下列信息的一个结构：正式的主机名、主机注册过的别名的列表、主机地址类型（即地址是否是 IP 地址）、地址长度和主机的一个或多个地址的列表。更多细节可见 UNIX 程序员手册。

函数 `gethostbyaddr` 与 `gethostbyname` 产生同样的信息。两者的区别在于 `gethostbyaddr` 以一个主机地址作为参数：

```
ptr = gethostbyaddr(addr, len, type)
```

参数 `addr` 是指向包含主机地址的字节序列的指针。参数 `len` 是给出地址长度的整数，参数 `type` 是指明地址类型（例如，它是一个 IP 地址）的一个整数。

正如前面提到的，过程 `sethostent`、`gethostent` 和 `endhostent` 提供了之后对主机数据库的一系列访问。

21.23 获取网络的信息

主机或者使用域名系统，或者为自己互联网中的网络维护一个简单的数据库。套接字库例程包括 5 个允许进程访问网络数据库的例程。过程 `getnetbyname` 根据给出的网络域名从数据库中获取或格式化某一项的内容。调用格式为：

```
ptr = getnetbyname(name)
```

此处的参数 `name` 是个指针，指向包含需要获取信息的网络名的字符串。返回值是一个结构的指针，该结构内容包含网络的正式名字、注册别名的列表、一个整数地址类型和一个 32 位网络地址（即，把主机部分设为零的 IP 地址）。

在给定地址时，进程若需要查询有关网络的信息，则调用库例程 `getnetbyaddr`。调用格式为：

```
ptr = getnetbyaddr(netaddr, addrtype)
```

参数 `netaddr` 是一个 32 位网络地址，参数 `addrtype` 是一个整数，指明了 `netaddr` 的类型。过程 `setnetent`、`getnetent` 和 `endnetent` 提供了之后对网络数据库的一系列访问。

21.24 获取协议信息

还有 5 个库例程提供对某台机器上使用的协议数据库的访问。每个协议都有一个正式名、注册别名和一个正式协议号。过程 `getprotobynumber` 允许调用方获取给定名字的协议的有关信息：

```
ptr = getprotobyname(name)
```

此处的参数 name 是个指针，指向包含要获取信息的协议名的字符串。返回值是一个结构的指针，该结构内容包含协议的正式名字、注册别名列表以及分配给此协议的唯一整数值。

过程 `getprotobyname` 允许进程使用协议号作为关键字搜索协议信息：

```
ptr = getprotobyname(number)
```

最后，过程 `setprotoent`, `getprotoent` 和 `endprotoent` 提供了之后对协议数据库的一系列访问。

21.25 获取网络服务信息

在第 11 章和第 12 章中都讲过，为熟知服务保留了一些 UDP 和 TCP 协议端口号。例如，TCP 端口 43 是为 whois 服务保留的。whois 允许一台机器上的客户与另一台机器上的服务器联系，并获取在服务器上有账号的用户的有关信息。在服务数据库中，whois 项指明服务名为 whois，协议为 TCP，且协议端口号为 43。有 5 个库例程可用于获取有关服务和使用协议端口的信息。

过程 `getservbyname` 把一个命名服务映射到一个端口号上：

```
ptr = getservbyname(name, proto)
```

此处的参数 name 指明了包含需获取信息的服务名的字符串的地址，并且参数 proto 是一个字符串，它给出了该服务所用协议的名称。在典型情况下，协议仅限于 TCP 和 UDP。返回值是一个结构的指针，该结构内容包含服务名、别名列表、服务所用协议的标识符以及分配给此服务的协议端口号（整数）。

过程 `getservbyport` 允许调用方根据分配给某服务的端口号，从服务数据库获取记录项。调用格式为：

```
ptr = getservbyport(port, proto)
```

参数 port 是分配给服务的整型协议端口号，参数 proto 指明服务所用的协议。与其他数据库一样，进程可以使用 `setservent`, `getservent` 和 `endservent` 进行之后的一系列服务数据库的访问。

21.26 客户举例

下列 C 语言程序示例了程序如何使用套接字 API 访问 TCP/IP 协议，这是 whois 客户和服务器的简单实现。正如 RFC 3912 中所定义的，whois 服务允许某台机器上的客户获取远程系统中用户的有关信息。在此实现中，客户是一个用户可调用的应用程序，带有两个参数：远程机器名和该机器上要获取信息的用户的名称。客户调用 `gethostbyname` 把远程机器名映射到一个 IP 地址，并调用 `getservbyname` 查找 whois 服务使用的熟知端口。一旦映射主机和服务名成功，客户就创建一个套接字，并指明套接字将使用可靠流交付（即 TCP）。然后，客户将套接字连接到指定目的机器的 whois 协议端口。

```
/* whoisclient.c - main */
```

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
```

```
/*-----*
```

```
* Program: whoisclient
```

```
*  
* Purpose: UNIX application program that becomes a client for the  
*           Internet "whois" service.  
*  
* Use:      whois hostname username  
*  
* Author:   Barry Shein, Boston University  
*  
* Date:    Long ago in a universe far, far away  
*  
*-----  
*/  
main(argc, argv)  
int argc;          /* standard UNIX argument declarations */  
char *argv[];  
{  
    int s;          /* socket descriptor */  
    int len;         /* length of received data */  
    struct scckaddr_in sa; /* Internet socket addr. structure */  
    struct hostent *hp; /* result of host name lookup */  
    struct servant *sp; /* result of service lookup */  
    char buf[BUFSIZ+1]; /* buffer to read whois information */  
    char *myname;     /* pointer to name of this program */  
    char *host;        /* pointer to remote host name */  
    char *user;        /* pointer to remote user name */  
  
    myname = argv[0];  
/*  
 * Check that there are two command line arguments  
*/  
if(argc != 3) {  
    fprintf(stderr, "Usage: %s host username\n", myname);  
    exit(1);  
}  
host = argv[1];  
user = argv[2];  
/*  
 * Look up the specified hostname  
*/  
if((hp = gethostbyname(host)) == NULL) {  
    fprintf(stderr, "%s: %s: no such host?\n", myname, host);  
    exit(1);  
}  
/*
```

```
* Put host's address and address type into socket structure
*/
bcopy((char *)hp->h_addr, (char *)&sa.sin_addr, hp->h_length);
sa.sin_family = hp->h_addrtype;
/*
 * Look up the socket number for the WHOIS service
 */
if((sp = getservbyname("whois", "tcp")) = NULL) {
    fprintf(stderr, "%s: No whois service on this host\n", myname);
    exit(1);
}
/*
 * Put the whois socket number into the socket structure.
 */
sa.sin_port = sp->s_port;
/*
 * Allocate an open socket
 */
if(s = socket (rp->h_addrtype, SOCK_STREAM, 0)) < 0) {
    perror("socket");
    exit(1);
}
/*
 * Connect to the remote server
 */
if(connect(s, &sa, sizeof sa) < 0) {
    perror("connect");
    exit(1);
}
/*
 * Send the request
 */
if(write(s, user, strlen(user)) != strlen(user)) {
    fprintf(stderr, "%s: write error\n", myname);
    exit(1);
}
/*
 * Read the reply and put to user's output
 */
while( (len = read(s, buf, BUFSIZ)) > 0)
    write(1, buf, len);
close(s);
exit(0);
}
```

21.27 服务器举例

服务器的例子只比客户稍复杂。服务器是循环的：在熟知 whois 端口上监听，反复接受下一个请求，使用 UNIX getpwnam 函数在 UNIX 口令文件中查找用户名，并向客户返回所请求的信息。

```
/* whoisserver.c - main */
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <pwd.h>

/*
 * Program: whoisserver
 *
 * Purpose: UNIX application program that acts as a server for
 *          the "whois" service on the local machine. It listens
 *          on well-known WHOIS port (43) and answers queries from
 *          clients. This program requires super-user privilege to
 *          run.
 *
 * Use:    whois hostname username
 *
 * Author: Barry Shein, Boston University
 *
 * Date: Long ago in a universe far, far away
 */
#define BACKLOG      5 /* # of requests we're willing to queue */
#define MAXHOSTNAME 32 /* maximum host name length we tolerate */

main(argc, argv)
int argc;           /* standard UNIX argument declarations */
char *argv[];
{
    int s, t;        /* socket descriptors */
    int i;            /* general purpose integer */
    struct sockaddr_in sa, isa; /* Internet socket address structure*/
    struct hostent *hp; /* result of host name lookup */
    char *myname;     /* pointer to name of this program */
    struct servent *sp; /* result of service lookup */
    char localhost[MAXHOSTNAME+1];/* local host name as character string */
}
```

```
myname = argv[0];
/*
 * Look up the WHOIS service entry
 */
if((sp = getservbyname("whois","tcp")) = NULL) {
    fprintf(stderr, "%s: No whois service on this host\n", myname);
    exit(1);
}
/*
 *      Get our own host information
 */
gethostname(localhost, MAXHOSTNAME);
if((hp = gethostbyname(localhost)) = NULL) {
    fprintf(stderr, "%s: cannot get local host info?\n", myname);
    exit(1);
}
/*
 * Put the WHOIS socket number and our address info
 * into the socket structure
 */
sa.sin_port = sp->s_port;
bcopy((char *)hp->h_addr, (char *)&sa.sin_addr, hp->h_length);
sa.sin_family = hp->h_addrtype;
/*
 * Allocate an open socket for incoming connections
 */
if((s = socket (hp->h_addrtype, SOCK_STREAM, 0)) < 0) {
    perror("socket");
    exit(1);
}
/*
 * Bind the socket to the service port
 * so we hear incoming connections
 */
if(bind(s, &sa, sizeof sa) < 0) {
    perror("bind");
    exit(1);
}
/*
 * Set maximum connections we will fall behind
 */
listen(s, BACKLOG);
/*
 * Go into an infinite loop waiting for new connections
*/
```

```
/*
while(1) {
    i = sizeof isa;
    /*
     *We hang in accept() while waiting for new customers
     */
    if((t = accept(s, &isa, &i)) < 0) {
        perror("accept");
        exit(1);
    }
    whois (t);/* perform the actual WHOIS service */
    close(t);
}

/*
 * Get the WHOIS request from remote host and format a reply.
 */
whois (sock)
int sock;
{
    struct passwd *p;
    char buf[BUFSIZ+1];
    int i;
    /*
     * Get one line request
     */
    if( (i = read(sock, buf, BUFSIZ)) <= 0)
        return;
    buf[i] = '\0'; /* Null terminate */
    /*
     * Look up the requested user and format reply
     */
    if((p = getpwnam(buf)) <= NULL)
        strcpy(buf, "User not found\n");
    else
        sprintf(buf, "%s: %s\n", p->pw_name, p->pw_gecos);
    /*
     * Return reply
     */
    write (sock, buf, strlen(buf));
    return;
}
```

21.28 小结

由于 TCP/IP 协议软件位于操作系统内，应用程序和 TCP/IP 协议之间的确切接口取决于操作系统的详细情况，不属于 TCP/IP 协议标准的范畴。我们研究了套接字 API，它最初是为 BSD UNIX 设计的，但事实上已经成为诸如 Microsoft 之类的运营商所用的标准。可以看出，套接字借鉴了 UNIX 的“打开一读一写一关闭”范例。为了使用 TCP，程序必须创建一个套接字，为它绑定地址，接受传入的连接，然后进行通信。最后，当套接字的使用结束时，程序必须关闭它。除了套接字抽象和套接字上运行的系统调用，BSD UNIX 还包括一些库例程，可帮助程序员创建和操作 IP 地址，在本地机器格式和网络标准字节顺序之间进行整数转换，搜索诸如网络地址之类的信息。

套接字 API 很流行，并受到许多运营商的广泛支持。在操作系统中没有提供套接字设施的厂商通常会提供一个套接字库，这样，即使底层操作系统使用不同的系统调用集，程序员在编写应用程序时也可照样使用套接字调用。

21.29 深入研究

在 UNIX 程序员手册中可以找到有关套接字函数的详细信息，其中第二部分包括对每个 UNIX 系统调用的描述，而第三部分包括对每个库过程的描述。UNIX 还通过 man 命令提供手册的在线副本。McKusick, Bostic, Karels, and Quarterman [1996] 更详细地探讨了 UNIX 系统。Hall et. al. [1993] 中包含了 Windows Sockets 的原始标准。

操作系统提供商通常在其系统上提供仿真套接字的过程库。查阅这些提供商的程序员手册可了解详细情况。Gilligan et. al. [RFC 3493] 探讨了 IPv6 的套接字扩展。

本系列教材的第三卷描述了如何构造客户和服务器程序，其中有四种版本的套接字，每种各使用不同的 API：BSD 套接字版、Windows 套接字版（适用于 Microsoft Windows）、Linux/POSIX 版以及一个为 AT&T 系统 V UNIX 而定义的使用**运输层接口**（Transport Layer Interface）的版本，同时这个版本在 Solaris 中也有效。

21.30 习题

1. 试在本地系统上运行 whois 示例客户和服务器。
2. 试构造一个接受多个并发连接的简单服务器（作为测试，让该进程处理一个连接，打印一个短报文，延迟任意时间后再打印另一个报文，然后退出）。
3. 何时 listen 调用显得重要？
4. 你的本地系统提供什么过程来访问域名系统？
5. 设计一个服务器，它使用单个 UNIX 进程（即单线程执行），但可以处理多个并发的 TCP 连接。提示：考虑 select。
6. 阅读有关套接字接口的替换方案，如**运输库接口**（Transport Library Interface，简称 TLI），并将其与套接字接口进行比较。两者之间的主要概念区别是什么？
7. 每个操作系统都会限制某程序在某一时刻同时使用的套接字数。在你的本地系统中一个程序可创建多少套接字？
8. 套接字 / 文件描述符机制及其相关联的 read 和 write 操作可被看成是面向对象设计的一种形式。试解释原因。
9. 考虑接口设计的另一种方案，它为每层的协议软件提供一个接口（例如，系统允许一个应用程序不使用 IP 发送和接收原始分组，或者不使用 UDP 或 TCP 发送和接收 IP 数据报）。

这种接口的优点是什么？缺点是什么？

10. 客户和服务器可在同一台计算机上运行并使用 TCP 套接字通信。解释为什么可能构造一个客户和一个服务器，它们不需要知道主机的 IP 地址就能在同一台机器上通信。
11. 试验本章中服务器的例子，看是否能足够快地产生 TCP 连接，超出服务器指定的 backlog 值。如果服务器运行在只有一个处理器的计算机上，传入的连接请求超出 backlog 的速度，是否会比运行在具有 5 个处理器的计算机上的服务器快？试解释原因。

第22章 启动引导与自动配置（DHCP）

22.1 引言

本章将说明如何把客户一服务器模型用于计算机的**启动引导**（bootstrapping）。连接到 TCP/IP 互联网的每台计算机都需要获得一个 IP 地址及其子网掩码，同时还有其他一些信息，包括路由器地址和名录服务器地址。本章将描述一种使主机在启动时能够自动判决这些信息的协议。自动的初始化过程是很重要的，因为它允许用户在不了解有关地址、掩码、路由器等概念，或者不知道应该如何配置协议软件的情况下，也能够将计算机连接到因特网上。

这里所描述的启动引导处理过程可能会出乎你的意料，因为它用 UDP 来传送报文。看起来用 UDP 好像不可能找到 IP 地址，因为每个 UDP 报文都需要包装在 IP 数据报中发送。但是我们将会看到，在第 3 章中曾讨论过的一些特殊 IP 地址使之成为可能。

22.2 启动引导的发展历程

正如第 5 章中所述，RARP 协议最初是为了让计算机能够获取一个 IP 地址而研发的。后来，一个名为**引导程序协议**（BOOTstrap Protocol，简称 BOOTP）的更通用的协议取代了 RARP。最终，作为 BOOTP 的继承者，研究人员又开发了**动态主机配置协议**（Dynamic Host Configuration Protocol，简称 DHCP）。因为 DHCP 是对 BOOTP 的继承，所以本章所做的描述通常对两者都适用。但是为了简化文字，我们将以 DHCP 来说明。

由于 DHCP 使用的是 UDP 和 IP，所以能够用一个应用程序来实现它。与其他应用协议类似，DHCP 遵守客户一服务器模型。DHCP 协议仅需交换一个分组，在这个交换过程中，计算机向服务器发送一个分组，以请求启动引导信息，而服务器则发送一个分组作为响应，在这个分组中指定的都是计算机启动时所需的项目，包括计算机的 IP 地址、路由器的地址以及名字服务器的地址。DHCP 同时也在响应分组中包含了一个**选项**（options）字段，以允许运营商发送一些由其自己的计算机使用的附加信息^①。

22.3 用 IP 来确定 IP 地址

我们讲过，DHCP 使用 UDP 携带报文，并且 UDP 报文要封装在 IP 数据报中交付。为了理解计算机如何在掌握其 IP 地址前就能用 IP 数据报发送 DHCP，可回忆第 3 章中的几个特殊用途 IP 地址。特别是用于目的地址时，由全 1 组成的 IP 地址（255.255.255.255）指的是受限广播。IP 软件在发现其本地 IP 地址信息之前，就可以接受和广播指明了受限广播地址的数据报。此处的要点是：

一个应用程序在发现本地网络的 IP 地址或机器的 IP 地址之前，可以使用 IP 受限广播地址强迫 IP 在本地网络上广播一个数据报。

假设客户机 A 想使用 DHCP 得到启动引导信息（包括其 IP 地址），同时假设位于同一物理网

^① 正如我们将会看到的，术语“选项”并不是非常贴切，因为根据规范的建议，选项字段应当用于通用性质的信息，如子网掩码等。

络上的服务器 B 将响应请求。由于 A 不知道 B 的 IP 地址或网络的 IP 地址，它必须使用 IP 受限广播地址来广播它的初始 DHCP 请求。响应情况会如何？B 能直接发回响应吗？不行，即使 B 知道 A 的 IP 地址也不行。要知道原因，可以想象 B 上的应用程序试图使用 A 的 IP 地址发送数据报的情况。在为数据报选择路由以后，B 上的 IP 软件将数据报传给网络接口软件。假定这里使用的是第 5 章描述的 ARP，接口软件必须将下一跳的 IP 地址映射到对应的硬件地址上。但是，由于 A 还没有收到 DHCP 响应，它无法识别自己的 IP 地址，因而不能回答 B 的 ARP 请求。因此，B 只有两种方案可选：或者广播响应，或者使用来自请求分组的信息，在 ARP 高速缓存中添加一项。在不允许应用程序修改 ARP 高速缓存的系统上，广播是唯一的解决方案。

22.4 DHCP 的重传策略

DHCP 将可靠性通信的责任都交给了客户。我们知道，由于 UDP 使用 IP 进行交付，报文可能出现延迟、丢失、无序或重复。此外，由于 IP 不提供数据检验，UDP 数据报在到达时可能有某些位遭到破坏。为防止破坏，DHCP 要求 UDP 必须使用检验技术。它也规定在发送请求和响应时应当将**不分片**（do not fragment）位设为 1，以适应几乎没有存储空间用于重装数据报的客户。DHCP 的设计是允许多个响应的，但它只接受和处理第一个响应。

为处理数据报的丢失问题，DHCP 使用常规的**超时**（timeout）和**重传**（retransmission）技术。当客户发送一个请求时，启动一个计时器。如果计时器超时还没有收到响应，那么客户就必须重传请求。当然，若出现电源故障，网络上的所有机器都将同时重新启动，完全有可能会使 DHCP 服务器因请求过多而超载。如果所有客户使用完全相同的重传定时，那么大多数或者所有客户可能又会同时重传请求。为了避免冲突，DHCP 规范建议使用随机时延。另外，规范建议一开始使用 0~4 s 之间的随机超时，而在每次重传后将定时加倍。当超时达到一个较大数值（60 s）以后，客户不再增加定时，但继续使用随机取值。每次重传后使定时加倍，可防止 DHCP 在原本就拥塞的网络上增加更多的通信量，而随机取值有助于避免同时传输。

22.5 DHCP 报文格式

为使实现尽量简单，DHCP 报文具有定长的字段，且响应与请求的格式相同。虽然我们说过客户和服务器都是程序，但 DHCP 协议在使用这些术语时比较随意，它把发送 DHCP 请求的机器称为**客户**（client），而把发送响应的任何机器都称为**服务器**（server）。图 22.1 所示为 DHCP 报文的格式。

字段 OP 指明报文是请求（1）还是响应（2）。与在 ARP 中一样，硬件类型（HTYPE）字段和硬件长度（HLEN）字段指明了网络硬件类型和硬件地址的长度（如以太网类型为 1，地址长度为 6）^①。客户把跳数（HOPS）字段置 0。如果 DHCP 服务器收到请求，并决定将请求转送到其他机器（如允许跨多个路由器的启动引导），它就会增加跳数字段的计数值。交互 ID（TRANSACTION ID）字段包含的是客户用于匹配请求和响应的一个整数值。秒数（SECONDS）字段报告自客户开始启动后逝去的秒数。

客户 IP 地址（CLIENT IP ADDRESS）字段及其后各字段中包含了最重要的信息。为了获得最大程度的灵活性，客户应尽量填写它所知道的信息，并将剩余部分置 0。例如，如果客户知道它想要从中获取信息的某个服务器的名字或地址，就应该能够填写服务器 IP 地址（SERVER IP ADDRESS）或服务器主机名（SERVER HOST NAME）字段。如果这些字段非零，那么只有与名字或地址字段匹配的服务器才能响应请求；若它们为零，收到请求的任何一台服务器都可以响应。

^① 硬件类型字段的值可以在最新的 RFC 编号文档中找到。

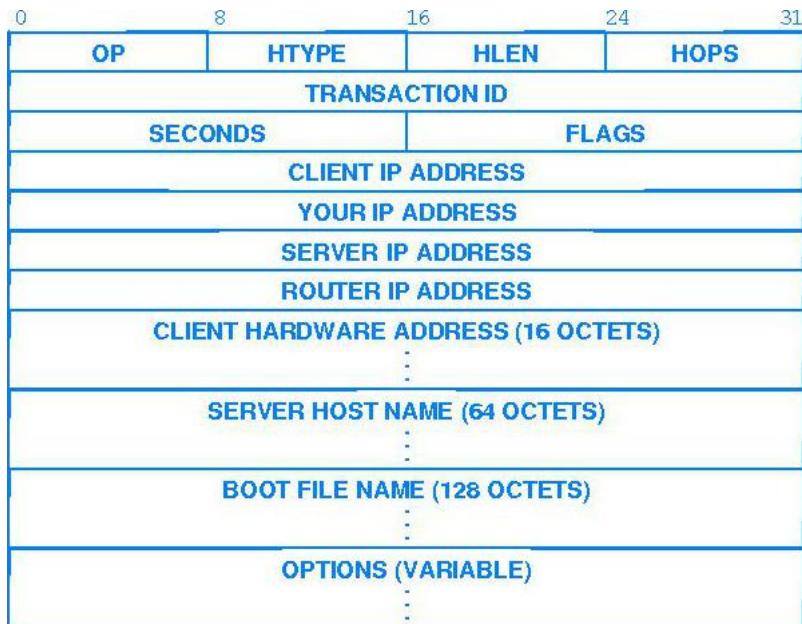


图 22.1 DHCP 报文的格式。为了使实现足够小，以便能装入 ROM，所有字段都是定长的

已经知道自己 IP 地址的客户也可以使用 DHCP（即获取其他信息）。已知 IP 地址的客户将其地址置于客户 IP 地址（CLIENT IP ADDRESS）字段中，其他不知道 IP 地址的客户则将其置 0。如果在一个请求报文中客户的 IP 地址为零，服务器会在你的 IP 地址（YOUR IP ADDRESS）字段中返回客户的 IP 地址。

16 位的标志（FLAGS）字段允许控制请求和响应。如图 22.2 所示，目前标志字段中只有最高位是有实际意义的。

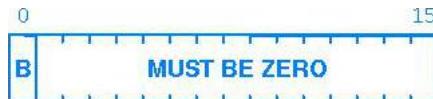


图 22.2 DHCP 报文的 16 位标志字段格式。最左边的位解释为广播请求。其他所有的位必须为 0

客户用标志字段的最高位来控制服务器在发送响应时是采用单播还是广播。要想了解为什么客户需要选择广播方式响应，可以回想一下，当客户与 DHCP 服务器通信时，它还没有 IP 地址，也就是说，客户无法响应 ARP 询问。因此，为了保证客户能够接收由 DHCP 服务器发送来的报文，它可以请求服务器使用 IP 广播方式发送响应，相当于硬件广播。IP 数据报的处理规则允许丢弃目的地址与计算机地址不匹配，并通过硬件单播到达的任何数据报，但是要求 IP 接受并处理任何发往 IP 广播地址的数据报。

有趣的是，DHCP 没有在报文中为嵌入式系统提供用于下载具体内存映像的空间。事实上，DHCP 提供了一个启动文件名（BOOT FILE NAME）字段，以供小型无盘系统使用。客户可利用此字段填写一个大概的名称，如“unix”，意思就是“我希望这台机器启动 UNIX 操作系统”。DHCP 服务器会查询它的配置数据库，将这个名称映射为一个具体的文件名，此文件中含有适合于客户硬件的内存映像，并在响应中返回这个完整规范的文件名。当然，配置数据库也允许完全自动的启动引导，此时客户将 DHCP 请求中的启动文件名字段置 0，并由 DHCP 为该机器选择一个内存映像，然后再由客户利用标准的文件传送协议（如 TFTP）来获取这个映像。这种方式的优点在于无盘客户可以使用一个大概的名称，而不需要对具体的文件名进行编码。并且，网络管理员也可以在不改

变嵌入系统 ROM 的条件下，调整启动映像所在的位置。

选项区中的项目全部使用 Type-Length-Value(TLV)方式编码，即每一项都包含一个**类型**(type)八位组和一个**长度**(length)八位组，最后还规定了指定长度的**值**(value)。

22.6 动态配置的必要性

早期引导程序协议的操作环境相对来说是静态的，其中每台主机都有一个永久的网络连接。管理人员创建一个配置文件，该文件为每台主机指定一组参数，其中包括一个 IP 地址。由于配置通常保持不变，该文件不会经常改变。典型情况下，配置将保持数星期不变。

但是，在目前的因特网中，ISP 的消费群体在不断发生变化，而无线连网和可移动便携机的发展，使计算机能够快速、方便地在不同地点之间移动。为了处理好自动地址分配问题，DHCP 允许计算机快速且动态地获取一个 IP 地址。也就是说，在配置 DHCP 服务器时，管理员需要提供一组 IP 地址。只要有一台新的计算机与网络相连接，它就会与服务器联络并请求一个地址。服务器从管理员指定的一组地址中选择一个，并把这个地址分配给发出请求的计算机。

为了做到完全通用，DHCP 允许三种地址分配方式，并由管理员选择 DHCP 对每个网络或每台主机的响应方式。与先前的 BOOTP 类似，DHCP 允许**人工配置**(manual configuration)，此时由管理员为每台具体的计算机指定一个具体的地址。DHCP 同时也允许一种称为**自动配置**(automatic configuration 或 autoconfiguration)的形式，此时管理员可以让 DHCP 服务器为第一次连接网络的计算机分配一个永久的地址。最后，DHCP 还允许完全的**动态配置**(dynamic configuration)，此时，服务器在一定的期限内把地址“租”给一台计算机。动态地址分配是 DHCP 最新颖、最强大的一面。

DHCP 根据客户的身份以及服务器配置文件决定如何进行处理。当客户与 DHCP 服务器联系时，客户发出一个标识符，通常是客户的硬件地址。服务器根据客户的标识符和客户所连接的网络，决定如何分配客户和 IP 地址。因此，管理人员可以完全控制地址的分配。我们通过对服务器的设置，让它为某些特定计算机静态地分配地址，同时为其他计算机动态地分配永久地址或临时地址。总结如下：

DHCP 允许计算机在没有人工干预和事先配置的情况下，获取一些在给定网络上运行所需的必要信息，包括一个 IP 地址。动态分配要受到管理上的约束。

22.7 DHCP 的租用概念

动态地址分配是临时的。我们说 DHCP 服务器将地址在有限时间内租(lease)给一个客户使用。服务器在地址分配时指明其租期。在租用期间，服务器不会将同一个地址再租给其他客户，但是在租期结束时，客户必须续租或停止使用该地址。

一个 DHCP 租期应该有多长？租期的最优时间与特定的网络环境和特定的主机需求有关。例如，为了保证能够尽快地循环使用地址，大学实验室学生使用的网上计算机的租期可能会很短(如 1 小时)。相反，一个公司的网络可能使用 1 天或 1 周的租期。而 ISP 则会根据与消费者之间的合约来制定租期。为适应各种可能的环境，DHCP 没有为租期定义一个固定的常量，而是让客户申请一个具体的租期，并由服务器通知客户它所认可的租期。因此，管理员可以决定服务器应该给一个客户分配多长时间的地址。在极端情况下，DHCP 服务器有一个保留值是**无限**(infinity)，以允许租期持续任意长的时间(即分配一个永久地址)。

22.8 多个地址和中继

一台多地址的机器会与多个网络相连接。当这种机器启动时，它可能需要为每个接口都获取配置信息。但是 DHCP 报文只提供一个接口的信息。有多个接口的计算机必须分开处理每个接口。因此，虽然我们在描述 DHCP 时似乎是一台机器只需要一个地址，但是读者必须记住，多地址计算机的每个接口在协议中的位置可能并不相同。

DHCP 使用了**中继代理**（relay agent）的概念，以允许计算机与非本地网络上的服务器联系。当中继代理（通常是一个路由器）接收到来自客户的广播申请时，它会将申请转发到服务器，然后返回服务器发给主机的响应。中继代理使多地址配置变得复杂，因为服务器可能会收到来自同一台机器的多个请求。然而，虽然 DHCP 使用了术语**客户标识符**（client identifier），但我们认为多地址客户发送的客户标识符是一个识别了特定接口的值（如一个唯一的硬件地址）。因此，即使收到的是通过中继代理发来的请求，服务器也能区分一台多地址主机发出的多个请求。

22.9 地址获取状态

当客户使用 DHCP 获取一个 IP 地址时，它处于 6 种状态之一。图 22.3 中的状态转换图绘出了引起客户状态转换的事件和报文。

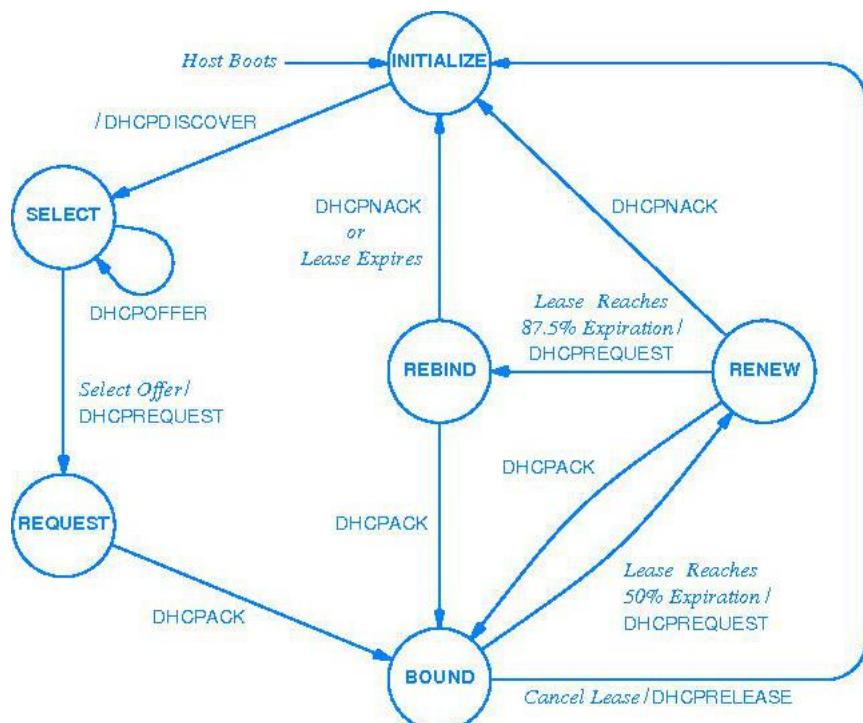


图 22.3 DHCP 客户的 6 个主要状态和状态之间的转换。各个转换标签列出了引起转换的传入报文或事件，接着是一个斜线，然后是客户发送的报文

当客户第一次启动时，进入 INITIALIZE（初始化）状态。为了获取一个 IP 地址，客户首先要与本地网络上的所有 DHCP 服务器联系。为此，客户广播一个 DHCPDISCOVER（DHCP 发现）报文，并进入 SELECT（选择）状态。由于 DHCP 协议是对 BOOTP 的扩充，所以客户在 UDP 数据报中发送 DHCPDISCOVER 报文时，数据报中的目的端口设为 BOOTP 端口（即端口 67）。本地网上的所有 DHCP 服务器都会收到这个报文，而只有那些被配置为响应给定客户的服务器才会发送 DHCPOFFER（DHCP 提供）报文。因此，客户可能会收到零个或多个响应。

处于 SELECT 状态时，客户收集来自 DHCP 服务器的 DHCPOFFER 响应。每个提供报文都包含了客户的配置信息，以及服务器向客户出租的一个 IP 地址。客户必须选择其中一个响应（如第一个到达的响应），并与服务器协商租用事宜。为此，客户给服务器发送一个 DHCPREQUEST(DHCP 请求) 报文，并进入 REQUEST(请求) 状态。作为对接受请求和租期生效的确认，服务器以一个 DHCPACK (DHCP 确认) 报文进行响应。客户收到确认后进入 BOUND (已绑定) 状态，此时客户可以开始使用此地址。归纳如下：

为了使用 DHCP，主机通过向本地网络中的所有服务器广播一个报文而成为客户。然后该主机收集由各个服务器提供的响应，从中选择一个，并与该服务器进一步验证接受事宜。

22.10 提前终止租期

我们认为 BOUND 状态是操作的正常状态，客户在使用得到的 IP 地址时通常保持在 BOUND 状态。如果客户有辅助存储器（如本地磁盘），就可以存储分配给它的 IP 地址，并在下次启动时申请同一个地址。但是，在某些情况下，处于 BOUND 状态的客户可能会发现它不再需要 IP 地址了。例如，假设便携机用户将机器连接到网络上，并使用 DHCP 获得一个 IP 地址，然后用该便携机读取电子邮件。协议指明的最短租期为一个小时，而这个规定可能比用户实际需要的时间长得多。

无论在什么情况下，当不再需要一个地址时，DHCP 允许客户提前终止租期，而不用等租期满。如果服务器可用的 IP 地址数量比连接到网络上的计算机数目少得多，那么提前终止租用就尤其重要。如果每个客户都能做到不再需要 IP 地址时立即终止租用，服务器就可以把地址分配给其他客户。

为了提前终止租期，客户向服务器发送一个 DHCPRELEASE (DHCP 释放) 报文。释放地址是阻止客户继续使用该地址的决定性动作。因此，发送释放报文后，客户就不能再使用此地址发送其他数据报了。根据图 22.3 所示的状态转换图，发出 DHCPRELEASE 报文的主机将会脱离 BOUND 状态，并且如果下次又想使用 IP 地址了，还必须重新从 INITIALIZE 状态开始。

22.11 租用续租状态

我们讲过，DHCP 客户获得了一个地址以后，就进入 BOUND 状态。进入 BOUND 状态的客户会设置三个计时器，分别用于控制租用续租、重新绑定和租用到期。DHCP 服务器给客户分配地址时，可以明确指明这几个计时器的值。如果服务器未指明计时器的值，客户就使用默认值。第一个计时器的默认值通常是总租期的一半。当第一个计时器到期时，客户必须尝试续租。为了请求续租，客户向提供租用的服务器发送 DHCPREQUEST 报文，然后客户进入 RENEW (续租) 状态等待响应。DHCPREQUEST 报文中包含了一个客户正在使用的 IP 地址，并请求服务器延长对此地址的租期。与最初的租用协商过程一样，客户可以请求续租期限，但是最终控制续租的还是服务器。服务器可以用两种方式响应客户的续租请求：指示客户停止使用该地址，或者同意客户继续使用该地址。如果服务器同意，就发送 DHCPACK，使客户返回 BOUND 状态并继续使用该地址。DHCPACK 中也可含有客户计时器的新值。如果服务器不同意继续使用，就发送一个 DHCPNACK，使客户立即停止使用该地址并返回 INITIALIZE 状态。

客户在发送请求延长租期的 DHCPREQUEST 报文后，保持在 RENEW 状态等待响应。如果没有收到响应，那么提供租用的服务器或者关机了，或者不可达。为了处理这种情况，DHCP 依靠第二个计时器，它在客户进入 BOUND 状态后开始设置。第二个计时器在租期的 87.5% 时到期，并使客户从 RENEW 状态转换到 REBIND (重新绑定) 状态。在进行这个转换时，客户假定原来的 DHCP 服务器是不可用的，并开始向本地网络上的所有服务器广播 DHCPREQUEST 报文。任意一个被设

置为可以向该客户提供服务的服务器都可以给出肯定响应（即延长租期）或否定响应（即拒绝继续使用该 IP 地址）。如果客户收到一个肯定响应，就返回到 BOUND 状态并重置两个计时器。如果客户收到否定响应，就必须转换到 INITIALIZE 状态，并且必须停止使用该 IP 地址。在它能够继续使用 IP 之前必须获取一个新地址。

客户进入 REBIND 状态后，它已经向原服务器和本地网络上的其他服务器发出延长租期的请求。很少有客户在第三个计时器到期，也就是租用到期时，还没有从任何一个服务器收到响应。如果发生这种情况，客户就必须停止使用 IP 地址，返回到 INITIALIZE 状态，并再次开始获取一个新地址。

22.12 DHCP 选项与报文类型

出乎人们意料的是，DHCP 没有在报文中为租用信息分配固定的字段。事实上，DHCP 仍然沿用了 BOOTP 的报文格式，只是利用选项（OPTIONS）字段来识别这个报文是 DHCP 报文。如图 22.4 所示，DHCP 的报文类型选项用于指明所发送的是什么类型的 DHCP 报文。

0	8	16	23		
CODE (53)	LENGTH (1)	TYPE (1 - 7)			
<hr/>					
TYPE FIELD	Corresponding DHCP Message Type				
1	DHCPDISCOVER				
2	DHCPOFFER				
3	DHCPREQUEST				
4	DHCPDECLINE				
5	DHCPACK				
6	DHCPNACK				
7	DHCPRELEASE				
8	DHCPINFORM				

图 22.4 DHCP 选项格式，用于指明发送的 DHCP 报文类型。表中列出了第三个八位组的可能取值

DHCP 响应中的每个选项字段都有一个类型和长度字段，合起来决定了选项的大小。图 22.5 中列出了一些可能的选项。

Quote Server	8	N	IP addresses of N/4 quote servers
Lpr Servers	9	N	IP addresses of N/4 lpr servers
Impress	10	N	IP addresses of N/4 Impress servers
RLP Server	11	N	IP addresses of N/4 RLP servers
Hostname	12	N	N bytes of client host name
Boot Size	13	2	2-octet integer size of boot file
RESERVED	128-254	-	Reserved for site specific use

图 22.5 DHCP 响应中选项字段的项目类型和内容, DHCP 响应长度可变

22.13 选项过载

DHCP 报文首部中的服务器主机名 (SERVER HOST NAME) 字段和启动文件名 (BOOT FILE NAME) 字段各自占用了许多个八位组。如果给定报文的这两个字段中有一个不含信息, 那么空间就浪费了。为了使 DHCP 服务器将这两个字段作为其他选项使用, DHCP 定义了一个选项过载 (Option Overload) 的选项。如果出现了这个选项, 则是告诉接收方应当忽略服务器主机名和启动文件名字段的通常含义, 而从中寻找其他的选项。

22.14 DHCP 与域名^①

虽然 DHCP 可以根据需要为计算机分配 IP 地址, 但 DHCP 无法自动完成将主机永久连接到因特网上所需的全部过程, 事实上, DHCP 协议没有指明任何与域名系统之间的交互。因此, 除非附加另外一种机制, 否则主机名与分配给主机的 IP 地址之间的绑定肯定是相互独立的。

虽然目前还没有一个标准, 但是有些 DHCP 的实现确实已经做到了与 DNS 之间的交互。例如, Linux 或 BSD 之类的 UNIX 系统试图让 DHCP 与 DNS 软件相协调, 称为 **named 绑定**^② (named bind) 或简称为 **绑定** (bind)。同样, 微软的 DHCP 软件与微软的 DNS 软件之间相互协调, 以保证分配了 DHCP 地址的主机同时也具有一个域名。这种协调机制也可逆向操作, 以保证在 DHCP 租用撤回时, DNS 也收到撤回相应域名的通知。

22.15 小结

通过使用动态主机配置协议, 一台计算机在启动时就可以获取必要的信息, 包括默认路由器的地址、域名服务器的地址以及一个 IP 地址。DHCP 允许服务器自动或动态地分配 IP 地址。在计算机可以快速接入和断开的环境中 (如无线网络), 动态分配是很必要的。

为了使用 DHCP, 计算机成为了客户。计算机向 DHCP 服务器广播一个请求, 从收集到的提供项中选择一个, 并与该服务器交换报文, 以获取对通知地址的租用。中继代理能够以客户的身份转发 DHCP 请求, 这也就意味着可以由一个 DHCP 服务器来处理一个网点上多个子网的地址分配。

客户在获得 IP 地址时会启动三个计时器。第一个计时器到期后, 客户尝试租用续租。如果在续租成功前, 第二个计时器又到期了, 客户就尝试通过任意一台服务器重新绑定其地址。如果在成功续租之前, 最后一个计时器也到期了, 那么客户会停止使用 IP 地址, 并返回到 INITIALIZE 状态, 重新获取新地址。可以用一个有限状态机来解释租用的获取和续租。

22.16 深入研究

有关 BOOTP 的详细内容参见 Croft and Gilmore [RFC 951], 它对 BOOTP 和 RARP 加以比较, 并可作为正式标准。Droms [RFC 2131] 和 Lemon and Cheshire [RFC 3396] 给出了 DHCP 的规范, 包括状态转换的详细描述。Alexander and Droms [RFC 2132] 和 Lemon et. al. [RFC 3442] 定义了 DHCP 选项编码。最后, Droms [RFC 1534] 讨论了 BOOTP 和 DHCP 的互操作性。

^① 第 23 章详细讨论了域名系统。

^② 术语 named 是**命名守护神** (name daemon) 的缩写。

22.17 习题

1. DHCP 没有专门包含用于服务器向客户返回日历的字段，但把它作为特定厂商信息（可选）的一部分。是否应当将时间包含在必需的字段中？试解释原因？
2. 试论证，将配置和内存映像的存储分开并不好。提示：参见 RFC 951。
3. DHCP 报文格式是不一致的，因为它有两个字段用于客户 IP 地址，有一个字段用于引导映像的名字。如果客户让其 IP 地址字段为空，服务器在第二个字段中返回客户的 IP 地址。如果客户让引导文件名字段为空，服务器就用一个明确的名字替代它。为什么？
4. 阅读标准，找出客户和服务器如何使用跳数（HOPS）字段。
5. 当 DHCP 客户通过硬件广播收到回答时，它怎样知道该回答是否给同一个物理网络上另一个 DHCP 客户？
6. 当机器使用 DHCP 而非 ICMP 获取子网掩码时，它给其他主机带来的负载会减少。试解释原因。
7. 阅读标准，了解 DHCP 客户和服务器在没有同步时钟的情况下如何就租期长度达成一致。
8. 考虑一个有磁盘的主机使用 DHCP 获取 IP 地址。如果主机在磁盘上存储其地址和租用到期的时间，然后在租用期间重启动机器，它还能使用该地址吗？试解释原因？
9. DHCP 要求最短租用期为 1 小时。你能想出由于 DHCP 的最短租用而造成不便的一种情况吗？具体说明情况。
10. 阅读 RFC，找出 DHCP 如何定义续租计时器和重绑定的计时器。服务器能否只设置其中的一个计时器？试解释原因？
11. 状态转换图中没有画出重传。阅读标准，找出客户应重传请求多少次？
12. DHCP 能否保证客户没有进行“电子欺骗”（即 DHCP 能否保证它没有将主机 A 的配置信息发送到主机 B）？如果是 BOOTP，答案是否有所不同？试解释原因？
13. DHCP 规定客户必须准备处理至少 312 八位组的选项。312 这个数字是怎么得出的？
14. 使用 DHCP 获取 IP 地址的计算机能否作为服务器使用？如果是这样，客户如何与该服务器联系？

第23章 域名系统 (DNS)

23.1 引言

前面的章节所描述的协议使用了称为因特网协议地址 (IP 地址) 的 32 位整数来识别机器。虽然这种地址能方便而紧凑地表示出在互联网中发送的分组的源地址和目的地址，但是用户更愿意为机器指定一个好读又好记的名字。

本章将考虑的是一种为大量机器分配有含义的高级名字的方案，并讨论高级机器名字与 IP 地址之间的映射机制。我们将会讨论从高级名字到 IP 地址之间的转换以及从 IP 地址到高级名字之间的转换。命名方案之所以引起我们的关注，主要有两个原因。第一，整个因特网已经在用它来为机器分配名字了。第二，由于它使用了分布在不同地点的一组服务器来实现名字和地址之间的映射，所以名字映射机制的实现为第 20 章所描述的客户—服务器模型提供了一个重要的实例。

23.2 机器的名字

最初，计算机系统要求用户必须能够理解诸如系统表和外围设备之类的对象的数值化地址。分时系统使计算技术有了更进一步的发展，允许用户为物理对象（如外设）和抽象对象（如文件）设计一个有意义的符号名字。类似的模式也出现在计算机网络发展过程中。早期的系统支持计算机之间的点对点连接，并使用低级硬件地址来指明不同的机器。而网际互连引入了**通用编址** (universal addressing) 的概念，并出现了将通用地址映射到低级硬件地址的协议软件。由于大多数计算环境中都含有多台计算机，所以用户需要用有意义的符号名字来识别它们。

早期机器的名字反映了它们所在的小环境。对于只有少数几台机器的网点来说，通常根据机器的使用目的来选择机器名。例如，诸如 accounting, production 和 development 之类的名字都是常见的机器名。与麻烦的硬件地址相比，用户认为这样的名字更受欢迎。

虽然地址和名字之间的区别在于是否直观，但区别只是人为因素造成的。任何名字只是由选自一个有限字母表的字符序列构成的标识符。只有当系统能够有效地将名字映射为它所表示的对象时，名字才是有用的。因此，我们认为 IP 地址是一个**低级名字** (low-level name)，而用户更喜欢的是机器的**高级名字** (high-level name)。

高级名字的格式很重要，因为它决定了如何将名字转换为低级名字或绑定到对象，以及怎样分配名字才是合法的。当机器很少时，选择一个名字十分容易，各管理员可以随便选择一个名字，只要验证这个名字没有其他人使用就可以了。例如，早在 1980 年，当普度大学计算机科学系的主计算机连接到因特网上时，系里为上网的计算机选择了“purdue”这个名字。当时存在潜在冲突的名字只有几十个。而到了 1986 年中期，因特网上主机的正式清单中就有 3100 个正式登记名和 6500 个正式的别名使用的是“purdue”。虽然在 20 世纪 80 年代时清单上的数目已经增长迅速，但大多数网点还有未进行登记的其他机器（如个人计算机）。时至今日，因特网连接着上亿台机器，要选择一个具有象征意义的名字实非易事。

23.3 平面名字空间

最初，整个因特网上的机器所使用的名字集合形成了一个**平面名字空间** (flat namespace)，其

中每个名字都是由字符序列组成的，没有更多的结构。在最初方案中，由中心网点，也就是**网络信息中心**（Network Information Center，简称 NIC）来管理名字空间，并决定新加入的名字是否合适（也就是说，它禁止淫秽的名字或保证新名字不与现有名字冲突）。

平面名字空间的主要优点在于简洁而短小。其主要缺点是，由于技术上和管理上的原因而不能容纳大量机器。首先，由于名字取自单一的标识符集合，潜在冲突会随着网点数目的增加而增加。第二，由于负责增加新名字的管理机构必须位于某个网点上，该中心网点的管理工作负载也会随着网点数目的增加而增长。为了理解此问题的严重性，可以想象一个具有几千个网点的快速增长的互联网，每个网点拥有几百台个人计算机和工作站。每次把一台新的个人计算机连到网上之前，该机器的名字都必须得到中心管理机构的认可。第三，由于名字到地址之间的绑定关系经常改变，用于维护每个网点完整映射表的正确副本的开销会很高，并且会随网点数目的增加而增长。从另一个角度看，如果名字数据库位于单个网点上，那么到该网点的网络通信量也会随着网点数目的增加而增大。

23.4 分级的名字

命名系统如何才能支持快速扩充的庞大的名字集合，同时又不需要一个中心网点进行管理？答案在于分散命名的机制，该机制将名字空间的各个部分授权给不同的管理机构，并且以分布方式负责名字和地址之间的映射。因特网使用的就是这种方案。在讨论细节之前，我们先考虑此方案背后的动机和人们的直观想法。

在定义名字空间的划分时必须做到既支持高效的名字映射，又能保证名字分配的自治控制。只使映射高效而进行的优化，得到的结果仍将保持平面的名字空间，只是通过把名字分给多台机器进行映射而降低了通信量。若只为使管理更容易而进行优化，将得到更容易委托管理的解决方案，但名字映射却开销较大或很复杂。

要了解应当如何划分名字空间，可以考虑一个大型组织的内部组织结构。在最高层，总裁负有全部责任。由于总裁不能顾及每一件事情，所以该组织又划分为各个部门，由经理负责管理每个部门。总裁允许各个部门在一定范围内自治。更具体地讲，掌管各部门的经理不需要获得总裁的直接批准就可以雇用或解雇职员、分配办公室以及授权下一级管理机构。

大型组织的等级划分除了便于授权管理以外，还引入了自治操作。例如，当某个办公室的工作人员需要诸如一个新雇员的电话号码之类的信息时，他一开始就可以询问本地的办事员（而该办事员又可能与其他部门的办事员联系）。关键问题在于，虽然管理权总是沿着公司的等级划分从上到下传递的，但信息可以从一个办公室跨级流动到另一个办公室。

23.5 名字的授权管理

分级命名机制工作起来很像一个大型组织的管理。名字空间将在最高层进行**划分**（partitioned），各分区内的名字管理权委派给指定的代理负责。例如，我们可以选择根据**网点名**（site name）来划分名字空间，并授权每个网点负责维护自己分区内的名字。由最高层来决定名字空间的划分，并为每个分区授予管理权，这样它就不会因为某个分区的内部变化而受到干扰。

对于按等级分配的名字，其语法通常反映了负责分配名字的管理机构的级别。例如，考虑具有如下名字格式的一个名字空间：

local.site

此处，site 是由中心管理机构授权的网点名，local 是由各网点负责控制的名字部分，而“.”^①是用

^① 在域名系统中，“.”分隔符的发音为“dot”。

于分隔它们的分隔符。如果最高管理机构同意增加一个新的网点 X，它就把 X 添加到有效网点列表中，并委托网点 X 负责管理所有以 “.X” 结尾的名字。

23.6 子集管理机构

在分级名字空间中，每一级的管理机构可以进一步向下划分名字空间。在按网点划分的例子中，网点本身可能是由几个行政小组构成的，网点的管理机构可以选择在各行政小组之间进一步划分自己的名字空间。其基本思想是不断地向下划分名字空间，直至每个分区足够小，以便于管理。

从语法上看，细分的名字空间将引入名字的另一个分段。例如，对已经用 site 划分过的名字进一步划分，会增加一个 group 段，将产生如下名字语法：

local.group.site

因为由最上层指派管理机构，所以组名没有必要在所有网点之间达成一致。一个大学的网点可选择 engineering, science 和 arts 之类的组名，而一个公司的网点则可以选择 production, accounting 和 personnel 之类的组名。

美国电话系统提供了分级命名语法的另一实例。10个电话号码数字划分为 3 个数字的区号(area code)、3 个数字的交换局号(exchange) 和交换局内 4 个数字的用户号(subscriber number)。每个交换局有权分配自己这部分名字空间中的用户号。虽然可以将任意一群用户分配到某个交换局，也可将任意一群交换局划归某个地区，但电话号码的分配并不是随心所欲的，它们都是被认真选择的，以便能够在电话网中方便地为电话呼叫选择路由。

电话系统这个例子很重要，因为它展现了 TCP/IP 互联网中使用的分级命名方案与其他分级方案之间的关键区别：在将某个组织所拥有的一组机器进行划分时，会按管理机构这条线索来划分，并没有说一定要通过物理位置来划分。例如，在某个大学中，同一个建筑物中可能容纳了数学系和计算机科学系，有可能虽然这两个组(系)的机器位于不同的管理域，但它们却连到了同一个物理网络上。还有可能属于同一个系的机器位于好几个物理网络上。基于这些原因，TCP/IP 命名方案允许任意委托分级名字空间的管理机构，而与物理连接无关。这个概念可归纳如下：

在因特网中，分级的机器名是根据组织结构进行分配的，其中每个组织都获得了部分名字空间的管理权，而不一定根据物理上的网络互连结构来分配。

当然，在许多网点中，组织的等级结构符合物理上的网络互联结构。例如，假设某个部门的计算机全部都连在同一个网络上，如果同时这个部门已分配得到了分级命名空间中的一部分，则所有名字落在这一部分中的机器正好连到同一个物理网络上。

23.7 因特网的域名

域名系统 (Domain Name System, 简称 DNS) 是为因特网提供从名字到地址的映射的系统。DNS 有两方面在概念上相互独立的内容。一方面的内容是抽象的，指明了名字的语法和名字的授权管理规则，另一方面的内容是具体的，指明了一个分布式计算系统的实现，这个系统能高效地将名字映射为地址。本节将讨论名字的语法，后面几小节将讨论实现。

域名系统使用称为**域名** (domain names) 的分级命名方案。与前面的例子一样，域名由被分隔符“.”隔开的子名字序列组成。在前面的例子中曾说过，名字中的一节可能代表了网点或组，但是在域名系统中，简单地将每一节称为一个标号 (label)。因此，域名

cs.purdue.edu

含有三个标号：cs, purdue 和 edu。在域名中，任何一个由标号形成的后缀都称为一个**域** (domain)。在上面的例子中，最底层的域是 cs.purdue.edu (普度大学计算机科学系的域名)，第二级域是

purdue.edu（普度大学的域名），顶级域是 edu（教育机构的域名）。如本例所示，域名书写时将本地标号放在第一位，而将顶级域放在最后。我们将会看到，按此顺序书写域名使得压缩含有多个域名的报文成为可能。

23.8 顶级域

图 23.1 中列举了几个顶级域名。

Domain Name	Meaning
aero	Air transport industry
arpa	Infrastructure domain
biz	Businesses
com	Commercial organization
coop	Cooperative associations
edu	Educational institution (4-year)
gov	United States government
info	Information
int	International treaty organizations
mil	United States military
museum	Museums
name	Individuals
net	Major network support centers
org	Organizations other than those above
pro	Credentialed professionals
country code	Each country (geographic scheme)

图 23.1 因特网的顶级域及其含义。虽然标号是小写的，但域名系统在比较域名时忽略大小写，因此 COM 与 com 完全一样

从概念上看，顶级域名出现了两种完全不同的命名分级方式：地理的和组织的。按地理划分就是把全世界的机器按国家和地区来划分。位于美国的所有机器都在顶级域 us 下。当其他国家想要在域名系统中登记机器时，中心管理机构给该国家分配一个新的顶级域，以该国家的国际标准双字母标识符作为其标号。US 域的管理机构已选择为每个州分配一个二级域。例如，弗吉尼亚州的域是。

va.us

作为不同于地理分级的另一种方案，顶级域也允许按组织类型划分。当一个组织想参与到域名系统中时，它选择希望按何种方式登记并请求批准。由**域名登记机构** (domain name registrar) 评审申请，并在已有的顶级域下为该组织分配一个子域^①。顶级域的拥有者可以决定如何以及怎样进一步划分其名字空间。例如，英国的双字母国家代码是 uk，它的大学和科研院所都登记在 ac.uk 域下面。

有一个例子可能有助于阐明命名等级和名字管理之间的关系。对于普度大学计算机科学系中的一台名为 xinu 的机器，其正式域名是：

xinu.cs.purdue.edu

其中的机器名是由计算机科学系的本地网络管理员认可和登记的。系管理员在此之前已经从普度大学的网络管理机构处获得了子域 cs.purdue.edu 的管理权，而大学网络管理机构也已经从因特网管理机构处获得管理子域 purdue.edu 的许可。因特网管理机构具有对 edu 域的控制，因此必须得到其许可才能添加新的大学。类似地，普度大学的网络管理员具有对 purdue.edu 子域的控制权，因而必须

^① 在标准中并没有定义术语“子域” (subdomain)。我们选用这个词是因为它与“子集”类似，有助于阐明域之间的关系。

得到他的许可才能添加新的第三级域。

图 23.2 所示为因特网域名等级中的一小部分。如图所示，IBM 公司是一个商业组织，它登记的域名为 ibm.com，普度大学登记的域名为 purdue.edu，而 National Science Foundation 是一个政府机构，它登记的域名为 nsf.gov。与此相反，Corporation for National Research Initiatives 选择了按地址划分，登记为 cnri.reston.va.us。

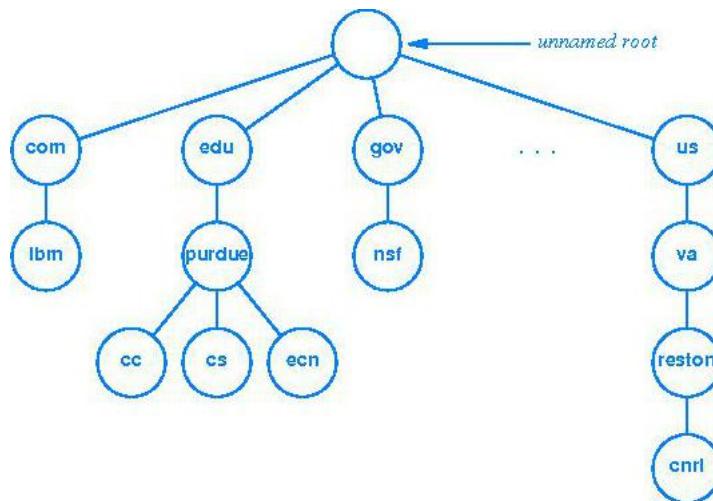


图 23.2 因特网域名等级（树）中的一小部分。命名树实际上很宽，也很平直，大多数主机项在第五层出现

23.9 名字的语法和类型

域名系统相当通用化，因为它允许在一个系统中内置多个命名等级。为了让客户能够在多种类型的项中加以区分，每个命名项在系统中存储时都分配有一个**类型** (type)，该类型指明了它是一台机器的地址，还是一个邮箱或一个用户等。当客户请求域名系统解析一个名字时，必须指明所需回答的类型。例如，当电子邮件应用程序使用域名系统解析一个名字时，它指明所需的回答应该是一个**邮件交换机** (mail exchanger) 的地址。远程登录应用程序指明它要找的是一台机器的 IP 地址。理解下面的观点很重要：

一个给定的名字可能会映射到域名系统中的多个项。客户在解析名字时要指明所需对象的类型，并由服务器返回此类型的对象。

除了指明所求结果的类型之外，域名系统还允许客户指明使用的协议族。域名系统按类别 (class) 划分整个名字集合，允许一个数据库存储用于多个协议族的映射^①。

根据名字的语法并不能判断它所命名的对象的类型或协议族的类别，尤其不能根据一个名字中的标号数量来判断名字是指向单个对象（机器）还是一个域。在我们的例子中可能有某台机器的名字是

gwen.purdue.edu

即使

cs.purdue.edu

用来命名一个子域。我们总结了以下重要观点：

仅从域名的语法中无法区分到底是子域的名字还是某个对象的名字，也无法识别命名对象

^① 实际上，使用多协议族的域服务器很少见。

的类型。

23.10 从域名映射到地址

除了名字的语法规则和管理机构的授权以外，域名方案还包括一个高效并且可靠的通用分布式系统，用于名字到地址的映射。从技术角度看，这个系统是分布式的，即位于多个网点中的一组服务器协同运作，以解决映射问题。系统中的大多数名字在本地即可完成映射，只有少量映射需要在互联网上通信，可见这样的系统是很高效的。它也是通用的，因为并没有局限用于机器名（虽然现在我们使用的例子是机器名）。最后，它是可靠的，因为某一台计算机出现故障不会妨碍整个系统的正确运行。

从名字映射到地址的域名机制由若干个独立的、相互协作的称为**名字服务器**（name server）的系统组成。名字服务器是提供从名字到地址的转换（从域名映射到 IP 地址）的服务器程序。通常，服务器程序在专用处理器上运行，并把这台机器本身称为名字服务器。客户软件称为**名字解析器**（name resolver），它在翻译名字时使用一个或多个名字服务器。

要想理解域名服务器如何工作，最简单的方法是想象它们安放在对应的命名等级树结构中，如图 23.3 所示。树的根是能够识别顶级域的服务器，并且它知道每个域该由哪个服务器来解析。给定一个要解析的名字以后，根可以为该名字选择一个正确的服务器。在下一级的一组服务器中，每一个服务器都负责回答对一个顶级域（如 edu）的查询。这一级的服务器知道哪个服务器可以解析它的域下面的某个子域。在树的第三级，名字服务器负责回答对一个子域（如 edu 下的 purdue）的查询。这种概念树可以一直向下发展，每一级在定义一个子域时都有一个服务器。

概念树中的链接并不表示物理网络的连接，而是表示某个给定服务器与它所知道的以及与它联系的其他服务器之间的关系。服务器本身可以位于互联网中的任意位置，因此服务器树是使用互联网进行通信的一个抽象体。

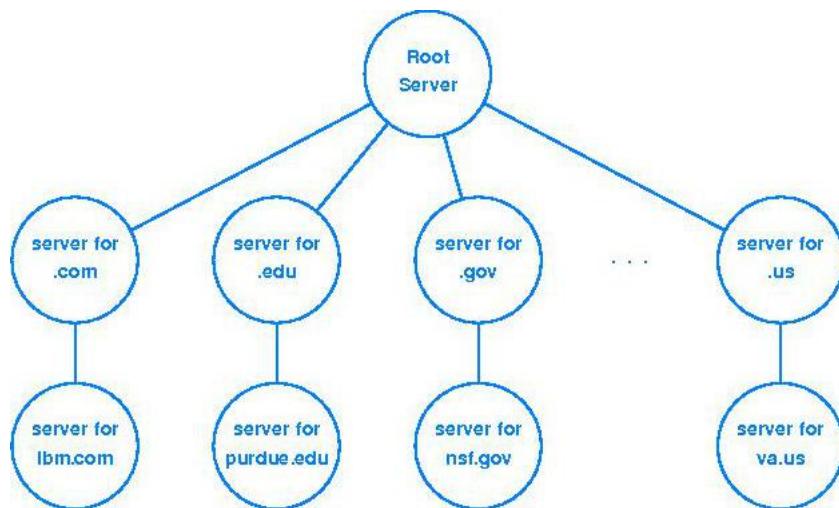


图 23.3 对应于命名等级树中域名服务器的概念布局。从理论上讲，每个服务器都知道它下一层的所有子域服务器的地址

如果域名系统中服务器的工作恰如我们的简化模型所示，则其连通性与授权之间的关系将很简单。如果某个子域得到了授权，那么申请授权的机构必须为该子域建立一个名字服务器，并将其链入树中。

实际上，命名等级与服务器树之间的关系并不像模型所示的那么简单。服务器树的层次并不多，

因为用一个物理服务器就能包含大部分命名等级的所有信息。在实际应用中，各组织经常从它的所有子域中收集信息并存放在一个服务器中。图 23.4 所示为图 23.2 的命名等级服务器的更实际的组织方式。

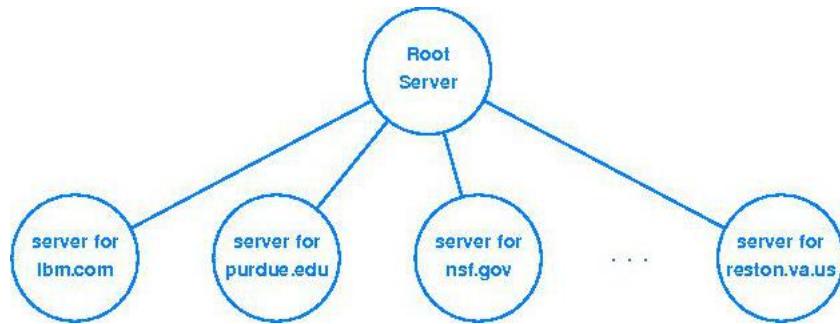


图 23.4 用于图 23.2 的命名等级服务器的更实际的组织方式。因为树很宽并且很平直，解析名字时只需联系很少的服务器

根服务器包含有关根以及顶级域的信息，而每个组织只使用一个名字服务器。由于服务器树显得比较矮，所以解析 `xinu.cs.purdue.edu` 之类的名字至多只需要联系两个服务器：根服务器和 `purdue.edu` 域的服务器（即根服务器知道哪个服务器可以处理 `purdue.edu`，而普度大学的全部域信息都位于一个服务器中）。

23.11 域名解析

虽然概念树使我们很容易理解服务器之间的关系，但它隐藏了几个微妙的细节。了解名字解析算法有助于解释这些细节。从概念上讲，域名解析自上而下进行，从根名字服务器开始，一直延续到叶上的服务器。使用域名系统有两种方式，或者每次联系一个名字服务器，或者请求名字服务器系统执行完全转换。无论哪种方式，客户软件都要形成一个域名查询，包括待解析的名字、对名字类别的说明、所需回答的类型以及一个代码，该代码用于指明是否要求名字服务器完全转换名字。它把查询发送给名字服务器以求解析。

当域名服务器收到查询时，检查名字是否处于自己有权管理的子域内。如果是，就根据自己的数据库把名字转换为地址，并将回答附加到查询中，然后返回给客户。如果名字服务器不能完全解析这个名字，它就检查客户指明的是哪种类型的交互。如果客户请求完全转换，按域名的术语则称之为**递归解析** (recursive resolution)，服务器就和能解析该名字的服务器联系，并将回答返回给客户。如果客户请求非递归解析，即**迭代解析** (iterative resolution)，该名字服务器就不能提供解析结果。它产生的回答指明了客户应联系的下一个解析名字的名字服务器。

当开始查找时，客户如何找到起始的名字服务器呢？当名字服务器不能解析名字时，它又如何找到能解析名字的其他名字服务器呢？答案很简单。客户必须知道怎样才能与至少一个名字服务器取得联系。而为了保证域名服务器与其他服务器之间的联系，域名系统要求每个域名服务器至少应该知道一个根服务器的地址^①。除此之外，服务器可能知道它的上一层次的服务器（称为父服务器）的地址。

域名服务器的所有通信都使用一个熟知协议端口，因此只要客户知道服务器所在机器的 IP 地

^① 为了更加可靠，域名服务器树的每个节点上都有多个服务器，并且根服务器也有备份的服务器，以提供负载均衡。

址，就知道如何与服务器通信。关于主机如何能找到在本地环境中运行名字服务器的机器，目前还没有一种标准的方法，而是留给设计客户软件的人员自己解决。在有些系统中，提供域名服务的机器地址在应用程序编译时就被绑定，而在其他一些系统中，这个地址存放在辅助存储器上的一个文件中。还有很多系统将自动获取域名服务器的地址作为启动引导过程的一部分^①。

23.12 高效率的转换

虽然沿名字服务器树向下解析查询似乎是很自然的事，但如果真的这样做，会因为以下三个原因而导致效率不高。第一，大多数名字解析涉及到的是本地名字，即与产生查询的机器在同一个名字空间分区中。沿着从上到下的路径逐级找到本地管理机构的做法，其效率会很低。第二，如果每个名字解析都从联络顶层服务器开始，将会造成根服务器的超载。第三，顶层机器的故障将妨碍名字解析，即使本地管理机构能够解析该名字。前面提到的电话号码分级有助于解释这些观点。虽然电话号码是按级分配的，但它们是按自底向上的方式解析的。由于大多数电话呼叫都是本地的，可由本地交换局进行解析，所以不需要逐级查询。此外，某个区号内的电话呼叫不需要与区号外的交换局联系。当这些做法用到域名系统中时，产生了两步名字解析机制，该机制维持了管理等级，但允许高效的转换。

我们已经说过，大多数发给名字服务器的查询涉及的是本地名字。在两步名字解析过程中，首先从本地的名字服务器开始解析。如果本地服务器不能解析该名字，就必须将查询请求发送到域名系统中的其他服务器。

23.13 高速缓存：高效率的关键

如果解析器将每个非本地名字的解析请求都送到根服务器，查询开销会非常高。即使查询能直接送到管理该名字的服务器，名字的查找仍然是互联网的一个沉重负担。因此，要提高名字服务器系统的整体性能，就需要降低非本地名字查询的开销。

因特网名字服务器使用**高速缓存** (caching) 来优化查询开销。每个服务器维护一个高速缓存，存放最近用过的名字以及从何处获得名字映射信息的记录。当客户请求服务器解析某个名字时，服务器首先按标准过程检查它是否被授权管理该名字。如果没有被授权，就查看自己的高速缓存，检查该名字最近是否被解析过。服务器向客户报告缓存的信息，但将这些信息标记为**非权限** (nonauthoritative) 绑定^②，还会给出获得此绑定的服务器 S 的域名。本地服务器还会附带发送其他一些额外信息，如将服务器 S 与 IP 地址的绑定告知客户。因此，客户很快就会收到回答，但是得到的信息有可能已过时。如果强调高效性，客户可以选择接受非权限的回答并进行处理。如果强调准确性，客户可以选择联系负责的管理机构，并验证名字与地址之间的绑定是否仍有效。

由于名字到地址的绑定不会经常改变，所以在域名系统中可以很好地使用高速缓存。但是，变化还是存在的。如果服务器将第一次查询得到的信息存入缓存后就不再改变，缓存中的内容会变得陈旧（即不正确）。为保持缓存正确，服务器为每项内容计时并丢弃超过合理时间的项。如果服务器已从缓存中删去某项后，又得到查询该项信息的请求，它就必须重新到授权管理该项的服务器上获取绑定信息。更重要的是，服务器并不是将所有项合在一起提供一个固定的定时值，而是允许每一项的管理机构配置其定时值。每次当管理机构响应一个请求时，它都将在响应中包含一个**生存周期** (Time To Live, 简称 TTL)，以指明保证绑定有效的时间。因此，管理机构可以为预计保持不变的项指定一个较长的定时值，以减少网络开销，而为预计经常变化的项指定一个较短的定时值，以

^① 详见第 22 章中讨论的 DHCP。

^② 原文 authoritative 直译是“有权威的”、“可相信的”、“命令式的”或“官方的”。这里译为“权限的”或“权限”可能更便于理解上下文的意思——译者注。

提高准确性。

高速缓存不但在本地域名服务器中很重要，在主机中同样也需要高速缓存。大多数解析软件会在主机中对 DNS 项进行缓存。这样，如果用户重复地查找同一个名字，后面几次查找就可以从本地高速缓存中解析，而不需要使用网络。

23.14 域名系统的报文格式

研究客户和域名服务器之间交换报文的细节，有助于从典型应用程序的角度阐明系统的运作情况。我们假设用户调用一个应用程序，并提供了与程序通信的机器名。在它使用 TCP 或 UDP 之类的协议与指定机器进行通信前，应用程序必须先找到该机器的 IP 地址。它将域名传给本地解析程序，并请求一个 IP 地址。本地解析程序检查自己的高速缓存，如果有这个域名就返回结果。如果本地解析程序没有答案，就形成一个报文并发送给服务器（即该程序成为一个客户）。虽然此例只解析了一个名字，但报文格式却允许客户在一个报文中请求解析多个名字。每个询问中都包括了客户待查 IP 地址的域名、查询类别（即互联网）的说明和所需对象的类型（如地址）。服务器返回一个类似的报文作为响应，其中包含一些询问的回答，服务器已有这些询问所需的绑定信息。如果服务器不能回答所有询问，那么响应中将包含有关客户能与之联系并获取答案的其他服务器的信息。

响应中也含有授权回答的服务器的信息和这些服务器的 IP 地址。图 23.5 所示为报文的格式。

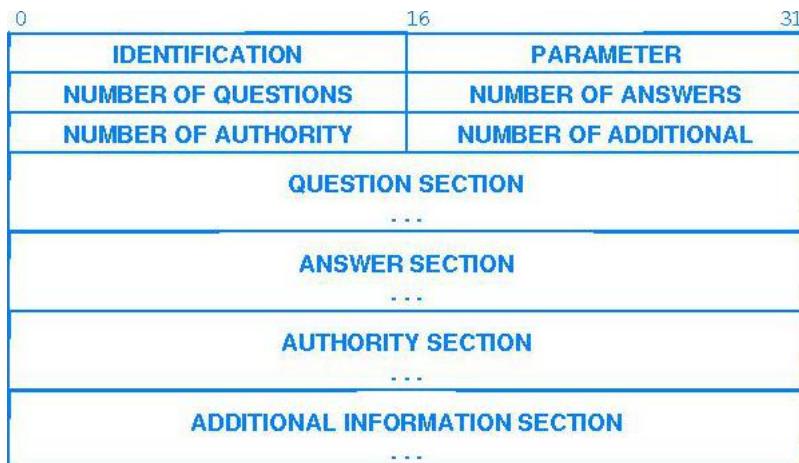


图 23.5 域名服务器的报文格式，其中的询问 (QUESTION) 区，回答 (ANSWER) 区，管理机构 (AUTHORITY) 区和附加信息 (ADDITIONAL INFORMATION) 区是变长的

如图 23.5 所示，每个报文以一个固定大小的报文首部开始。报文首部中含有一个客户用来匹配查询和响应的唯一标识 (IDENTIFICATION) 字段，以及一个参数 (PARAMETER) 字段，用来指明请求的操作和响应码。图 23.6 给出了参数字段中各位的解释。

Bit of PARAMETER field	Meaning
0	Operation: 0 Query 1 Response
1-4	Query Type: 0 Standard 1 Inverse 2 Server status request 4 Notify 5 Update
5	Set if answer authoritative
6	Set if message truncated
7	Set if recursion desired
8	Set if recursion available
9	Set if data is authenticated
10	Set if checking is disabled
11	Reserved
12-15	Response Type: 0 No error 1 Format error in query 2 Server failure 3 Name does not exist 5 Refused 6 Name exists when it should not 7 RR set exists 8 RR set that should exist does not 9 Server not authoritative for the zone 10 Name not contained in zone

图 23.6 域名服务器报文中参数字段各位的含义。位的编号由 0 开始从左至右

数目字段给出了出现在报文后面相应各区中的项数。例如，询问数目（NUMBER OF QUESTION）字段给出了报文的询问区（QUESTION SECTION）中的项数。

询问区（QUESTION SECTION）包含了需要回答的查询。客户只需要填写询问区。服务器在响应中返回询问和回答。每个询问由待查域名（QUERY DOMAIN NAME）字段、查询类型（QUERY TYPE）字段和 QUERY CLASS（查询类别）字段组成，如图 23.7 所示。



图 23.7 域名服务器报文询问区中的项的格式，其中域名是变长的，客户填写询问，服务器返回这些询问及其回答

虽然待查域名字段是变长的，但在下一节将会看到域名的内部表示方法可以使接收方知道其确切长度。查询类型对询问的类型进行编码（如这个询问指的是机器名还是邮件地址）。查询类别字段允许将域名用于任意对象，因为正式的因特网名字只是其中一种可能类别。值得注意的是，虽然图 23.5 中所描绘的报文格式沿用了以 32 位的倍数来表示的习惯，但待查域名字段可能含有任意个八位组，不需要加以填充。因此，发往或来自域名服务器的报文可能含有奇数个八位组。

在域名服务器报文中，每个回答区、管理机构区和附加信息区都是由描述域名和映射关系的一组**资源记录**（resource records）组成的。每个资源记录描述一个名字。图 23.8 显示了这一格式。

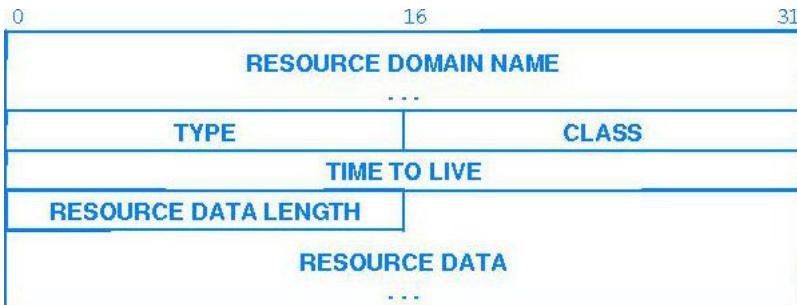


图 23.8 在域名服务器返回报文中，尾端的几个区所采用的资源记录的格式

资源域名（RESOURCE DOMAIN NAME）字段含有这个资源记录涉及的域名。该字段可以为任意长度。类型（TYPE）字段指明资源记录包含的数据的类型。类别（CLASS）字段指明数据的类别。生存时间（TIME TO LIVE）字段包含一个 32 位整数，指明这个资源记录中的信息在缓存中保持有效的秒数，它适用于请求了名字绑定并希望把结果放入高速缓存的客户。最后两个字段包含绑定的结果，其中资源数据长度（RESOURCE DATA LENGTH）字段指明了资源数据（RESOURCE DATA）字段中数据的八位组个数。

23.15 压缩的名字格式

在报文中表示域名时，域名存储为一个标号序列。每个标号以一个指明其长度的八位组开始。因此，接收方在重建域名时，重复性地先读取一个八位组的长度 n，然后读 n 个八位组长的标号。长度八位组为 0 表示域名的结束。

域名服务器经常会为一个查询返回多个回答，而在许多情况下，域名的后缀有重叠现象。为了节省回答分组的空间，名字服务器通过为每个域名存储一个副本来自压缩名字。当从报文中提取域名时，客户软件必须检查名字的每一段，看它是一个文本字符串（格式为 1 八位组的计数字段，后面跟着组成名字的字符），还是指向字符串的指针。如果遇到指针，客户必须沿着指针，找到报文中的一个新位置，才能找到其余名字。

指针总是出现在字段的前端，在计数字节中进行编码。如果 8 位的段计数字段的前两位是 1，客户就必须将后面的 14 位看成是一个整数指针。如果前两位是 0，则之后 6 位指明的是在计数八位组后面跟着的标号的字符数。

23.16 域名的缩写

电话号码分级系统说明了本地解析的另一个有用特征：**名字缩写**（name abbreviation）。当解析进程能自动提供名字的某些部分时，缩写提供了一种缩短名字的方法。通常用户在拨打本地电话号码时省略了区号。所拨数字就形成了一个缩写名，它假设所拨号码与用户号码同在一个区号内。缩写也能很好地用于机器名。给出一个类似 xyz 的名字，解析过程可以假设它与解析名字的机器位于同一个本地管理范围内。因此，解析程序可以自动提供名字中缺失的部分。例如，在普度大学的计算机科学系中，缩写名

xinu

与完整域名

xinu.cs.purdue.edu

等价。大多数客户软件用一个**域名后缀列表**（dommn name suffix list）来实现域名的缩写。本地管

理员配置一组可能出现的后缀，用于在查询时附加到名字后。当解析进程遇到一个名字时，它遍历列表，依次追加每个后缀，并试图查询形成的名字。例如，普度大学计算机科学系的后缀列表包括：

.cs.purdue.edu
.cc.purdue.edu
purdue.edu
null

因此，本地解析进程首先将 cs.purdue.edu 附加到名字 xinu 后面。如果查找失败，就将 cc.purdue.edu 附加到这个名字后并查询。例子里列表的最后一个后缀是空串，意思是如果其他查询全都失败，解析程序将尝试不带后缀进行查询。管理员可利用后缀列表使缩写更方便，或者限制应用程序只能使用本地名。

我们说过由客户负责扩充缩写的名字，但还要强调的是，这种缩写本身并不是域名系统的一部分。域名系统只允许查询完整的域名。因而，依赖于名字缩写的程序在构造它们的环境以外可能无法正常使用。概括如下：

域名系统只能将完整的域名映射到地址，缩写本身并不是域名系统的一部分，但是通过客户软件引入缩写，可以让用户更方便地使用本地名字。

23.17 反向映射

我们说过，除了从机器名到 IP 地址的映射之外，域名系统还可以提供其他映射。**反向查询** (inverse query) 允许客户请求服务器“反向”映射，即根据回答来生成得到此回答的询问。当然，不是所有的回答都对应唯一的询问。即使它的确只对应一个询问，可能服务器又无法提供这个询问。虽然反向查询从一开始就是域名系统的一部分，但使用得并不多，因为通常需要搜索整个服务器集合，才能找到能解析某个查询的服务器。

23.18 指针查询

有一种类型的反向映射可以作为一种鉴别机制，被服务器用来验证一个客户是否有权访问该服务。这种反向映射就是由服务器将客户的 IP 地址映射到域名。例如，位于 example.com 公司中的一台服务器通过配置可以做到只向本公司内部的客户提供服务。当客户联系服务器时，服务器将客户的 IP 地址映射为相应的域名，然后在同意其访问之前，先验证这个域名是否以 example.com 结尾。这种反向查找如此重要，以至于域名系统为了提供此类查找而专门支持了一个特殊的域和一种特殊的询问格式，称为**指针查询** (pointer query)。在指针查询中，提交给域名服务器的询问中指明了一个 IP 地址，该 IP 地址按照域名格式编码为可打印字符串（即点分数字的文本表示）。指针查询请求名字服务器返回指定 IP 地址的机器的正确域名。

指针查询的生成并不难。我们考虑以点分十进制形式书写的 IP 地址，它具有如下格式：

aaa.bbb.ccc.ddd

为了形成一个指针查询，客户重新整理点分十进制表示的地址，形成如下格式的字符串：

ddd.ccc.bbb.aaa.in-addr.arpa

这种新格式是特殊的 in-addr.arpa 域中的一个名字^①。由于本地名字服务器可能既没有 arpa 域的管理权，也没有 in-addr.arpa 域的管理权，它需要与其他名字服务器联系，以完成解析。为了使指针查询的解析效率更高，因特网根域服务器维护着一个数据库，其中含有有效 IP 地址和能解析各个地

^① 由于 IP 地址将最高位八位组写在最前面，而域名则将最低位八位组写在最前面，所以在形成域名时需要将 IP 地址的各个八位组反向排列。

址组的域名服务器的有关信息。

23.19 对象类型与资源记录的内容

我们曾经提到，域名系统除了可以将主机名转换为 IP 地址，也可以用于将域名转换为邮件交换机的地址。域名系统非常通用，以至于可用于任何分级名字。例如，你可能决定将可用的计算服务与每个服务的名字到电话号码之间的映射关系一起存储，通过这个号码就可以找到相应服务的有关信息。或者，^①也可以将协议产品的名字与名字到提供产品的厂商地址之间的映射关系一起存储。

回想一下，系统通过在每个资源记录中包含一个类型（type）字段来适应各种各样的映射。在发送请求时，客户必须在查询中指明其类型^①。服务器在所有返回的资源记录中都会指明数据的类型。根据图 23.9 的表格，类型决定了资源记录中的内容。

Type	Meaning	Contents
A	Host Address	Canonical main name for an alias
CNAME	Canonical Name	Name of CPU and operating system
HINFO	CPU & OS	Information about a mailbox or mail list
MINFO	Mailbox Info	Mailbox preference and name of host that acts as mail exchanger for the domain
MX	Mail Exchanger	Preference and name of host that acts as mail exchanger for the domain
NS	Name Server	Name of authoritative server for domain
PTR	Pointer	Domain name (like a symbolic link)
SOA	Start of Authority	Multiple fields that specify which parts of the naming hierarchy a server implements
TXT	Arbitrary text	Uninterpreted string of ASCII text

图 23.9 域名系统资源记录类型的几个例子。目前已定义了 50 多种类型

大部分数据是 A 类型的，意味着它是由连接因特网的主机的名字和主机 IP 地址组成的。第二个最有用的域类型是 MX，它分配给电子邮件交换机作为名字，允许一个网点指定多台能够接收邮件的主机，用户发送电子邮件时使用的电子邮件地址格式为 user@domain-part。邮件系统利用域名系统解析 domain-part，其查询类型为 MX。域名系统返回一组资源记录，其中每个记录包含一个优先级字段和一台主机的域名。邮件系统按优先级从高到低遍历资源记录集（数字越小意味着优先级越高）。对于每个 MX 资源记录，邮件发送方提取主机的域名并再次使用一个 A 类型的查询，将此名字解析为一个 IP 地址。然后它试图与该主机联系并发送邮件。如果该主机不可用，邮件发送方将继续尝试联系记录表中的其他主机。

为了使查询的效率更高，服务器总是会在响应的附加信息区（ADDITIONAL INFORMATION SECTION）中返回它所知道的其他信息。如果是 MX 记录，域名服务器可以使用附加信息区为回答区中的域名返回 A 类型的资源记录。这样做就明显减少了邮件发送方发往域服务器的查询数量。

23.20 获得子域的管理权

在一个机构正式成为合法的第二级域管理机构以前，它必须允诺自己的域名服务器运作符合因特网标准。当然，域名服务器必须遵守指定报文格式和对请求的响应规则的协议标准。服务器也必须知道处理每个子域（如果存在）的服务器的地址以及至少一个根服务器的地址。

实际上，域系统比我们描绘的复杂得多。大多数情况下，一个物理服务器可以处理命名等级的多个部分。例如，在普度大学，有一个名字服务器既要管理第二级域 purdue.edu，又要负责地理域 laf.in.us。一个特定的名字服务器管理的名字子树形成了一个权限区（zone of authority）。另一个导致实际应用复杂性的原因在于，无论解析一个请求要花多长时间，服务器必须做到能够同时处理许多请求。通常，服务器支持并发任务，允许服务器在处理先到的请求的同时即可处理后到的请求。当服务器收到要求它将请求发给其他服务器的递归请求时，并发处理请求显得尤为重要。

由于因特网管理机构要求对每个域名服务器的信息进行备份，这就使服务器的实现变得更加复杂。信息必须出现在至少两个服务器上，并且这两个服务器必须运行在不同的机器上。实际应用的

^① 查询还可以指定几种附加类型（例如，有一种查询的类型是请求所有资源记录）。

要求是相当苛刻的：这些服务器一定不能具有哪怕是一个故障交汇点。避免同时出现故障就意味着两个名字服务器不能连接到同一个网络上，甚至不能有相同的电源来源。因此，为满足此需求，一个网点必须至少找到一个同意运行其名字服务器备份的其他网点。当然，无论在服务器树的任何位置，服务器都必须知道如何找到子域的主名字服务器和备份名字服务器，并且，如果主服务器不可用，它必须能够将查询改送到备份名字服务器。

23.21 动态 DNS 更新和通知

在第 19 章对 NAT 和第 20 章对 DHCP 的讨论中都曾提到过要与 DNS 进行交互。在 NAT 盒的情况下，需要从 ISP 那里获取一个动态地址，如果域名服务器和 NAT 系统相结合，那么域名服务器就只能位于 NAT 盒的后面。在 DHCP 的情况下，当主机获取一个动态地址时，该主机使用的 DNS 必须更新主机的当前地址。为了解决以上这些问题，并且为了多方共享管理事务（例如，允许多个登记机构共同管理顶级域），IETF 开发了一种称为**动态 DNS**（Dynamic DNS）的技术。

动态 DNS 有两方面的内容：**更新**（update）和**通知**（notification）。正如其名字所暗示的，动态 DNS 更新允许动态地对服务器存储的信息进行修改。因此，在 DHCP 服务器向一台主机分配一个 IP 地址时，它可以用动态更新机制告诉 DNS 服务器此次分配情况。通知报文解决的是传播变化信息的问题。实际上，因为 DNS 使用了备份服务器，所以在主服务器中所做的改变必须传播到每个备份服务器。当动态改变发生时，主服务器向所有备份服务器发送一个通知，让每个备份都请求对管理区的信息的更新。因为 DNS 通知避免了发送不必要的副本，所以与只使用一个小的超时值来提高更新频率相比，DNS 通知占用更少的带宽。

23.22 DNS 安全扩展

由于域名系统是因特网基础设施中最重要的环节之一，因此域名系统经常作为应当受到保护的关键部位而被人们谈及。实际上，如果主机的 DNS 查询结果得到的是不正确的回答，那么应用软件或用户就会受骗上当，相信一个假冒的万维网站或泄漏机密信息。为了有助于保护 DNS，IETF 已研发出一种称为**DNS 安全保证**（DNS Security，简称 DNSSEC）的技术。

DNSSEC 提供的主要服务包括对报文起源的**鉴别**（authentication）以及数据的**完整性**（integrity）支持。也就是说，如果主机使用了 DNSSEC，它就能验证 DNS 报文是否确实由具有权限的 DNS 服务器（即负责对查询中所涉及的名字进行处理的服务器）发送来，并且到达的报文中的数据没有被改变过。另外，DNSSEC 还能鉴别否定回答：主机能够获取一个经过鉴别的报文，这个报文的内容是说某个特定的域不存在。

尽管 DNSSEC 提供了鉴别和数据的完整性，它还是不能解决所有的问题。实际上，DNSSEC 没有提供机密性，也不能抵御“拒绝服务”的攻击。前者的含义是，即使主机使用了 DNSSEC，一个在网上偷窥的非内部人士仍能掌握主机所查找的名字（即偷窥者也许能猜到为什么有一桩生意正在联系中）。对“拒绝服务”攻击的无能为力，意味着即使主机和服务器都使用了 DNSSEC，还是不能保证来往于它们之间的报文都能被对方收到。

为了提供鉴别和数据完整性，DNSSEC 使用了数字签名机制，即除了被请求的信息之外，来自 DNSSEC 服务器的回答中还包含了一个数字签名，以允许接收方验证报文内容是否被改变过。DNSSEC 的一个最引人注意的特点在于数字签名机制的管理方式。与许多安全机制一样，DNSSEC 机制使用了**公钥加密**（public key encryption）技术。出乎人们意料的是，为了分发公钥，DNSSEC 使用的还是 DNS。也就是说，服务器包含了等级树管理区以下各部分的公钥（例如，负责.com 的服务器含有用于 example.com 的公钥。为了保证整个系统的安全性，等级树最顶层的公钥（即根服务器的密钥）必须手工配置到解析器中。

23.23 小结

等级命名系统允许授权名字管理机构，使系统可适应任意大的名字集合，而不会使中心网点因管理任务过重而不堪负荷。名字解析独立于管理机构的授权，即使管理机构的授权总是从上到下进行的，还是可以创建从本地服务器开始解析的高效的等级命名系统。

我们已考察了因特网域名系统（DNS），知道它提供了一种等级命名方案。DNS 使用分布式查询，其中域名服务器将每个域名映射成一个 IP 地址或邮件交换机的地址。客户首先试图在本地解析名字，当本地服务器不能解析该名字时，客户必须选择是通过名字服务器树进行迭代解析，还是请求由本地服务器递归解析。最后，我们看到了域名系统可支持多种绑定，包括从 IP 地址到高级名字的绑定。

DNSSEC 提供了一种可用于保证 DNS 安全的机制。它会鉴别收到的响应并保证回答的完整性。DNSSEC 使用公钥加密技术，并设法利用 DNS 来分发公钥集。

23.24 深入研究

有许多 RFC 都以 DNS 及其扩展建议为中心。Mockapetris [RFC 1034]讨论一般的因特网域名系统，并给出了总体原则，同时 Mockapetris [RFC 1035]提供域名系统所用的协议标准。Mockapetris [RFC 1101]讨论如何使用域名系统对网络名进行编码，并提出许多对其他映射有用的扩展。Postel and Reynolds [RFC 920]陈述了因特网域名服务器必须满足的要求。Stahl [RFC 1032]给出建立一个域的管理原则，Lottor [RFC 1033]提供了运行域名服务器的原则，而 Eastlake et. al. [RFC 2929]讨论 IANA 的有关内容。Vixie [RFC 1996]定义了 DNS 通知，而 Vixie et. al. [RFC 2136]则定义了动态更新。Eastlake [RFC 2535]给出了安全扩展。Klensin [RFC 2821]说明域命名与电子邮件地址之间的关系。Wellington [RFC 3008]建议了 DNSSEC，并且通过 RFC 3658, 3755, 3757, 3845 不断更新其细节内容。

23.25 习题

1. 机器名不应该在编译期间绑定到操作系统中，试解释原因。
2. 你是愿意让机器从一个远程文件获取名字，还是愿意从名字服务器获取名字？为什么？
3. 为什么每个名字服务器需要知道其父服务器的 IP 地址，而不是其父服务器的域名？
4. 设计一个允许改变命名等级的命名方案。例如，考虑两个大公司，它们各自具有独立的命名等级，并且假设这两个公司合并了，你能设法使原有名字仍正常工作吗？
5. 阅读标准，找出域名系统如何使用 SOA 记录。
6. 因特网域名系统也可以容纳邮箱名，考察它是如何用于邮箱名的。
7. 标准建议当程序需要查找 IP 地址对应的域名时，应首先给本地服务器发送一个反向查询，并仅在本地查询失败时才使用 in-addr.arpa 域。为什么？
8. 你会如何使域命名方案适应缩写？例如，有两个网点，一个登记在.edu 下，另一个登记在顶层服务器下。分别解释每个网点如何处理各种类型的缩写。
9. 找到对域名系统的官方描述，并构造一个客户程序。尝试查询名字 merlin.cs.purdue.edu。
10. 扩充上面的习题，使之包括一个指针查询。尝试查找地址 128.10.2.3 的域名。
11. 找到 dig 程序，并用它来查找前两个习题中的名字。
12. 如果扩充域名的语法，使域名在顶级域标号后包含一个点，名字和缩写就没有二义性了。这种做法的优点和缺点是什么？

-
13. 阅读有关域名系统的 RFC。DNS 服务器在一个资源记录的 TIME-TO-LIVE 字段中可存放的最大值和最小值是多少？
 14. 域名系统是否允许部分匹配查询（即，名字的某部分用通配符表示）？为什么？
 15. 普度大学计算机科学系选择将类型 A 的资源记录项放入它的域名服务器：

localhost.cs.purdue.edu 127.0.0.1

试解释：如果一个远程网点尝试 ping 一个域名为 localhost.cs.purdue.edu 的机器，会发生什么现象？

第24章 远程登录和桌面（TELNET 和 SSH）

24.1 引言

本章以及后续五章的内容将研究应用层的因特网服务和支持这些服务的协议，继续我们对互联网技术的探讨。这些服务形成了 TCP/IP 整体中的一部分。它们决定了互联网用户将如何感知互联网的存在，并充分展示出这一技术的强大威力。我们将会看到应用层服务提供了越来越多的通信功能，并允许用户和程序与远程机器上的自动服务及远程用户进行交互。我们将看到高层协议是以应用程序实现的，并了解它们如何依赖前面章节所描述的网络层服务。本章首先探讨远程访问。

24.2 远程交互式计算

我们已经知道了客户—服务器模型是如何提供一些特殊计算服务的，如用于多台机器的日期—时间服务。而可靠的数据流协议，如 TCP，也允许交互式使用远程机器。例如，构想一个提供远程文字处理服务的服务器。为了实现这种服务，我们需要一个接收请求的服务器，以及一个发出请求的客户。为了调用远程文字处理器，用户需要运行一个客户程序，由客户建立到服务器的 TCP 连接，然后开始向服务器发送键入的信息，并显示从服务器返回的输出。

如何将我们所构想的远程文字处理服务进一步普遍化？如果每个计算服务都要使用一个服务器，那么带来的问题就是机器会很快被服务器进程挤垮。通过构建一种机制，以允许用户先在远程机器上建立一个会话，然后再运行任意应用程序，就可以消除大多数专用服务器，并提供更广泛的通用性。在基于文本的计算机系统中，这样的会话称为**远程登录**（remote login）功能。在基于“视窗”的操作系统中，此类会话称为**远程桌面**（remote desktop）功能。

当然，提供远程登录并非易事。计算机系统在设计时，处理键盘输入、追踪鼠标轨迹以及控制显示等都是在本地完成的。例如，操作系统可以为某些键入分配特殊的含义（如 Windows 机器的 CONTROL-ALT-DELETE 或 Linux 系统的 CONTROL-C）。如果客户把所有这些键入都传到远程服务器，那么它可能无法终止或控制本地客户进程。

24.3 TELNET 协议

TCP/IP 协议族包括一个称为 TELNET 的简单文本远程终端协议，允许用户通过互联网登录到另一台计算机上。TELNET 建立一个 TCP 连接，然后将用户从键盘键入的信息直接传递到远程计算机，就好像用户是用连在远程机器上的键盘进行操作一样。TELNET 还将远程机器的文本输出送回到用户屏幕上。这种服务称为**透明**（transparent）服务，因为它给人的感觉好像用户的键盘和显示器是直接连在远程机器上的。

TELNET 提供三种基本服务。第一，它定义了一个**网络虚拟终端**（Network Virtual Terminal，简称 NVT），为远程系统提供标准的接口。客户程序不必详细了解所有可能的远程系统，它们在构造时只需要使用标准接口。第二，TELNET 包括一种允许客户和服务器协商选项的机制，同时还提供了一组标准选项（例如，其中有一个选项用于控制通过连接传输的数据是使用标准 7 位 ASCII 字符集还是 8 位字符集）。最后，TELNET 对称处理连接的两端。实际上，TELNET 既没有强制要求客户从键盘输入，也没有要求客户必须在屏幕上显示输出。因此，TELNET 允许任意程序成为客

户。此外，任何一端都可以发起选项协商。

图 24.1 说明了如何使用应用程序实现 TELNET 客户和服务器。

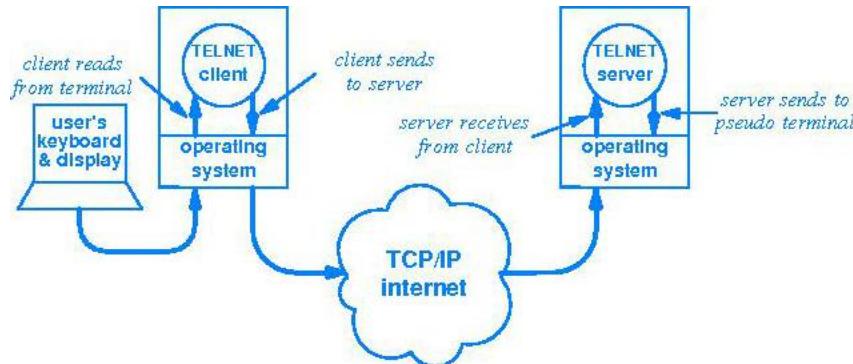


图 24.1 在 TELNET 远程终端会话中，数据从用户键盘传到远程操作系统时的数据通路

如图 24.1 所示，当用户调用 TELNET 时，用户机器上的应用程序就成为了客户。客户建立一条到服务器的 TCP 连接，并将在此连接上进行通信。一旦建立了连接，客户就从用户键盘接受键入并送到服务器，同时它接受服务器发送回来的字符并显示在用户屏幕上。客户还要检查用户是否键入了用来控制客户的**转义符** (escape character)。服务器接受来自客户的 TCP 连接，然后在 TCP 连接和本地操作系统之间对数据进行中继。

实际上，服务器比图中所描绘的复杂得多，因为它必须处理多个并发连接。通常，一个主服务器进程等待新的连接，并为处理每个连接创建一个新的从进程副本。因此，图 24.1 所示的“TELNET 服务器”只代表处理某一个连接的从进程。图中并未画出监听新请求的主服务器，也没有显示处理其他请求的从进程。

我们用术语**伪终端** (pseudo terminal)^①来描述操作系统的入口点，它允许 TELNET 服务器之类的运行程序向操作系统传输字符，并且使字符似乎来自本地键盘。除非操作系统提供了这样的设施，否则不可能构造一个 TELNET 服务器。如果系统支持伪终端抽象，则可以用应用程序实现 TELNET 服务器。每个从服务器将 TCP 流从一个客户连接到一个特定的伪终端。

将 TELNET 服务器安排为应用层程序既有优点也有缺点。最显著的优点是，与把代码嵌入操作系统相比，服务器的修改和控制更容易。最大的缺点是效率不高。键入的所有信息从用户键盘通过操作系统传到客户程序，再从客户程序返回操作系统，通过底层互联网传到服务器机器。到达目的主机后，数据必须通过操作系统向上传给服务器应用程序，并且再从服务器应用程序返回到服务器操作系统的伪终端入口点。最后，远程操作系统将字符传给用户正在运行的应用程序。同时，输出（如果选择了字符回送选项，还包括远程字符回送）按照同样的路径从服务器回送到客户。

了解操作系统的读者会赞成图 24.1 所示的实现，其中每次键入都需要计算机多次切换进程环境。在大多数系统中，由于服务器所在机器上的操作系统必须从伪终端把字符传回给另一个应用程序（如命令解释器），还需要额外的进程环境切换。虽然进程环境切换的开销很昂贵，但由于用户键入速率不会很高，所以这种机制还是比较实际的。

24.4 适应异构性

为了使 TELNET 有可能在多个系统之间进行互操作，它必须适应异构计算机和操作系统的具

^① UNIX 称系统入口点为伪 tty，因为面向字符的设备称为 tty。

体细节。例如，一些系统需要每行文本用 ASCII 回车控制符（CR）结束，另外一些系统则需要使用 ASCII 换行符（LF），而其他的系统需要用两个字符的回车—换行（CR-LF）序列^①。另外，大多数交互系统为用户提供了通过一个输入键来中断程序运行的方法。但是，中断程序使用的特殊键在各个系统中可能不同（如一些系统使用 Control-C，而其他一些系统使用 ESCAPE）。

为了适应异构环境，TELNET 定义了在因特网上传输数据和命令序列的方式，此定义称为网络虚拟终端（NVT）。如图 24.2 所示，客户软件把来自用户终端的键入和命令序列转换为 NVT 格式，并发送到服务器。服务器软件将收到的数据和命令从 NVT 格式转换为远程系统要求的格式。对于返回的数据，远程服务器将数据从远程机器的格式转换为 NVT 格式，并且本地客户又将数据从 NVT 格式转换为本地机器的格式。

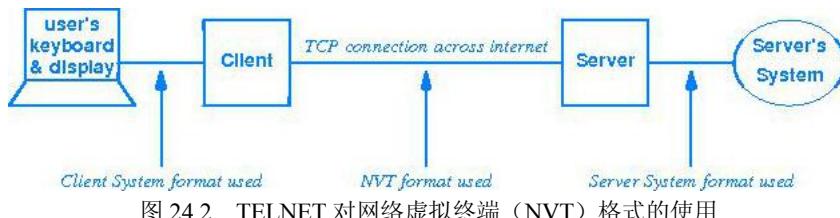


图 24.2 TELNET 对网络虚拟终端（NVT）格式的使用

NVT 格式的定义相当直观。所有通信涉及的都是 8 位字节。从启动开始，NVT 使用标准 7 位 US ASCII 字符表示数据，而高位置 1 则保留用于命令序列。所有可打印字符值的含义与标准 US ASCII 字符集一样。NVT 标准对控制字符的定义如图 24.3 所示^②。

ASCII Control Code	Decimal Value	Assigned Meaning
NUL	0	No operation (has no effect on output)
BEL	7	Sound audible/visible signal (no motion)
BS	8	Move left one character position
HT	9	Move right to the next horizontal tab stop
LF	10	Move down (vertically) to the next line
VT	11	Move down to the next vertical tab stop
FF	12	Move to the top of the next page
CR	13	Move to the left margin on the current line
other control		No operation (has no effect on output)

图 24.3 US ASCII 控制字符的 TELNET NVT 解释。TELNET 没有指定制表站的位置

除了图 24.3 中的控制字符解释，NVT 还规定标准行结束为两字符的 CR-LF 序列。当用户按下本地终端上对应于行结束的一个键（如 ENTER 或 RETURN 键）时，TELNET 客户必须将它映射为 CR-LF，以用于传输。TELNET 服务器将 CR-LF 转换为远程机器相应的行结束字符序列。

24.5 传递用于控制远端的命令

我们已提到过，大多数系统提供了允许用户终止一个正在运行的程序的机制。通常，本地操作系统将该机制绑定为某个键或键入序列。例如，除非用户指定其他方式，否则许多 UNIX 系统保留了将 CONTROL-C 产生的字符作为中断键的用法。按下 CONTROL-C 将使 UNIX 终止一个运行中的程序，程序并不接受把 CONTROL-C 作为输入。系统可能还保留了其他字符或字符序列，用于其他控制功能。

^① 回车—换行序列通常写为 crlf，其读音是 curl-if。

^② NVT 对控制字符的解释跟在常用 ASCII 解释之后。

TELNET NVT 通过定义如何把控制功能从客户传输到服务器来适应这些控制功能。从概念上讲，我们认为 NVT 可以接受超过 128 个字符的键盘输入。我们假定用户键盘上有一些额外（虚拟）的键，它们所对应的是一些用于控制进程的典型功能。例如，NVT 定义了一个概念上的“中断”键，用于请求程序终止。图 24.4 列出了 NVT 允许的控制功能。

Signal	Meaning
IP	Interrupt Process (terminate running program)
AO	Abort Output (discard any buffered output)
AYT	Are You There (test if server is responding)
EC	Erase Character (delete the previous character)
EL	Erase Line (delete the entire current line)
SYNCH	Synchronize (clear data path until TCP urgent data point, but do interpret commands)
BRK	Break (break key or attention signal)

图 24.4 TELNET NVT 可以识别的控制功能。从概念上讲，客户除了从用户那里接受数据以外，还接受控制功能，它们被传输到服务器所在的系统上并在该系统上得到解释

为了在 TCP 连接上传递控制功能，TELNET 使用一个**转义序列** (escape sequence)，对控制功能进行编码。转义序列用一个保留八位组来指示下一八位组将是控制码。在 TELNET 中，说明转义序列开始的那个保留八位组称为**按命令解释** (Interpret As Command，简称 IAC) 八位组。图 24.5 列出了可能的命令和每个命令使用的十进制编码。

Command	Decimal Encoding	Meaning
IAC	255	Interpret next octet as command (when the IAC octet appears as data, the sender doubles it and sends the 2-octet sequence IAC-IAC)
DON'T	254	Denial of request to perform specified option
DO	253	Approval to allow specified option
WONT	252	Refusal to perform specified option
WILL	251	Agreement to perform specified option
SB	250	Start of option subnegotiation
GA	249	The “go ahead” signal
EL	248	The “erase line” signal
EC	247	The “erase character” signal
AYT	246	The “are you there” signal
AO	245	The “abort output” signal
IP	244	The “interrupt process” signal
BRK	243	The “break” signal
DMARK	242	The data stream portion of a SYNCH (always accompanied by TCP Urgent notification)
NOP	241	No operation
SE	240	End of option subnegotiation
EOR	239	End of record

图 24.5 TELNET 的命令和对应的编码。编码只有在前面出现一个 IAC 字符时才有意义。当 IAC 出现在数据中时需发送两次

如图 24.5 所示，由 NVT 键盘上的概念键产生的每个信号都对应一个命令。例如，要请求服务器中断运行程序，客户必须发送两个八位组的序列 IAC IP (255 后面跟着 244)。还有一些命令允许

客户和服务器协商所要使用的选项，以及使通信同步。

24.6 强制服务器读取一个控制功能

与普通数据一起发送控制功能，并不能保证总是能够得到所希望的结果。要知道原因，可以考虑用户向服务器发送“**中断进程**”(interrupt process)控制功能的情况。通常，只有当远程机器上的运行程序出现故障，或者用户希望服务器终止程序时，才需要使用这种控制。然而遗憾的是，如果服务器网点中的应用程序停止读取输入，那么操作系统的缓冲区最终会被填满，服务器也无法将数据写到伪终端。服务器停止从 TCP 连接读取数据，从而又使 TCP 连接的缓冲区变满，于是服务器所在机器上的 TCP 开始将接收窗口大小通告为零，从而阻止了数据从此连接流入。

如果用户在缓冲区已满时生成一个中断控制功能，这个控制功能永远也无法到达服务器。也就是说，客户可以形成命令序列 IAC IP，并将其写入 TCP 连接，但由于 TCP 已停止把数据发送到服务器所在的机器，服务器将不可能读到这个控制序列。要点如下：

TELNET 不能只依靠客户和服务器之间的常规数据流来传输控制序列，因为需要被控制的出错应用程序可能无意中阻塞了数据流。

为了解决此问题，TELNET 必须使用**带外**(out of band)信号。TCP 用的是**紧急数据**(urgent data)机制来实现带外数据信令。只要 TELNET 在数据流中放入了控制功能，它就会发送一个 SYNCH 命令。然后 TELNET 再附加一个称为**数据标记**(data mark)的保留八位组，它会导致 TCP 在发送报文段时设置紧急数据位，以此通知服务器。携带紧急数据的报文段将绕过流量控制直达服务器。作为对紧急信令的响应，服务器将读取并抛弃所有数据，直到找到该数据标记。服务器在遇到数据标记后将返回正常处理过程。

24.7 TELNET 选项

我们对 TELNET 的简单描述中省略了一个最复杂的方面：选项。在 TELNET 中，选项是可协商的，这样就使客户和服务器可以重新配置它们之间的连接。例如，我们曾说过通常数据流传递的是 7 位的数据，将八位组中的第 8 位置 1 就可以用于传递类似中断进程命令的控制信息。然而，TELNET 也提供一个选项，允许客户和服务器传递 8 位的数据（当传递 8 位的数据时，如果保留八位组 IAC 出现在数据中，仍然需要变成两个）。在开始这种传输之前，客户和服务器必须进行协商，并且双方都必须同意传递 8 位的数据。

TELNET 选项的范围很广，其中一些选项对其功能的扩展是举足轻重的，而另一些选项只涉及一些微小细节。例如，原协议的设计用于半双工环境，其中通信一方必须在希望发送更多数据之前，先通知另一端“继续”。有一个选项可控制 TELNET 是在半双工还是全双工模式下工作。另一个选项允许远程机器上的服务器决定用户终端类型。终端类型对于产生光标定位序列的软件（如远程机器上运行的全屏幕编辑器）很重要。图 24.6 列出了 TELNET 选项的几个例子。

Name	Code	RFC	Meaning
Transmit Binary	0	856	Change transmission to 8-bit binary
Echo	1	857	Allow one side to echo data it receives
Suppress-GA	3	858	Suppress (no longer send) Go-ahead signal after data
Status	5	859	Request for status of a TELNET option from remote site
Timing-Mark	6	860	Request timing mark be inserted in return stream to synchronize two ends of a connection
Terminal-Type	24	884	Exchange information about the make and model of a terminal being used (allows programs to tailor output like cursor positioning sequences for the user's terminal)
End-of-Record	25	885	Terminate data sent with EOR code
Linemode	34	1116	Use local editing and send complete lines instead of individual characters

图 24.6 大量 TELNET 选项的几个示例

24.8 TELNET 选项的协商

TELNET 协商选项的方式很有意思。由于有时必须由服务器来初始化某个选项才有意义，因而协议被设计成允许任何一端发出申请。这样，从选项过程来看，这个协议可以说是对称的 (symmetric)。接收的一方在响应时可以肯定地接受请求或拒绝请求。在 TELNET 术语中，请求是 WILL X，意思是“你是否同意我使用选项 X”；而响应则是 DO X 或 DON'T X，意思是“我同意你使用选项 X”或“我不同意你使用选项 X”。由于 DO X 要求接收的一方开始使用选项 X，而 WILL X 或 WON'T X 意味着“我将开始使用选项 X”或“我不会使用选项 X”，从而出现了对称现象^①。

我们可以总结如下：

TELNET 使用一种对称选项协商机制，允许客户和服务器重新配置参数，以控制它们之间的交互。由于所有 TELNET 软件都了解基本的 NVT 协议，即使客户和服务器中有一方理解选项而另一方不理解，它们之间也能互操作。

24.9 安全外壳

除了 TELNET 以外，一种称为**安全外壳** (Secure Shell，简称 SSH) 的技术也越来越受欢迎。与 TELNET 一样，SSH 通过 TCP 建立与远程计算机之间的连接，并允许远程登录。但是，与 TELNET 相比，SSH 提供了两个意义重大的增强性能。首先，SSH 提供安全的通信。其次，SSH 向用户提供了一种能力，允许用户经过登录时所用的同一条连接来执行额外的独立数据传送。尽管 SSH 有着商业性质的出身，但目前它已经是 IETF 的建议标准。

SSH 提供了三种机制，以构成它所提供的服务的基础。

- **一个运输层协议**，提供了服务器鉴别、数据保密性以及具有良好前向保密性的数据完整性（即，如果某个会话中的密钥有泄密的危险，就是知道了这个密钥也不会影响此前会话的安全性）。因此，即使第三方得到了通过 SSH 连接发送的分组副本，发送的数据仍然是保

^① 为了避免双方都将对方的确认看成是请求，从而可能导致循环，协议规定，对于已经在用的选项的请求，不给出确认。

密的。

- **一个用户鉴别协议**, 用于服务器鉴别用户。因此服务器可以准确掌握是哪一个用户在试图建立连接。
- **一个连接协议**, 可以在一条底层 SSH 连接上复用多条逻辑通信通道。

为了提供最大的灵活性, SSH 并没有强制要求对通信的所有层面都使用某个具体的保密协议。事实上, SSH 的服务器鉴别使用了公钥加密, 而对于用户鉴别, 允许使用迭代密码或者公钥加密。另外, 对于鉴别阶段之后的通信任务, SSH 允许在几个共享的加密协议中进行选择。这个密钥(也称为会话密钥)由客户和服务器在应用程序数据开始传送之前先协商确定。因此, 除了在连接建立之初需要选择会话密钥之外, SSH 能够避免公钥加密的额外开销。

注意, 虽然我们认为 SSH 是一种安全远程登录服务, 但 SSH 提供的是一条通用的安全连接以及在一条连接上复用多个服务的设施。因此, 远程登录仅仅是一个在 SSH 连接上复用的服务。例如, 在下一章中将会看到有可能通过一条安全的 SSH 连接实现文件传送的复用。

端口转发。SSH 复用机制最强大的一面称为**端口转发**(Port Forwarding)。共有两种类型: 本地的和远程的。转发的基本思想很简单: SSH 连接可以用来作为两台计算机之间的安全隧道, 并且用户可以对 SSH 进行配置, 使其自动通过隧道衔接一个入口 TCP 连接和一条新连接。如果连接到隧道的客户端时出现了衔接处理, 称为**本地**(local) 端口转发。如果连接到隧道的服务器端时出现了衔接处理, 称为**远程**(remote) 端口转发。例如, 假设网点 1 的计算机 C₁(客户端) 建立了一条到达网点 2 的计算机 C₂的一条 SSH 连接, 并指明到达 C₁ 端口 2000 的入口 TCP 连接都会自动通过隧道转发到 C₂, 然后再与到达计算机 C₃(也在网点 2) 80 端口的连接互相衔接。一旦 SSH 建立好连接并且启用转发功能, 只要网点 1 中有计算机建立一条到达 C₁ 端口 2000 的连接, SSH 将自动设法将数据转发到 C₃ 的 80 端口。并且, 通信的两端看起来都是本地的: 网点 1 中的客户只看见到达本地计算机 C₁ 的 TCP 连接, 而 C₃ 上的服务器只看见来自 C₂ 的 TCP 连接。

因为端口转发允许任意一个应用程序都可以在两个网点之间传递数据, 而不是用独立的客户和服务器软件为每个应用程序实现加密, 所以 SSH 可以通过配置, 允许一个用户的所有应用程序都使用一条 SSH 连接进行通信, 这就是端口转发的主要优点。

24.10 其他远程访问技术

尽管 TELNET 和 SSH 是 TCP/IP 协议中的一部分, 人们还设计了其他一些远程访问协议。有几个例子体现出了这些协议的多样性。

rlogin: 最早出现的可以代替 TELNET 的是 rlogin, 它是作为 BSD UNIX 操作系统的一部分而设计的。Rlogin 最先提出了**可信主机**(trusted hosts)的思想, 这个思想被后来的很多服务所采纳, 包括 SSH。从本质上讲, 这种机制允许系统管理员选择一组机器, 在这些机器上共享登录名和文件访问保护, 并为用户登录建立等价关系。用户通过管理基于远程主机和远程用户名的远程登录, 来控制对其用户账号的访问。因此, 用户可能在一台机器上用登录名 X 而在另一台机器上用登录名 Y, 而且还可能不需要每次都键入口令, 就从一台机器远程登录到另一台机器上^①。

虚拟网络计算(Virtual Network Computing, 简称 VNC)。与 TELNET, rlogin 或 SSH 不同, VNC 提供的是**远程桌面**(remote desktop)能力, 而不是文本接口。也就是说, VNC 允许一台计算机上的用户查看另一台计算机桌面上完全相同的副本, 并使用键盘和鼠标实现与远程计算机的交互, 即用户可以看到各个窗体、图标以及其他图形。更重要的是, VNC 软件可以跨多个平台运行。其结果是, 即使服务器运行在使用 Windows 操作系统的计算机上, 客户也可以在使用 Linux 操作系统的计算机上运行。

^① 使用 IP 地址和用户名进行鉴别已经不再被认为足够安全, 还要求有其他附加的机制, 如加密。

远程桌面协议 (Remote Desktop Protocol, 简称 RDP)。微软公司已经为自己的操作系统定义了一个远程桌面协议。与其他远程桌面系统一样, RDP 允许远程计算机查看确切的桌面副本以及运行应用程序。RDP 还可以跨平台使用 (如目前存在一种 Linux 客户), 这与其他远程桌面技术也一样。

24.11 小结

与 TCP/IP 相关的大量功能很多都来自各种各样的应用层协议。这些服务通常遵循客户—服务器模型, 其中服务器在一个熟知端口上运行, 以便客户知道如何与之联系。

我们概述了一个远程访问系统, 即 TELNET, 它是 TCP/IP 标准中的一部分。TELNET 使用 TCP 连接, 允许客户把控制命令 (如中断进程) 和数据传递给服务器, 也允许客户和服务器协商选项。

安全外壳 SSH 提供了一种经过鉴别的、保密的远程登录功能。因为 SSH 远程登录功能建立在通用的、安全的复用机制上, 所以其他服务可以通过一条 SSH 连接进行转发。端口转发机制允许用户对 SSH 进行配置, 使任意应用程序能够共享一条 SSH 连接。

人们还设计了一些使用 TCP/IP 协议的远程访问系统, 用于在客户和提供远程访问的服务器之间提供连通性, 其中包括 rlogin, VNC 和 RDP 等一些实例。VNC 和 RDP 提供的是远程桌面能力, 而非文本交互。

24.12 深入研究

目前已经建议的高层协议有很多。Edge [1979] 将端到端的协议和逐跳 (hop-by-hop) 的方法进行了比较。Saltzer, Reed, and Clark [1984] 讨论了如何让最高层的协议来执行端到端的确认和差错检测。

Postel [RFC 854] 含有 TELNET 远程登录规范。在此之前, 有 30 多份 RFC 都在讨论 TELNET 选项、弱点、试验和更改建议, 包括含有早期标准的 Postel [RFC 764]。Postel and Reynolds [RFC 855] 给出了选项规范并探讨了子协商。在 RFC 856~861, 884, 885, 1041, 1091, 1096, 1097, 1184, 1372, 1416 和 1572 中可以找到一长串的选项列表。

有关 SSH 的信息可以查阅 IETF 文档库:

<http://www.ietf.org/html.charters/secsh-charter.html>

维护 VNC 的英国公司 RealVNC 的信息可从以下网站找到:

<http://www.realvnc.com>

24.13 习题

1. 阅读有关 TELNET 和 SSH 的资料。它们在功能上的明显区别是什么?
2. 什么是远程过程调用?
3. Folklore 说操作系统总是在更新淘汰中, 而协议却是永远存在的。观察你所处的本地计算网点, 检验这个说法, 看看操作系统和通信协议哪一个变化更频繁。
4. 构造一个 TELNET 客户软件。
5. 使用 TELNET 客户将你的键盘和显示器连接到本地系统中用于 echo 或 chargen 的 TCP 协议端口, 看看会发生什么情况。
6. TELNET 使用 TCP **紧急数据** (urgent data) 机制迫使远程操作系统对控制功能做出快速反应。阅读标准, 找出远程服务器在扫描输入流时认可哪些命令?

-
7. 如果对方总是确认请求，对称的 DO/DON'T 和 WILL/WON'T 选项协商是怎样产生响应的无尽循环的？
 8. RFC 854 (TELNET 协议规范) 的文本文件只有 854 行。你认为这有什么重要意义吗？
 9. 下载 VNC 软件，并在异构的操作系统中分别运行客户和服务器。你能不能找到什么功能是远程用户不能使用的？
 10. 能不能用 SSH 完成 rlogin 提供的所有功能？试解释原因。
 11. 如果你家的网点提供了 SSH 访问，设计一个端口转发机制，以允许你使用 SSH 隧道从远程网点（如当你在旅行时）阅读和发送电子邮件报文。提示：假设你的本地电子邮件软件访问的是 IMAP 和 SMTP 服务器。

第25章 文件传送与存取（FTP, TFTP 和 NFS）

25.1 引言

本章将继续探讨应用协议，研究作为 TCP/IP 协议族一部分的文件存取与传送协议。本章将描述这些协议的设计及能力。我们将学习基于 TCP（参见第 12 章）和 TELNET（参见第 24 章）的最常用的文件传送协议。

25.2 远程文件存取、传送和存储网络

许多网络系统向计算机提供存取远程机器上的文件的能力。设计人员已开发出多种不同的远程文件存取方法，每种方法针对一组特定目标进行优化。例如，一些设计者使用远程文件存取来降低总体开销。在这种体系中有一个中央**文件服务器**（file server），为那些有很少或根本没有本地硬盘的廉价计算机提供辅存。这些无盘机器并不一定是常规的计算机，它们可以是用于清点库存之类日常工作的便携式手持扫描设备，并且这些机器可以通过无线网络与文件服务器进行通信。

还有一些设计使用远程的存储设备来保管归档数据。在这些设计中，用户拥有带本地存储设备的常规计算机，并以正常方式操作。每台计算机都要定期通过网络把文件副本（或整个盘的副本）发送到归档设备并存储，以避免数据的偶然丢失。

最后，有些设计强调了多个程序、多个用户或多个网点之间的数据共享能力。例如，某公司有一个在线的未结算定单数据库，可以让所有工作组共享。这种功能称为**存储区域网络**（Storage Area Network，简称 SAN）系统。

25.3 在线共享存取

文件共享有两种不同形式：**在线存取**（on-line access）和**完整文件复制**（whole-file copying）。在线存取意味着允许多个程序同时存取一个文件。对文件所做的改动即时生效，并对存取该文件的所有程序都有效。完整文件复制意味着当程序想存取一个文件时，必须获得一个本地的副本。复制通常用于只读数据，但如果必须修改文件，程序则必须在本地修改副本，并将修改后的文件传回原网点。

许多用户认为在线数据共享功能只能由分布式数据库系统提供，它允许用户（客户）从远程网点对数据库进行事务处理（即存取或修改数据库中的项）。然而，文件共享机制的存在更为普遍。例如，如果操作系统允许文件通过一个共享的在线存取协议被存取。那么远程操作系统就能直接运行存取该文件的软件，而不需要用户事先调用一个类似数据库客户端的特殊应用程序。事实上，用户可以运行任意应用程序并存取远程文件，就好像它们是本地文件一样。我们可以说，远程文件与本地文件系统**一体化**（integrated）了，并且整个文件系统提供对共享文件的**透明存取**（transparent access）。

透明存取的优点很明显：不需要对应用程序做任何可见的改动即可存取远程文件。用户对本地的和远程的文件一样进行存取，允许它们对共享数据进行任意计算。缺点相对而言不太明显。用户可能没有想到这样做会带来什么结果。例如，考虑一个使用本地和远程文件的应用程序。如果网络或远程机器出现故障，即使用户的机器仍在运行，应用程序也不能正常工作。即使远程机器在工作，

它可能超负载或者网络可能拥塞，从而使应用程序运行缓慢，或引起通信协议向用户报告出现超时，这是用户不希望看到的结果。这个应用程序似乎不太可靠。

尽管有优点，但要实现一体化的、透明的文件存取仍然比较困难。在异构环境中，这台机器上的可用文件名可能无法映射到那台机器的文件名空间。类似地，远程文件存取机制必须处理文件的所有者、存取授权和存取保护等在不同系统上各自为政的概念。最后，由于文件的表示方法和允许的操作在不同机器间各不相同，在所有文件上实现全部操作可能很难或不太可能。

25.4 通过文件传送的共享

与一体化的、透明的在线存取不同的另一种方案是**文件传送机制** (file transfer mechanism)。用传送机制存取远程数据的过程分为两个步骤：用户首先要获得一个文件的本地副本，然后对副本进行操作。大多数传送机制操作的并不是本地文件系统（即未实现一体化），用户必须调用一个专用的客户程序来传送文件。客户与远程机器上的服务器联系并请求一个文件的副本。一旦传送完成，用户终止客户并使用本地机器上的应用程序读取或修改本地副本。完整文件复制的一个优点是操作的高效性，一旦应用程序获得远程文件的本地副本，就可以高效地处理文件副本。

与在线共享一样，异构机器间的完整文件传送比较困难。客户和服务器必须就存取授权、文件所有者和存取保护、数据格式等方面达成一致。数据格式尤其重要，因为在传送过程中可能会对文件有稍许改变（例如，行结束符可能会改变）。其结果是，如果文件从机器 A 传送到机器 B，然后再传回机器 A，这时得到的文件与最初的文件可能有所不同。

25.5 主要的 TCP/IP 文件传送协议 FTP

文件传送是最常用的 TCP/IP 应用之一，而且由它带来的通信量是因特网通信量中不可忽视的一部分。在 TCP/IP 实现之前，就已经有了用于 ARPANET 的标准文件传送协议。这些早期的文件传送软件版本演化成了目前使用的标准，称为**文件传送协议** (File Transfer Protocol，简称 FTP)。

25.6 FTP 的特点

既然已经有了 TCP 之类的可靠的端到端传送协议，文件传送看起来似乎很简单。但是，正如前几节中提到的，异构机器之间的存取授权、命名和表示方法等细节问题，使文件传送协议变得复杂。另外，FTP 还提供了许多超出自身传送功能的设施。

- **交互存取**。虽然 FTP 是为程序使用而设计的，但是大多数 FTP 的实现另外提供了一个交互接口，允许用户与远程服务器之间的交互。
- **格式规范**。FTP 允许客户指明存储数据的类型和表示方法。例如，用户可指明一个文件包含的是文本还是二进制数据，以及文本文件是使用 ASCII 字符集还是 EBCDIC 字符集。
- **存取授权控制**。FTP 要求客户在请求文件传送前把登录名和口令发送给服务器，以获得存取许可。服务器将拒绝不能提供有效登录名和口令的客户的存取。

25.7 FTP 进程模型

与其他服务器类似，大多数 FTP 服务器的实现允许多个客户的并发存取。客户使用 TCP 连接到服务器。正如第 20 章中描述的，一个主服务器进程等待连接，并为每个连接的处理建立一个从进程。然而，与大多数服务器不同的是，从进程没有执行所有必要的计算。事实上，从进程只接受

和处理来自客户的**控制连接** (control connection)，另外还需要使用一个额外的进程和一条额外的 TCP 连接来处理每次**数据传送** (data transfer) 操作。控制连接承载的是告诉服务器将传送哪个文件的命令，而每次数据传送操作 (即每个文件的传送) 都需要分别在客户端和服务器端建立一条新的 TCP 连接和一个新的进程。图 25.1 描绘的就是这个概念，其中进程体系结构的具体细节取决于所用的操作系统。

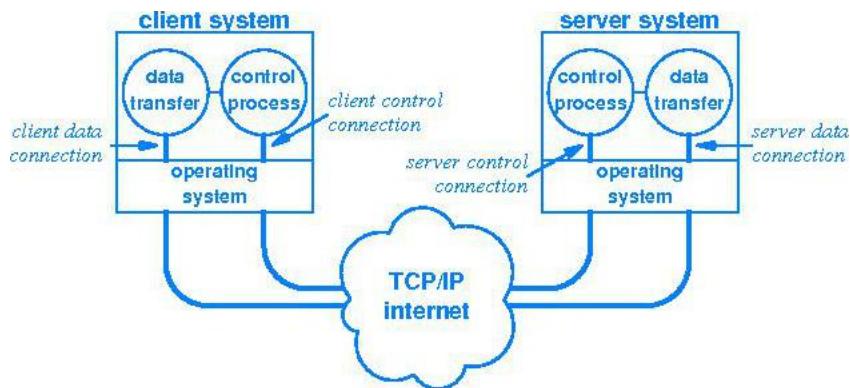


图 25.1 一个 FTP 客户和服务器。两者之间有一条 TCP 控制连接，与此相关的数据传送进程之间有一条独立的 TCP 连接

如图 25.1 所示，客户的控制进程使用一个 TCP 连接连到服务器的控制进程，而相关的数据传送进程使用它们自己的 TCP 连接。通常，只要用户保持 FTP 会话在活动状态，控制进程和控制连接就一直存在，而数据传送连接只维持一个文件的传送。主要思想总结如下：

数据传送连接和使用连接的数据传送进程可以在需要时动态创建，但控制连接在整个会话期间持续。一旦控制连接消失，会话也就终止了，并且两端的软件将终止所有的数据传送进程。

25.8 TCP 端口号和数据连接

在客户构造一条到服务器的初始连接时，客户使用的是一个任意分配的本地协议端口号，并与服务器的一个熟知端口 (21) 联系。如第 20 章所述，服务器只需使用一个协议端口就可以接受来自多个客户的连接，因为 TCP 用连接的两端来识别一个连接。随之而来的问题是，“当控制进程为某个数据传送建立一条新的 TCP 连接时，它们使用什么协议端口？”显然，它们不能使用与控制连接相同的一对端口号。事实上，客户从本地获取一个未用的端口，并通过控制连接将这个端口号发送出去，然后等待服务器构造一条到达指定端口的 TCP 连接。也就是说，在构造数据连接时，FTP 客户变成了服务器，而 FTP 服务器则变成了客户！有意思的是，FTP 服务器在构造数据连接时，使用的是一个熟知端口：为 FTP 数据传送而保留的端口 (20)。

我们可以看出协议为什么要使用两条连接：客户控制进程获取一个任意的本地端口用于文件传送，在客户机上创建一个传送进程用于监听该端口，通过控制连接将端口号告知服务器，然后等待服务器建立一条到达该端口的 TCP 连接。总而言之：

除了向服务器传递用户命令以外，FTP 还通过控制连接，让客户和服务器的控制进程相互协调对动态分配的 TCP 协议端口的使用，并创建使用这些端口的数据传送进程。

因为 FTP 在最初设计时没有考虑如何与安全防火墙以及 NAT 系统一起正常工作，所以又增加

了对它的扩展。这个扩展称为**被动 FTP** (passive FTP)，它允许由客户初始化每条数据传送连接。也就是说，FTP 可以穿过防火墙或 NAT 系统使用，而不需要特殊处理。大多数 FTP 服务器和客户支持被动 FTP 扩展。

FTP 在控制连接上传递的数据使用什么格式？虽然 FTP 设计者本来可以发明一种新的规范，但他们没有这样做。相反，他们让 FTP 使用第 24 章描述的 TELNET 虚拟网络终端协议。FTP 不完全像 TELNET 协议，它不允许选项协商。事实上，FTP 只能使用基本的 NVT 定义，因此 FTP 控制进程的管理比 TELNET 连接的管理简单得多。尽管有其局限性，但是使用 TELNET 的定义而不是发明一种新规范，使 FTP 得以大大简化。

25.9 从用户的角度看 FTP

虽然 FTP 最初是以一个交互式应用程序实现的，但是现在很少有用户会直接调用 FTP。事实上，大多数用户通过浏览器（将在第 27 章中描述）访问 FTP。例如，假设某人将一本教材的在线副本放在 [ftp.cs.purdue.edu](ftp://ftp.cs.purdue.edu) 计算机的/pub/comer 目录^①下的 tcpbook.tar 文件中。为了获得这个副本，用户可以先启动浏览器，然后输入以下 URL：

<ftp://ftp.cs.purdue.edu/pub/comer/tcpbook.tar>

浏览器将会读取这个文件的副本。另外，如果用户输入的 URL 对应的是一个目录，则浏览器会显示该目录下的文件名列表：

<ftp://ftp.cs.purdue.edu/pub/comer>

25.10 匿名 FTP

虽然 FTP 中的存取授权机制有助于保护敏感文件不被随意存取，但是要求输入登录名和口令的措施使用户连公共文件也无法存取，除非用户已经得到一个账户。为了提供对公共文件的存取，大多数 TCP/IP 网点允许使用**匿名 FTP** (anonymous FTP)。匿名 FTP 访问意味着客户不需要账号或口令。事实上，用户指明的登录名是 anonymous 和一个相当于用户电子邮件地址的口令，或者就是 guest 口令。允许匿名登录的服务器将限制匿名者只能存取向公众开放的文件。

25.11 安全文件传送 (SSL-FTP, scp 和 sftp)

虽然 FTP 的初级版本中包括了口令机制，以允许服务器判别用户，但是这个口令的发送是未经加密的。另外，通过 FTP 发送的数据也是未经加密的。因此，如果有人在一条连接上窃听，很可能获得正在发送的数据的副本或者用户的登录名和口令。为此已经开发了数种机制，用于为文件传送提供额外的安全措施。

安全套接字层 FTP (Secure Sockets Layer FTP，简称 SSL-FTP)。有一种 FTP 扩展使用了安全套接字层技术为 FTP 加密，这是应用程序（如万维网）使用的事实上的标准。在使用 SSL-FTP 时，所有传送都是保密的，无论是正在传送的数据还是登录口令。另外，因为基本的 SSL 遵守套接字 API，所以使用这个扩展时只需要对 FTP 的客户和服务器做很少的修改。

安全文件传送程序 (secure File Transfer Program，简称 sftp)。安全文件传送程序 sftp 是为了取代 FTP 而开发的（即 sftp 协议与 FTP 几乎没有共同点）。虽然 sftp 可以执行任意文件的传送，但是它建立在使用 SSH 隧道的基础上，不能单独使用。因此，sftp 传送要通过一条 SSH 连接进行复用，这条 SSH 连接同时也向其他应用程序提供安全性。

^① 有些操作系统使用术语**文件夹** (folder)，而不是**目录** (directory)。

安全复制 (Secure Copy, 简称 scp)。安全复制是 FTP 的另一个替代，是从 UNIX 的远程复制程序 (remote copy program, 简称 rcp) 衍生而来的，使用加密技术提供安全传送。与 sftp 一样，scp 传送使用的是 SSH 通道，并且远程计算机上的文件存取要经过 SSH 服务器的中继。不过，与 sftp 不同的是，scp 使用与 UNIX 文件复制命令 cp 相同的语法。因此，scp 不仅可以用在命令行中，也可以用在脚本中。

25.12 TFTP

虽然 FTP 是 TCP/IP 协议族中最常用的文件传送协议，但它对编程而言也是最复杂、最困难的。许多应用程序既不需要 FTP 提供的全部功能，也无法承受 FTP 的复杂性。例如，FTP 需要客户和服务器管理多个并发的 TCP 连接，这对于没有复杂操作系统的嵌入式计算机而言有些困难，或者根本不可能实现。

TCP/IP 协议族含有第二个文件传送协议，提供并不复杂且开销也不大的服务。该协议称为**简单文件传送协议** (Trivial File Transfer Protocol, 简称 TFTP)，是为客户和服务器之间不需要复杂交互的应用程序设计的。TFTP 只限于简单的文件传送操作，不提供存取授权。由于 TFTP 的局限性较大，TFTP 软件比 FTP 小得多。

对于许多应用而言，程序小是很重要的。例如，嵌入式系统的生产厂商可以在只读存储器 (ROM) 中保存 TFTP 代码，并在机器开机时用它来获取一个初始的内存映像。ROM 中的程序称为系统**引导程序** (bootstrap)^①。使用 TFTP 的优点是允许引导程序代码在启动后使用与操作系统相同的底层 TCP/IP 协议。因此，一台计算机可以通过位于另一个物理网络的服务器引导。

与 FTP 不同的是，TFTP 不需要可靠的数据流运输服务。它运行在 UDP 或其他任何不可靠的分组交付系统上，使用超时和重传来保证数据的到达。发送方用固定大小 (512 字节) 的块来发送文件，并在发送下一个块之前等待对每个块的确认。接收方每收到一个块后都要加以确认。

TFTP 的规则很简单。发送的第一个分组用于请求文件传送和建立客户与服务器之间的交互。这个分组指明了文件名，以及文件将被读取 (传给客户) 还是写入 (传给服务器)。文件块从 1 开始连续编号。每个数据分组包含一个首部，指明它所运载的块的编号，并且每个确认都包含被确认的块的编号。少于 512 字节的块标志着文件结束。可以在数据或确认的位置上发送差错报文，差错将终止文件传送。

图 25.2 所示为五类 TFTP 分组的格式。初始分组必须使用操作码 1 或 2，分别指明是**读请求** (read request) 还是**写请求** (write request)。文件名 (FILENAME) 字段指明文件名，模式 (MODE) 字段指明客户将读文件，写文件，还是既读又写。

^① 第 22 章讨论了 DHCP 启动引导的细节。

2-octet opcode	n octets	1 octet	n octets	1 octet
READ REQ. (1)	FILENAME	0	MODE	0
2-octet opcode	n octets	1 octet	n octets	1 octet
WRITE REQ. (2)	FILENAME	0	MODE	0
2-octet opcode	2 octets	up to 512 octets		
DATA (3)	BLOCK #	DATA OCTETS...		
2-octet opcode	2 octets			
ACK (4)	BLOCK #			
2-octet opcode	2 octets	n octets	1 octet	
ERROR (5)	ERROR CODE	ERROR MESSAGE	0	

图 25.2 五种 TFTP 报文类型。鉴于一些字段是变长的，因而字段未按长度显示。前两个八位组的操作码用于识别报文格式

一旦发出了读或写请求，服务器使用客户的 IP 地址和 UDP 协议端口号识别后续操作。因此，无论是数据报文（运载文件块的报文）还是确认报文（确认数据块的报文），都不需要指明文件名。图 25.2 中的最后一个报文类型用来报告差错。丢失的报文可以在超时后重传，但其他大多数差错只简单地导致交互的终止。

由于对称性，TFTP 重传与普通的重传不一样，由两端各自实现超时和重传。如果发送端超时，它将重传丢失的数据块。如果负责确认的一端超时，它将重传丢失的确认。让两端都参与重传有助于保证在丢失单个分组时传送不会失败。

对称重传虽然保证了健壮性，但也可能导致过量重传。这个问题称为 Sorcerer's Apprentice Bug（巫士徒弟的错误），当数据分组 k 的一个确认延迟，但并没丢失时，会出现此问题。发送方将重传数据分组，然后接收方进行确认。两个确认最终又到达了发送方，每个确认都会导致数据分组 k+1 的传送。接收方对两个数据分组 k+1 都进行确认，而发出的两个确认又将分别引起发送方发送数据分组 k+2。如果底层互联网复制了分组，也会造成过量重传。一旦出现，每个数据分组正好传送两次的循环将无限继续下去。在最近的 TFTP 版本中对这种过量重传问题做了修正。

25.13 NFS

网络文件系统（Network File System，简称 NFS）最初是由 Sun Microsystems 公司开发的，目前已经作为 IETF 标准发布，以提供透明、一体化的共享式文件存取。图 25.3 描绘了 NFS 如何嵌入到操作系统中。

当一个应用程序执行时，它调用操作系统打开文件，以便从文件中读取数据，或将数据写入文件。文件存取机制接受请求，并自动将请求传给本地文件系统软件或 NFS 客户，具体取决于文件是位于本地磁盘还是远程机器上。当 NFS 客户软件收到请求后，就使用 NFS 协议联系相应的远程机器上的服务器，并执行所请求的操作。远程服务器做出响应后，客户软件将结果返回给应用程序。

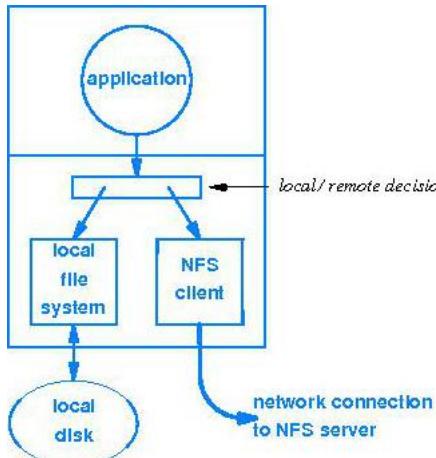


图 25.3 操作系统中的 NFS 的实现。当应用程序请求一个文件操作时，操作系统必须把请求传递给本地文件系统或 NFS 客户软件

25.14 NFS 的实现（RPC 和 XDR）

设计人员并没有打造一个全新的 NFS 协议，而是选择了用三个独立的部分来构造 NFS 协议，即：NFS 协议本身、一个通用的**远程过程调用**（Remote Procedure Call，简称 RPC）机制和一个通用的**外部数据表示**（eXternal Data Representation，简称 XDR）。这样做的目的是使这三个部分之间相互独立，让其他软件（包括应用程序和其他协议）也可以使用 RPC 和 XDR。

从程序员的角度看，NFS 本身并没有提供其他程序可以调用的新过程。事实上，一旦管理员配置好 NFS，程序存取远程文件所用的操作就与存取本地文件一样。不过，RPC 和 XDR 都为程序员提供了构造分布式程序所需的机制。例如，程序员可以将程序分成客户端和服务器端，使用 RPC 作为主要通信机制。在客户端，程序员指明一些过程是**远程的**（remote），迫使编译器将 RPC 代码合并到这些过程中。在服务器端，程序员实现所需的过程并使用其他 RPC 设施将其声明为服务器的一部分。当正在运行的客户程序调用一个远程过程时，RPC 自动收集参数的值，形成一个报文，再将报文送到远程服务器，然后等待响应，并将返回的值存放在指定参数中。实质上，与远程服务器的通信只是作为远程过程调用的副作用而自动发生的。RPC 机制隐藏了所有协议细节，使得几乎不了解底层通信协议的程序员也可以编写分布式程序。

另一个相关工具 XDR 为程序员提供了一种在异构机器之间传递数据的方式，它不需要程序员书写转换硬件数据表示的过程。XDR 通过定义一种与机器无关的表示来解决此问题。在通信通道的一端，一个程序调用 XDR 过程将本地硬件表示转换为与机器无关的表示。一旦数据传到另一台机器，接收程序调用 XDR 过程将与机器无关的表示转换为机器的本地表示。

25.15 小结

存取远程文件中的数据有两种方式：完整文件复制和共享在线存取。文件传送协议 FTP 是 TCP/IP 协议族中的主要文件传送协议。FTP 使用了完整文件复制，并提供列出远程机器上的目录和双向文件传送的能力。简单文件传送协议 TFTP 为只需要文件传送功能的应用程序提供了一个小巧且简单的，可取代 FTP 的方案。由于 TFTP 小到足以放入 ROM，所以可用于嵌入式系统的启动引导。

网络文件系统（NFS）由 Sun Microsystems 公司设计，可提供共享的在线文件存取。NFS 使用 UDP 传送报文，并使用了该公司的远程过程调用（RPC）和外部数据表示（XDR）机制。

25.16 深入研究

Postel [RFC 959] 含有 FTP 协议标准。Horowitz and Lunt [RFC 2228], Allman and Ostermann [RFC 2577] 以及 Housley and Hoffman [RFC 2585] 都讨论了安全性方面的扩展。有 30 多份 RFC 研究 FTP，提出了修改建议或定义协议的新版本。其中，Lottor [RFC 913] 描述了简单文件传送协议。DeSchon and Braden [RFC 1068] 说明如何使用 FTP 第三方传送进行后台文件传送。Allman et. al. [RFC 2428] 探讨了在 IPv6 以及 NAT 下的 FTP。本章描述的简单文件传送协议出自 Sollins [RFC 1350]。Finlayson [RFC 906] 描述了 TFTP 在引导计算机系统中的使用，Malkin and Harkin [RFCs 2347, 2348, 2349] 讨论了各种选项。

Sun Microsystems 公司已发布了定义网络文件系统和相关协议的三个 RFC。RFC 1094 含有 NFS 标准，RFC 1057 定义 RPC，RFC 1014 定义 XDR。有关 RPC 和 NFS 的更详细内容可参见本系列教材的第三卷。

25.17 习题

1. 为什么即使采用了 TCP 之类的端到端可靠数据流传送协议，文件传送协议还是应当计算所收到文件数据的检验和？
2. 查阅有关资料，FTP 是否为其传送的文件计算检验和？
3. 如果用于数据传送的 TCP 连接中断，但控制连接没有中断，FTP 将会发生什么情况？
4. 为控制和数据传送分别单独使用 TCP 连接的主要优点是什么？提示：考虑一些异常情况。
5. 概述一种使用 TFTP 引导无盘机器的方法。注意，它每一步使用的 IP 地址是什么？
6. 实现一个 TFTP 客户。
7. 试验 FTP 或一个其他等价协议，看看在一个局域网的两个相当大的系统之间传送一个文件能有多快。在网络忙和网络闲时分别进行试验。试解释结果。
8. 尝试从本机到本机的 FFP，然后再尝试从局域网上一台机器到另一台机器的 FTP。这两个数据传送速率令你惊讶吗？
9. 比较局域网上 FTP 和 NFS 的传送速率。你能解释这一区别吗？
10. 解释 RPC 的定义。它会处理数据报的丢失吗？重复呢？迟延呢？损伤呢？
11. 在前一个问题的基础上进一步展开，考虑在 RPC 上运行 NFS，NFS 在全球因特网上能够正常运行吗？试解释原因。
12. XDR 方案在什么情况下效率不高？
13. 考虑将浮点数从内部格式转换为外部格式并转换回内部格式。在选择外部格式中的指数和尾数大小时的折中方案是什么？
14. FTP 默认使用 ASCII 模式（即文本模式）来传送文件。这种默认模式明智吗？试论证默认的 ASCII 模式可能是“有害的”。

第26章 电子邮件（SMTP, POP, IMAP 和 MIME）

26.1 引言

本章继续探讨网际互联技术，研究电子邮件服务以及支持它的协议。本章将描述如何组织一个邮件系统，解释别名扩展，并阐明邮件系统软件如何使用客户—服务器模式传送每一个报文。

26.2 电子邮件

电子邮件 (electronic mail, 简称 e-mail) 工具允许用户通过因特网发送一些函件。电子邮件是使用最广泛的应用服务之一，因为它提供了快速而方便的信息传送方式，包括简短的说明性文字或大容量的函件，并且允许个人之间或群组之间的通信。

电子邮件与其他大多数网络应用有本质上的区别，因为邮件系统必须考虑远端目的地暂时无法接通的情况。为了处理延迟交付问题，邮件系统使用一种称为**假脱机** (spooling) 的技术^①。当用户发送一个邮件报文时，系统将邮件的副本以及邮件的发送方、接收方、目的机器和存储时间一起保存在专用存储区（称为缓冲区^②）中。然后，系统以后台方式启动与远程机器之间的传送，以允许发送方继续进行其他计算活动。图 26.1 描绘了这一概念。

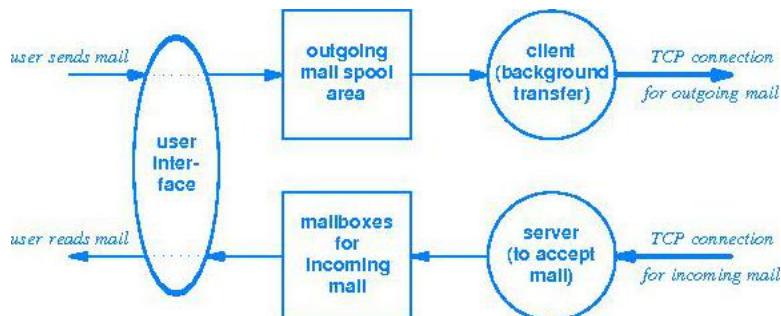


图 26.1 电子邮件系统在概念上的组成部分。用户激活一个邮件接口应用程序来存放或读取邮件。所有传送都在后台进行

后台邮件传送进程成为一个客户，并使用域名系统将目的机器名映射为 IP 地址，然后试图建立到达目的机器上邮件服务器的 TCP 连接。如果成功，传送进程将一份报文副本传递给远程服务器，该服务器将此副本存放在远程系统的缓冲区内。一旦客户和服务器都认为已经收到并存储了该副本，客户即可删除本地副本。如果客户不能建立 TCP 连接或连接失败，传送进程将记录尝试交付的时间和交付终止的时间。后台传送进程定期扫描整个缓冲区，检查是否有未交付的邮件，一般来说这个周期为每隔 30 分钟一次。一旦它找到一个未交付的邮件，或者一旦用户放入一个新的待发邮件，后台进程将再次尝试交付。如果过了几小时后，邮件客户进程发现邮件报文还是不能交付出去，它就会通知发送方，而如果是过了某个期限（如 3 天）之后，邮件软件会将此邮件返回给发

^① 实际上这就是缓冲技术——译者注。

^② 有时邮件缓冲区也称为**邮件队列** (mail queue)，虽然这种说法从技术上讲并不准确。

送方。

26.3 邮箱名字和别名

在我们对邮件交付过程的简单描述中隐藏了三个重要的思想。第一，用户通过给出一个字符串来指明每个收件方，这个字符串包含了由@分隔的两项内容：

local-part@domain-name

其中，domain-name 是邮件将被交付到达的邮件目的地的域名^①，local-part 是该机器上的邮箱地址。例如，本书作者的邮件地址是：

comer@purdue.edu

第二，此规范中使用的名字与分配给机器的其他名字完全无关。通常，邮箱与用户的登录标识符相同，而机器的域名会作为邮件目的地的域名。不过，其他各种设计方法也是可行的。例如，可以按职位指派邮箱，如 department-head。由于域名系统中包含了一个单独的用于邮件目的地的查询类型，因此可以让邮件目的地域名与分配给机器的正常域名脱离关系。因此，发给 example.com 用户的邮件到达的机器可能与 example.com 的 ping 请求到达的机器不是同一台机器。第三，我们的概述中没有考虑邮件转发，也就是说，到达某台机器上的邮件还需要转发到另一台机器上。

26.4 别名扩展与邮件转发

大多数电子邮件服务器提供**邮件转发**（mail forwarding）软件，其中包括了**邮件别名扩展**（mail alias expansion）机制。邮件转发器允许将传入报文的副本再发送到一个或多个目的地。通常，服务器会查询一个小型的邮件别名数据库，把传入的收件方地址映射为一组地址 A，然后再向 A 中的每个地址转发一份副本。

因为别名映射可以是“多对一”或“一对多”的，所以它增加了邮件系统的功能，并为用户带来了方便。一个人可以拥有多个邮件标识符，或者一群人合起来使用一个邮件别名。在后一种情况下，与一个标识符关联的一组收件方称为**电子邮件列表**（electronic mailing list）。图 26.2 所示为支持邮件别名和邮件列表扩展的邮件系统的构成方式。

如图所示，传入的和外发的邮件都要经过邮件转发器，并由邮件转发器扩展别名。因此，如果别名数据库指明要将邮件地址 x 映射替换为 y，则别名扩展将重写目的地址 x，将它改为 y，然后别名扩展程序再判定 y 指的是本地地址还是远程地址，从而知道应当将邮件放到传入邮件队列中，还是外发邮件队列中。

邮件别名扩展可能会带来危险。设想有两个网点建立了相互冲突的别名。例如，假设网点 A 将邮件地址 x 映射为网点 B 上的邮件地址 y，而网点 B 将邮件地址 y 映射为网点 A 上的邮件地址 x。发送到网点 A 的地址 x 的邮件将会在两个网点之间来回循环不休^②。

^① 从技术上讲，域名指明的是一个邮件交换机，而不是机器名。

^② 实际上，大多数邮件转发器会在邮件交换次数达到某个预定阈值后终止该邮件的转发。

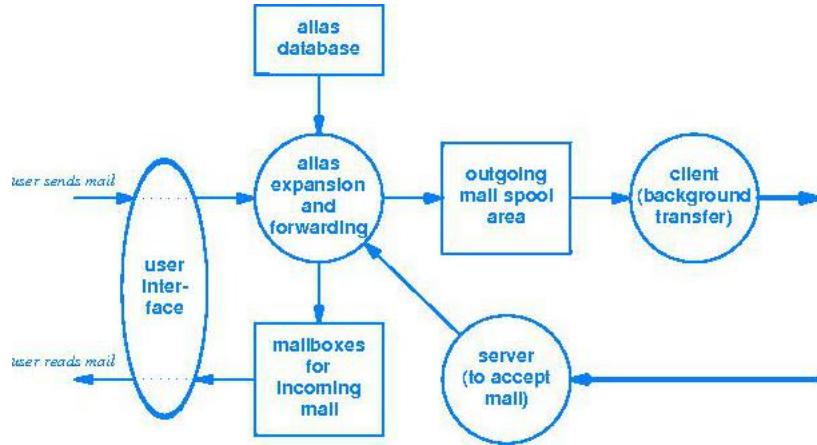


图 26.2 对图 26.1 中的邮件系统的扩展，支持邮件别名和转发功能。传入和外发的邮件都要通过别名扩展机制

26.5 电子邮件服务的 TCP/IP 标准

前面讲过，TCP/IP 协议的努力方向是为最大范围的计算机系统和网络提供互操作性。为了拓展电子邮件的互操作性，TCP/IP 将邮件标准一分为二。其中一个标准（在 RFC 2822 中给出）用于指明邮件报文使用的语法格式^①。另一个标准则指明两台计算机之间电子邮件交换的细节。

根据 RFC 2822，邮件报文以文本方式表示，并且分为两部分：首部和主体，中间用一个空行隔开。邮件报文标准指明了邮件首部的确切格式以及每个首部字段的语义解释，但它将主体格式留给发送方决定。实际上，标准指明邮件首部中包含的是分成若干行的可读文本，每行由一个关键字后跟冒号和一个值组成。有一些关键字必须存在的，有一些是可选的，其余则不加解释。例如，首部必须包含一行指明了目的地的文本。该行以 To: 开头，后半行的内容则是收件方的电子邮件地址。以 From: 开头的行中含有发送方的电子邮件地址。可选的关键字包括：发送方可以指定回执应送到哪个地址（即允许发送方指定将回执送到某个地址而不是发送方的邮箱），如果有，则用以 Reply-to: 开头的行指明回执发往的地址。如果没有此行，接收方将使用 From: 行中的信息作为回执地址。

选择使用这样的邮件报文格式是为了方便在异构机器之间的处理和传送。保持邮件首部格式的直观，使它可用于各种系统。将报文限制为可读文本，则避免了为选择一种标准的二进制表示法以及在标准表示法和本地表示法之间进行转换而带来的问题。

26.6 简单邮件传送协议

除了报文格式之外，TCP/IP 协议族还定义了机器和机器之间交换邮件的标准。也就是说，该标准定义了一台机器上的客户将邮件传送到另一台机器上的服务器时所使用的确切报文格式。这个标准传送协议称为**简单邮件传送协议**（Simple Mail Transfer Protocol，简称 SMTP）。读者可能会猜到，SMTP 比早期的**邮件传送协议**（Mail Transfer Protocol，简称 MTP）简单。SMTP 重点关注底层邮件交付系统如何通过一个互联网，将邮件从一台机器传到另一台机器，并没有指明邮件系统如何从用户那里接受邮件，或用户接口如何将收到的邮件交给用户。而且，SMTP 没有定义如何存储邮件或邮件系统以什么频率来尝试发送邮件。

^① 最初的标准是在 RFC 822 中定义的。为了使编号匹配，IETF 将其更新推迟至 RFC 2822。

SMTP 非常直观。客户和服务器之间的通信由可读的 ASCII 文本组成。与使用其他应用协议一样，程序读取命令缩写以及行首的 3 位数字，其余文字用来帮助人们手工调试邮件软件。虽然 SMTP 严格定义了命令的格式，但是人们可以轻易地读到客户与服务器之间交互时的命令列表，因为每个命令占据单独一行。最初由客户建立一条到达服务器的可靠数据流连接，并等待服务器发送一个 220 READY FOR MAIL 报文（如果服务器超载，可能暂时推迟 220 报文的发送）。收到 220 报文后，客户发送一个 HELO^①命令（如果客户支持 FRC 2821 中的扩展，客户就发送一个 EHLO 命令）。一行的结束标志着一个命令的结束。服务器通过标识自己进行响应。一旦建立通信，发送方就可以传送一个或多个邮件报文，然后终止连接。接收方必须确认命令，也可以异常终止整个连接或当前报文的传送。

邮件事务是从 MAIL 命令开始的，给出了发送方的标识符和一个 FROM:字段，该字段包含接受差错报告的地址。收件方准备好用于接收新邮件的数据结构，并通过发送响应 250 来回答 MAIL 命令。响应 250 的含义是一切都好。完整的响应由文本“250 OK”组成。

成功执行 MAIL 命令后，发送方发出一系列 RCPT 命令，这些命令标识了邮件的收件方。接收方必须通过发送 250 OK 或发送差错报文 550 No such user here 来确认每个 RCPT 命令。

在收到所有 RCPT 命令的确认后，发送方发出一个 DATA 命令。DATA 命令本质上是通知接收方，称发送方已经准备好传送一个完整的邮件。接收方用报文 354 Start mail input 响应，并指明用于终止邮件报文的字符序列。终止序列由 5 个字符组成：回车符、换行符、点、回车符和换行符^②。

下面将举例阐明 SMTP 交换过程。设想主机 Alpha.edu 上的用户 Smith 向主机 Beta.gov 上的用户 Jones, Green 和 Brown 发送邮件。如图 26.3 所示，主机 Alpha.edu 上的 SMTP 客户软件与主机 Beta.gov 上的 SMTP 服务器软件联系，并开始交换。

在这个例子中，由于服务器不能将 Green 识别为有效邮件地址（也就是说，它既不是用户也不是邮件列表），它拒绝了收件方 Green。SMTP 协议没有详细定义客户如何处理这种错误，必须由客户自己决定。虽然可以在出错时完全放弃交付，但大多数客户不会这样做。事实上，它们继续将邮件交付给所有有效的收件方，然后将问题报告给初始的发送方。通常，客户用电子邮件报告差错。差错报文中含有关于差错小结和引起问题的邮件的首部。

一旦客户的所有邮件报文都已发送完成，就可以发出一个 QUIT 命令。另一端用命令 221 响应，表示同意终止连接。然后两端将从容关闭这条 TCP 连接。

SMTP 比这里概述的内容复杂得多。例如，如果有一个用户转移了，服务器可能知道用户新的邮箱地址。SMTP 允许服务器把新地址通知给客户，以便客户以后可以使用它。在向客户通知新地址时，服务器可能选择转发这个引起了通知报文的邮件，也可能要求客户自己负责转发。另外，SMTP 还包括**运输层安全**（Transport Layer Security，简称 TLS）扩展，以允许加密 SMTP 会话。

^① HELO 是 hello 的缩写。

^② SMTP 用 CR-LF 来结束一行，并禁止邮件的主体中出现只有一个点的行。

```
S: 220 Beta.gov Simple Mail Transfer Service Ready
C: HELO Alpha.edu
S: 250 Beta.gov

C: MAIL FROM:<Smith@Alpha.edu>
S: 250 OK

C: RCPT TO:<Jones@Beta.gov>
S: 250 OK

C: RCPT TO:<Green@Beta.gov>
S: 550 No such user here

C: RCPT TO:<Brown@Beta.gov>
S: 250 OK

C: DATA
S: 354 Start mail input; end with <CR><LF>.<CR><LF>
C: ...sends body of mail message...
C: ...continues for as many lines as message contains
C: <CR><LF>.<CR><LF>
S: 250 OK

C: QUIT
S: 221 Beta.gov Service closing transmission channel
```

图 26.3 SMTP 传送举例, 从 Alpha.edu 到 Beta.gov, 其中无法识别收件方 Green。以“C:”开头的行是由客户(Alpha)发送的, 而以“S:”开头的行是由服务器发送的

26.7 邮件取回和邮箱操纵协议

前面介绍的 SMTP 传送方案暗示了服务器必须在任何时候都做好接收电子邮件的准备。如果此处的服务器运行在具有永久因特网连接的计算机上, 那么可以正常工作, 但是, 对于连接时断时续的计算机来说(如膝上型计算机在移动时断开连接), 则无法正常工作。在这样的计算机上运行电子邮件服务器是没有任何意义的, 因为只有在用户的计算机连到因特网上时, 服务器才是有效的, 否则任何联络服务器的企图都将徒劳无功, 而发送给用户的电子邮件始终无法交付。由此带来的问题是“没有永久连接的用户怎样接收电子邮件呢?”

这个问题的答案在于一个两阶段的交付过程。在第一个阶段, 一台具有永久因特网连接的计算机为每个用户分配了一个邮箱。这台计算机运行着一个常规的 SMTP 服务器, 它总是准备好接收电子邮件。第二个阶段, 用户连接到因特网上, 然后运行一个从永久性邮箱中取回邮件的协议。这个协议把邮件报文传送到用户使用的计算机上, 然后用户可以在这台计算机上阅读邮件。

有两个协议允许远程用户从永久邮箱中取回邮件。虽然这两个协议的功能类似, 但却采取了两种截然相反的方法: 一个允许用户下载报文的副本, 而另一个则允许用户查阅和操纵服务器上的报文。下面两个小节分别描述了这两个协议。

26.7.1 邮局协议

把邮件从永久邮箱传送到本地计算机上的最流行的协议称为**邮局协议**(Post Office Protocol)版本3, 简称POP3, 这个协议的安全版本称为POP3S。用户调用POP3客户, 该客户创建一个TCP连接, 连到邮箱所在计算机的POP3服务器上。用户首先要发送登录名和口令, 以鉴别会话。一旦

鉴别被接受，客户发送命令，取回一个或多个邮件的副本，然后从永久邮箱中删除邮件。邮件按 2822 标准格式的文本文件进行存储和传送。

注意，具有永久邮箱的计算机必须运行两个服务器，SMTP 服务器接受发送给用户的邮件，并把传入的每个邮件添加到该用户的永久邮箱中，而 POP3 服务器允许用户从邮箱中提取邮件并将其删除。为了确保操作正确，这两个服务器必须协调使用邮箱，如果邮件通过 SMTP 到达的同时，用户通过 POP3 提取邮件，邮箱必须能够保持有效状态。

26.7.2 网际报文访问协议

网际报文访问协议 (Internet Message Access Protocol) 版本 4，简称 IMAP4，是 POP3 的一种替代协议，允许用户查阅和操纵报文。IMAP 也已定义了一个安全版本，称为 IMAPS。与 POP3 类似，IMAP4 定义了称为**邮箱** (mailbox) 的抽象，邮箱和服务器位于同一计算机上。用户运行一个 IMAP4 客户与服务器进行联络，以操纵报文，这也和 POP3 类似。但是，与 POP3 不同的是，IMAP4 允许用户从多个地点（如从工作地点和从家里）访问邮件报文，并保证所有副本都是同步并且一致的。

IMAP4 还提供了邮件取回和处理的扩展功能。用户可以获得有关报文的信息或检查首部字段，而不需要取回整个报文。另外，用户可以搜索一个指定的字符串，并取回报文中的某些部分。部分取回对于慢速拨号连接尤其重要，因为这意味着用户不需要下载无用的信息。

26.8 非 ASCII 码数据的 MIME 扩充

多用途网际邮件扩充 (Multipurpose Internet Mail Extension，简称 MIME) 是为了通过电子邮件发送非 ASCII 数据而定义的。MIME 并没有改动或取代 SMTP、POP3 和 IMAP4 这些协议。相反，它允许用 ASCII 码将任意数据编码，然后在标准电子邮件报文中传送。为了适应任意数据类型和表示法，每个 MIME 邮件中包含了一些信息，用来告知收件方它所使用的数据类型和编码格式。MIME 的信息位于 2822 邮件首部中，MIME 首部中的一些行指明了 MIME 的版本、发送数据的类型以及将数据转换为 ASCII 码所用的编码格式。例如，图 26.4 所示为一个 MIME 报文，其中包含了一幅标准的 JPEG^①格式的照片。这个 JPEG 图像已经用 base64 编码格式转换为 7 位 ASCII 码表示。

```
From: bill@acollege.edu
To: john@example.com
MIME-Version: 1.0
Content-Type: image/jpeg
Content-Transfer-Encoding: base64
```

...data for the image...

图 26.4 一个 MIME 报文的例子。首部中的行标识了数据类型和所用的编码格式

在图 26.4 中，首行 MIME-Version: 声明了报文是用 1.0 版本的 MIME 协议构成的。Content-Type: 声明项指明数据是一幅 JPEG 图像，Content-Transfer-Encoding: 首部声明使用 base64 编码将图像转换成 ASCII 码。

为了查看图像，接收方的邮件系统首先必须将 base64 编码转换成二进制码，然后运行一个应用程序在用户屏幕上显示 JPEG 图像。选择 base64 可提供 64 个在各种版本的 ISO 英语字符集中表

^① JPEG 是用于数字图片的**联合图像专家小组** (Joint Picture Experts Group) 标准。

示都相同的 ASCII 字符^①，因此可以保证接收方从编码数据中提取的图像与原始图像完全一致。

MIME 标准规定 Content-Type 声明必须含有两个标识符，一个是**内容类型** (content type)，一个是**子类型** (subtype)，中间用“/”分开。在这个例子中，image 是内容类型，jpeg 是子类型。

标准定义了 7 种基本内容类型以及每种类型的子类型和传送编码。例如，image 类型的子类型必须为 jpeg 或 gif，而 text 类型却不能使用这两种子类型。除了标准类型和子类型之外，MIME 允许发送方和接收方定义专用的内容类型^②。图 26.5 列出了 7 种基本内容类型。

Content Type	Used When Data In the Message Is
text	Textual (e.g. a document).
image	A still photograph or computer-generated image
audio	A sound recording
video	A video recording that includes motion
application	Raw data for a program
multipart	Multiple messages that each have a separate content type and encoding
message	An entire e-mail message (e.g., a memo that has been forwarded) or an external reference to a message (e.g., an FTP server and file name)

图 26.5 可以出现在 MIME Content-Type 声明中的 7 种基本类型及其意义

26.9 MIME 多部分报文

MIME 多部分的内容类型很有用，因为它在很大程度上增加了灵活性。标准为多部分报文定义了 4 种可能的子类型，每种子类型都提供了重要功能。**混合** (mixed) 子类型允许单个报文含有多个相互独立的子部分，每个子部分可以有独立的类型和编码。使用混合的多部分报文可以在一个报文中包含文本、图形和音频，或者发送一个带有附加数据段的函件，类似于商业信件中的随信附件。**替代** (alternative) 子类型允许在一个报文中包含同一数据的多种表示。在向使用不同硬件和软件系统的多个收件方发送函件时，替代多部分类型的报文很有用。例如，用户发送文档时既包含普通 ASCII 文本，也包含格式化过的文档，从而允许拥有图形功能计算机的用户在阅读时选用格式化的文档^③。**并行** (parallel) 子类型允许一个报文含有必须放在一起查看的子部分（例如，必须在一起播放的视频和音频子部分）。最后，**文摘** (digest) 子类型允许一个报文含有一组其他报文（例如，从一次讨论中搜集的电子邮件集合）。

图 26.6 所示为多部分报文的一种主要用途：电子邮件报文可以包含解释报文用途的简短文本，以及其他非文本信息的部分。在图中，报文第一部分的注解说明了在第二部分中含有一幅照片图像。

图 26.6 也展示了 MIME 的几个细节问题。例如，在基本声明之后，首部行中可以含有 X = Y 格式的参数。多部分内容类型说明之后紧跟的关键字 Boundary= 定义了用于分隔报文各部分的字符串。在这个例子中，发送方选择了字符串 StartOfNextPart 作为分界符。如果有子部分的内容类型和传送编码格式的声明，则应该紧跟在分界行之后。这个例子中的第二个子部分声明为一幅 GIF 图像。

^① 这个字符集包括 26 个大写字母、26 个小写字母、10 个数字、加号和符号“/”。

^② 为了避免可能产生的名字冲突问题，标准要求在选择专用内容类型的名字时，都要以“X-”打头。

^③ 有一种广泛使用的电子邮件系统利用替代类型 MIME 选项，以 ASCII 和 HTML 两种格式发送报文。

```

From: bill@acollege.edu
To: john@example.com
MIME-Version: 1.0
Content-Type: Multipart/Mixed; Boundary=StartOfNextPart

--StartOfNextPart
Content-Type: text/plain
Content-Transfer-Encoding: 7bit
John,
      Here is the photo of our research lab that I promised
      to send you. You can see the equipment you donated.

Thanks again,
Bill

--StartOfNextPart
Content-Type: image/gif
Content-Transfer-Encoding: base64
...data for the image...

```

图 26.6 MIME 混合多部分报文的例子。报文的每个部分可以有独立的内容类型

26.10 小结

电子邮件是使用最广泛的应用服务之一。与大多数 TCP/IP 服务类似，它服从客户—服务器模型。邮件系统将外发的和传入的邮件进行缓存，允许客户和服务器的传送活动在后台进行。

TCP/IP 协议族分别为邮件报文格式和邮件传送提供了独立的标准。邮件报文格式称为 2822，使用空行分隔邮件的首部和主体部分。简单邮件传送协议（SMTP）定义了一台机器上的邮件系统如何将邮件传送到另一台机器上的服务器上。邮局协议版本 3（POP3）和网际报文访问协议版本 4（IMAP4）指明了用户如何取回邮箱中的内容。它们允许用户在一台具有连续不中断的因特网连接的计算机上拥有一个永久的邮箱，并允许用户从一台不具有连续因特网连接的计算机上访问邮箱中的内容。

多用途网际邮件扩充（MIME）提供了使用 SMTP 传送任意数据的机制。MIME 在电子邮件首部中增加了几行，用来定义数据类型和所用的编码。MIME 的混合多部分类型允许一个报文中含有多种数据类型。

26.11 深入研究

本章描述的协议都在因特网 RFC 中有定义。Klensin [RFC 2821] 描述了简单邮件传送协议，并给出了许多例子。Resnick [RFC 2822] 给出了邮件报文的确切格式。许多 RFC 都对此进行了补充并且在不断变化着。Freed and Borenstein [RFCs 2045, 2046, 2047, 2048, 2049] 定义了 MIME 标准，包括首部声明的语法、创建新内容类型的过程、内容类型的解释以及本章提到的 base64 编码格式。Dierks and Allen [RFC 2246] 定义了 SMTP 使用的运输层安全扩充。

26.12 习题

1. 一些邮件系统要求用户指明邮件到达目的地所需经过的机器序列。每台机器中的邮件协议仅仅将邮件传给下一台机器。举出这种方案的三个缺点。
2. 看看你的计算机系统是否允许直接调用 SMTP。

-
3. 构造一个 SMTP 客户并使用它传送一个邮件报文。
 4. 尝试能否通过一个邮件转发器发送邮件并返回给你自己。
 5. 为你所在网点处理的邮件地址格式列一张表，并写出一组分析地址的规则。
 6. 了解如何把 Linux 系统配置成作为一个邮件转发器。
 7. 考察你的本地邮件系统多长时间尝试交付一次，连续尝试多长时间后才会放弃交付？
 8. 许多邮件系统允许用户指定将收到的邮件直接交给程序，而不是存放在邮箱中。试构造一个用于接受收到邮件的程序，它将你的邮件放入一个文件，然后发送回执，告诉发送方你正在休假。
 9. 仔细阅读 SMTP 标准。然后使用 TELNET 连接到远程机器上的 SMTP 端口，并请求远程 SMTP 服务器扩展邮件别名。
 10. 用户收到了一个邮件，该邮件的 To 字段指明为字符串 important-people。在此邮件的发送机器上，别名 important-people 中没有任何有效的邮箱标识符。仔细阅读 SMTP 规范，看看为什么可能发生这种情况。
 11. POP3 把邮件取回和删除分离开，允许用户取回和查看邮件，但并不将其从永久邮箱中删除。这种分离方式的优缺点分别是什么？
 12. 阅读有关 POP3 的资料，其中的 TOP 命令是怎样操作的，为什么这个命令很有用？
 13. 阅读有关 IMAP4 的资料。当多个并发的客户同时访问某个给定的邮箱时，IMAP4 是如何保证其一致性的？
 14. 仔细阅读 MIME RFC。在 MIME 外部引用中可以指定什么服务器？

第27章 万维网（HTTP）

27.1 引言

本章继续探讨使用 TCP/IP 技术的应用，重点讨论一个已成为最具影响力的应用：**万维网**（World Wide Web，简称 WWW）。本章在简单介绍其概念后，将讨论把万维网页面从服务器传送到万维网浏览器时所用的基本协议。我们在讨论中不仅要涉及基本传送机制，也会涉及到缓存机制。

27.2 万维网的重要性

在因特网的早期发展历史中，FTP 数据传送大约占了因特网通信量的三分之一，比其他任何应用都要多。但是从 20 世纪 90 年代初开始，万维网的发展非常迅速，到 1995 年，万维网通信量已经超过 FTP，成为因特网主干带宽的最大使用者，并自此一直保持着领先地位。

万维网的影响力不是单从通信量统计数据上就能完全了解的。与任何其他因特网应用相比，知道和使用万维网的人是最多的。事实上，对于大多数用户而言，因特网就是万维网。

27.3 体系结构组成

从概念上讲，万维网由通过因特网能访问到的大量文档集合构成，这些文档称为**万维网页面**（web page）。一个万维网页面就其分类来说就是一个**超媒体**（hypermedia）文档。在 hypermedia 一词中，后缀 media 表示文档还可以包含除文本以外的其他项目（如图形图像），而之所以使用前缀 hyper，是因为文档中可以包含指向其他相关文档的**可选择链接**（selectable link）。

在因特网上实现万维网需要使用两个主要构建模块。**万维网浏览器**（web browser）由一个应用程序构成，用户调用该程序访问和显示万维网页面。浏览器作为客户联络相应的**万维网服务器**（web server），得到指定页面的副本。由于一个给定的服务器可以管理多个万维网页面，所以在请求时浏览器必须指明确切的页面。

万维网页面使用的数据表示标准取决于万维网页面的内容。例如，包含单幅图形图像的页面可以使用 Graphics Interchange Format（GIF）或 Joint Picture Experts Group（JPEG）之类的标准图形表示法。混合了文本和其他项目的页面用超文本标记语言（HyperText Markup Language，简称 HTML）来表示。HTML 文档由包含文本以及内嵌式命令的文件组成，内嵌式命令称为**标记**（tag），用来指导显示方式。标记用小于号（<）和大于号（>）括起来。有一些标记需要成对使用，并对位于一对标记之间的所有项都起作用。例如，<CENTER>和</CENTER>这两个命令使标记之间的所有项在浏览器窗口中显示时都需要居中对齐。

27.4 统一资源定位符

每个万维网页面都被分配了一个唯一的名称，用来标识该页面。这个名称称为统一资源定位符（Uniform Resource Locator，简称 URL）^①，它的开头部分是用于访问该项的**方案**（scheme）说明。

^① URL 是更为通用的**统一资源标识符**（Uniform Resource Identifier，简称 URI）的特殊类型。

实际上，方案指明了传送协议。URL 其余部分的格式则与具体的方案有关。例如，遵守 http 方案的 URL 具有以下形式^①：

`http://hostname[:port]/path[;parameters][?query]`

其中，斜体字部分表示提供的项，方括号代表可选项。现在，知道 `hostname` 字符串指明的是作为服务器的那台计算机的域名或 IP 地址就足够了，`:port` 是可选的协议端口号，只在服务器不使用熟知端口（80）的情况下才使用这一选项。`path` 是标识了服务器上某个具体文档的字符串，`;parameters` 是可选的字符串，指明由客户提供的可选参数；`?query` 是当浏览器发送询问时使用的可选字符串。用户未必能直接看到或使用其中的可选部分。事实上，用户输入的 URL 只包含 `hostname` 和 `path`。例如：

<http://www.cs.purdue.edu/people/comer/>

这个 URL 指的是作者在普度大学的万维网页面。该服务器运行在计算机 www.cs.purdue.edu 上，文档的名称为 `/people/comer/`。

协议标准对前面介绍的**绝对**（absolute）URL 形式和**相对**（relative）URL 形式加以区分。用户很少看到相对 URL，只有在已经确定了服务器时，相对 URL 才有意义。一旦与某个服务器已经建立了通信，相对 URL 就很有用了。例如，在与服务器 www.cs.purdue.edu 通信时，只需要用字符串 `/people/comer/` 就可以指定前面用绝对 URL 命名的文档。归纳如下：

每个万维网页面都分配有一个唯一的标识符，即统一资源定位符（URL）。URL 的绝对形式包含完整的说明，而 URL 的相对形式则省略了服务器的地址。只有在已知服务器时，相对形式才有用。

27.5 示例文档

下面将举例说明如何从文档的**可选择链接**（selectable link）中产生 URL。对于这样的链接，文档中都包含了一对值：在屏幕上显示的项以及用户选择该项后跟随的 URL。在 HTML 中，`<A>` 和 `` 这对标记被称为**锚**（anchor），用于定义链接。将 URL 添加到第一个标记中，而将要显示的项放在两个标记之间。例如，下面的 HTML 文档中包含了一个可选择链接：

```
<HTML>
The author of this text is
<A HREF="http://www.cs.purdue.edu/people/comer">
Douglas Comer.</A>
</HTML>
```

当显示文档时，在屏幕上出现一行文本：

The author of this text is Douglas Comer.

浏览器在 `Douglas Comer` 下加了下划线，表示它对应于可选择链接。浏览器在内部存储了 `<A>` 标记中的 URL，当用户选定链接时浏览器会转到该 URL。

27.6 超文本传送协议

在浏览器和万维网服务器或中间机器和万维网服务器之间进行通信时所使用的协议称为**超文本传送协议**（HyperText Transfer Protocol，简称 HTTP）。HTTP 有以下几个特点：

- **应用层**。HTTP 在应用层上操作。它假设一种可靠的面向连接的运输层协议，如 TCP，而它本身不提供可靠性或重传机制。

^① 有些文献把起始字符串 `http` 称为**注记**（pragma）。

-
- **请求 / 响应**。一旦建立了运输会话，一端（通常是浏览器）必须向另一端发送 HTTP 请求，并由另一端响应。
 - **无状态**。每个 HTTP 请求都是自包容的，服务器不保留以前的请求或会话的历史记录。
 - **双向传输**。在大多数情况下，由浏览器请求万维网页面，而服务器则把页面的副本传送给浏览器。但是，HTTP 也允许从浏览器到服务器的传送（如用户提交所谓的“表单”时）。
 - **能力协商**。HTTP 允许浏览器和服务器协商一些细节，如在传送中使用的字符集。发送方可以指明它提供的能力，而接收方则指明它接受的能力。
 - **支持缓存**。为了减少响应时间，浏览器将它接收的每个万维网页面的副本放入缓存。如果用户再次请求该页，HTTP 允许浏览器询问服务器，确定自从缓存之后该页的内容是否已经改变。
 - **支持中介**。HTTP 允许在从浏览器到服务器之间的路径上的某台机器作为**代理服务器**（Proxy Server），代理服务器将万维网页面放入缓存并从中响应浏览器的请求。

27.7 HTTP 的 GET 请求

在最简单的情况下，浏览器直接联络万维网服务器，以得到一个页面。浏览器从 URL 开始，提取出主机名部分，使用 DNS 把主机名映射成相应的 IP 地址，然后用 IP 地址建立到服务器的 TCP 连接。一旦形成了 TCP 连接，浏览器和万维网服务器就使用 HTTP 进行通信，浏览器发送请求以读取某个页面，服务器通过发送页面的副本进行响应。

浏览器发送 HTTP GET 命令从服务器上请求一个万维网页面^①。这个请求由一个文本行构成，以关键字 GET 开头，然后是 URL 以及 HTTP 版本号。例如，要从服务器 www.cs.purdue.edu 得到上面例子中的万维网页面，浏览器可以发送如下请求：

GET <http://www.cs.purdue.edu/people/comer/HTTP/1.1>

一旦建立了 TCP 连接，就不需要发送绝对 URL，下面的相对 URL 读取的是同一页：

GET /people/comer/HTTP/1.1

归纳如下：

在浏览器和万维网服务器之间使用超文本传送协议（HTTP）。浏览器向服务器发送 GET 请求，服务器则通过发送被请求的内容作为响应。

27.8 错误消息

当万维网服务器接收到非法请求时应该如何响应呢？在大多数情况下，请求由浏览器发送，然后该浏览器试图显示服务器返回的任何内容。因此，服务器经常在有效的 HTML 中生成错误消息。例如，一个服务器生成了以下错误消息：

```
<HTML>
<HEAD> <TITLE>400 Bad Request</TITLE>
</HEAD>
<BODY>
  <H1>Bad Request</H1> Your browser sent a request
  that this server could not understand.
</BODY>
</HTML>
```

^① 标准使用了面向对象的术语**方法**（method），而不是**命令**（command）。

文档的“首部”（也就是在<HEAD>和</HEAD>之间的条目）是浏览器内部使用的，对用户只显示“正文”。标记<H1>和</H1>使浏览器把 Bad Request 作为标题显示出来（即大写并粗体显示），结果在用户的屏幕上得到两行输出结果：

Bad Request

Your browser sent a request that this server could not understand.

27.9 持久连接和长度

HTTP 的第一个版本遵循与 FTP 一样的模式，即为每个数据传送使用一个新的 TCP 连接。客户打开一条 TCP 连接，发送一个 GET 请求。服务器传送被请求的项目的副本，然后关闭 TCP 连接。因此，客户会一直从 TCP 连接读取数据，直到遇到**文件尾** (end of file)，然后客户关闭它这一端的连接。

HTTP 的版本 1.1 从根本上改变了基本 HTTP 模式。版本 1.1 并不是为每个传送使用一条 TCP 连接，而是采用**持久连接** (persistent connection) 作为默认方法。也就是说，一旦客户打开了和特定服务器之间的一条 TCP 连接，客户就让该连接在多个请求和响应过程中一直存在。当客户或服务器准备关闭连接时，则通知另一端，然后关闭该连接。

持久连接的主要优点在于减少额外开销。TCP 连接越少，就意味着响应时间、底层网络上的额外开销、缓冲区使用的内存和使用 CPU 的时间就会更少。我们可以用**管道化** (pipelining) 请求来进一步优化使用持久连接的浏览器（也就是一个接一个地发送请求，不必等待响应）。在必须为某个页面取回多幅图片，而底层互联网又有较高的吞吐量和较长的延迟时，管道技术特别有用。

使用持久连接的主要缺点在于，需要标识通过连接发送的每一项的开头和结尾。处理这种情况有两种可能的技术：或者先发送数据的长度，然后再发送数据项，或者在数据项后面发送一个**标记值** (sentinel value)，标记数据项的结束。HTTP 不能预先保留一个标记值，这是因为传送的数据项包括图形图像，图像可以包含任意八位组序列。因此，为了避免标记值和数据之间的多义性，HTTP 采用了先发送长度，然后发送具有该长度的数据项的方法。

27.10 数据长度和程序输出

在发送之前让服务器知道数据项的长度可能不太容易，甚至根本不可能。要理解其原因，读者必须知道有些万维网页面是根据请求生成的。也就是说，服务器使用了像**公共网关接口** (Common Gateway Interface，简称 CGI) 这样的机制，以允许服务器上运行的计算机程序动态地创建万维网页面。当到达的请求对应的是 CGI 生成的页面时，服务器运行相应的 CGI 程序，将程序的输出作为响应返回给客户。动态万维网页面的生成使创建的信息具有时效性（例如，运动项目比赛中当前分数的列表），但这意味着服务器不能提前知道准确的数据大小。此外，由于以下两个原因，在发送数据之前把数据保存到文件中也是不可取的：它使用服务器的资源，并且延迟了传送。因此，为了提供动态万维网页面，HTTP 标准指明，如果服务器预先不知道数据项的长度，那么服务器可以通知浏览器，它将在传送完项目之后关闭连接。归纳如下：

为了允许 TCP 连接在多个请求和响应中持久存在，HTTP 要在每个响应前先发送长度。如果服务器不知道数据的长度，它就要通知客户，发送响应，然后关闭连接。

27.11 长度编码和首部

服务器发送长度信息时使用什么样的表示法呢？有趣的是，HTTP 借用了电子邮件的基本格式，

使用 2822 格式和 MIME 扩充^①。与标准的 2822 报文类似，每个 HTTP 传送都包含一个首部、一个空行和要发送的数据项。此外，首部中的每一行都包含一个关键字、一个冒号和信息。图 27.1 列出了几种可能的首部以及它们的含义。

Header	Meaning
Content-Length	Size of item in octets
Content-Type	Type of the item
Content-Encoding	Encoding used for item
Content-Language	Language(s) used in item

图 27.1 首部中可能出现的项的例子。Content-Type 和 Content-Encoding 直接取自 MIME

作为一个例子，考虑图 27.2，它显示了通过持久 TCP 连接传送 HTML 文档时使用的几个首部。

```
Content-Length: 34
Content-Language: en
Content-Encoding: ascii

<HTML> A trivial example. </HTML>
```

图 27.2 HTTP 传送示意图，包括用于指明属性的首部行、空行和文档本身。如果是持久连接，需要给出 Content-Length 首部

除了图中所示的例子以外，HTTP 还包括各种各样的首部，以允许浏览器和服务器交换元信息。例如，我们曾说过，如果服务器不知道数据项的长度，那么服务器会在发送完数据项后关闭连接。但是，服务器不会不事先警告就采取行动，它会通知浏览器，希望关闭连接。为此，服务器在数据项之前包含了一个 Connection 首部，以代替 Content-Length 首部：

Connection:close

当浏览器接收到一个 Connection 首部时，它就知道服务器要在传送后关闭连接，禁止浏览器继续发送更多的请求。下一节将描述其他首部的作用。

27.12 协商

除了指明有关正在发送的数据项的详情以外，HTTP 还使用首部来允许客户和服务器**协商**（negotiate）能力。可协商的一组能力包括有关连接（例如，访问是否是经过授权）、表示法（例如，是否接受 jpeg 格式的图形图像或使用哪种类型的压缩格式）、内容（例如，文本文件是否必须是英文）以及控制（例如，页面保持有效的时间长度）等各个方面的特征。

有两种类型的协商：**服务器驱动**（server-driven）和**代理驱动**（agent-driven）（即浏览器驱动）。服务器驱动是从浏览器发出请求开始的。在指明所要求数据项的 URL 的同时，请求也会指明首选列表。服务器从可用的表示法中选出符合浏览器首选要求的一项。如果有多项符合条件，则服务器进行“最佳猜测”。例如，如果一个文档以多种语言存储，而请求指定首选英文，那么服务器发送英文版的文档。

代理驱动协商意味着浏览器用两步过程完成选择。首先，浏览器向服务器发送请求，询问可用的内容，服务器返回可能的内容列表。浏览器从中选择一个，发送第二个请求，以获得该数据项。

^① 参见第 26 章对电子邮件和 2822 格式以及 MIME 的讨论。

代理驱动协商的缺点是要求与服务器进行两次交互，优点是浏览器得到了对选项的完全控制权。

浏览器使用 HTTP Accept 首部指明哪种媒体或表示法是可接受的。在首部中列出了格式的名称以及分配给每个名称的首选值：如：

Accept: text/html, text/plain; q = 0.5, text/x-dvi; q = 0.8

这个例子指明浏览器希望接受 text/html 媒体类型，但是如果该类型不存在，则浏览器将接受 text/x-dvi，并且如果 text/x-dvi 也不存在，则接受 text/plain。第二条以及第三条相关联的数值可以看成是优先级（preference level），不存在 $q = 1$ 的值， $q=0$ 意味着不接受该类型。对于“quality”有意义的媒体类型（如音频）来说，如果某个媒体类型是其他形式在品质上减少 $q\%$ 后最好的可用类型，可以把 q 值解释为接受该媒体类型的意愿值。

有多种 Accept 首部，分别对应于前面描述的 Content 首部。例如，浏览器可以发送以下任一项：

Accept-Encoding:

Accept-Charset:

Accept-Language:

它们分别用于指定编码、字符集和浏览器希望接受的语言。

归纳如下：

HTTP 使用类似于 MIME 的首部运载元信息。浏览器和服务器都要发送首部，以允许它们对希望得到的文档的表示法和使用的编码进行协商。

27.13 条件请求

HTTP 允许发送方有条件地进行请求。也就是说，当浏览器发送请求时，它包括的首部可以限制在何种条件下才应该响应请求。如果不符合指定的条件，则服务器不返回请求的数据项。条件请求通过避免不必要的传送，从而允许浏览器优化取回过程。If-Modified-Since 请求指明了一个最简单的条件，它允许浏览器避免传输数据项，除非数据项自指定的日期后已经更新。例如，浏览器的 GET 请求可以包括首部：

If-Modified-Since: Fri, 01 Apr 2005 05:00:01 GMT

如果数据项比 2005 年 1 月 1 日还要早，则会避免传送。

27.14 代理服务器和缓存

代理服务器是万维网体系结构中的一个重要组成部分，因为它提供了优化机制，能够降低等待时间并减少服务器的负担。有两种形式的代理服务器存在：**非透明的**（nontransparent）和**透明的**（transparent）。正如它们的名字所暗示的，非透明服务器对用户来说是可见的，也就是说，用户必须配置浏览器来联络一个代理，而不是与原始资源所在的服务器联络。透明代理不需要对浏览器的配置进行任何改动。事实上，透明代理会检查所有通过代理的 TCP 连接，并拦截到达 80 端口的任何连接。无论是哪一种情况，代理都会将万维网页面进行缓存，然后从缓存中取出页面，以回答后来的请求。

HTTP 包括对代理服务器的显式支持。协议确切指出代理服务器如何处理每个请求，代理服务器应该如何解释首部，浏览器如何与代理服务器协商，代理服务器又如何与服务器协商。此外，已经设计了几个专门用于代理服务器的 HTTP 首部。例如，有一个首部用于代理向某个服务器鉴别自己的身份，另一个首部用于处理过数据项的每个代理服务器都记录下自己的身份，因而最终的接收者就可以收到一份所有中介代理的列表。最后，HTTP 允许服务器控制代理对每个万维网页面的处理。例如，服务器可以在响应中包括 Max-Forwards 首部，以限制在交付给浏览器之前，处理数据

项的代理服务器的数目。如果服务器指定代理的数目为 1，如：

Max-Forwards:1

那么从服务器到浏览器沿途中，最多只有一个代理可以处理数据项。数字为 0 时禁止任何代理处理数据项。

27.15 缓存

缓存的目的是提高效率：通过消除不必要的传送，缓存减少了等待时间和网络通信量。缓存最明显的特点是存储。在第一次访问一个万维网页面时，由浏览器或中介代理，或者由这两者将该万维网页面的副本存储在磁盘上。当以后请求同一页面时，可以简化查找过程，从缓存而不是从服务器上得到万维网页面的副本。

所有缓存方案的中心问题都和时限有关：“数据项应在缓存中保留多长时间呢？”一方面，保留缓存的副本时间太长会使副本变得陈旧，这意味着对原页面的改动没有反映在缓存的副本中。另一方面，如果保留缓存的副本时间不够长，效率就会降低，这是因为下一个请求又必须回到服务器。

HTTP 允许服务器以两种方式控制缓存。首先，当服务器响应对万维网页面的请求时，可以指定缓存的细节，包括该页面能否被缓存，能否由代理缓存该页面，哪些人可以共享缓存的页面，缓存副本的到期时间，以及可应用于该副本的转换限制等。其次，HTTP 允许浏览器强制万维网页面 **重新确认** (revalidation)。为此，浏览器发送对该页面的请求，使用首部指明其最大年龄（也就是自从存储万维网页面的副本以来的时间）不能大于 0。没有一个缓存中的副本能够满足这个请求，因为副本都有一个非 0 的年龄。这样，只有由初始服务器来响应该请求。沿途的中介代理将接收到新的副本放入缓存，而发出请求的浏览器也会接收到新的副本。

归纳如下：

缓存是万维网高效操作的关键。HTTP 允许服务器控制能否缓存页面、如何缓存页面以及页面的生存时间。浏览器可以绕过缓存，强制请求页面，从拥有该页面的服务器上得到新的副本。

27.16 其他 HTTP 功能

我们前面对 HTTP 的描述完全集中在读取方面，也就是说，一个客户（通常是一个浏览器）发出 GET 请求，从一个服务器上读取万维网页面的副本。但是 HTTP 还包括一些其他功能，允许客户和服务器之间进行更复杂的交互。实际上，HTTP 还提供了 PUT 和 POST 方法，以允许客户向服务器发送数据。这样才可能创建一段脚本，提示用户输入 ID 和口令，然后把结果传送到服务器。

出乎意料的是，虽然它允许任一方向的传送，但是底层的 HTTP 协议仍然是无状态的（即在交互期间不要求保持一条持久的运输层连接）。因此，通常需要用额外的信息来协调一系列的传送过程。例如，服务器在响应一个 ID 和口令时，它可能会发送一个称为 cookie 的用于识别的整数，并在客户的后续传送中返回。

27.17 HTTP、安全性和电子商务

虽然 HTTP 定义了一种用来访问万维网页面的机制，但是它并没有提供安全性。因此，用户在进行要求传送诸如信用卡号之类的信息的万维网购物时，必须确保其交易是安全的。这里有两方面的问题：传送数据的保密性和对提供商品的卖方万维网网站的鉴别。我们将在第 30 章中看到，为了确保其保密性，使用的是加密方法。另外，有一种授权机制可用于鉴别商家。

目前已经开发出一种用于万维网交易的安全技术。该技术称为 SSL 上的 HTTP (HTTP over SSL, 简称 HTTPS)，它是在安全套接字层 (SSL) 上运行的 HTTP 协议。HTTPS 解决了与电子商务相关的两个问题：因为它们是加密的，所以数据传送是保密的；因为 SSL 使用了证书树，所以商家是经过鉴别的。

27.18 小结

万维网是由存储在一组万维网服务器上并通过浏览器来访问的超媒体文档组成。每个文档都分配有一个唯一标识了该文档的 URL。URL 指明了用来检索文档的协议、服务器的位置以及文档在该服务器上的路径。

超文本标记语言 (HTML) 允许文档中含有文本以及内嵌的用于控制格式的命令。HTML 还允许文档中包含其他文档的链接。

浏览器和服务器使用超文本传送协议 (HTTP) 进行通信。HTTP 是应用层协议，显式支持协商、代理服务器、缓存和持久连接。有一种相关的技术称为 HTTPS，它利用 SSL 提供了安全的 HTTP 通信。

27.19 深入研究

Berners-Lee [RFC 1630] 和 Berners-Lee et. al. [RFC 1738] 定义了 URL。有各种各样的 RFC 含有关于扩展的提议。Mealling [RFCs 3401, 3402, 3403, 3404] 考虑了如何在域名系统中存储 URL。

Connolly and Masinter [RFC 2854] 含有目前版本的 HTML 的标准。

Fielding et. al. [RFC 2616] 定义了 HTTP 的 1.1 版本，它添加了许多以前没有的特性，包括另外增加了对持久连接和缓存的支持。Franks et. al. [RFC 2617] 考虑了在 HTTP 中的访问鉴别。

目前许多与万维网相关的标准都被一个称为**万维网协会** (World Wide Web consortium) 的商业群体所控制，包括除 HTML 以外的一些标记语言。有关信息可从以下地址找到：

www.w3c.org

27.20 习题

1. 阅读 URL 的标准。在 URL 末尾出现的带“#”的字符串表示什么含义？
2. 继续扩展前一个问题。把带有“#”后缀的 URL 发送给万维网服务器是合法的吗？试解释原因。
3. 浏览器如何区别包含 HTML 的文档和包含任意文本的文档。为了得出结论，可用浏览器读取一个文件进行试验。浏览器是使用文件名还是文件内容来确定如何解释该文件的？
4. HTTP TRACE 命令的目的是什么？
5. HTTP PUT 命令和 HTTP POST 命令之间的区别是什么？分别在什么时候有用？
6. 什么时候使用 HTTP 保活 (Keep-Alive) 首部？
7. 任意万维网服务器都能作为代理服务器吗？为了得出结论，可选择一个任意的万维网服务器，然后把浏览器配置为使用它作为代理。结果令你感到惊讶吗？
8. 下载并安装 squid 透明代理缓冲。squid 使用操作系统的什么设施来缓存万维网页面。
9. 阅读了解 HTTP 的 must-revalidate 缓存控制指示符。给出应该使用这个指示符的万维网页面的例子。
10. 如果浏览器在请求前没有发送一个 HTTP Content-Length 首部，服务器会如何响应？

11. 阅读有关 HTTPS 的资料，并解释 HTTPS 对缓存技术的影响。当使用 HTTPS 时，在什么情况下代理可以对万维网页面进行缓存？

第28章 IP 上的话音和视频传输（RTP, RSVP 和 QoS）

28.1 引言

本章重点介绍如何在 IP 网络上传输话音和视频之类的实时数据。除了讨论传输此类数据所使用的协议之外，本章还要考虑两个范围更广的问题。首先探讨的是商业 IP 电话服务所需的协议和技术，其次将探讨 IP 网络中的路由器如何才能保证足够高的服务质量，以提供高质量的视频和音频再现。

28.2 数字化和编码技术

在话音或视频能够通过分组网络发送之前，必须使用一些硬件设备将模拟信号转换为数字形式，这些硬件设备称为**编码 / 解码器**（coder/decoder，简称 codec）。最常见的编码器是**波形编码器**（waveform coder），它以一定的时间间隔测量输入信号的振幅，并把每个采样值转换成数字值（即整数值）^①。在接收端，解码器接受整数序列输入，重新建立与数字值相匹配的连续模拟信号。

目前存在多种数字编码标准，主要是在再现品质的高低和数字表现的大小之间取得不同的平衡。例如，常规的电话系统使用**脉码调制**（Pulse Code Modulation，简称 PCM）标准，指明每 125 微秒（即每秒 8000 次）采样一个 8 位值。结果得到的数字电话数据的速率是 64 kbps。PCM 编码会产生惊人的输出量，存储 128 秒的音频剪辑需要 1 MB 内存。

有三种方法可以降低数字编码产生的数据量：每秒采集较少的样值、每个样值使用较少的位编码，或者使用数字压缩方案来减小最后的输出。目前存在的各种系统都使用了这三种技术中的一种或几种，因此才可能看到一些生成音频编码速率只有 2.2 kbps 的产品。但是每项技术都有其缺点。采集较少的样值或者使用较少位编码的主要缺点是音频品质较低，系统不能再现大范围内的声音。压缩的主要缺点是延迟。如果经过了压缩，就必须先还原数字输出。另外，压缩得越厉害，要求的处理过程就越多，所以最佳的压缩或者要求 CPU 的速度很快，或者会带来较长的延迟。因此，当延迟不重要（如编码 / 解码器的输出将存储在文件中）时，压缩是最有用的方法。

28.3 音频和视频的传输及再现

许多音频和视频应用程序被归类为**实时**（real-time）的，因为它们要求及时地传输和交付^②。例如，交互式的电话通话就是实时的交换，因为音频的交付必须没有很明显的延迟，否则用户会认为系统令人不满意。及时传输不仅仅意味着延迟时间要短，这是因为最后得到的信号除非与初始信号的顺序完全相同，并且时序也完全相同，否则仍无法理解。因此，如果发送方每 125 微秒采样一次，那么接收方必须以完全相同的速率把数字值转换成模拟量。

网络如何保证数据流在交付时的速率和发送方使用的速率完全一样呢？常规的电话系统引进了这样一种解决方案：**等时的**（isochronous）体系结构。等时的设计意味着必须把整个系统，包括数字电路，设计成交付输出信号的时序和产生输入信号所用的时序完全一样。因此，任意两点之间

^① 另外还有一种可替代的设备是**话音编码 / 解码器**（或称声码器）（voice coder/decoder，简称 vocodec），用于对人的话音，而不是普通波形进行识别和编码。

^② 及时性比可靠性更重要，数据有丢失可以简单地跳过去。

有多条路径的等时系统在设计时必须使所有路径具有完全相同的延迟。

IP 互联网不是一个等时系统。前面已经介绍过，数据报可以重复、延迟或无序到达。无规律的延迟称为**抖动** (jitter)，这在 IP 网络中尤其普遍。要想让通过所谓的 IP 网络传输和再现的数字信号具有实际意义，就需要额外协议的支持。为了处理数据报的重复和无序交付，每个传输必须包含一个序号。为了处理抖动，每个传输必须包含一个**时间戳** (timestamp)，告诉接收方应该何时回放分组中的数据。使用独立的序号和时间信息能够使接收方准确地重建信号，而不依赖于分组的到达状况。当数据报丢失或者当发送方在沉默期间停止编码时，这些时间信息就非常重要，它允许接收方在回放时暂停由时间戳指定的一段时间。总结如下：

因为 IP 互联网不是等时系统，所以在发送数字化的实时数据时需要额外的协议支持。除了一些用来检测重复或对分组重排序的基本序号信息之外，每个分组还必须携带一个独立的时间戳，用于告诉接收方分组中的数据应该在何时播放。

28.4 抖动和回放延迟

如果网络中有抖动，接收方应该如何准确地重建信号呢？接收方必须实现一个**回放缓冲区** (playback buffer)^①，如图 28.1 所示。

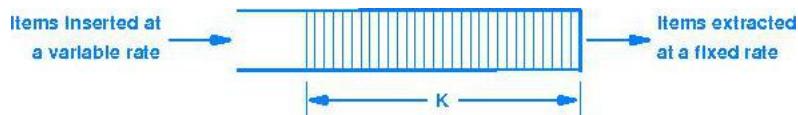


图 28.1 用于补偿抖动的回放缓冲区的概念组织图。缓冲区存储数据的时间长度为 K 个时间单位

当会话开始时，接收方先要推迟回放，将传入的数据存放在缓冲区中。当缓冲区中的数据达到预先约定的称为**回放点** (playback point) 的阈值时，输出开始。在图中标记为 K 的回放点表示要播放的数据有 K 个时间单位长。这样，当接收方已经累积了 K 个时间单位长度的数据时，回放开始。

在回放的同时，数据报连续不断地到达。如果没有抖动，则新数据抵达的速率和提取以及播放旧数据的速率完全一样，意味着缓冲区总是准确地包含 K 个时间单位的未播放数据。如果有一个数据报经历了短时间的延迟，回放也不受影响。在数据提取时，缓冲区的大小稳定地递减，回放可以保持 K 个时间单位不被打断。当延迟的数据报抵达时，重新填满缓冲区。

当然，回放缓冲区并不能补偿丢失的数据报。在这种情况下，回放最终会达到缓冲区中未填写的那个位置，而输出则会暂停相应于丢失数据的一段时间。此外，K 值要在丢失和延迟之间折中选取^②。如果 K 太小，那么一个较小的抖动就会使系统在需要的数据抵达之前耗尽回放缓冲区。如果 K 太大，那么系统会免受抖动影响，但是在底层网络传送延迟的基础上再添加额外的延迟，对用户而言延迟就非常明显了。除了这些缺点以外，大多数通过 IP 互联网发送实时数据的应用程序都会依赖回放缓冲区作为解决抖动的主要方案。

28.5 实时运输协议

在 IP 互联网上传输数字音频或视频信号所用的协议称为**实时运输协议** (Real-Time Transport Protocol, 简称 RTP)。有趣的是，RTP 不包含确保及时交付的机制，而必须由底层系统来保证。RTP

^① 回放缓冲区也称为**抖动缓冲区** (jitter buffer)。

^② 虽然可利用网络延迟和抖动而动态确定 K 的值，但是很多回放缓冲区机制使用的是一个恒量 K 值。

提供了两个关键特性：每个分组中有一个序号，以允许接收方检测无序交付或丢失，另外还有一个时间戳，以允许接收方控制回放。

因为设计 RTP 是为了让它传送包括音频和视频等广泛的实时数据，所以 RTP 不强制统一的语义解释。事实上，每个分组以一个固定的首部开头，首部中的字段指明了如何解释其余的首部字段以及如何解释有效载荷。图 28.2 所示为 RTP 固定首部的格式。

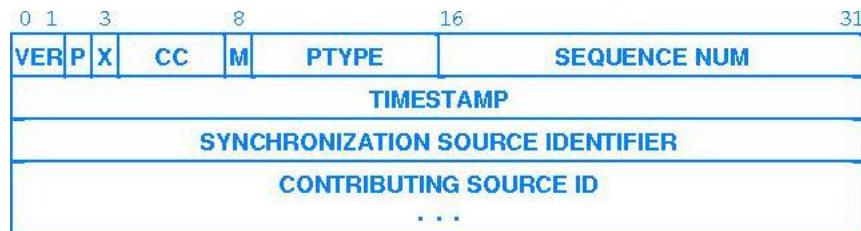


图 28.2 RTP 使用的固定首部格式图示。每个报文都以这样的首部开头，其中对字段的具体解释以及需要附加哪些首部字段，都取决于有效载荷类型（PTYPE）字段

如图 28.2 所示，每个分组的开始是版本（VER）字段，其中包含的是 2 位 RTP 版本号，当前的版本号为 2。在 16 位序号（SEQUENCE NUM）字段中包含了分组的序号。在具体的某个会话过程中，第一个序号是随机选择的。一些应用程序定义了可选的首部扩展，放在固定首部和有效载荷之间。如果应用类型允许扩展，则使用 X 位来指明分组中是否存在扩展。对首部中的大多数其他字段的解释依赖于指明了有效载荷类型的 7 位 PTY 字段。P 位指明在有效载荷后是否需要补零填充。当要求把数据分配在固定大小的块中，以便进行加密时，会用到 P 位。对 M（代表 marker，标记）位的解释取决于应用程序，应用程序如果需要标记数据流中的点（例如，在发送视频时每帧的开头），就会使用 M 位。

有效载荷的类型也会影响对时间戳（TIMESTAMP）字段的解释。时间戳是一个 32 位的值，它给出了数字数据的第一个八位组的采样时间，会话的初始时间戳是随机选择的。标准规定时间戳必须连续递增，即使在没有检测到信号和没有发送值期间也一样，但是它没有规定具体的粒度。事实上，粒度是由有效载荷类型决定的，这意味着每个应用程序可以选择自己的粒度，以允许接收方用正好适合应用程序的时间精度在输出中放置数据项。例如，如果通过 RTP 传输的是音频数据流，那么时钟每滴答一下就采样一次的逻辑时间戳粒度是合适的^①。但是，如果正在传输的是视频数据，那么为了得到平滑的回放效果，时钟每滴答一下可能需要采样多次。无论在什么情况下，如果这两个分组中的数据是同时采样的，标准允许两个分组中的时间戳完全相同。

28.6 数据流、混合和多播

RTP 的关键之处在于它对转换（translation）（即在中间站改变数据流的编码）或混合（mixing）（即从多个源站接收数据流，把它们合并成一个数据流，然后发送）的支持。想要理解为什么需要混合，可以设想位于多个不同网点上的人参加一个 IP 电话会议。为了尽量减少 RTP 流的数量，他们可以指派一个混合器（mixer），并设法让每个网点建立一条到达混合器的 RTP 会话。混合器将这些音频数据流合并起来（很可能是先把它们转换回模拟信号，再对最后得到的结果信号重新采样），然后把得到的结果用一个数字数据流发送出去。

RTP 首部中的字段对发送方进行了标识，并指示是否发生过混合。同步源标识

^① 为了强调时间粒度依赖于所度量的信号的类型，TIMESTAMP 有时也称为 MEDIA TIMESTAMP。

(SYNCHRONIZATION SOURCE IDENTIFIER) 字段指明了数据流的源。每个源必须选择一个唯一的 32 位标识符，如果发生冲突，协议包括解决冲突的机制。如果经过混合器合并了多个数据流，混合器就变成新的数据流的同步源。但是，有关初始源的信息并没有丢失，这是因为混合器使用了变长的参与源 ID (CONTRIBUTING SOURCE IP) 字段，提供被混合的数据流的同步源 ID。4 位的 CC 字段给出参与源的数目，最多可以列 15 个源。

RTP 设计可用于 IP 多播，并且它的混合功能在多播环境中特别具有吸引力。要理解其原因，可以设想一次有很多参与者的电话会议。单播要求发送站向每个参与者发送一份 RTP 外发分组的副本。但是如果用多播，发送站只需要发送一个分组副本，将它交付给所有的参与者。此外，如果使用了混合，则所有的源都可以通过单播到达混合器，混合器将其合并成一个数据流，然后进行多播。这样，混合技术和多播技术的结合能够使交付给每个参与主机的数据报数量大大减少。

28.7 RTP 封装

RTP 这个名字暗示了它是一个运输层协议。当然，如果 RTP 像常规的运输协议一样工作，那么它将要求把每个报文直接封装到 IP 数据报中。但事实上 RTP 并不像运输协议那样工作，实际应用中不会出现在 IP 中直接封装的情况，虽然这样做并没有什么错。相反，RTP 运行在 UDP 之上，这意味着每个 RTP 报文被封装到一个 UDP 数据报中。使用 UDP 的主要优点在于并发性，也就是说，在一台计算机上可以有多个使用 RTP 的应用程序，并且不会互相干扰。

与以前介绍的许多应用协议不同，RTP 没有使用保留的 UDP 端口号。相反，每次会话使用的端口都需要经过分配，并且必须把端口号告诉远程应用程序。根据习惯，RTP 会选择偶数的 UDP 端口号。下一节将要讨论使用下一个端口号的伴生协议 RTCP。

28.8 RTP 控制协议

迄今为止，对实时传输的描述都集中在使接收方能够重现内容的协议机制上。不过，实时传输还有另一个方面的内容也同样重要：在会话期间监视底层网络以及提供端点之间的带外 (out of band) 通信。在使用自适应方案的情形下这样的机制尤其重要。例如，当底层网络变得拥挤时，应用程序可以选择较窄的带宽编码，或者当网络延迟或抖动变化时，接收方可能需要改变回放缓冲区的大小。最后，可以利用带外机制，发送与实时数据并行的信息（如伴随视频流的字幕）。

作为 RTP 的伴生协议及其不可分割的一部分，**RTP 控制协议**(RTP Control Protocol, 简称 RTCP) 提供了所需的控制功能。RTCP 允许发送方和接收方之间相互传输一系列报告，其内容是有关正在传送的数据以及网络性能的额外信息。RTCP 报文封装在 UDP 中进行传输^①，并且发送时使用的端口号比它们所属的 RTP 流的端口号大 1。

28.9 RTCP 操作

图 28.3 所示为用于让发送方和接收方之间交换有关会话信息的五种基本报文类型。发送方若想关闭一条数据流，就传输一个 bye (结束) 报文。**application specific** (应用程序特定) 报文定义了一种新的报文类型。例如，应用程序为了伴随视频流发送结尾字幕信息，可以选择定义一种新的 RTCP 报文。

^① 因为有些报文很短，标准允许多个 RTCP 报文合并成一个 UDP 数据报进行传输。

Type	Meaning
200	Sender report
201	Receiver report
202	Source description message
203	Bye message
204	Application specific message

图 28.3 五种 RTCP 报文类型。每条报文以标识了类型的固定首部开头

接收方会定期传输**接收方报告** (receiver report) 报文，向源站汇报它的接收状态。这些报文很重要，原因有两个。首先，它们不仅允许所有接收方都参与到一个会话中，而且也让发送方了解到这些接收方的接收状态。其次，它们使接收方能够调整报告的速率，避免使用过多的带宽或淹没了发送方。自适应方案保证全部的控制通信总量不超过实时数据通信量的 5%，接收方报告报文不超过控制通信量的 75%。每个此类报文都标志出一个或多个同步源，并且为每个源包含一个独立的分区，每个分区都指明了已经从该源接收到的分组的最高序号、分组丢失的累计数量和百分比、自上一个 RTCP 报告从源站抵达以来的时间以及两次到达之间的抖动。

发送方会定期传输**发送方报告** (sender report) 报文，提供一个绝对时间戳。为了理解为什么需要时间戳，可以回忆一下，RTP 允许每个数据流为自己的时间戳选择一个粒度，第一个时间戳是随机选择的。发送方报告报文中的绝对时间戳是必要的，因为它是唯一能够向接收方提供对多个数据流进行**同步** (synchronize) 的机制。实际上，因为 RTP 要求每种媒体类型都有一个单独的数据流，所以发送视频及其伴随的音频需要两个数据流。绝对时间戳的信息允许接收方同时播放两个数据流。

发送方除了要周期性地传输发送方报告报文以外，还要传输**源描述** (source description) 报文，以提供有关拥有或控制源的用户的常规信息。在每个报文中，每个外发的 RTP 数据流都有一个相应的分区，它的内容是人能读懂的。例如，其中唯一的一个必填字段由数据流拥有者的规范名称 (canonical name) 构成。字符串的形式如下：

user@host

其中 host 是计算机的域名或点分十进制格式的 IP 地址，user 是登录名。在源描述信息中可选的字段包含更进一步的详细信息，如用户的电子邮件地址（可能和规范名称不同）、电话号码、网点的地理位置、创建数据流所用的应用程序或工具，以及其他有关源的文本注解。

28.10 IP 电话和信令

实时传输有一个方面的用途特别重要：将 IP 作为电话服务的底层技术。这种思想称为 **IP 电话** (IP telephony) 或 **IP 语音传输** (voice over IP)，已被许多电话公司实现。问题是，在 IP 可以代替现有的等时电话系统之前，还需要哪些额外技术？虽然答案不会很简单，但是研究人员目前正在研发三个组件。第一，我们已经知道要想正确地通过 IP 互联网传输数字信号，需要使用一个类似 RTP 的协议。第二，需要一种机制以建立和终止电话通话。第三，研究人员正在寻找使 IP 互联网具有等时网络的类似功能的方法。

在电话行业中用术语**信令** (signaling) 来表示建立电话通话的过程。特别地，在传统的**公共交换电话网** (Public Switched Telephone Network, 简称 PSTN) 中使用的信令机制是 **7 号信令** (Signaling System 7, 简称 SS7)。SS7 在发送任何音频之前先要执行呼叫例程。在拨出电话号码之后，SS7 在网络上形成一个电话回路，使被叫电话振铃，当电话被接听时连通该回路。SS7 还要处理一些诸如呼叫转移和错误状况（如被叫电话忙）等细节问题。

在可以使用 IP 进行电话通话之前，必须能够使用信令功能。此外，为了让电话公司采用，IP 电话必须和现有的电话标准兼容，必须让 IP 电话系统和传统的电话系统在所有级别上都能进行互操作。因此，除了必须在 IP 使用的话音编码和标准 PCM 编码之间做到互相转换之外，还必须做到在 IP 使用的信令和 SS7 之间相互转换。结果是这两种信令机制将具有同样的功能性。

取得这种互操作性的常规方法是在 IP 电话系统和传统电话系统之间使用一个网关 (gateway)。网关的任意一侧都可以发起呼叫。当请求信令抵达时，网关对这个请求进行转换和转发，网关也必须对返回的响应进行转换和转发。最后，在信令完成且通话建立后，网关必须双向转发话音，把这一侧使用的编码转换成另一侧使用的编码。

已经有两个工作小组提出了 IP 电话的标准。ITU 定义了一个协议族，称为 H.323，而 IETF 提议的信令协议称为会话发起协议 (Session Initiation Protocol，简称 SIP)。下一节将概要描述这两种方法。

28.10.1 H.323 标准

ITU 最初创建 H.323 是为了在局域网技术上实现话音的传输。现在这个标准已经扩展到允许在 IP 互联网上传输话音，并且电话公司很有可能会采用该标准。H.323 不是单独的一个协议，而是指明了多个协议应当如何组合起来，形成一个实用的 IP 电话系统。例如，除了网关之外，H.323 还定义了一个称为网闸 (gatekeeper) 的设备，每个网闸为使用 IP 的电话提供一个联络点。为了获得呼出的许可，并且为了使电话系统能够将传入的呼叫送到正确的目的站，每个 IP 电话必须向一个网闸注册，H.323 中包括了必要的协议。

除了为传输实时话音和视频而指定的一个协议以外，H.323 框架结构还允许参与者传送数据。因此，参加音频—视频会议的一对用户之间还可以共享屏幕上的白板、发送静止的图像或交换文档的副本。

图 28.4 列出了 H.323 所依赖的四个主要协议。

这个协议族里的协议涵盖了 IP 电话的各个方面，包括电话注册、信令、实时数据编码和传送（音频和视频）以及控制。

图 28.5 举例说明了组成 H.323 的协议之间的关系。如图所示，整个协议族最终依赖于 IP 上运行的 UDP 和 TCP。

Protocol	Purpose
H.225.0	Signaling used to establish a call
H.245	Control and feedback during the call
RTP	Real-time data transfer (sequence and timing)
T.120	Exchange of data associated with a call

图 28.4 H.323 的 IP 电话所用的协议

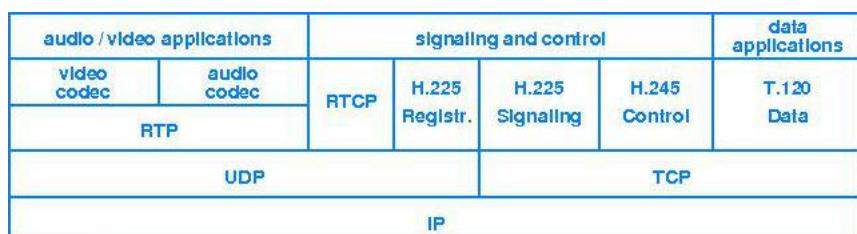


图 28.5 组成 ITU 的 H.323 IP 电话标准的协议之间的关系。省略了用于处理像安全性和 FAX 传输这样的细节问题

的协议

28.10.2 会话发起协议

IETF 已建议了另一种可以代替 H.323 的协议，称为会话发起协议（SIP），它只涉及信令，不推荐具体的编码 / 解码器，也不要求对实时传输使用 RTP。因此，SIP 没有提供 H.323 的所有功能。

SIP 使用客户—服务器交互方式，并且把服务器分成两种类型。**用户代理服务器**（user agent server）在一个 SIP 电话中运行，它分配有一个标识符（如 user@site），可以接收传入的通话。第二种类型的服务器是中介服务器（也就是在两个 SIP 电话之间的服务器），用于处理如建立通话和转发通话这样的任务。中介服务器的功能类似于**代理服务器**（proxy server），能够将传入的通话请求转发到通向被叫电话的路径上的下一个代理服务器，或者类似于**重定向服务器**（redirect server），告诉呼叫者如何达到目的地。

为了提供有关通话的信息，SIP 要依靠一个伴生协议：**会话描述协议**（Session Description Protocol，简称 SDP）。SDP 在电话会议中尤其重要，因为参与者动态地加入和离开通话。SDP 指明了一些细节，如媒体编码、协议端口号和多播地址。

28.11 服务质量之争

服务质量（Quality of Service，简称 QoS）这个术语指的是网络系统对丢失、延迟、吞吐量和抖动等方面在统计性能上的保证。等时系统因为在设计时要满足严格的性能条件，所以说它提供了 QoS 保证，而使用尽最大努力交付的分组交换网络被认为是不提供 QoS 保证的。通过 IP 传送实时语音和视频时需要 QoS 保证吗？如果需要，应该如何实现呢？主要的争论围绕着两个问题展开。一方面，设计电话系统的工程师坚持认为，长途电话的高品质语音再现需要底层系统为每个电话通话提供延迟和丢失的 QoS 保证。另一方面，设计 IP 的工程师则坚持认为没有 QoS 保证的因特网工作得也无可挑剔，并且在因特网上增加每流量的 QoS 是行不通的，因为路由器上的处理会使系统昂贵并且速度减慢。

伴随着 QoS 引发的争论已经诞生了不少提议、实现方案和实验。虽然因特网的操作不具有 QoS，但因特网已经用于音频的发送了。从电话系统模型中衍生出的技术（如 ATM）可以为每个单独的连接或单独的流提供 QoS 保证。IETF 在对定义了每流量服务质量的**综合服务**（Integrated Services，简称 IntServ）进行研究之后，决定采用保守的**区分服务**（Differentiated Services，简称 DiffServ）方法，把通信量划分为不同的 QoS 类。区分服务方案为简化转发过程而牺牲了更精细的粒度控制，有时称其为**服务类别**（Class of Service，简称 CoS）。

28.12 QoS 和利用率以及容量

对 QoS 的争论使人联想到早期对资源分配的争论，如对内存分配和处理器调度的操作系统策略展开的争论。它们的中心问题都是利用率。当网络有足够的资源满足所有的通信时，QoS 限制就根本没有必要了。当通信量超出网络容量时，没有哪个 QoS 系统能满足所有用户的需求。也就是说，利用率为 1% 的网络不需要 QoS，而对于利用率为 101% 的网络，任何 QoS 都不起作用。实际上，QoS 机制的支持者声称，复杂的 QoS 机制能实现两个目标。首先，通过把现有资源在多个用户之间分配，可以让系统更“公平”。其次，通过对每个用户的通信量进行整形，可以允许网络以较高的利用率运行，而没有崩溃的危险。

反对复杂 QoS 机制的一个主要论点是从底层网络性能的提高得来的。网络容量已显著增加，只要容量继续保持高速增长，QoS 机制只不过代表了一些不必要的开销。但是，如果需求的增长超

过容量的增长，QoS 就有可能成为一件有利可图的事情，把较高的价格和较高级别的服务相联系，ISP 就可以根据价格分配容量。

28.13 综合业务资源预留

如果需要 QoS，IP 网络该如何提供它呢？IETF 在研究 IntServ 时，开发出两个能够提供 QoS 的协议：**资源预留协议**（Resource ReSerVation Protocol，简称 RSVP）和**公共开放策略服务**（Common Open Policy Services，简称 COPS）协议。这两个协议都要求改变基本的底层结构，一对端点之间的所有路由器必须为每个流量针对预留资源（例如带宽）问题达成一致意见。这包括两个方面。首先，在发送数据之前，端点必须发送指明所需资源的请求，沿途所有路由器必须同意提供该资源，这个过程可以看成是某种形式的信令。其次，在数据报经过某个流时，路由器需要监视和控制通信量的转发。这个监视操作有时称为**通信量管制**（traffic policing），它的存在确保了在流上发送的通信量没有超过指定的边界条件。有两个原因使得对排队和转发的控制成为必要。路由器必须实现一个排队策略，以满足延迟限制的保证，另外路由器还必须平滑分组的突发。有时把平滑分组的突发称为**通信量整形**（traffic shaping），因为网络通信量经常是突发性的，所以通信量整形很有必要。例如，指定平均吞吐率为 1 Mbps 的数据流可能在某一毫秒内有 2 Mbps 的通信量，而在下一毫秒又完全没有通信量。路由器可以让传入的数据报暂时排队等待，并以 1 Mbps 的稳定速率发送它们，以此消除突发现象。

RSVP 处理预留请求和回答。它既不是路由传播协议，也不强制在流量建立时就启用策略。事实上，它只需在发送任何数据之前运行。为了初始化一个端对端的流量，端点首先发送 RSVP 的路径（path）报文，以确定到达目的地的路径。运载报文的数据报利用路由器报警（routeralert）选项来保证路由器检查该报文。在接收到对路径报文的回答后，端点发送请求（request）报文，为流量预留资源。该请求指明所要求的 QoS 边界条件，在通往目的地的沿途中对请求进行转发的每个路由器都必须同意预留请求所指定的资源。如果沿途有任何路由器拒绝请求，那么路由器会使用 RSVP 向源站发送一个拒绝回答。如果沿途的所有系统都同意请求，则 RSVP 返回肯定回答。

每个 RSVP 流都是**单工**（simplex）（即单向传输）的。如果一个应用程序需要双向的 QoS 保证，那么两个端点都必须使用 RSVP 分别请求一个数据流。因为 RSVP 使用现有的转发机制，所以既不能保证两个方向上的流量能通过同样的一组路由器，也不是说其中一个方向的流量被批准就意味着另一个方向上的流量一定会被批准：归纳如下：

一个端点使用 RSVP 请求在 IP 互联网上建立一条具有指定 QoS 条件的单工流量。如果沿途所有路由器都接受请求，它们就会批准该流量，否则会拒绝。如果某个应用程序需要两个方向上的 QoS，那么每个端点必须使用 RSVP 请求独立的流量。

28.14 综合服务执行

当 RSVP 请求抵达时，路由器必须考虑两方面的内容：可行性（即路由器是否具有能够满足请求的资源）和策略（即请求是否符合策略限制条件）。可行性是一个本地决策，路由器可以确定如何管理带宽、内存和处理能力这些本地性能。但是策略的执行要求全局合作，也就是说，所有的路由器都必须同意相同的一组策略。

为了实现全局策略，IETF 体系结构使用了两级模型，并且两级之间的交互使用客户—服务器模式。当一个路由器接收到 RSVP 请求时，它就变成客户，与一个称为**策略判决点**（Policy Decision Point，简称 PDP）的服务器协商，以确定请求是否符合策略。PDP 不处理通信量，它只不过对请求进行评估，检查它是否符合全局策略。如果 PDP 认可请求，则路由器必须以**策略执行点**（Policy

Enforcement Point, 简称 PEP) 的身份进行操作, 确保通信量不会超出经过认可的策略。

COPS 协议定义了路由器和 PDP 之间的客户一服务器交互作用 (或者, 如果某个机构有多级策略服务器, 那就是路由器和本地 PDP 之间的客户一服务器交互作用)。虽然 COPS 定义了它自己的报文首部, 但是其底层格式的许多细节与 RSVP 共享。实际上, COPS 使用的格式与 RSVP 请求报文中的单个数据项的格式一样。这样, 当路由器接收到 RSVP 请求时, 它可以提取与策略相关的数据项, 把它们放在一个 COPS 报文中, 然后把结果发送给 PDP。

28.15 区分服务和每跳行为

第 6 章所描述的区分服务与综合服务之间有两个明显的不同之处。首先, 区分服务不是为每个流量, 而是为一个类 (即一组流量) 分配服务的。第二, 与 RSVP 方案的端对端预留不同, 区分服务允许沿途的每个结点为某个给定的类定义它将接收到的服务。例如, 路由器可以选择划分带宽。使称为**加速转发** (Expedited Forwarding, 简称 EF) 的区分服务类接收 50% 的带宽, 而剩余 50% 的带宽则在称为**确保转发** (Assured Forwarding, 简称 AF) 的类之间划分。但是, 沿途的下一个路由器可以将 90% 的带宽给 EF 类, 而 AF 类只能划分剩下的 10%。我们用短语每跳行为 (per-hop behavior) 来描述这种方式, 同时说明了区分服务不能提供端对端的保证的特点。

28.16 通信量调度

要实现任何一种形式的 QoS, 路由器都要为外发通信量分配优先级, 并选择在什么时间发送哪个分组。在一组分组中进行挑选的过程称为**通信量调度** (traffic scheduling), 而这个机制则称为**通信量调度器** (traffic scheduler)。在构造一个调度器时, 需要考虑以下四方面的内容:

- **公正性:** 调度器要保证一个流量所消耗的资源 (即带宽) 不能超出分配给该流量的范围^①。
- **延迟:** 给定流量中的分组不应当有过分的延迟。
- **自适应性:** 如果给定的流量没有分组需要发送, 调度器就应当把多余的带宽划分给其他流量, 并且要按照给它们分配资源的比例进行划分。
- **计算性额外开销:** 因为调度器是在快速通道上操作的, 所以它绝不能导致大量的计算性额外开销。实际上, 类似**通用处理器调度** (Generalized Processor Scheduling, 简称 GPS) 这种理论上的算法是不能使用的。

最简单且实用的通信量调度方案称为**权重循环** (Weighted Round Robin, 简称 WRR), 因为它给每个流量分配了一个权重, 并且从该流量发送数据时要依据它的权重。例如, 我们可以设想有三个流量 A, B 和 C, 它们的权重分别为 2, 2 和 4。如果这三个流量都有分组在等待发送, 那么调度器从流 C (权重为 4) 中发送的数据是流 A 和流 B (权重各为 2) 的两倍。

如果按照以下顺序选择流量, WRR 调度器好像能够达到要求的权重:

C C A B

也就是说, 调度器不断重复以下选择:

C C A B C C A B C C A B ...

这个选择式样似乎达到了所要求的权重, 因为 1/2 的选择来自流 C, 1/4 来自 B, 还有 1/4 来自 A。另外, 从整个序列来看, 它以规则的间隔时间为每个队列服务, 也就是说, 没有哪个流量会有不必要的延迟 (即从某个给定的流量发送分组的速率是恒定的)。

虽然上面列举的序列可以使分组率与分配的权重相匹配, 但是因为数据报没有统一的长度, WRR 方法还是不能达到让数据率与权重相匹配的目标。例如, 如果流 C 的数据报的平均长度是流

^① 这一节讨论的都是流量之间的调度。但是, 读者应当能够理解, 在使用区分服务时, 调度是针对类别之间的。

A 数据报平均长度的一半，那么以流 A 两倍的频率选择流 c 中的数据将会使这两个流量的数据率相等。

为了解决这个问题，人们设计了一种经过改进的算法，以适应可变长度的分组。这个算法称为**亏损循环**（Deficit Round Robin，简称 DRR），它以发送的总字节数而不是分组数来计算权重。该算法最初为每个流量分配的字节数与它应得的带宽成比例。当选择某个流量后，只要没有超过规定的字节数，DRR 算法会尽可能多地把分组传输出去。然后，DRR 算法会计算余数（即分配的字节数和实际发送的分组数之差），在下一个循环发送时就要加上这个余数。因此，DRR 总是为每个流量保持一个连续不断的“亏损”总额。即使在某个循环上获得的亏损额很小，但是经过几个循环后，这个值会不断增加，直至足够容纳另一个分组。这样，经过一段时间后，DRR 为某个流量发送数据的比例会近似于该流量的权重。

类似 WRR 和 DRR 这样的循环调度算法既有优点也有缺点。主要的优点在于其高效性，一旦分配好权重，对于如何选择分组只需要很少的计算，这也是它们之所以受欢迎的原因所在。事实上，如果所有分组都具有相同大小，且选择互为倍数的权重，那么用一个数组就可以达到根据权重选择分组的目标，而不需要通过计算。

尽管循环算法很受欢迎，但是它们确实有不尽人意的地方。首先，某个给定流量所经历的延迟与其他有通信量发送的流量的数目相关。在最糟糕的情况下，某个给定流量可能需要等待调度器从其他流量中发送完一个或多个分组。第二，由于循环算法在某个时刻集中从给定队列中发送分组，接着算法会为其他队列服务，而这个队列就要经历延迟，所以循环算法会带来抖动。

28.17 通信量管制

实现了通信量调度的系统同时也需要一个**通信量管制器**（traffic policer），它可以验证到达的通信量没有超出规定的统计参数。要理解为什么需要这样一种机制，可以考虑一个系统，它的调度器为区分服务类 Q 分配了 25% 的外出带宽。如果有三个进入流量都与类别 Q 相匹配，那么这些流量会竞争使用为 Q 分配的带宽。因此，系统必须对每个进入流量进行管制，以确保没有哪个流量超过它应得的那一份。

对于通信量管制已经建议了几种机制。有一种早期的机制是基于**漏桶**（leaky bucket）方法的，它使用了一个计数器来控制分组速率。该算法会定期递增计数器，每当有分组到达时，算法会递减计数器。如果计数器变为负数，就说明进入流量已经超过了分配给它的分组速率。

与前面指出的一样，使用固定的分组速率在因特网中是没有意义的，因为数据报的长度并不一致。因而，又建议了一种更复杂的管制方案，以适应变长的分组。例如，有一种**权标桶**（token bucket）机制，它将前面所介绍的方案进行扩展，使计数器对应于比特而不是分组。计数器根据要求的数据率定期递增，并且当有分组到达时递减的值正好是这个分组的长度。

在实际应用中，上述管制机制并没有要求一个定时器来定期更新计数器。相反，每次当一个分组到达时，管制器会检查时钟，判断该流量自上次处理后已逝去的时间，并用这个时间来计算计数器的递增值。

28.18 小结

一种称为编码 / 解码器的硬件单元将模拟信号（如音频）编码成数字形式。电话的数字音频编码标准是脉码调制（PCM），它以 64 kbps 的速率产生数字值。其他一些编码标准牺牲保真度，以达到较低的比特率。

RTP 用于通过 IP 网络传送实时数据。每个 RTP 报文中都包含了两条关键的信息：一个序号和一个媒体时间戳。接收方使用序号来按顺序放置报文并检测丢失的数据报，使用时间戳来确定何时

播放编码的值。还有一个相关的控制协议 RTCP，提供有关源的信息，并允许一个混合器合并多条数据流。

人们对提供实时传输时是否需要服务质量（QoS）保证的争论持续不休。在公布区分服务方法之前，IETF 开发了一个综合服务模型，并设计了一对可用于提供每流量 QoS 的协议。端点使用 RSVP 来请求具有指定 QoS 的流量，而中间的路由器可以批准请求，或者拒绝请求。当 RSVP 请求抵达时，路由器使用 COPS 协议联络策略判决点，并验证请求是否满足策略要求。

QoS 的实现需要一个通信量调度机制，以便从多个外发队列中选择分组，同时也需要通信量管制来监视进入的流量。亏损循环算法是一种流行的通信量调度算法，而漏桶算法则是一种流行的通信管制算法，原因都在于它们的计算效率高，并且能处理变长的分组。

28.19 深入研究

Schulzrinne et. al. [RFC 3550]给出了 RTP 和 RTCP 的标准。Perkins et. al. [RFC 2198]指明了通过 RTP 的冗余音频数据的传输，而 Schulzrinne [RFC 3551]指明了 RTP 在音频一视频会议中的使用。Schulzrinne, Rao, and Lanphier [RFC 2326]描述了实时通信量的数据流使用的相关协议。

Braden et. al. [RFC 1633]描述了综合服务，而 Zhang et. al. [RFC 2205]则包含 RSVP 的规范。Durham et. al. [RFC 2748]定义了 COPS，Herzog et. al. [RFC 2749]描述了 RSVP 下 COPS 的使用。Blake et. al. [RFC 2475]定义了区分服务，Heinanen et. al. [RFC 2597]讨论了确保转发的每跳行为，同时 Davie et. al. [RFC 3246]讨论了加速转发。Parekh and Gallager [1993, 1994]详细描述了通用处理器共享（Generalized Processor Sharing），而 Shreedhar and Varghese [1995]讨论了亏损循环调度。

28.20 习题

1. 阅读**实时数据流协议**（Real Time Streaming Protocol，简称 RTSP）。RTSP 和 RTP 之间的主要区别是什么？
2. 讨论：虽然常常把带宽引用为 QoS 机制可保证的性能的例子，但延迟才是更基础的资源。
提示：如果资金较为宽裕，你认为应该减缓哪个限制？
3. 如果抵达的 RTP 报文的序号远远大于希望的序号，协议该如何做？为什么？
4. 序号在 RTP 中是必要的吗？是不是可以用一个时间戳来代替它呢？试解释原因。
5. 你是否愿意让因特网对所有的通信量都要求 QoS？为什么？
6. 测量你的因特网连接的利用率。如果所有的通信量都要求 QoS 预留，服务会更好一些还是会更差一些？试解释原因。

第29章 网络管理 (SNMP)

29.1 引言

除了提供网络层服务的协议和使用这些服务的应用程序以外，还需要一个子系统，以允许管理员能够调试出现的问题、控制路由选择以及发现违反协议标准的计算机。我们把这些行为称为**网络管理**（network management）。本章将探讨 TCP/IP 网络管理所隐含的思想，并描述一个用于网络管理的协议。

29.2 管理协议的级别

最初，许多广域网将管理协议作为其链路层协议的一部分。如果一个分组交换机出现故障，网络管理员可以指示相邻的分组交换机发送一个特殊的**控制分组**（control packet）。控制分组使接收方暂停正常的运作，并响应来自管理员的命令。管理员可以查询分组交换机，以确定问题所在、检查或改变路由、测试某个通信接口或重启交换机。一旦故障修复，管理员就可指示交换机恢复正常运作。由于管理工具是底层协议的一部分，因而即使高层协议不能正常工作，管理员也能控制交换机。

因特网与同构的广域网不同，它的链路层协议不是单一的。事实上，因特网由多种物理网络类型组成，并且其中的设备来自多个不同的运营商。因此，因特网需要一个全新的网络管理模式，以提供下面三种重要性能。第一，由一个管理员就能控制所有的异构设备，包括 IP 路由器、网桥、调制解调器、工作站和打印机。第二，被管实体不需要共享相同的链路层协议。第三，管理员所控制的一组机器可能位于因特网的任意位置上。特别重要的是，管理员可能需要控制与自己的机器不在同一物理网上的一台或多台机器。因此，除非管理软件使用的是能够提供通过互联网的端到端连接的协议，否则管理员不可能与被管机器通信。其结果是，TCP/IP 网络的网络管理协议运行在运输层之上。

在 TCP/IP 互联网中，管理员需要检查和控制路由器以及其他网络设备。因为这些设备连接到任意网络，所以网络管理协议运行在应用层，并且利用 TCP/IP 运输层协议进行通信。

将网络管理软件设计在应用层工作有几个优点。由于协议可设计成与底层网络硬件无关，所以同一组协议可用于所有的网络。由于协议可设计成与被管机器的硬件无关，所以同一组协议可用于所有被管设备。从管理员的观点看，使用同一组管理协议意味着统一性，所有路由器响应完全相同的一组命令。此外，由于管理软件使用 IP 进行通信，管理员的机器不必直接连到每个物理网络或路由器上，就能够控制整个 TCP/IP 互联网上的路由器。

当然，在应用层构造管理软件也有缺点。除非操作系统、IP 软件和运输协议软件等都正常工作，否则管理员无法联络到需要实施控制的路由器。例如，如果一个路由器的路由表被破坏，可能没有办法将其纠正或从远程网点上重启动机器。如果路由器上的操作系统崩溃，即使路由器仍然能处理硬件中断和分组转发，也不可能联系到实现了网络管理协议的应用程序。

29.3 体系结构模型

尽管有潜在的缺点，但让 TCP/IP 管理软件运行于应用层的实际效果非常好。在大型的互联网上，管理员的机器不需要直接连到含有被管实体的所有物理网络，这就突出体现了将网络管理软件放在高层的优点。图 29.1 所示为这种体系结构的一个例子。

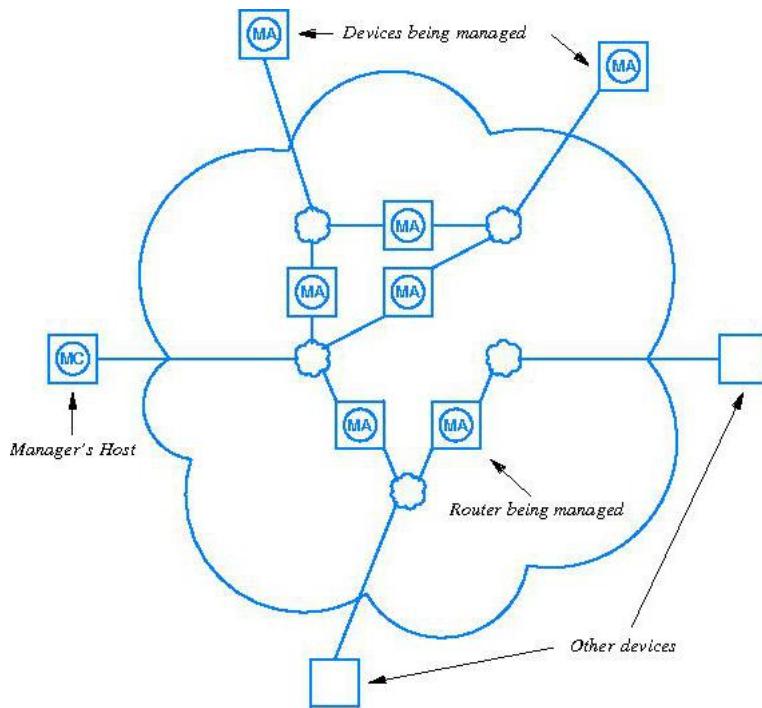


图 29.1 网络管理的一个例子。管理员激活管理客户 (MC) 软件，与在互联网设备上运行的管理代理 (MA) 软件联络

如图 29.1 所示，客户软件通常在管理员的工作站上运行。每个参与的网络系统都要运行一个服务器程序^①。从技术上讲，这个服务器软件称为**管理代理** (management agent) 或简称**代理** (agent)。管理员激活本地主机上的客户软件并指明与之通信的代理。当客户联系上服务器后，它发送询问以获取信息或发送修改路由器状态的命令。当然，在一个大型互联网中，并不是所有设备都由同一个管理员管理。大多数管理员只控制本地网点上的设备，一个大型网点可能有多位管理员。

网络管理软件使用了鉴别机制，以保证只有授权的用户才能访问或控制某个具体的设备。有些管理协议支持多级鉴别，允许管理员在每个设备上指定权限。例如，某个路由器可配置成允许几个管理员获取信息，同时只允许从这几个管理员中挑选出的一个子集来修改信息或控制路由器。

29.4 协议框架结构

TCP/IP 网络管理协议把管理问题分成了两大部分，并为每个部分分别定义了独立的标准。第一个部分与信息通信相关。协议指明了在管理员主机上运行的客户软件如何与代理进行通信。协议定义客户与服务器交换的报文的格式和含义，以及名字与地址的表现形式。第二部分是关于被管理的数据的，协议指明了被管设备必须保存的数据项和每个数据项的名称以及用于表示名称的语法。

^① 我们用术语“网络系统”(network system) 涵盖了类似路由器和台式计算机的常规设备以及打印机和传感器之类 的设备。

29.4.1 一个标准网络管理协议

网络管理的 TCP/IP 标准就是**简单网络管理协议**（Simple Network Management Protocol，简称 SNMP）。该协议已经发展了三代，相应地，现在的版本是 SNMPv3，以前的版本是 SNMPv1 和 SNMPv2。它们之间的变化不大：三个版本都使用相同的通用体系结构，并且许多特性都是向后兼容的。

除了指明像报文格式这样的细节以及运输协议的使用以外，SNMP 标准还定义了一组操作以及每个操作的含义。我们将会看到，它所使用的方法是最精简的，即用很少几个操作来提供所有的功能。

29.4.2 管理信息的标准

被管设备必须保存管理员可访问的控制和状态信息。例如，路由器保存有关网络接口状态、传入和外发分组通信量、丢失的数据报，以及产生的差错报文等这些统计信息，而调制解调器则保存有关发送和接收的比特数（或字符数）的统计信息。虽然 SNMP 允许管理员访问这些统计信息，但它并没有指明究竟在哪些设备上可以访问哪些数据。事实上，有另外一个独立的标准指明了每种设备类型的细节。这个标准就是**管理信息库**（Management Information Base，简称 MIB），它指明了被管设备必须保存的数据项、对每个数据项允许进行的操作及其含义。例如，IP 的 MIB 规定了软件必须保存从各个网络接口到达的所有八位组的总数，并规定网络管理软件只能读取该计数值。

TCP/IP 的 MIB 将管理信息划分为许多类。类别的选择是很重要的，因为用于指明数据项的标识符中要包含一个类别代码。图 29.2 列出了其中几个例子。

MIB category	Includes Information About
system	The host or router operating system
interfaces	Individual network interfaces
at	Address translation (e.g., ARP mappings)
ip	Internet Protocol software
icmp	Internet Control Message Protocol software
tcp	Transmission Control Protocol software
udp	User Datagram Protocol software
ospf	Open Shortest Path First software
bgp	Border Gateway Protocol software
rmon	Remote network monitoring
rip-2	Routing Information Protocol software
dns	Domain name system software

图 29.2 MIB 信息类别举例。类别被编码到指明一个对象的标识符中

使 MIB 的定义与网络管理协议无关，这对于运营商和消费者来说都是有好处的。运营商可以在产品（如路由器）中包含 SNMP 代理软件，并保证在定义新的 MIB 项之后该软件仍遵守标准。消费者可使用同一个网络管理客户软件来管理多个设备，这些设备的 MIB 版本稍有不同。当然，没有新的 MIB 项的设备不能提供这些项的信息。但是，由于所有被管设备使用同一种语言进行通信，所以对于任何一个查询它们都能进行分析，并提供被请求的信息或发送差错报文，以解释被请求的项尚不存在。

29.5 MIB 变量的例子

SNMP 的早期版本把所有变量都收集到一个很大的 MIB 中，并让这个完整的变量集收录在一个 RFC 中。为了避免 MIB 规范变得越来越笨拙，IETF 决定允许发布独立的 MIB 文档，每一个文

档指明特定设备类型的变量。其结果是，已经定义了 100 多个独立的 MIB 作为标准过程的一部分，这些 MIB 指明了 10,000 多个变量。例如，与以下这些设备相关的 MIB 变量目前都有独立的 RFC：硬件网桥、不间断电源、以太网交换机和 DSL 调制解调器。另外，许多运营商还为自己的硬件或软件产品定义了特定的 MIB 变量。

讨论几个和 TCP/IP 协议有关的 MIB 数据项将有助于阐明这些内容。图 29.3 列举了一些 MIB 变量及其类别。

MIB Variable	Category	Meaning
sysUpTime	system	Time since last reboot
ifNumber	interfaces	Number of network interfaces
ifMtu	interfaces	MTU for a particular interface
ipDefaultTTL	ip	Value IP uses in time-to-live field
ipInReceives	ip	Number of datagrams received
ipForwDatagrams	ip	Number of datagrams forwarded
ipOutNoRoutes	ip	Number of routing failures
ipReasmOKs	ip	Number of datagrams reassembled
ipFragOKs	ip	Number of datagrams fragmented
ipRoutingTable	ip	IP Routing table
icmplnEchos	icmp	Number of ICMP Echo Requests received
tcpRtoMin	tcp	Minimum retransmission time TCP allows
tcpMaxConn	tcp	Maximum TCP connections allowed
tcpInSegs	tcp	Number of segments TCP has received
udpInDatagrams	udp	Number of UDP datagrams received

图 29.3 MIB 变量及其类别举例

图 29.3 中列举的大多数项都是数值量，也就是说，每个值可存放在一个整数中。但是，MIB 也定义了更复杂的结构：例如，MIB 变量 ipRoutingTable 指的是一个完整的路由表。另外还有其他 MIB 变量定义了路由表中的项的内容，以允许网络管理协议引用表中的某一项，包括前缀、地址掩码和下一跳字段。当然，MIB 变量只给出每个数据项的逻辑定义，即路由器使用的内部数据结构可能与 MIB 的定义不同。当一个查询到达路由器时，路由器上的代理软件负责在 MIB 变量和路由器存储信息所用的数据结构之间进行映射。

29.6 管理信息的结构

除了指明 MIB 变量及其含义的标准以外，另外还有一个标准指明了一组定义和识别 MIB 变量的规则。这些规则称为**管理信息结构**（Structure of Management Information，简称 SMI）规范。为了使网络管理协议简单化，SMI 对 MIB 中允许的变量类型做出限制，指明对这些变量的命名规则，并创建定义变量类型的规则。例如，SMI 标准包括对一些术语的定义，如 IPAddress（定义为 4 八位组的字符串）和 Counter（定义为 $0 \sim 2^{32} - 1$ 的整数），并指明它们是用于定义 MIB 变量的一些术语。更为重要的是，SMI 中的规则描述了 MIB 如何指向由许多个值构成的表格（如 IP 路由表）。

29.7 使用 ASN.1 的正式定义

SMI 标准指明必须使用 ISO 的**抽象语法记法 1**（Abstract Syntax Notation 1，简称 ASN.1^①）定义和引用所有 MIB 变量。ASN.1 是一种正式语言，有两个主要属性：一个是在人们阅读的文档中使用的记法，另一个是在通信协议中使用的对于同一信息的紧凑编码表示形式。在这两种情况下，精确且正式的记法在表示和含义中去掉了任何可能的二义性。例如，使用 ASN.1 的协议设计者必

^① ASN.1 在读法上要把“.”读做“dot”，也就是“A-S-N dot 1”。

须说明数值量的准确格式和范围，而不是说某个变量含有一个整数值。当某个实现包含了异构计算机，而且这些计算机对数据项并不都使用相同的表示法时，这种严谨性就非常重要了。

除了指明每一项的名称和内容之外，ASN.1 还定义了一组基本编码规则（Basic Encoding Rules，简称 BER），用于详细指明这些名称和数据项在报文中应该如何编码。因此，一旦使用了 ASN.1 来表达 MIB 文档，其中的变量就能直接且机械地转换成报文中使用的编码格式。归纳如下：

TCP/IP 网络管理协议使用了称为 ASN.1 的正式记法，用来定义管理信息库中的变量名字和类型。精确的记法消除了变量在格式和内容上的二义性。

29.8 MIB 对象名的结构和表示法

我们已说过，ASN.1 指明了如何表示数据项及其名字。但是，要理解 MIB 变量使用的名字，就需要我们对底层名字空间有所了解。MIB 变量使用的名字取自 ISO 和 ITU 管理的**对象标识符**（object identifier）名字空间。对象标识符名字空间所隐含的一个关键思想就是由它提供的名字空间要对所有可能的对象都能进行分配。这个名字空间不限于网络管理中使用的变量，它包括任意对象的名字（如每个国际协议标准文档都有一个名字）。

对象标识符名字空间是绝对的（全局的），这意味着用它所构造的名字在全世界都是唯一的。与大多数庞大的、绝对的名字空间类似，对象标识符名字空间也是分级的。名字空间的管理权在每一级都被进一步细分，允许各级管理机构获得分配部分名字的管理授权，而不必在每次分配时向中心管理机构查询^①。

对象标识符分级树的根是没有名字的，但有三个直接后代，分别由 ISO 和 ITU 管理，或由 ISO 和 ITU 联合管理。为了便于识别，分别为这三个后代分配了一个较短的文本字符串和一个整数（文本字符串在人们需要理解对象名时使用；计算机软件则使用整数，以便形成名字的紧凑且编码化的表示法）。ISO 为其他国家或国际标准组织（包括美国标准组织）的使用分配了一棵子树，而美国国家标准和技术研究院^②为美国国防部分配了一棵子树。最后，IAB 已请求国防部在名字空间中为它分配一棵子树。图 29.4 所示为对象标识符分级树的相关部分，并显示出 TCP/IP 网络管理协议所用节点的位置。

在分级树中，一个对象的名字是从根到对象沿途各个节点的数字标号组成的数字序列。这个序列的书写方法是用点来分隔开各个组成部分。例如，名字 1.3.6.1.2 表示标号为 mgmt 的节点，即**因特网管理**（Internet management）子树。MIB 已被分配为 mgmt 子树下面标号为 mib 且数值为 1 的节点。由于所有 MIB 变量都位于该节点下面，所以它们的名字都有一个前缀 1.3.6.1.2.1。

^① 读者应当还记得，在第 23 章中讨论的域名系统如何对一个分级的名字空间进行细化授权管理。

^② 美国国家标准和技术研究院（U.S. National Institute for Standards and Technology）原名为美国国家标准局（National Bureau Of Standards）。

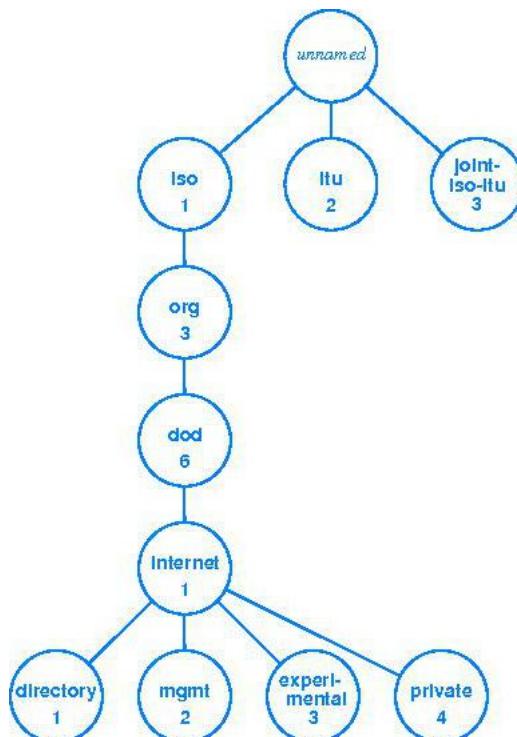


图 29.4 分级的对象标识符名字空间中用于命名 MIB 变量的部分。对象名由从根一直到该对象的沿途的数值标号组成

前面已提过 MIB 将变量划分为几类。现在可以解释这些类别的准确意义：这几个类别是对象标识符名字空间中 mib 节点下的子树。图 29.5 通过显示 mib 节点下的部分命名子树来说明这一概念。

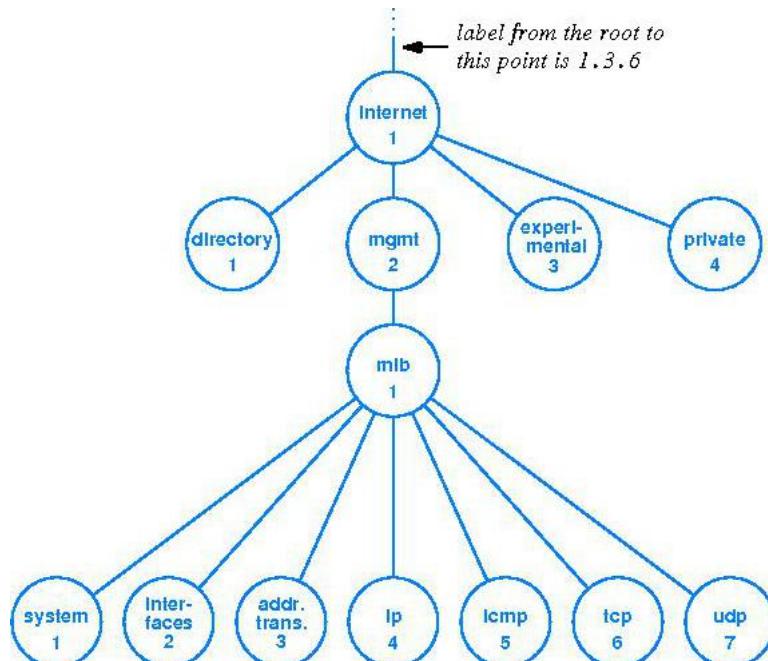


图 29.5 部分 IAB mib 节点下的对象标识符名字空间。每个子树对应 MIB 变量的一种类型

用两个例子就能说明命名语法。图 29.5 表明给标号为 ip 的类别分配的数字值是 4，因此，所有对应于 IP 的 MIB 变量的标识符都有前缀 1.3.6.1.2.1.4。如果想写出对象名的文本标号而不是用数值表示，它的名字为：

iso.org.dod.internet.mgmt.mib.ip

名字空间中 ip 节点下的一个名为 ipInReceives 的 MIB 变量被分配了数值标识符 3，因而该变量的名字为：

iso.org.dod.internet.mgmt.mib.ip.ipInReceives

相应的数值表示为：

1.3.6.1.2.1.4.3

当网络管理协议在报文中使用 MIB 变量时，每个变量名后面还要加一个后缀。对于简单变量，后缀 0 指具有该名字的变量的实例。因此，当它出现在发送给路由器的报文中时，ipInReceives 的数值表示就是：

1.3.6.1.2.1.4.3.0

它指向该路由器上的 ipInReceives 的实例。注意，完全无法推算分配给一个变量的数值或后缀。必须查询已发布的标准，找到分配给每个对象类型的数值。因此，程序完全通过查找一张对照表来提供文本格式与底层数值之间的映射，不存在能够完成转换的闭合式计算。

第二个例子较为复杂，考虑 MIB 变量 ipAddrTable，它含有每个网络接口的 IP 地址的列表。这个变量是在名字空间 ip 节点下的一棵子树，分配给该变量的数值为 20。因此，对它的引用需要前缀：

iso.org.dod.internet.mgmt.mib.ip.ipAddrTable

等价的数值表示为：

1.3.6.1.2.1.4.20

用编程语言的术语，我们认为 IP 地址表是个一维数组，数组中的每个元素都由一个结构（记录）组或，该结构含有 5 项：一个 IP 地址，一个与该项对应的接口的整数编号，一个 IP 子网掩码，一个 IP 广播地址和一个用来指明路由器重装数据报的最大长度的整数。当然，路由器的存储器中不可能有这样一个数组。路由器可以用许多变量来保存该信息，或者可能需要根据指针找到它。但是，MIB 为数组提供了一个名字，就好像数组存在一样，并允许各个路由器上的网络管理软件将**表引用**（table reference）映射为相应的内部变量。关键在于：

虽然 MIB 标准似乎指明了有关数据结构的具体细节，但是它们并没有规定其实现。相反，MIB 的定义提供的是一个统一的虚拟接口，让管理员用来访问数据。而代理则必须在 MIB 的虚拟项和内部实现之间进行转换。

使用 ASN.1 记法，我们可把 ipAddrTable 定义为：

ipAddrTable ::= SEQUENCE OF ipAddrEntry

其中的 SEQUENCE 和 OF 是关键字，它们将 ipAddrTable 定义为一个 ipAddrEntry 的一维数组。数组中每一项的定义由 5 个字段组成（假设 ipAddress 已经定义了）：

```
ipAddrEntry ::= SEQUENCE {
    ipAdEntAddr
        ipAddress,
    ipAdEntIfIndex
        INTEGER,
    ipAdEntNetMask
        ipAddress,
    ipAdEntBcastAddr
```

```
 ipAddress,  
ipAdEntReasmMaxSize  
    INTEGER(0..65535)  
}
```

进一步的定义必须为 ipAddrEntry 和 ipAddrTable 中的每一项分配一个数值。例如，定义：

```
ipAddrEntry {ipAddrTable 1}
```

指明 ipAddrEntry 位于 ipAddrTable 下并被分配了数值 1。类似地，定义：

```
ipAdEntNetMask {ipAddEntry 3}
```

表示 ipAdEntNetMask 在 ipAddrEntry 下并被分配了数值 3。

我们说过 ipAddrTable 像个一维数组。但是，程序员使用数组的方式和网络管理协议使用 MIB 中表的方式有很大区别。程序员将数组看成是一组元素的集合，并通过索引来选择某个元素。例如，程序员可能会写 xyz[3] 表示从数组 xyz 中选择了第三个元素。ASN.1 语法不使用整数索引。事实上，MIB 表通过在名字后面添加一个后缀来选择表中的某个元素。对于 IP 地址表的例子，标准规定用于表项选择的后缀是由一个 IP 地址形成的。依照语法，IP 地址（点分十进制表示）被连接到对象名末尾形成完整的引用名。因此，为指明 IP 地址表中对应于地址 128.10.2.3 的网络掩码字段，就要使用如下名字：

```
iso.org.dod.internet.mgmt.mib.ip.ipAddrTable.ipAddrEntry.ipAdEntNetMask.128.10.2.3
```

相应的数值表示为：

```
1.3.6.1.2.1.4.20.1.3.128.10.2.3
```

虽然在变量名末尾附加一个序号似乎比较麻烦，但它为客户搜索表提供了一个强有力的手段，使客户在搜索表时不需要知道表的总项数以及作为索引的数据类型。下一节将描述网络管理协议如何使用这一特性逐个遍历表。

29.9 简单网络管理协议

网络管理协议指明了由管理员激活的网络管理客户程序与主机或路由器上执行的网络管理服务器程序之间的通信。网络管理协议除了定义交换报文的格式和含义，以及这些报文中的名字和取值的表示法以外，还要定义被管路由器之间的管理关系。也就是说，它们为管理系统提供了鉴别机制。

用户可能会认为，网络管理协议中含有大量的命令。例如，在一些早期协议中，它们支持的命令可允许用户管理系统的重启、增加或删除路由、禁用或启用某个网络接口以及删除高速缓存中的地址绑定。围绕着命令来构建管理协议的主要缺点是它所导致的复杂性。协议需要对每个数据项的每个操作都有一个单独的命令。例如，删除路由表中某一项的命令不同于禁用接口的命令。因此，必须更改协议，以适应新的数据项。

SNMP 采用了另一种很有意思的网络管理方法：SNMP 的所有操作都遵守**取一存** (fetch-store) 模式^①，而不是定义了大量的命令。从概念上讲，SNMP 只含有两个命令，它允许管理系统读取一个数据项的值，或者把一个值存储到数据项中。所有其他操作都被定义为这两个操作的附带效果。例如，虽然 SNMP 没有一个显式的重启动操作，但可通过声明一个数据项，给出到下一次重启动前的时间，并且允许管理员为该数据项设置值（包括零），从而定义了与重启动等价的操作。

使用取一存模式的主要优点是稳定性、简单性和灵活性。SNMP 特别稳定，这是因为即使 MIB 增加了新的数据项，并为它定义了新的操作（这只是设置这些项的附带效果），SNMP 的定义仍然保持未变。SNMP 易于实现、理解和调试，这是因为它避免了区别对待每个命令所带来的复杂性。

^① 取一存模式是从称为 HEMS 的管理协议系统衍生而来的。详情可以参阅 Partridge and Trewitt [RFCs 1021, 1022, 1023, 1024]。

最后，SNMP 特别灵活，因为在一个优秀的框架结构中可以适应任意命令。

当然，从管理员的角度来看，SNMP 仍然是隐式的。网络管理软件的用户接口可以把操作看成是强制命令（如重启）。这样，管理员使用 SNMP 的方式与使用其他网管协议的方式之间几乎看不出差异。实际上，运营商已经在出售提供图形用户接口的网络管理软件。这种软件显示了网络连通图，并使用鼠标点击式的交互界面。

如图 29.6 所示，SNMP 提供的操作比上面描述的两种更多一些。

Command	Meaning
get-request	Fetch a value from a specific variable
get-next-request	Fetch a value without knowing its exact name
get-bulk-request	Fetch a large volume of data (e.g., a table)
response	A response to any of the above requests
set-request	Store a value in a specific variable
inform-request	Reference to third-part data (e.g., for a proxy)
snmpv2-trap	Reply triggered by an event
report	Undefined at present

图 29.6 一组可能的 SNMP 操作。get-next-request 操作使管理系统能够遍历表中各项

get-request 和 set-request 操作提供基本的读取和存储操作，response 提供对这些操作的响应。SNMP 规定操作必须是**原子的** (atomic)，也就是说，如果一个 SNMP 报文指明了对多个变量的操作，那么服务器或者完成所有操作，或者什么也不做。实际上，如果其中任何一个操作出错，则所有的操作都不会执行。trap 操作允许管理系统在事件发生时让服务器发送信息。例如，通过对 SNMP 服务器的编程，让 SNMP 服务器在所连的任何一个网络不可用时（即一个接口发生故障），就给管理系统发出一个 trap 报文。

29.9.1 用名字搜索表

我们说过，ASN.1 不提供通常意义上的数组声明和数组索引机制。但是，可以通过为表的对象标识符附加一个后缀来表示表中的元素。遗憾的是，对于没有掌握某个表的所有有效后缀的客户程序来说，它就需要逐项检查表中的各项。get-next-request 操作允许客户在不知道表中有多少项的前提下遍历表格。其规则相当简单，在客户发送 get-next-request 时，要提供一个有效的对象标识符的前缀 P。代理检查它所控制的所有变量的对象标识符集合，并为字典序大于 P 的下一个对象标识符发送 response 命令来响应请求。也就是说，代理必须掌握所有变量的 ASN.1 名字，并且能够选择出对象标识符大于 P 的第一个变量。由于 MIB 使用后缀对表进行索引，因此客户可以发送对应于表格的对象标识符前缀，然后收到表中的第一个元素。客户可以再发送表中第一个元素的名字，并收到第二个元素，依次类推。

让我们来看一个搜索的例子。前面讲过 ipAddrTable 使用 IP 地址来标识表中各项。客户如果不知道某个给定路由器的 ipAddrTable 表中有哪些 IP 地址，就不能形成一个完整的对象标识符。但是，客户可以使用 get-next-request 操作，通过发送下面的前缀来搜索该表：

iso.org.dod.internet.mgmt.mib.ip.ipAddrTable.ipAddrEntry.ipAdEntNetMask

它的数值表示为：

1.3.6.1.2.1.4.20.1.3

服务器返回 ipAddrTable 表中第一项的网络掩码字段。客户使用服务器返回的完整对象标识符再次请求表中的下一项。

29.10 SNMP 的报文格式

与大多数 TCP/IP 协议不同，SNMP 报文没有固定的字段。相反，它们使用标准 ASN.1 编码。因此，SNMP 报文用手工进行编码和理解时都比较困难。在研究 ASN.1 记法表示的 SNMP 报文的定义之后，我们将简述 ASN.1 编码方案，并列举一个 SNMP 报文编码的例子。

图 29.7 所示为 SNMP 报文如何用 ASN.1 记法描述。一般情况下，语法中的每一项由具有说明性的名字和对该项类型的声明组成，如下所示的项：

```
msgVersion INTEGER (0..2147483647)
```

声明 msgVersion 是一个小于等于 2,147,483,647 的非负整数。

```
SNMPv3Message ::=  
SEQUENCE {  
    msgVersion INTEGER (0..2147483647),  
    -- note: version number 3 is used for SNMPv3  
    msgGlobalData HeaderData,  
    msgSecurityParameters OCTET STRING,  
    msgData ScopedPduData  
}
```

图 29.7 SNMP 报文的 ASN.1 记法格式。两个短划线后面的文本是注释

如图 29.7 所示，每个 SNMP 报文由四个主要部分组成：一个标识了协议版本的整数、一个附加的首部数据、一组安全参数和一个运载有效载荷的数据区。除此之外，还必须为其中用到的每一项提供精确的定义。例如，图 29.8 举例说明如何定义 HeaderData（首部数据）区的内容。

```
HeaderData ::= SEQUENCE {  
    msgID INTEGER (0..2147483647),  
    -- used to match responses with requests  
    msgMaxSize INTEGER (484..2147483647),  
    -- maximum size reply the sender can accept  
    msgFlags OCTET STRING (SIZE(1)),  
    -- Individual flag bits specify message characteristics  
    -- bit 7 authorization used  
    -- bit 6 privacy used  
    -- bit 5 reportability (i.e., a response needed)  
    msgSecurityModel INTEGER (1..2147483647)  
    -- determines exact format of security parameters that follow  
}
```

图 29.8 SNMP 报文中对 HeaderData 区的定义

SNMP 报文中的数据区被划分为许多 **协议数据单元** (Protocol Data Unit, 简称 PDU)。每个 PDU 由一个请求（客户发送）或一个响应（代理发送）构成。SNMPv3 允许每个 PDU 以纯文本或加密的形式发送。因此，语法指明了一个 CHOICE。从编程术语看，这一概念称为**区别联合** (discriminated union)。

```
ScopedPduData ::= CHOICE {  
    plaintext ScopedPDU,  
    encryptedPDU OCTET STRING -- 加密的 ScopedPDU 值  
}
```

加密的 PDU 以产生它的引擎^①标识符开头。引擎 ID 后面跟着的是内容的名称以及加密报文的八位组。

^① 在 SNMPv3 中，使用了 SNMP 所支持服务的**应用程序** (application) 与**引擎** (engine) 不同，引擎是传输请求和接收响应的底层软件。

```

ScopedPDU ::= SEQUENCE {
    contextEngineID OCTET STRING,
    contextName OCTET STRING,
    data ANY          --例如下面定义的一个 PDU
}

```

在 ScopedPDU 的定义中，标为 data 的项为 ANY 类型，这是因为字段 contextName 定义了该项的具体细节。**SNMPv3 报文处理模型**（SNMPv3 Message Processing Model，简称 v3MP）指明了该数据必须由图 29.9 所示的 SNMP PDU 之一构成。

定义指明每个协议数据单元必须是八种类型中的一种。要对 SNMP 报文进行完整定义，还必须进一步定义八种独立类型的语法。例如，图 29.10 所示为 get-request 的定义。

```

PDU ::= CHOICE {
    get-request
        GetRequest-PDU,
    get-next-request
        GetNextRequest-PDU,
    get-bulk-request
        GetBulkRequest-PDU,
    response
        Response-PDU,
    set-request
        SetRequest-PDU,
    inform-request
        InformRequest-PDU,
    snmpV2-trap
        SNMPv2-Trap-PDU,
    report
        Report-PDU,
}

```

图 29.9 一个 SNMP PDU 的 ASN.1 定义。必须进一步指明每个请求类型的语法

```

GetRequest-PDU ::= [0]
IMPLICIT SEQUENCE {
    request-id
        Integer32,
    error-status
        INTEGER (0..18),
    error-index
        INTEGER (0..max-bindings),
    variable-bindings
        VarBindList
}

```

图 29.10 一个 get-request 报文的 ASN.1 定义。该报文正式定义为一个 GetRequest-PDU

在标准中还有更进一步的定义来指明其他未定义的项。其中，error-status 和 error-index 都是 1 八位组的整数，在请求中所含的值为零。如果有错误发生，则在响应中发送的值指示了发生错误的原因。最后，VarBindList 包含的是一个对象标识符的列表，客户希望查找它们的值。在 ASN.1 的术语中，定义指明 VarBindList 是对象的名字和值组成的序列，ASN.1 把这些名字和值的序偶对表

30 67 02 01 03
SEQUENCE len=103 INTEGER len=1 vers=3

示成两个项的序列³⁰因此在可能的最简单请求中，VarBindList 就是这样两个项的序列：一个名字和一个空值。

02 02 08 00
INTEGER len=2 maxmsgsize=2048

29.11 SNMP 报文编码举例

string len=1 msgFlags=0x04 (bits mean noAuth, noPriv, reportable)

ASN.1 的编码格式使用变长字段来表示各项。通常，每个字段以一个首部开头，指明对象的类型及其字节单位长度。例如，SEQUENCE 的开头都是一个八位组，其值为 30 (十六进制)，下一个八位组指明了组成 SEQUENCE 的八位组的数目。

图 29.11 是一个 SNMP 报文举例，演示了如何把值编码成八位组的方法。该报文是一个 get-request 报文，指明的数据项是 sysDescr (数字对象标识符为 1.3.6.1.2.1.1.1.0)。因为这个例子所示的是一个实际报文，其中包含了很多细节。² 特别是该报文包含了一个以前从未讨论过的 msgSecurityParameters 区。这个报文用 UsmSecurityParameters 来构成安全参数，然而该报文中的其他数据区都应当可以与前面的定义联系起来。

02	01	00	msgAuthoritativeEngineTime=0						
INTEGER			len=1						
04	09		43	6F	6D	65	72	42	6F
string	len=9		-----msgUserName value is "ComerBook"-----						
6F	6B								
<hr/>									
04	00		msgAuthenticationParameters (none)						
string	len=0								
04	00		msgPrivacyParameters (none)						
string	len=0								
30	2C		ScopedPDU						
SEQUENCE	len=44								
04	0C		00	00	00	63	00	00	
string	len=12		contextEngineID						
00	A1		c0	93	8E	23			
<hr/>									
04	00		contextName = "" (default)						
string	len=0								

图 29.11 数据项 sysDescr 的 SNMPv3 get-request 报文编码格式，其中各个八位组用十六进制表示，下方是含义注解。有关联的八位组按行分组，在报文中它们是连续的

如图 29.11 所示，报文的开始是一个 SEQUENCE 的编码，其长度为 103 八位组^①。序列中的第一项是 1 八位组的整数，指明了协议的 version，值为 3 表明这是一个 SNMPv3 报文。其后的字段定义了报文的 ID 以及发送方在接收回答时能够接受的最大报文长度。安全性信息，包括用户名（ComerBook），跟在报文首部的后面。

GetRequest-PDU 占据了报文的尾部。标为 ScopedPDU 的序列指明了如何解释报文的其余部分。八位组 A0 把操作指定为 get-Request。A0 的第 5 位表示这是一个非初级数据类型，而最高位代表八位组的解释是**特定于上下文的** (context specific)。也就是说，十六进制的值 A0 只在这个上下文中指明的是一个 GetRequest-PDU，而不是一个普遍意义上的保留值。紧跟这个请求八位组后，长度八位组指明这个请求的长度为 26 八位组。请求 ID 为 2 八位组，但差错状态和差错索引都为 1 八位组。最后，序偶序列包含的是一个绑定，即把一个对象标识符绑定到一个 null 值。标识符的编码和预期结果一样，只是前两个数字标号被合并成 1 八位组。

^④ 在 SNMP 报文中，序列项频繁出现，因为 SNMP 用 SEQUENCE 取代了 array 和 struct 之类的常规编程语言中的结构。

29.12 SNMPv3 的新特性

SNMP 版本 3 的出现和发展是对早期版本基本结构的遵循和扩展，其主要变化体现在安全性和管理领域。它有双重目标。首先，把 SNMPv3 设计成既具有常规安全策略，又具有灵活的安全策略，使得能够实现管理员和被管设备之间的交互作用，从而严格遵守由一个机构指定的安全策略。其次，系统被设计成能够容易地进行安全性管理。

为了获得通用性和灵活性，SNMPv3 包括几个安全方面的内容，并且允许单独配置每一项。例如，v3 支持**报文鉴别**（message authentication），保证由有权利的管理员来发出指令；支持**保密性**（privacy），保证在管理员的工作站和被管设备之间传递报文时没有其他人能读取报文；支持**授权**（authorization）和**基于视图的访问控制**（view-based access control），保证只有授权的管理员能访问特定的项。为了使安全系统易于配置和更改，v3 允许**远程配置**（remote configuration），这意味着授权的管理员能更改以上列出的安全项目的配置，而不需要物理地出现在该设备面前。

29.13 小结

网络管理协议允许管理员监督和控制路由器及主机。网络管理客户程序运行在管理员的工作站上，与运行于受控设备上的一个或多个服务器（称为代理）相联系。由于互联网是由异构机器和网络组成的，所以 TCP/IP 网络管理软件以应用程序的身份执行，并且在客户与服务器之间通信时使用 TCP/IP 运输协议（如 UDP）。

标准的 TCP/IP 网络管理协议是 SNMP（简单网络管理协议）。SNMP 定义了一个低层的管理协议，它提供两种基本操作：从一个变量读取值，或将一个值存储到变量中。在 SNMP 中，其他所有操作的发生都是修改变量值时所产生的副作用。SNMP 定义了管理员的计算机与被管实体之间传输的报文的格式。

与 SNMP 相伴的一组标准定义了由被管实体维护的一系列变量。这些变量集组成了**管理信息库**（Management Information Base，简称 MIB）。MIB 变量使用 ASN.1 进行描述，而 ASN.1 是一种为名字和对象提供简洁编码格式和严谨的可读记法的正式语言。ASN.1 使用分级名字空间，以保证所有 MIB 名字都是全球唯一的，同时还允许由**子群**（subgroup）来分配部分名字空间。

29.14 深入研究

Case et. al. [RFC 3410]向人们全面介绍了 SNMPv3，给出其背景和动机，讨论各种版本之间的变化。它还包含对和 v3 有关的 RFC 的总结，并解释哪个 v2 标准仍在使用中。许多其他 RFC 讨论了协议的某一个方面。例如，Wijnen et. al. [RFC 3415]提出基于视图的访问控制模型，而 Case et. al. [RFC 3412]讨论了报文处理。

ISO 标准 8824 和 8825 含有 ASN.1 的规范并指明其编码。McCloghrie et. al. [RFCs 2578, 2579, 2580]定义了 MIB 模块使用的语言，并提供了对数据类型的定义。Presuhn [RFC 3418]定义了版本 3 的 MIB。

29.15 习题

1. 用网络分析工具捕捉一个 SNMP 分组，并为各个字段解码。
2. 阅读标准，找出 ASN.1 如何将对象标识符的前两个数字值编码为 1 八位组。它为什么要这样做？

-
3. 阅读 SNMPv2 和 SNMPv3 这两个标准并进行比较。在什么情况下 v2 的安全特性有效？在什么情况下无效？
 4. 设想 MIB 设计者需要定义一个对应于二维数组的变量， ASN.1 记法如何适应对这样一个变量的引用？
 5. 为 MIB 变量定义全球唯一的 ASN.1 名字的优点和缺点是什么？
 6. 参考标准并对照图 29.11 中具有相应定义的每一项。
 7. 如果你身边有客户代码，尝试使用它读取本地路由器中的 MIB 变量。允许任意管理系统读取所有路由器中变量的优点是什么？缺点是什么？
 8. 阅读 MIB 规范并找出对应 IP 路由表的变量 `ipRoutingTable` 的定义。设计一个程序，使用 SNMP 联系多个路由器，并研究其路由表中是否有哪一项会引起转发环路。确切地说，这个程序会产生哪些 ASN.1 名字？
 9. 考虑一个 SNMP 代理的实现。完全按照 SNMP 描述的方式在存储器中安排 MIB 变量是否可行？为什么？
 10. 讨论：SNMP 用词不当，因为 SNMP 不是“简单的”网络管理协议。
 11. 阅读第 30 章中描述的 IPsec 安全标准。如果有某个机构使用 IPsec，那么 SNMPv3 中的安全性还有必要吗？试解释原因。
 12. 用 SNMP 来管理所有设备是否有意义？为什么？提示：考虑一个简单的硬件设备，如拨号调制解调器。

第30章 互联网的安全性和防火墙设计（IPsec 和 SSL）

30.1 引言

如同需要用锁来保证有形财产的安全一样，计算机和数据网络也具有保护信息安全的防范机制。在互联网中，安全性既重要又困难。说它重要，是因为信息具有重要价值，也就是说，信息可被直接买卖，也可以间接地使用信息创造出可获得高利润的新产品或服务。说它困难，是因为安全性意味着除了要了解参与的用户、计算机、服务和网络在何时以及怎样相互信赖之外，还要了解网络硬件和协议的技术细节。哪怕只有一点缺陷，都有可能损害全网的安全性。更重要的是，由于TCP/IP支持各种不同的用户、服务和网络，还由于互联网可以跨越很多政治的和组织的边界，因此参与其中的个人和组织在处理数据的信任程度或策略方面可能并不统一。

本章将讨论构成互联网安全基础的两种基本技术：边界安全和加密。边界安全允许一个组织机构决定自己对外人所提供的服务和网络，以及外人所能使用的资源的范围。加密用于处理其他大部分安全性事务。下面先简单介绍一些基本概念和术语。

30.2 对资源的保护

从广义上讲，术语**网络安全**（network security）和**信息安全**（information security）指的就是保密，即确保网络上的信息和服务不被未经授权的用户所使用。安全暗示了安全和保险，包括对数据的完整性的保证，防止对计算资源的未经授权随意访问，防止随意窃听或偷窥以及随便打断服务。当然，如同不能保证物理财产的绝对安全一样，网络也没有绝对的安全。各种组织机构之所以要在网络安全上做出巨大的努力，其道理和保护其建筑物以及办公室的安全是一样的：使犯罪更困难的各种基本安全手段会使罪犯有所收敛。

为了提供信息的安全性，对物理的和抽象的资源都要求加以保护。物理资源包括无源的存储设备（如磁带和磁盘）以及有源的设备（如用户的计算机）。在网络环境中，物理安全扩大到了构成网络基础结构的电缆、网桥以及路由器等设备。虽然人们很少提及物理安全，但它的确在整个安全规划中经常起着重要的作用。显然，物理安全可以防止窃听。好的物理安全还可以抵挡外来攻击（例如，让一个路由器无法工作，导致一些分组选择另一条比较不安全的路径）。

由于信息的变化多端，使保护诸如信息之类的抽象资源比提供物理安全更困难。信息安全包括许多方面的防护：

- **数据完整性**。一个安全的系统必须保护信息不被未经授权的用户修改。
- **数据有效性**。系统必须保证外人不能阻止对数据的合法访问（如，任何一个外人都不能阻碍客户访问一个万维网网站）。
- **保密性或机密性**。系统必须防止数据在通过网络时被外人复制，或在复制之后了解其内容。
- **授权**。虽然物理上的安全措施对人和资源的类别划分常常是粗放的（如，禁止所有非雇员使用某个特定的门厅），但是信息安全通常要求更细致的限制（例如，一个雇员档案中的某些部分只能由人事部门的工作人员查阅，还有一些部分只能让老板看到，而其他一些部分则对会计人员开放）。
- **鉴别**。系统必须允许两个通信实体互相验证对方的身份。
- **避免重放**。要防止外人俘获分组副本，并在之后利用它们，系统就必须防止接受重新传送

的分组副本。

30.3 信息策略

一个组织机构在能够推行网络安全之前，必须先要评估风险，并针对信息的访问和保护制定明确的策略。该策略要求指明允许谁来访问哪条信息，每个人在向其他人散布信息时所必须遵守的规则，以及对违反规定的情况将如何处置的声明。

信息策略首先要从人开始，这是因为：

人往往是任何安全措施中最薄弱的环节，而一个存心不良、粗心或者对组织的信息策略漠然无视的工作人员可以破坏最好的安全机制。

30.4 互联网安全

互联网的安全是很难保证的，因为数据报在从源站到目的站的传送过程中常常要经过许多中间网络，还要通过一些路由器，这些路由器可能是发送方和接收方都不拥有或无法控制的。因此，由于数据报可能会在中途被截获或损害，所以不能信任其内容。例如，假设一个服务器试图使用**信源鉴别**（source authentication）来验证请求是否由合法客户发出。信源鉴别要求服务器检查每个传入数据报的源 IP 地址，并且只接受授权列表中的计算机发出的请求。信源鉴别很脆弱，因为它很容易被破坏。实际上，中间路由器可以观察发送到服务器以及从服务器发出的通信量，并记录合法客户的 IP 地址。之后，中间路由器就可以制造一个使用同样源地址的请求（并截获回答）。关键问题是：

在一个不安全的互联网中，使用远程机器的 IP 地址进行鉴别的授权方案是远远不够的。

控制了中间路由器的冒名顶替者通过冒充已被授权的客户，可以获得访问权。

更强大的鉴别方案要求经过**加密**（encryption）处理。仔细地选择加密算法确实可以使中间机器不能解密报文或制造合法的报文。

30.5 IP 安全 (IPsec)

IETF 已经设计了一套提供因特网安全通信的协议。这些协议合在一起称为 IPsec (IP security 的缩写)，它们在 IP 层提供鉴别和保密服务，既能和 IPv4 一起使用，也能和 IPv6 一起使用^①。更重要的是，IETF 选择了让系统既灵活又具有扩展性，而不是完全彻底地指明其功能或必须使用的加密算法。例如，使用了 IPsec 的应用程序可以选择是使用鉴别程序确认发送方的身份，还是使用加密程序，加密程序也能保证有效载荷是可信的。并且这种选择是非对称的（如，在一个方向上进行鉴别，而在另一个方向上没有鉴别）。此外，IPsec 没有限制用户使用哪个具体的加密或鉴别算法。相反，IPsec 提供了一个通用的框架结构，允许每对通信端点自己选择算法和参数（如密钥大小）。为了保证协同工作的能力，IPsec 确实也包括了一组所有实现必须识别的加密算法。要点是：

IPsec 不是单个的安全协议，而是提供了一组安全算法，再加上一个通用的框架结构，以允许每对通信实体使用任何一种算法为通信提供适当的安全保证。

^① 本章所举的例子都是针对 IPv4 的，第 31 章将描述 IPv6 的细节并说明在 IPv6 数据报中 IPsec 首部的表现形式。

30.6 IPsec 鉴别首部

IPsec 没有改变基本的数据报首部或创建一个 IP 选项，而是使用一个独立的**鉴别首部**（Authentication Header，简称 AH）来承载鉴别信息。图 30.1 所示为鉴别首部针对 IPv4 的最直观用法。

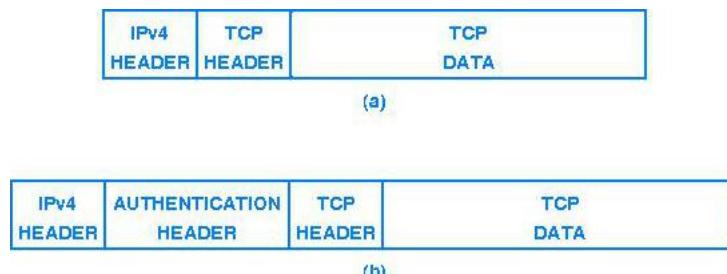


图 30.1 (a) 一个 IPv4 数据报；(b) 添加了 IPsec 鉴别首部以后的同一数据报。新的首部直接插到 IP 首部之后

如图 30.1 所示，IPsec 将鉴别首部直接插在原始 IP 首部之后，运输首部之前。此外，IP 首部中的协议（PROTOCOL）字段的值变成 51，以指示鉴别首部的存在。

如果 IPsec 修改了 IP 首部中的协议字段，那么接收方如何确定在数据报中传送的信息的类型呢？在鉴别首部中还有一个下一首部（NEXT HEADER）字段，用于指明其类型，也就是说，IPsec 将原始的协议字段的值写在下一首部字段中。当数据报抵达时，接收方使用鉴别首部中的安全信息来验证发送方，并使用下一首部的值进一步分解数据报。图 30.2 所示为这个首部的格式。

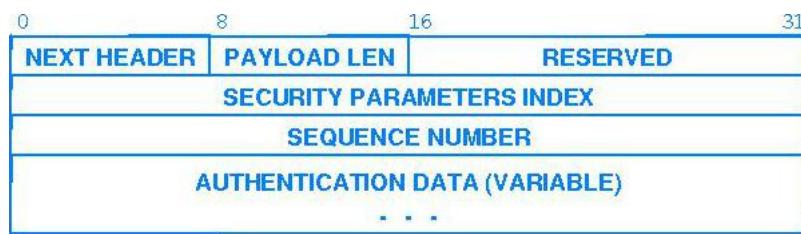


图 30.2 IPsec 鉴别首部格式。下一首部字段记录了 IP 协议字段的原始值

有趣的是，有效载荷长度（PAYLOAD LEN）字段并没有指明数据报中数据区的大小，而是指明了鉴别首部的长度。其余几个字段用于保证安全性。序号（SEQUENCE NUMBER）字段为每个发送的分组包含一个唯一的序号，当选定某个安全算法后，序号即从零开始，单调递增。安全参数索引（SECURITY PARAMETERS INDEX）字段指明了使用的安全方案，而鉴别数据（AUTHENTICATION DATA）字段包含所选安全方案的数据。

30.7 安全关联

要了解使用安全参数索引的原因，应当注意到安全方案在定义细节时会提供许多变量。例如，安全方案包括一个鉴别算法、一个算法使用的密钥、密钥保持有效的生存时间、接收方同意使用算法的生存时间以及授予了使用该方案的权利的源地址列表。仔细斟酌之后你会发现，这些信息无法放在首部中。

为了节省首部的空间，IPsec 设法让每个接收方把有关安全方案的所有细节都收集到一个称为**安全关联**（Security Association，简称 SA）的摘要中。

每个 SA 都给定了一个数字，称为**安全参数索引**（security parameters index），通过这个数字来标识 SA。在发送方能够使用 IPsec 与接收方进行通信之前，发送方必须了解某个具体的 SA 的索引值。然后，发送方把该值放在每个外发数据报的安全参数索引字段中。

索引值不是全局指定的。相反，各个目的站根据需要创建 SA，并给每个 SA 分配一个索引值。目的站可以指定每个 SA 的生存时间，一旦某个 SA 变为无效，就可以重新使用它的索引值。因此，如果不询问目的站就无法解释索引（如，索引 1 的含义在两个不同的目的站上含义可能完全不同）。归纳如下：

目的站使用安全参数索引来标识一个分组的安全关联。这些索引值不是全局的，需要用目的地址和安全参数索引的组合来标识一个 SA。

30.8 IPsec 封装安全有效载荷

IPsec 除了要处理鉴别之外，还要处理保密性，对此 IPsec 使用了**封装安全有效载荷**（Encapsulation Security Payload，简称 ESP），它比鉴别首部复杂得多。在数据报中，如果协议字段的值为 50，就是告诉接收方数据报承载的是 ESP。图 30.3 所示为其概念上的组织结构图。

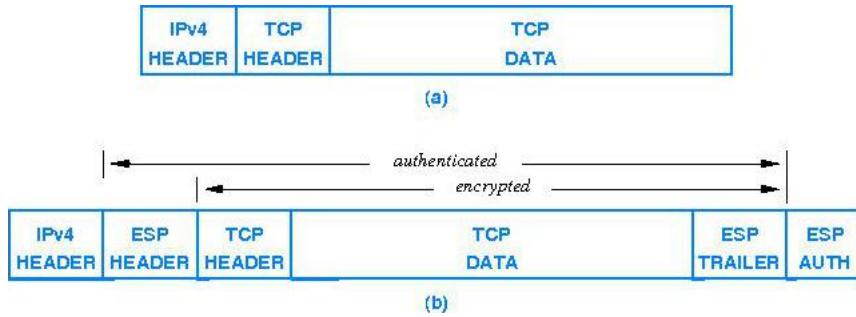


图 30.3 (a) 一个数据报；(b) 使用 IPsec 封装安全有效载荷的同一数据报。实际上，加密就代表着其中的字段不易辨别

如图 30.3 所示，ESP 给数据报添加了三个区。ESP 首部（ESP HEADER）紧跟在 IP 首部之后，加密的有效载荷之前。ESP 尾部（ESP TRAILER）与有效载荷一起加密，可变长的 ESP 鉴别（ESP AUTH）在加密区之后。

ESP 使用的许多项与鉴别首部中使用的相同，但是重新排了序。例如，ESP 首部由八个八位组组成，它标识的是安全参数索引和一个序号。

ESP 尾部包括可选的填充项、填充长度（PAD LENGTH）字段和下一首部（NEXT HEADER）字段，其后跟随的是可变大小的鉴别数据。

填充是可选的，它的存在有以下三个原因。第一，一些加密算法要求在加密的报文后有零出现。第二，注意下一首部字段在上面所示的 4 八位组字段中是最靠右的。这个靠右对齐很重要，因为 IPsec 要求跟在 ESP 尾部后面的鉴别数据要与 4 八位组的左边界对齐，所以需要补零填充以保证对齐。第三，一些网点可能会选择给每个数据报填充随机数量的零，这样在传送沿途某个中间点的偷听者就不能用数据报的大小猜出它的用途。

30.9 鉴别和可变首部字段

IPsec 鉴别机制的设计要做到能够保证抵达的数据报和源站发出的数据报完全一样。但是这样的保证根本不可能。要理解其原因，可以回想一下，IP 是一个从机器到机器的层，也就是说，这一层的所有规则只能应用于一跳的距离上。实际上，每个中间路由器会递减生存时间字段的值，并重新计算检验和。

IPsec 把这些在传输过程中会改变的 IP 首部字段称为**可变字段**（mutable field）。为了防止这些字段的改变引起鉴别错误，IPsec 在鉴别计算中特别省略了这些字段。因此，当数据报抵达时，IPsec 只鉴别不会改变的字段（如源地址和协议类型）。

30.10 IPsec 隧道

回顾第 19 章，VPN 技术使用加密和 IP-in-IP 隧道技术来保证网点之间传送的保密性。IPsec 经过专门设计完全能够适应加密隧道。实际上，标准为鉴别首部和封装安全有效载荷都定义了隧道化的版本。图 30.4 所示为隧道模式下的数据报设计。

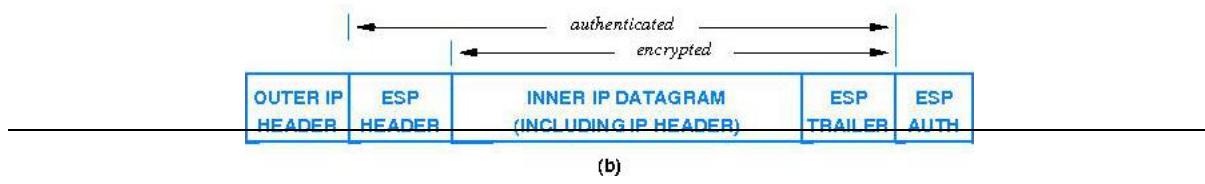


图 30.4 图示为(a) 鉴别和(b) 封装安全有效载荷的 IPsec 隧道模型。整个内部的数据报是受保护的

30.11 必要的安全算法

IPsec 定义了一个必要的（也就是所有实现必须提供的）最小化的算法集合。标准针对各种不同的情况定义了不同的算法使用方法。图 30.5 列出了这些必要的算法。

Authentication	
HMAC with MD5	RFC 2403
HMAC with SHA-1	RFC 2404
Encapsulating Security Payload	
DES in CBC mode	RFC 2405
HMAC with MD5	RFC 2403
HMAC with SHA-1	RFC 2404
Null Authentication	
Null Encryption	

图 30.5 IPsec 必要的安全算法

30.12 安全套接字

在 20 世纪 90 年代中期，当安全性对于因特网商务活动的重要性变得很明显时，几个小组提出了用于万维网的安全机制。虽然 IETF 没有正式采纳这些机制，但是其中的一个建议已经变成了事实上的标准。

这项称为**安全套接层**（Secure Socket Layer，简称 SSL）的技术最初是由 Netscape 公司开发。从这个名字上可以看出 SSL 和套接字 API 同处于一层。当客户用 SSL 与服务器联络时，SSL 协议允许两端分别向另一端鉴别自己。然后两端可以协商并选择一个双方都支持的加密算法。最后，SSL 允许两端都建立一条加密的连接（也就是说，使用所选的加密算法来保证其连接的保密性）。IETF 将 SSL 作为**运输层安全**（Transport Layer Security，简称 TLS）协议的基础。SSL 和 TLS 之间的关系如此密切，以至于它们使用同一个熟知端口，并且绝大多数 SSL 的实现都支持 TLS。

30.13 防火墙和互联网的访问

控制**互联网访问**（internet access）的机制所要处理的问题就是筛选特定的网络或机构，防止并不期望的通信。这种机制可以防止外人获取信息、改变信息或打断机构内联网的通信。成功的访问控制要求仔细地组织网络拓扑结构、中间信息分级以及分组过滤等各种限制。

目前已出现了一种作为互联网访问控制基础的技术，称为**互联网防火墙**（internet firewall）^①。一个组织机构可以在它与外部网络（如全球因特网）的连接上放置一个防火墙。防火墙把互联网分成两个区域，非正式地称为**内部的**（inside）和**外部的**（outside）区域。

^① 防火墙这个术语来自建筑业，在建筑业中，防火墙指厚的、可以防火的部分，使建筑物的某个部分免受烈火侵入。

30.14 多重连接和最薄弱的链接

虽然概念听起来很简单，但构造防火墙的细节使之变得非常复杂。首先，一个组织机构的内联网可能有多个外部连接。该组织必须在每个外部连接上安装防火墙，以形成**安全防线**（security perimeter）。为保证这个防线是有效的，必须把所有的防火墙配置成使用相同的访问限制。否则，就有可能绕过某个防火墙的限制，而通过其他的防火墙进入这个组织机构的内联网^①。

归纳如下：

有多个外部连接的组织必须在每个外部连接上安装防火墙，还必须协调所有这些防火墙。
所有防火墙上的限制访问若有不一致之处，就会使组织处于危险之中。

30.15 防火墙的实现和分组过滤器

如何实现防火墙呢？从理论上讲，防火墙只是简单地封锁组织内和组织外的计算机之间的未授权通信。在实际应用中，实现的细节取决于网络技术、连接的容量、通信量负载以及组织的策略。所以，没有对所有组织普遍适用的单一解决方法，也就是说，防火墙要设计成可配置的。这种机制的非正式名称是**分组过滤器**（packet filter），它要求管理员指明路由器应当如何处理每个分组。例如，管理员可以选择过滤掉（即阻塞）所有来自某个特定源站的数据报，同时允许来自其他源站的数据报通过，或者管理员可以选择阻塞所有以某个 TCP 端口为目的地的数据报，同时允许以其他端口为目的地的数据报通过。

为了以网络速度运行，分组过滤器需要对硬件和软件进行优化，以完成任务。许多商业路由器都包含了一个用于高速分组过滤的独立硬件（如路由器中的一块板子）。一旦管理员对其进行配置，这个过滤器就以线路速度运作，而不会给分组带来延迟。

因为 TCP/IP 并没有为分组过滤器制定一个标准，所以每个路由器运营商都可以自由选择其分组过滤器的性能以及管理员对过滤器进行配置的接口。有些路由器允许管理员为每个接口分别配置过滤器行为，而有些则为所有接口使用同一个配置。通常，分组过滤器允许管理员指定以下内容的任意组合：源站 IP 地址、目的站 IP 地址、协议、源站协议端口号以及目的站协议端口号等。例如，图 30.6 所示就是一个过滤器规则。

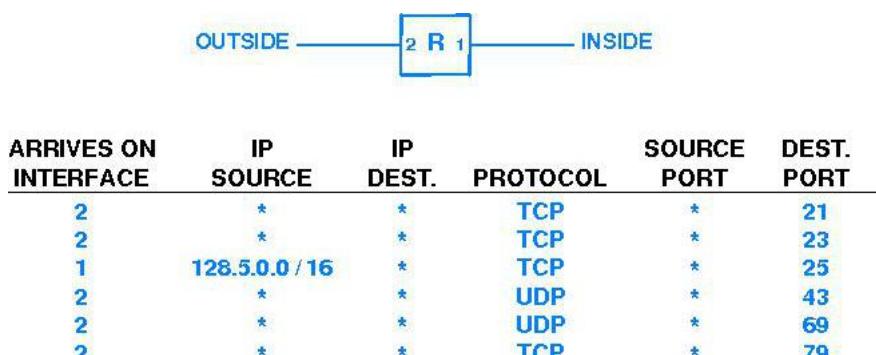


图 30.6 具有两个接口的路由器及其数据报过滤器规则的一个例子。包含分组过滤器的路由器构成了防火墙的基本构件

^① 安全性的强度取决于系统的最薄弱环节，这一著名观点称为**最薄弱环节公理**（weakest link axiom），该名字出自一句古谚，一条链子是否结实要看它最细的链环。

在这个例子中，管理员选择了阻塞目的地为 FTP (TCP 端口号 21)、TELNET (TCP 端口号 23)、WHOIS (UDP 端口号 43)、TFTP (UDP 端口号 69) 或 FINGER (TCP 端口号 79) 的传入数据报，并阻塞从任何与 128.5.0.0 的 16 比特前缀相匹配的主机地址发出的，并且目的站是远程电子邮件服务器 (TCP 端口号 25) 的所有外发数据报。

30.16 安全性与分组过滤器规则

虽然图 30.6 中的过滤器配置例子指明了一个小型的应当阻塞的服务列表，但在一个高效的防火墙中，这种方法并不可行。其原因有以下三点。首先，熟知端口号的数量很大并且在不断增长。因此，为了列出每一个服务，要求管理员不断地更新这个表，只要有一点疏忽就会陷防火墙于危险境地。其次，互联网中的大多数通信量并不是发往或来自这些熟知端口号的。除了程序员可以为其专用客户一服务器应用程序选择端口号以外，**远程过程调用** (Remote Procedure Call，简称 RPC) 之类的服务也会动态地分配端口号。第三，列出这些熟知服务的端口号会使防火墙在**隧道** (tunneling) 技术下露出破绽。只要内部的某个主机或路由器同意接受来自外部的封装过的数据报，那么除去封装层后。将数据报再转发给那些本应被防火墙限制访问的服务，隧道技术就会破坏安全性。

防火墙怎样才能有效地使用分组过滤器呢？答案在于将过滤器的想法逆转过来：防火墙不是指明应该过滤哪些数据报，而应当配置成阻塞所有的数据报，除了到那些被批准允许与之对外通信的特定网络、主机以及协议端口的数据报。因此，管理员首先假设所有通信都是不允许的，然后在启用任何一个端口之前必须仔细检查该组织的信息策略。实际上，许多分组过滤器都允许管理员指明一组可以通过的数据报，而不是一组要阻塞的数据报。总结如下：

为了使防火墙有效，利用数据报过滤的防火墙应当限制所有 IP 源站、IP 目的站、协议以及协议端口的访问，组织已经明确决定可对外开放的计算机、网络以及服务除外。分组过滤器允许管理员指明哪些数据报是可通过的，而不是指明要阻塞哪些数据报，这样做可以使对分组过滤器的限制易于指定。

30.17 客户受限访问的后果

通过防止外人访问组织机构中的任意服务器，全面禁止来自未知协议端口的数据报，似乎可以解决许多潜在的安全性问题。但这种防火墙有一个令人意想不到的后果：它也防止了防火墙内部任一计算机作为客户访问防火墙外部的服务。要理解为什么会这样，回想一下，虽然服务器工作于某个熟知端口上，但客户并不是这样。当一个客户程序开始执行时，它要求操作系统为其选择一个协议端口号，这个协议端口号既不是熟知端口，也不是客户计算机正在使用的端口。当客户试图与组织以外的服务器通信时，它会产生一个或多个数据报，并将这些数据报发送给服务器。每一个外发数据报把客户的协议端口号作为源端口号，把服务器的熟知端口号作为目的端口号。当这种数据报从内部发出时，防火墙并不阻拦。但当服务器在产生响应时，它要交换两个协议端口号，客户端成了目的端口，而服务器的端口成了源端口。可是，当载有这个响应的数据报抵达防火墙时，防火墙将会阻拦它，因为目的端口是未经批准的。因此，我们可以得到一个重要的思想：

如果除了对应于本组织对外提供的服务的那些端口之外，防火墙限制其他所有传入的数据报，那么这个组织机构内的一个任意应用程序就不能成为组织机构外部某个服务器的客户。

30.18 有状态防火墙

那么如何才能允许一个组织内部的任意客户访问因特网上的服务，同时又不允许以任意协议端口为目的地的传入数据报通过呢？这个问题的答案在于一种称为**有状态防火墙**（stateful firewall）的技术。从本质上讲，这种防火墙要监视外出连接，并相应地调整过滤规则以接纳响应分组。

作为一个有状态防火墙的例子，设想在某组织内部的一个客户程序建立了一条到达万维网服务器的 TCP 连接。如果客户的源 IP 地址是 I_1 ，源 TCP 端口是 P_1 ，并且所连接的万维网服务器的 IP 地址为 I_2 ，端口为 80，那么用于初始化连接的外发 SYN 报文段将通过防火墙，这时防火墙会记录以下的四维组：

$(I_1, P_1, I_2, 80)$

当服务器返回 SYN+ACK 响应时，防火墙要将自己保存的四维组中的两个端点与之进行匹配，并允许这个传入报文段的通过。

有意思的是，有状态防火墙不允许组织机构内部的客户与任意外部目的站初始化连接。相反，有状态防火墙的所有行为都是由一组分组过滤器规则驱动的。因此，防火墙管理员仍然可以选择是接受还是拒绝某个特定的分组。在分组被允许通过防火墙的情况下，这个过滤器规则能够进一步指明是否要记录状态信息，以便将来允许响应的返回。

有状态防火墙中的状态是如何管理的呢？总的来说有两种方法：防火墙可以使用**软状态**（soft state）和**连接监视**（connection monitoring）。软状态的实现要通过设置一个定时器，并在超时后删除那些不活动的状态信息。而在连接监视时，防火墙要观察流量中的分组，并在流量终止时删除状态信息（如在一条 TCP 连接上接收到一个 FIN 时）。即使状态防火墙试图监视连接，通常也需要用软状态作为候补，以处理像 UDP 流量之类的没有显式终止的情况。

30.19 内容保护和代理

上面所讨论的机制都是有关访问的。安全性的另一个方面在于其内容。例如，我们知道一个导入的文件或电子邮件报文有可能含有病毒。通常，这些问题只能由系统对传入的内容进行检查来解决。

在分组层检查内容是不可行的，因为分组可以无序到达，内容也可能被划分到多个分组中。因此，检查内容的唯一切实可行的办法包括从一条连接上提取内容，然后在允许它进入组织之前先对提取的内容进行检查。

实施检查内容的机制必须以**应用代理**（application proxy）的方式工作。例如，某组织可以运行一个文件传送代理，它会拦截每个外发的 FTP 请求，获取被请求的文件的副本，扫描副本以检查是否有病毒，并且如果认为副本是干净的，就完成向客户的副本传送过程。类似地，一个组织也可以运行一个万维网代理。注意，应用代理可以是**透明的**（transparent）（即除了延迟之外，客户并没有意识到有一个代理曾经拦截了请求）或**非透明的**（nontransparent）（即必须配置客户，以使用某个具体的代理）。

尽管许多组织使用了有状态防火墙来保护自己不受外人的随意窥探，但是很少有组织会对内容进行检查。因此，经常会发现一个组织机构对于来自外部的任意攻击具有免疫力，但是当一个警惕性不高的职员引入了一个能够通过构造外出连接来欺骗防火墙的程序后，这个组织仍然会被电子邮件病毒和特洛伊木马问题所折磨。

30.20 监视与日志

监视是防火墙设计中最重要的方面之一。负责防火墙的网络管理员需要注意各种绕过安全性的企图。除非防火墙报告有意外事件出现，否则管理员可能意识不到问题的发生。

监视可以是**主动的** (active) 或**被动的** (passive)。在主动监视时，只要有意外事件发生，防火墙就会通知管理员。主动监视的主要优点是速度很快，管理员可以立即发现潜在的问题。主要的缺点是经常会产生大量信息，以至于管理员不能理解或注意到真正的问题。因此，大多数管理员更偏爱被动监视，或者采用一种组合方式，即被动监视伴以由主动监视报告少数有高度危险的事件。

在被动监视时，防火墙将每个事件记录在磁盘上的一个文件中。被动监视通常还会记录正常的通信量信息（例如简单的统计数据）以及被过滤掉的数据报。管理员可以在任何时候阅读这些日志，并且大多数管理员使用计算机程序进行此项工作。被动监视的主要优点在于它对事件的记录，管理员可以参考日志文件来观察趋势，并且在确实出现安全问题时，检查那些引起问题的事件的历史记录。更重要的是，管理员可以定期地分析日志（例如每天），以判断试图访问该组织的次数在一定时间里是增加了还是减少了。

30.21 小结

由于互联网可以连接那些互不信赖的组织，这样就带来了安全性问题。有一些技术可以用来确保当信息穿过互联网时仍然是安全的。安全套接字层 (SSL) 给套接字 API 添加了加密和鉴别机制。IPsec 允许用户选择两个基本方案：一个方案提供对数据报的鉴别，另一个方案提供鉴别和加密的组合。IPsec 通过插入鉴别首部或使用封装安全有效载荷来修改数据报，后一种方案会插入一个首部和一个尾部，并加密发送的数据。IPsec 提供了一个通用的框架结构，允许每对通信实体选择一种加密算法。因为安全性经常和隧道一起使用（如在 VPN 中），所以 IPsec 定义了一个安全隧道模式。

防火墙机制用于控制互联网的访问。一个组织可以在它的每个外部连接上设置一个防火墙，以此来保证组织内部的网络不受未经授权的通信量的影响。防火墙含有一个分组过滤器，系统管理员必须对它进行配置，以指明在每个方向上有哪些分组可以通过。有状态防火墙通过配置可以让防火墙自动允许响应分组通过，只要它已建立外出连接。

尽管防火墙提供了访问保护，但并没有检查或控制内容。因此，使用防火墙来限制访问并不能保护网点免除所有麻烦。要保护一个网点能够抵御像病毒和其他不希望的内容的攻击，该组织机构就必须使用应用代理，以便在接受数据之前先检查内容。

30.22 深入研究

20世纪90年代中期，IETF宣布了安全的重要性，要求每个工作组都在设计中考虑安全性的实现问题。因此有许多RFC针对的是有关互联网安全方面的问题，并建议了各种策略、过程和机制。Dierks and Allen [RFC 2246]定义了TLS版本1。Kent and Atkinson [RFC 2401]定义IPsec体系结构。Kent and Atkinson [RFC 2402]指明了IPsec鉴别首部，而[RFC 2406]则指明了封装安全有效载荷。

许多RFC都描述了某个具体应用协议的安全性。例如，Wijnen et. al. [RFC 3415]提出基于视图的安全性模型，Blumenthal and Wijnen [RFC 3414]提出基于用户的安全模型。这两个模型都是用于SNMPv3的。

Cheswick et. al. [2003]讨论了防火墙和其他有关TCP/IP互联网安全操作的主题。Kohl and Neuman [RFC 1510]描述kerberos鉴别服务，而Borman [RFC 1411]则讨论如何将kerberos用于

TELNET 鉴别。

30.23 习题

1. 阅读商用路由器的分组过滤器说明，它提供了哪些特性？
2. 收集一份进入你所在网点的通信量的日志。分析这个日志，看看来自或者传到某个熟知协议端口的通信量的百分比。结果有什么令人惊讶之处吗？
3. 如果你的计算机中有一个加密软件，测量对一个 10 MB 的文件进行加密、传送和解密所需的时间。将其与不加密传送时所需的时间相比较。
4. 调查你所在网点中的用户，判断他们是否在电子邮件中发送了敏感的信息。这些用户是否知道 SMTP 用 ASCII 码传输报文，而任何监视网络通信量的人都可以看到电子邮件的内容？
5. 调查你所在网点的 IT 职员，看看他们是否曾经配置过 SMTP 服务器，使之在任何可能的情况下自动使用 SSL 或其他加密技术（例如，对于所有内部数据传递都使用了 SSL）。
6. 调查你所在网点上的职员，看看有多少人使用调制解调器和个人计算机导入和导出信息，问问他们是否了解本组织的信息策略。
7. 其他一些协议族，如 AppleTalk 或 Netware，能够使用防火墙吗？为什么？
8. 防火墙可以和 NAT 一起使用吗？会带来什么后果？
9. 在军事上，只把信息发送给那些“需要知道”的人。对于你的组织上的所有信息来说，这种策略是否可行？为什么？
10. 为什么要把管理组织安全策略的人和管理组织的计算机以及网络系统的人分成两个不同的组，试给出两个理由。
11. 有一些组织使用防火墙把组织内部的若干用户组分隔开。给出使用内部防火墙可以提高网络性能的例子，再给出其降低网络性能的例子。
12. 如果你的组织使用 IPsec，找出它正在使用的算法。它的密钥有多长？

第31章 下一代 IP (IPv6)

31.1 引言

TCP/IP 技术的发展与全球因特网的发展息息相关。成千上万的用户访问着分布于世界各地的网点，他们把因特网作为他们日常工作的必备环境，看起来因特网已经度过了早期发展阶段，现在已拥有十分稳定的产品设施。然而，尽管表面上看来如此，实际上无论是因特网还是 TCP/IP 协议族都不是一成不变的。研究人员和工程师们在不断探索新技术的使用，并改进底层机制。

本章的目的是讨论目前正在前进中的发展历程，并考察其中一项最具影响力的工作：IP 修订版的建议。如果运营商采纳了该建议，它将对 TCP/IP 协议和全球因特网产生巨大影响。

31.2 为什么要改变

在 20 世纪 90 年代初，研究人员认为现在的因特网不足以承担像音频和视频这样的新型应用程序。他们更进一步地争论说，因为因特网的迅猛发展，它的规模每 9 个月或更短的时间就会翻一番，这样下去很快就会耗尽所有可用地址。然而时至今日，这两个问题都已变得很明朗了。首先像电话这样的应用也能在目前的因特网上很好地运作，其次 CIDR 地址技术和 NAT 提供了必要的地址扩展。根据最新预测，直到 2022 年以前，仍然有足够的地址供我们使用（如果将未使用的地址重新利用，可以延长到 2028 年）。

31.3 超越 IPv4

网际协议版本 4 (IPv4) 是第一个被实际应用的版本，自 20 世纪 70 年代后期 IPv4 出现以来，它就几乎没有改动过。版本 4 的长久性体现了它的设计非常灵活和强壮。自从 IPv4 设计至今，处理器的性能已经提高了三个数量级，典型的存储器大小提高了 400 倍，因特网主干链路的带宽也提高了 150,000 倍。局域网技术已经出现，因特网上的主机数量也已从不多的百十来个激增到数以亿计。

第 9 章描述了更新 IP 的主要动机：最终地址空间的局限性。在设计 IP之初，32 位的地址空间是绰绰有余的。那时只有屈指可数的组织机构使用局域网，没有谁拥有一台 PC 机。但是今天，即使是很小的公司也拥有自己的局域网和多台计算机。如果再为每个移动电话分配一个 IP 地址，那么地址很快就会耗尽。

31.4 通向新版 IP 之路

IETF 为设计新版的 IP 协议工作了很多年。由于 IETF 致力于产生一种开放的标准，他们邀请了很多团体的代表来参加标准的制定过程。计算机制造商、硬件和软件运营商、用户、管理员、程序员、电话公司以及有线电视产业都对下一版 IP 提出了自己的要求，并且都在专门的建议中加以说明。

许多被建议的设计是服务于某种特定目的或特定团体的。最终，一个称为 SIP (Simple IP)^① 的设计成为了一个扩展建议的基础，在这个扩展建议中还包含一些来自其他建议的思想。这个扩展的版本名为简单 IP Plus (Simple IP Plus，简称 SIPP)，它最终被选择为下一版 IP 的设计基础。

31.5 下一代 IP 的名字

IETF 决定为 IP 的修订版分配版本号 6，将其命名为 IPv6 (起初它称为“IP 下一代”，IPng)。在经过一系列失误和误解后，版本号 5 被跳过。为了避免混淆不清，IETF 选择用 6 作为其版本号。

31.6 IPv6 的特点

建议的 IPv6 协议保留了 IPv4 赖以成功的许多特点。事实上，IPv6 的设计师们将 IPv6 的总体特征设计得基本上与 IPv4 保持一致，只是做了一点修改。例如，IPv6 仍然支持无连接的交付（即每个数据报独立地选择路由），允许发送方选择数据报的大小，要求发送方指明数据报在到达终点前的最大跳数。正如我们将会看到的，IPv6 保留了由 IPv4 选项提供的大多数概念，包括分片和源路由选择功能。

尽管许多概念是相似的，但 IPv6 还是改变了协议的许多细节。例如，IPv6 使用更大的地址空间，增加了一些新特性。更重要的是，IPv6 完全修订了 IPv4 数据报的格式，用一系列固定格式的首部取代了 IPv4 中可变长度的选项字段。下面将在探讨这些主要变化及造成这些变化的内在动机，之后研究有关细节。

IPv6 带来的变化可以归为七类：

- **更大的地址空间。**新的地址尺寸是 IPv6 最显著的变化。IPv6 把 IPv4 的 32 位地址翻了四倍，增大到 128 位。
- **扩展的地址层次。**IPv6 利用更大的地址空间为分级地址结构添加了更多层次（例如，允许 ISP 向每个客户按地址块分配地址）。
- **灵活的首部格式。**IPv6 使用了一种全新的不兼容数据报格式，并包括一组可选的首部。
- **增强的选项。**IPv6 允许数据报包含可选的控制信息。IPv6 的选项还提供了一些 IPv4 所不具备的新功能。
- **对协议扩展的保障。**IPv6 没有指明所有的细节，相反，IPv6 的扩展能力使 IETF 能够让协议适应底层网络或新的应用程序的变化。
- **支持自动配置和重新编号。**IPv6 允许孤立网上的计算机自动分配本地地址。它的设计还允许管理员动态地重新给网络编号。
- **支持资源分配。**IPv6 包括了流的抽象和用于区分服务 (DiffServ) 规范的专门位。IPv6 的区分服务与 IPv4 的区分服务相同。

31.7 IPv6 数据报的一般形式

IPv6 完全改变了以前的数据报格式。如图 31.1 所示，IPv6 数据报开始是一个固定大小的基本首部 (base header)，其后紧跟的是零个或多个扩展首部 (extension header)，之后才是数据。

^① 现在，首字母缩写 SIP 指的是会话发起协议 (Session Initiation Protocol)。

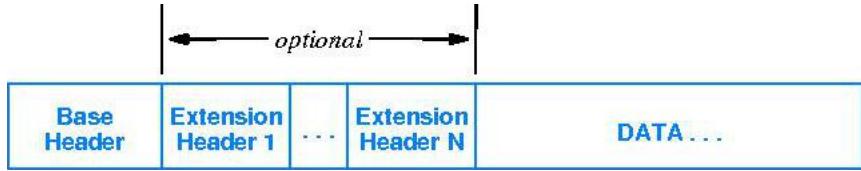


图 31.1 有多个首部的 IPv6 数据报的一般形式。只有基本首部是必要的，扩展首部可选

31.8 IPv6 基本首部格式

虽然 IPv6 必须容纳更长的地址，但它的基本首部所包含的信息却比 IPv4 数据报首部少一些。IPv4 数据报首部中的选项和一些固定字段在 IPv6 中被转移到了扩展首部中。总之，数据报首部的变化反映了协议的变化。

- 对齐方式已经从 32 位的整数倍改为 64 位的整数倍。
- 取消了首部长度字段，数据报长度字段被有效载荷长度（PAYLOAD LENGTH）字段所取代。
- 源地址和目的地址字段的大小都增加到了 16 八位组。
- 分片信息已从基本首部的固定字段转移到了一个扩展首部中。
- 生存时间（TIME-TO-LIVE）字段被跳限制（HOP LIMIT）字段取代。
- 服务类型（SERVICE TYPE）字段改名为通信量类别（TRAFFIC CLASS）字段，并用流标号（FLOW LABEL）字段扩展。
- 协议（PROTOCOL）字段由一个新的指明了下一首部类型的字段代替。

图 31.2 所示为 IPv6 基本首部的内容和格式。IPv6 基本首部中的不少字段都和 IPv4 首部中的字段直接对应。与 IPv4 一样，IPv6 基本首部最前面 4 位的版本（VERS）字段指明了协议的版本，在 IPv6 数据报中，VERS 字段的值都是 6。如同 IPv4 一样，源地址（SOURCE ADDRESS）和目的地址（DESTINATION ADDRESS）字段指明发送方和期望的接收方的地址，只不过在 IPv6 中要求每个地址字段为 16 八位组。IPv6 的跳限制（HOP LIMIT）字段对应 IPv4 的生存时间（TIME-TO-LIVE）字段，不同之处是 IPv4 将生存时间解释为跳数和最长时间的组合，而 IPv6 将该值解释为数据报在被丢弃前可以进行的最大跳数的严格阈值。

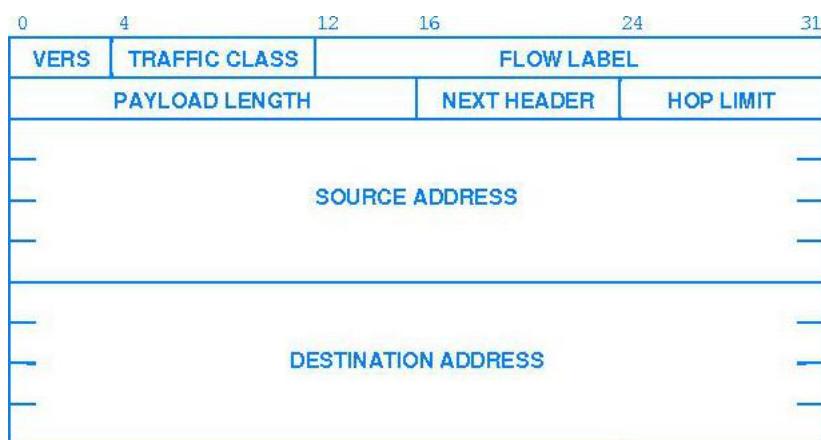


图 31.2 40 八位组的 IPv6 基本首部的格式。每个 IPv6 数据报以一个基本首部开始

IPv6 以一种新的方式处理数据报长度的说明。首先，因为基本首部的长度固定为 40 八位组，所以基本首部不必包括一个首部长度字段。第二，IPv6 以一个 16 位有效载荷长度（PAYLOAD LENGTH）字段取代了 IPv4 的数据报长度字段，该字段指明除去首部自身的长度外，数据报所携带的八位组数。因此，IPv6 数据报可以容纳 64 K 八位组的数据。

基本首部中有两个字段用于转发判决。IPv4 的服务类型（SERVICE CLASS）字段已经重命名为通信量类别（TRAFFIC CLASS）。此外，IPv6 有一个新机制用于支持资源预留并允许路由器将每个数据报与一个给定的资源分配相关联。作为其底层抽象概念，**流**（flow）指的是互联网上的一条路径，这条路径上沿途的路由器要保证指明的服务质量。基本首部的流标号（FLOW LABEL）字段中包含的信息被路由器用于将数据报与特定的流和优先级相关联。例如，两个需要发送视频的应用程序可以建立一个流，它们所需的带宽和时延在这个流上可以得到保证。另一种方式是，网络的提供者可能要求用户指明他所期望的服务质量，然后使用一个流来限制特定计算机或特定应用程序所发送的通信量。注意，流也可由某个组织用于管理网络资源，以保证所有应用程序能够公平地使用网络。在把数据报和特定的流相关联时，路由器要使用数据报源地址和流标识符的组合。现小结如下：

每个 IPv6 数据报都以 40 八位组的基本首部开始，该基本首部包含源地址和目的地址、最大跳数限制、通信量类别、流标号以及下一首部的类型。因此，IPv6 的数据报除了数据之外，至少还要包括 40 八位组。

31.9 IPv6 的扩展首部

选择一个固定的基本首部后面跟一组可选的扩展首部的模式，是对通用性和有效性的折中。要彻底地具有通用性，IPv6 还需要包含支持分片、源路由以及鉴别等功能的机制。然而，为所有这些机制在数据报首部中安排固定的字段是不合算的。因为大多数数据报并没有使用所有的机制，而 IPv6 的巨大地址空间加剧了效率上的降低。例如，在一个局域网中发送数据报时，包含空地址字段的数据报首部会在每个帧中占相当大的比重。更重要的是，设计人员认识到没有人能够预言今后可能还会需要什么功能。

IPv6 扩展首部模式的操作方法与 IPv4 的选项相似，发送方可以选择在给定的数据报中要包含或忽略哪些扩展首部。因此，扩展首部提供了最大限度的灵活性。现小结如下：

IPv6 的扩展首部与 IPv4 的选项相似。每个数据报只包含那些提供它要使用的功能的扩展首部。

31.10 IPv6 数据报的解析

每个基本首部和扩展首部都含有一个下一首部（NEXT HEADER）字段，被中间路由器和最终目的站用来解析数据报。处理过程是顺序进行的，即每个首部中的下一首部字段都说明了跟在后面的是什么。例如，图 31.3 所示为三个数据报的下一首部字段，这三个数据报分别包含零个、一个和两个扩展首部。

为了提高处理速度，IPv6 要求把中间路由器使用的扩展首部放在最终目的站使用的扩展首部之前。在 IPv6 中，术语**逐跳首部**（hop-by-hop header）指的是中间路由器必须处理的扩展首部。因此可以说逐跳首部位于端对端首部之前。

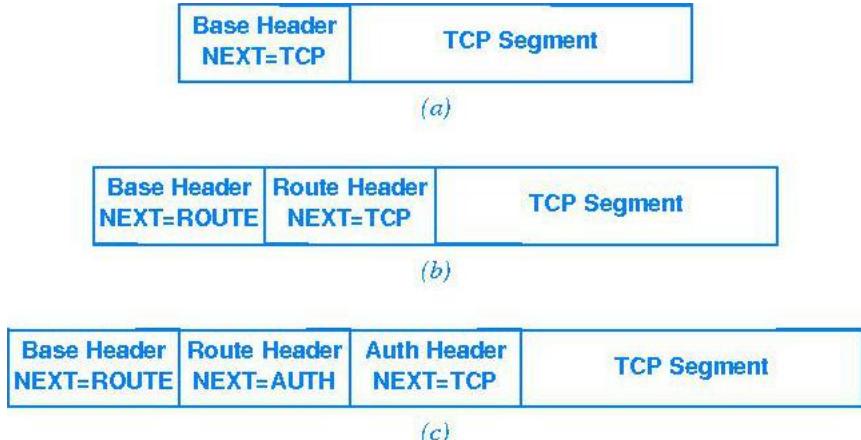


图 31.3 三个数据报，它们分别是：(a) 只有一个基本首部；(b) 一个基本首部和一个扩展首部；(c) 一个基本首部和两个扩展首部。每个首部中的下一首部字段指明了跟在这个首部后面的类型

31.11 IPv6 的分片和重装

IPv6 允许数据报的分片，并且有一个扩展首部用于指明数据报是一个分片。与 IPv4 一样，IPv6 也把数据报重装的任务安排给了数据报的最终目的站。不过，设计者们还是做了一些修改，避免了路由器的分片。回顾 IPv4，当数据报需要通过某个网络，而它的长度对该网络的 MTU 来说又太大时，IPv4 要求中间路由器对数据报进行分片。在 IPv6 中，分片是端对端的，在中间路由器上不会发生分片。负责分片的源站有两个选择：或者使用 1280 八位组的**保证的最小 MTU** (guaranteed minimum MTU)，或者使用**路径 MTU 发现** (Path MTU Discovery) 技术，以识别到达目的站的沿途中最小的 MTU。无论使用哪种方法，都是由源站对数据报分片，以使每个分片小于预计的路径 MTU。图 31.4 所示为**分片扩展首部** (Fragment Extension Header) 中的内容。

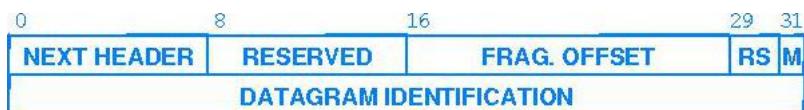


图 31.4 分片扩展首部的格式

IPv6 保留了基本的 IPv4 分片功能。每个分片必须是 8 八位组的倍数，M 字段中有一位标记了最后一个分片，这类似于 IPv4 中的更多分片(MORE FRAGMENTS)位，而数据报标识(DATAGRAM IDENTIFICATION)字段承载的是一个唯一的标识符，接收方使用此标识符来收集分片^①。最后，RS 字段目前是保留未用的字段，在传输中将这两个位设置为 0，由接收方忽略。

31.12 端到端分片的后果

采用端到端分片的动机是要减少路由器上的额外开销，使每个路由器在单位时间内处理更多的

^① 为了适应速度更高的网络，IPv6 将 IPv4 的标识字段扩展到了 32 位。

数据报。IPv4 分片需要的 CPU 开销确实相当大。在一个常规的路由器中，如果路由器对收到的所有数据报都进行分片，其 CPU 的利用率可以达到 100%。但是，端到端的分片带来了一个严重后果：它改变了 IPv4 的最基本假设，即路由是动态改变的。

为了理解端到端的分片所带来的后果，我们回顾一下 IPv4，它被设计为允许在任何时候改变路由。例如，如果一个网络或者路由器出了故障，通信量可以重新选择另一条不同的路径。这种系统的主要优点是它的灵活性，通信量可以另外选择一条路径而不必中断服务，也不必通知源站和目的站。然而，在 IPv6 中，就不能这样轻易改变路由了，因为改变路由可能会改变路径 MTU。如果新路由的路径 MTU 小于原来的路径 MTU，那么或者由中间路由器对数据报分片，或者必须通知最初的源站。我们把这个问题总结如下：

使用端到端分片的网际协议要求发送方发现到达每个目的站的路径 MTU，并将任何大于此路径 MTU 的外发数据报分片。端到端的分片不能适应路由的改变。

为了解决路由变更影响路径 MTU 这个问题，IPv6 包括了一条新的 ICMP 差错报文。当路由器发现需要分片时，就把报文送回源站。当源站接收到这样的报文时，就要再执行一次路径 MTU 发现功能并判断新的最小 MTU，然后根据新的值将数据报分片。

31.13 IPv6 源路由

IPv6 保留了由发送方指明一条不严格的源路由的能力。与 IPv4 通过选项提供源路由所不同的是，IPv6 使用了一个独立的扩展首部。如图 31.5 所示，路由选择首部中的前四个字段是固定的。路由选择类型（ROUTING TYPE）字段指明路由选择信息的类型，目前有定义的类型只有类型 0，对应于不严格的源路由。特定类型数据（TYPE-SPECIFIC DATA）字段包含的是数据报必须经过的路由器的地址列表。剩余分片（SEG LEFT）字段指明列表中包含的地址总数。最后，首部扩展长度（HDR EXT LEN）字段指明路由选择首部的大小。

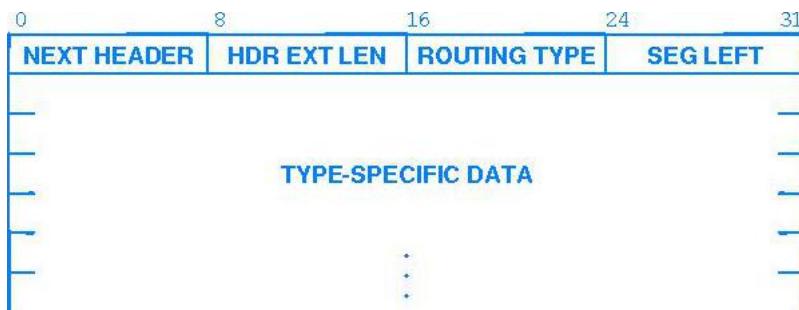


图 31.5 IPv6 路由选择首部的格式。目前只定义了类型 0（不严格的源路由）

31.14 IPv6 的选项

看起来似乎 IPv6 的扩展首部完全取代了 IPv4 的选项，但设计者们还是建议了两个附加的 IPv6 扩展首部来容纳其他扩展首部所不包含的杂项信息。这两个附加首部分别是逐跳扩展首部和端对端扩展首部。正如它们的名字所暗示的，这两个首部将需要在每一跳查看的一组选项和只有在目的站才进行解释的一组选项区分开。

虽然这两个选项首部各有一个唯一的类型码，但它们都使用图 31.6 所示的格式。

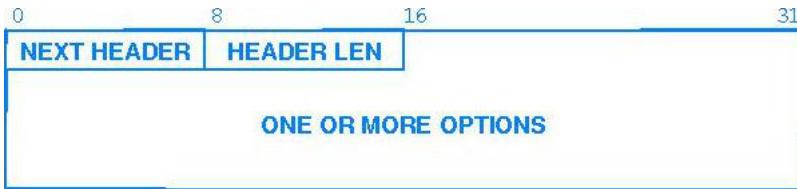


图 31.6 IPv6 选项扩展首部的格式。逐跳和端到端选项首部都使用相同的格式。用前一个首部的下一首部字段来区分这两种类型

照例，下一首部字段给出了跟在后面的首部的类型。由于选项首部没有固定的长度，首部长度（HEADER LEN）字段指明了该首部的全长。一个或多个选项（ONE OR MORE OPTIONS）区域代表了由多个选项组成的序列。图 31.7 表明了每个选项如何编码，包括类型、长度和值^①。选项既不对齐，也不填充。



图 31.7 IPv6 选项扩展首部中的单个选项的编码。每个选项包括 1 八位组的类型和 1 八位组的长度，接着是选项的零个或多个八位组的数据

如图 31.7 所示，IPv6 的选项沿用了 IPv4 选项的格式。每个选项以 1 八位组的类型（TYPE）字段开始，接着是 1 八位组的长度（LENGTH）字段。如果选项需要附加的数据，长度字段后面跟的是构成值（VALUE）字段的八位组。

如果主机或路由器不明白选项的含义，每个选项的类型字段的两个高位将指明它们此时应该如何处理这个数据报：

此外，TYPE 字段中的第三位指明选项是否可以在传输中被改变。拥有这样的信息对鉴别是很重要的。在处理鉴别问题时，传送中可以改变的选项的内容可看成是一些 0。

31.15 IPv6 地址空间的大小

在 IPv6 中，每个地址占据 16 八位组，这是 IPv4 地址长度的四倍。这样巨大的地址空间可以保证 IPv6 能承受任何合理的地址分配方案。事实上，如果设计者们以后决定变更编址方案，地址空间也大得足以适应地址的重新分配。

要体会 IPv6 的地址空间的大小是困难的。理解这个问题的一种方法是将其与人口数量联系起来：这个地址空间是如此之大，以至于地球上的每个人都可以有足够的地址来构成他们各自的像目前这个因特网一样大的互联网。理解该问题的第二种方法是和可用的物理空间相联系：地球表面大

^① 在文献中，类型、长度和值的编码有时称为 **TLV 编码** (TLV encoding)。

约是 5.1×10^8 平方公里，它意味着地球表面每平方米上的地址就超过 10^{24} 个。另外可以从地址耗尽的角度来理解。例如，考虑将所有可能的地址分配出去需要多长时间。16 八位组的整数可以存储 2^{128} 个值，因此这个地址空间大于 3.4×10^{38} 。如果以每微秒分配一百万个地址的速度进行分配，需要 10^{20} 年的时间才能将所有可能的地址分配完毕。

31.16 IPv6 的冒号十六进制记法

虽然 IPv6 解决了地址容量不够的问题，但巨大的地址范围又带来了一个有趣的新问题：维护互联网的人们必须阅读、输入和操纵这些地址。很明显，二进制表示是不可取的。但 IPv4 所使用的点分十进制记法也不能把这些地址充分简洁地表示出来。为了理解其中的道理，我们可以考虑用点分十进制表示的一个 128 位数的例子：

104.230.140.100.255.255.255.255.0.0.17.128.150.10.255.255

为了使地址再稍简洁一些并且容易输入，IPv6 的设计者们建议使用**冒号十六进制记法** (colon hexadecimal notation, 简写为 colon hex)，它把每十六位的量用十六进制值表示，各量之间用冒号分隔。例如，把前面所给的点分十进制数表示的值改为冒号十六进制表示，并且使用相同的字间距印刷，就变成了：

68E6:8C64:FFFF:FFFF:0:1180:96A:FFFF

冒号十六进制记法和点分十进制记法相比具有很明显的优点，它只需要较少的数字和较少的分隔符。此外，冒号十六进制记法还包含两项技术，使得它非常有用。首先，冒号十六进制记法允许零压缩 (zero compression)，即一连串重复的零可以用一对冒号取代，例如：地址

FF05:0:0:0:0:0:B3

可以写成：

FF05::B3

为了保证零压缩不会引起二义性，建议中规定，在任一地址中，只能使用一次零压缩。零压缩技术对已建议的分配方案特别有用，因为会有许多地址包含连续的零串。第二，冒号十六进制记法可以结合点分十进制记法的后缀。下面将会看到，这种结合旨在用于 IPv4 向 IPv6 的转换阶段。例如，下面的字符串是一个合法的冒号十六进制表示：

0:0:0:0:0:128.10.2.1

注意，虽然被冒号分隔的每个值指定一个 16 位的量，但每个点分十进制部分的值指定的是一个八位组的值。零压缩当然也可以用于上面的数值，这样可以生成看上去与 IPv4 地址十分相似的等效冒号十六进制记法：

::128.10.2.1

最后，通过允许在地址后面跟一个斜杠和一个指定位数目的数字，IPv6 可扩展为类似于 CIDR 的记法。例如：

12AB::CD30:0:0:0/60

指的是地址的前六十位或 12AB00000000CD3（用十六进制表示）。

31.17 三种基本 IPv6 地址类型

如同 IPv4 一样，IPv6 把一个地址与一条特定的网络连接相关联，而不是与一台特定的计算机相关联，因此地址的分配也类似于 IPv4：一个 IPv6 路由器具有两个或多个地址，而只有一条网络连接的 IPv6 主机仅需要一个地址。在 IPv4 中给一个物理网络分配一个前缀，在 IPv6 中仍然保留了（并且扩展了）这种分级的地址结构。不过，为了使分配和修改地址更方便，IPv6 允许给某个网络分配多个前缀，并允许一台计算机向某个接口同时分配多个地址。

除了允许每个网络连接同时拥有多个地址以外，IPv6 还扩充了（在某些情况下是统一了）IPv4 的特殊地址。一般来讲，一个报文的目的地址可以归为以下三类之一：

单播 (unicast) 目的地址指明一台计算机（主机或路由器），数据报将选择一条最短的路径到达目的站。

任播 (anycast) 目的站是共享一个地址的一组计算机的集合（它们可能在不同的位置上），数据报将选择一条最短的路径，并只交付给组中的一个成员（即最近的一个成员）^①。

多播 (multicast) 目的站是一组计算机，它们可能位于不同的地方。数据报将通过硬件多播或广播（如果可以）交付给该组中的每一个成员。

31.18 广播和多播的二重性

IPv6 没有采用术语**广播** (broadcast) 或**定向广播** (directed broadcast) 来表示将数据报交付给一个物理网络或逻辑 IP 子网中的所有计算机。事实上，它采用的术语是**多播** (multicast)，并将广播看成是多播的一个特例。对了解网络硬件的人而言，选择这种方式看上去有些古怪，因为支持广播的硬件技术比支持多播的多。事实上，硬件工程师更愿意把多播看成是广播的一种受限形式。就像处理广播分组一样，硬件将多播分组发送给网络中的所有计算机，然后由每台计算机上的接口硬件对收到的所有多播分组进行过滤，只接收那些由管理软件指定该硬件接口可以接收的分组。

从理论上讲，选择多播还是受限形式的广播是无关紧要的，因为它们可以互相模仿。也就是说，广播和多播是提供同一功能性的互相重复的东西。要领会其中的原因，我们可以考虑它们如何相互模仿对方。如果广播可用，则可以把一个分组发送给网络中的所有机器，然后让每台计算机中的软件决定是接受还是丢弃接收到的分组，通过这种方式可以将分组发送到所期望的一组机器上。如果多播可用，可以让网络中的所有计算机都监听类似于第 16 章中讨论的**全系统** (all systems) 群组的一个多播群组，以此方式将分组交付给网络中的所有机器。

31.19 模拟广播——工程上的选择

广播和多播在理论上是相互具有二重性的，但了解这一点对从两者中做出选择并无帮助。为了弄清 IPv6 的设计者们为何放弃广播而选择多播作为中心抽象，就要考虑应用程序，而不是考虑底层的硬件。应用程序需要与另一个应用程序或一组应用程序进行通信。直接的通信最好由单播来处理，而群组通信最好由多播或广播处理。为了最大程度地提供灵活性，组的成员身份不应由网络连接来决定，这是因为组成员可以位于任何地点。在一个像因特网这样的大型互联网中，对所有的组通信使用广播是无法承受的。

IPv6 的设计者们在 IPv6 中预定义了代替 IPv4 网络广播地址的多播地址也就不足为奇了。因此，除了自己的单播地址以外，要求每个路由器都接受地址为本地环境下的**全路由器** (All Routers) 多播群组的分组。

31.20 建议的 IPv6 地址空间分配

对于如何划分 IPv6 地址空间的问题引起了很大的争论。争论围绕着两个中心问题：怎样管理地址分配以及怎样将一个地址映射到一条路由。第一个问题集中在设计管理机构层次这一实际问题上。目前因特网使用的是两级层次结构：网络前缀（由 ISP 分配）和主机后缀（由各个组织自己分配）。而 IPv6 与此不同，它的巨大的地址空间允许具有多级层次结构或多个层次体系。第二个问题

^① 以前把任播地址称为**群集** (cluster) 地址。

的重点是计算效率。与分配地址的管理机构层次无关，路由器必须检查每个数据报，并选择一条通往目的站的路径。为了使高速路由器保持较低的花费，选择路径所需的处理时间一定要小。

如图 31.8 所示，IPv6 的设计者建议采用类似 IPv4 所用的方案来分配地址块。尽管地址的最前面八位已经足够标识该地址的类型，但是地址空间并没有被分割成相等的块。

Binary Prefix	Type Of Address	Part Of Address Space
0000 0000	Reserved (IPv4 compatibility)	1/256
0000 0001	Unassigned	1/256
0000 001	NSAP Addresses	1/128
0000 01	Unassigned	1/64
0000 1	Unassigned	1/32
0001	Unassigned	1/16
001	Global Unicast	1/8
010	Unassigned	1/8
011	Unassigned	1/8
100	Unassigned	1/8
101	Unassigned	1/8
110	Unassigned	1/8
1110	Unassigned	1/16
1111 0	Unassigned	1/32
1111 10	Unassigned	1/64
1111 110	Unassigned	1/128
1111 1110 0	Unassigned	1/512
1111 1110 10	Link-Local Unicast Addresses	1/1024
1111 1110 11	IANA - Reserved	1/1024
1111 1111	Multicast Addresses	1/256

图 31.8 建议将 IPv6 地址划分为与 IPv4 的类 (classe) 相似的类型 (type)。与 IPv4 一样，地址的前缀决定该地址的类型

如图 31.8 所示，目前只分配了 15% 的地址空间。当对地址空间的需求量增大时，IETF 将使用其余的部分。尽管分配稀疏，但已经对地址进行了选择，使处理过程的效率更高。例如，地址的最高位八位组用于区别多播（全 1）和单播（既有 0 也有 1）。

31.21 嵌入的 IPv4 地址和过渡

虽然图中标明前缀 0000 0000 是保留的，但设计者们计划将其中一小部分用来对 IPv4 地址编码。实际上，任何地址若开始的 80 位全是 0，接着 16 位全是 1 或全是 0，那么它的低 32 位就是一个 IPv4 的地址。16 位字段的值指示这个结点是否仍有常规的 IPv6 单播地址。图 31.9 所示为这两种形式。

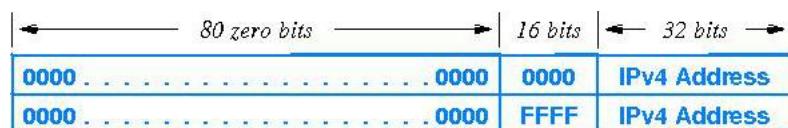


图 31.9 IPv4 地址在 IPv6 地址中的编码。如果结点还包含常规的 IPv6 地址，则 16 位字段包含 0000，否则包含 FFFF

在从 IPv4 向 IPv6 过渡时需要这种编码，其原因有两点。第一，计算机在分配合法的 IPv6 地

址之前，可能会选择先将它的软件从 IPv4 向 IPv6 升级。第二，运行 IPv6 软件的计算机可能需要与只运行 IPv4 软件的计算机通信。

有了将 IPv4 地址编码到 IPv6 地址中的方法，并不能解决两种版本的互操作问题。除了地址编码以外，还需要进行转换。为了使用转换器，IPv6 计算机会生成含有 IPv4 目的地址编码的 IPv6 数据报。IPv6 计算机把数据报发送给转换器，这个转换器再用 IPv4 与目的站通信。当转换器收到来自目的站的回答时，再将这个 IPv4 数据报转换为 IPv6 数据报，并将它发回给 IPv6 源站。

因为高层软件会鉴别地址的完整性，所以看起来地址转换似乎会失败。尤其是 TCP 和 UDP 在检验和计算中使用了**伪首部** (pseudo header)，在伪首部中包含源站和目的站的协议地址，所以改变这些地址可能会影响检验和的计算。不过，设计者们经过精心策划，让 IPv4 机器上的 TCP 和 UDP 能够和 IPv6 机器上的相应运输协议进行通信。为了避免检验和的不匹配，IPv4 地址的 IPv6 编码是经过选择的，无论对 IPv4 地址还是编码后的 IPv6 地址，其 16 位二进制反码检验和是一致的。要点是：

除了选择新的国际协议的技术细节以外，致力于 IPv6 的 IETF 也十分关注寻找一种将当前协议过渡到新协议的方法。具体来讲，目前的 IPv6 建议允许在 IPv6 地址中含有 IPv4 地址的编码，而且地址转换不会改变伪首部的检验和。

31.22 未指明的地址和环回地址

与 IPv4 一样，有几个 IPv6 地址是分配了特殊含义的。例如，全 0 的地址：

0:0:0:0:0:0:0

是**未指明的地址** (unspecified address)，不能分配给任何计算机或作为目的地址，只能在计算机不知道自己的地址的情况下，被引导程序用来作为源地址。

与 IPv4 类似，IPv6 也有**环回地址** (loopback address)，用于测试软件。IPv6 的环回地址是：

0:0:0:0:0:0:1

发送到环回地址的任何数据报都将被交付给本地机器，而决不允许在外发数据报中作为目的地址。

31.23 单播地址的结构

最新版 IPv6 的单播编址结构与 IPv4 的子网编址方案类似，其中每个地址被划分为三个概念部分。在 IPv6 中，这三个部分分别对应于：一个用于数据报路由选择的全局唯一的前缀；一个子网 ID，用于区分某个给定网点上的多个物理网络；一个接口 ID，用于标识子网中的具体连接。图 31.10 体现了 IPv6 地址如何划分成三个部分。

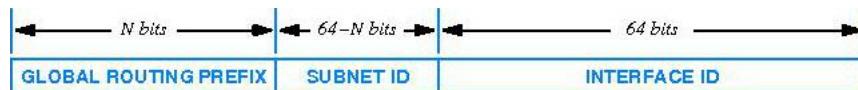


图 31.10 IPv6 单播地址划分成三个概念部分。接口 ID 始终占据 64 位

31.24 接口标识符

如图 31.10 所示，IPv6 单播地址的低 64 位标识了具体的网络接口。与 IPv4 不同，IPv6 后缀选

择得很大，足以适应用接口硬件地址的直接编码。把硬件地址编码到 IP 地址有两个重要意义。首先，IPv6 没有使用 ARP 技术把 IP 地址解析成硬件地址。相反，IPv6 使用了新版的 ICMP (ICMPv6) 中的**邻站发现协议** (neighbor discovery Protocol)，允许结点判断哪一台计算机是和它直接连接的邻站。其次，为了保证互操作性，所有计算机对硬件地址必须使用同样的编码。其结果是，IPv6 标准明确指明了如何对不同形式的硬件地址进行编码。在最简单的情况下，硬件地址直接放在 IPv6 地址中，另外一些格式使用比较复杂的变换。

有两个编码的例子将有助于阐明以上概念。例如，IEEE 定义了一个标准的 64 位全球唯一地址格式，称为 EUI-64。当 EUI-64 地址编码到一个 IPv6 地址中时，唯一需要的改动是将高位八位组中的第 6 位翻转，这个位指示的是地址是否全球唯一。

对于常规的 48 位以太网地址，则要求比较复杂的改动。图 31.11 所示为这种编码方式。如图所示，原始地址中的比特在编码后是不连续的。事实上，它们中间插入了一个值为 FFFE_{16} (十六进制) 的 16 位。此外，指示地址是否是全球范围的第 6 位从 0 变成了 1。如图 31.11 所示，地址中的其他位，包括群组位 (标记为 g)、生产接口的公司的 ID (标记为 c) 以及生产商的扩展名都进行了复制。

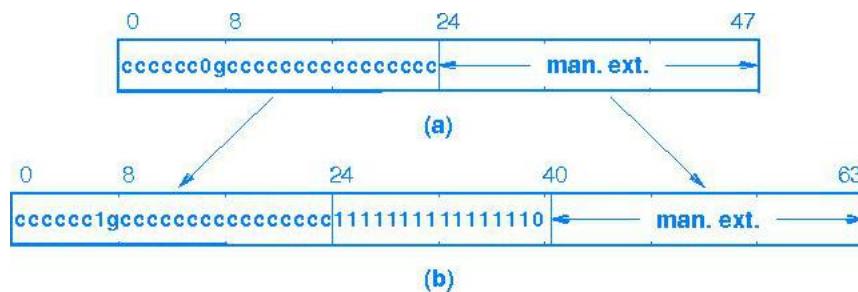


图 31.11 (a) 48 位以太网地址的格式，其中标记为 c 的位标识公司，man.ext. 指明了生产商的扩展名；(b) 在 IPv6 单播地址的低 64 位对该地址的编码形式

31.25 本地地址

除了前面已经描述过的全球单播地址以外，IPv6 还包括了具有本地范围意义的单播地址前缀。如图 31.8 所示，标准定义了一个**本地链路地址** (link-local address)，这些地址都限制在一个网络范围内。另外还建议了几种受限地址的形式。路由器以下面的方式来遵守这些范围规则：路由器决不会把一个含有本地范围的地址转发到指定范围之外（即离开本地网络）。

本地链路地址提供了在单个物理网络上进行通信的手段，数据报不会有被转发到互联网上的危险。例如，IPv6 结点在执行邻站发现时要使用本地链路地址。其范围规则指明，只有和发送方在同一物理网络上的计算机才能接收到邻站发现报文。类似地，与**孤立网络** (isolated network) (即没有连接路由器的网络) 连接的计算机可以使用本地链路地址进行通信。

31.26 自动配置和重新编号

IPv6 的设计可以支持**无服务器自动配置** (serverless autoconfiguration)^①，以允许计算机不要求管理员指明地址就能进行通信。前面讨论的两个特性，即本地链路地址和嵌入式接口标识符，使自动配置的实现成为可能。在自动配置开始时，计算机首先要将本地链路前缀：

1111 1110 10

^① 无服务器自动配置又称为**无状态自动配置** (stateless autoconfiguration)。

和 54 个 0 位以及它的 64 位接口标识符结合起来，生成本地链路地址。

一旦计算机验证了本地链路地址是唯一的，就用该地址发送一个**路由器恳求** (router solicitation)，用于向路由器恳求额外信息。如果网络上有路由器，则该路由器就会发送一个**路由器通告** (router advertisement) 进行回答，告诉计算机可用于全球地址的前缀。当路由器的通告到达时，该计算机将发送通告的路由器作为自己的默认路由器。最后，通告中有一个标志告诉计算机是依靠自动配置还是使用常规的管理配置 (即 DHCP)。

为了实现网络的**重新编号** (renumbering)，IPv6 允许路由器限制计算机保留前缀的时间。为此，路由器的通告为每个前缀指定两个时间值：有效生存时间和最佳生存时间。主机必须监听额外的路由器通告。当前缀的最佳生存时间到期时，前缀仍然是有效的，但是主机必须尽可能地为所有通信使用另一个前缀。当有效生存时间过期时，即使当前仍有通信在进行中，主机也必须停止使用该前缀。

支持重新编号的思想很简单：通告一个新的（即额外的）前缀，等待一段不长的时间，然后废止旧前缀的使用。但是实际上，要实现这种转换是很困难的。当改变出现时，生存时间很长的运输层连接将被打断，与当前地址绑定的服务器将不可能接收到用新地址进行的联络，并且路由选择也必须改变。在转换期间，每个接口都有多个地址，并且这些地址的生存时间都要与 DNS 协调，以防止再次使用了缓存中的旧地址资源记录副本。因此，从实践的角度来看，重新编号并不简单。

31.27 小结

IETF 已经定义了下一代网际协议，因为给它分配的版本号是 6，所以称为 IPv6。IPv6 保留了目前协议（即 IPv4）的许多基本概念，但是改变了大多数细节。与 IPv4 一样，IPv6 提供无连接的、尽最大努力的数据报交付服务。不过，IPv6 的数据报格式与 IPv4 的完全不同，IPv6 提供了一些新的特性，如鉴别技术，并支持流标签。

IPv6 的每个数据报的布局为一组首部及跟在首部后的数据。数据报总是以 40 八位组的基本首部开始，该首部包含源站和目的站地址、通信量类别以及流标识符。基本首部后面可能跟着零个或多个扩展首部，再后面是数据。扩展首部是可选的，IPv6 使用这些扩展首部来存储在 IPv4 选项中编码的大部分信息。

IPv6 的地址有 128 位长，其地址空间是如此之大，以至在可预见的未来不可能耗尽。IPv6 使用地址前缀来确定位置以及解释地址字段中的其他部分。除了常规的单播和多播地址以外，IPv6 还定义了任播地址。可以把一个任播地址分配给一组计算机，而向该地址发送的数据报将只交付给该集合中的一台计算机（也就是离源站最近的计算机）。

IPv6 支持自动配置和重新编号。在一个孤立网络上的每台主机都会产生一个用于通信的唯一的本地链路地址。主机还可以使用本地链路地址来发现路由器和得到全局前缀信息。为了实现重新编号，所有的前缀都分配了生存时间，如果正在使用的前缀的生存时间到期，那么主机就必须使用一个新的前缀。

31.28 深入研究

已经有很多包含与 IPv6 有关信息的 RFC 出现。Deering and Hinden [RFC 2460] 指明了基本的协议。Thomson and Narten [RFC 2462] 描述无状态地址自动配置。Narten, Nordmark, and Simpson [RFC 2461] 讨论邻站发现。Conta and Deering [RFC 2463] 指明 IPv6 的伴生协议 ICMPv6。Crawford [RFC 2464] 描述在以太网上传输时的 IPv6 的封装。

许多 RFC 讨论的重点都是 IPv6 的编址。Hinden and Deering [RFC 3513] 描述了包括前缀的含义在内的基本编址体系结构。Hinden, O'Dell, and Deering [RFC 3587] 考虑了可聚集的全局单播地址

的格式。Hinden and Deering [RFC 2375]指明多播地址的分配。Johnson and Deering [RFC2526]描述保留的任播地址。有关 64 位 EUI 格式的信息可以在以下地址找到：

<http://standards.ieee.org/regauth/oui/tutorials/EUI64.html>

31.29 习题

1. 目前的 IPv6 标准没有首部检验和，这种方法的优缺点各是什么？
2. 各扩展首部应当如何排列才能使处理时间最小？
3. 虽然 IPv6 的地址是按层次分配的，但是路由器在选择路由时并不需要将地址完全解析。为有效的路由选择设计一个算法和数据结构。提示：可以考虑最长匹配方法。
4. 证明 128 位地址空间比实际需要的大，而 96 位地址就可以提供足够的容量。
5. 假设你所在的组织想采用 IPv6。设计一种地址分配方案，为组织内的每台主机分配一个地址。你是否会选择一种层次分配方案呢？为什么？
6. 将以太网地址编码到 IPv6 地址中有什么主要优点，主要缺点呢？
7. 考虑这样一台主机，它通过用标准本地链路前缀和 48 位以太网地址一起编码而形成本地链路地址。得到的地址能保证在网络上是唯一的吗？为什么？
8. 阅读 IPv6 的邻站发现协议。在前面的习题中，标准是否指明主机必须使用邻站发现协议来验证地址的唯一性？为什么？
9. 如果要求你为 IPv6 单播地址的顶层、下一层和网点 ID 字段选择大小，你会为各个字段分配多大的长度？为什么？
10. 阅读 IPv6 的鉴别和安全首部。为什么要建议这两个首部？
11. IPv6 的最小 MTU (1280) 会怎样影响它的灵活性？