[概述 2](#_Toc508005515)

[物理层 5](#_Toc508005516)

[链路层 6](#_Toc508005517)

[PPP链路 9](#_Toc508005518)

[以太网 10](#_Toc508005519)

[ARP 12](#_Toc508005520)

[帧发送 13](#_Toc508005521)

[帧接受 14](#_Toc508005522)

[介质访问控制 15](#_Toc508005523)

[令牌环 18](#_Toc508005524)

[FDDI 19](#_Toc508005525)

[HDLC 20](#_Toc508005526)

[网桥 21](#_Toc508005527)

[交换机 22](#_Toc508005528)

[网络层 23](#_Toc508005529)

[拥塞控制 23](#_Toc508005530)

[IP地址 24](#_Toc508005531)

[路由策略(路由选择) 26](#_Toc508005532)

[路由机制(分组转发) 27](#_Toc508005533)

[广播与多播(组播) 34](#_Toc508005534)

[分片 38](#_Toc508005535)

[NAT 40](#_Toc508005536)

[IPv6 41](#_Toc508005537)

[传输层 42](#_Toc508005538)

[UDP 42](#_Toc508005539)

[TCP 45](#_Toc508005540)

[TCP发送 53](#_Toc508005541)

[TCP接收 55](#_Toc508005542)

[TCP连接 56](#_Toc508005543)

[ICMP 60](#_Toc508005544)

[IGMP 61](#_Toc508005545)

[应用层 63](#_Toc508005546)

[模型 63](#_Toc508005547)

[DNS 64](#_Toc508005548)

[FTP 67](#_Toc508005549)

[SMTP、POP3 70](#_Toc508005550)

[HTTP 73](#_Toc508005551)

[DHCP 76](#_Toc508005552)

[SNMP 77](#_Toc508005553)

[TELNET 78](#_Toc508005554)

# 概述

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | 物理层 | 链路层 | 网络层 | 传输层 | | 应用层 |
| TCP | UDP |
| 服务 | 发送、接收数据 | 相邻主机间通信 | 任意主机间通信 | 进程之间端对端  逻辑通信 | | 产生报文 |
| 中继系统 | 中继器、集线器 | 网桥、交换机 | 路由器 | 网关 | | 网关 |
|  |  |  | 不可靠 | 可靠 | 不可靠 |  |
|  |  |  | 无连接 | 有连接 | 无连接 |  |
| 方向 |  |  |  | 全双工 |  |  |
|  |  |  |  | 单播  (点对点)  (一对一) | 单播、广播、多播 |  |
| 传输单位 |  |  |  | 字节 | 报文 |  |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 通信端  (TCP连接) | **socket=(IP地址:端口号)**  网络中真正进行通信的实体=某主机中的某进程=TCP连接的端点，使用socket表示  两个socket唯一地确定一条TCP连接：TCP连接={socket1,socket2}={(IP1:port1),(IP:port2)} | | | | |
| IP地址 | 每一台主机的每一个接口，对应一个全世界唯一的IP地址(32bit)  IP地址=网络号+主机号=000.000.000.000~255.255.255.255  网络号：全0→本网络，全1→环回测试，  主机号：全0→本主机，全1→该网络所有主机  具有本地磁盘的主机，从磁盘上的配置文件中读取其IP地址,无盘机则需要采用其他方法 | | | | |
| 网络号 | 地址  掩码 | IP地址&掩码=网络前缀→地址块，高位部分(11…1)网络前缀，低位部分(00…0)主机号  如果随IP地址还给出了子网掩码，那么该IP地址的子网编址就是确定的：  子网掩码是一个32bit的值，其中值为1的比特留给网络号和子网号，值为为0的比特留给主机号 | | |
| CIDR  记法 | IP地址/前缀位数，如128.14.35.7/20，CIDER技术把小的网络汇聚成大的超网  地址块：相同网络前缀的IP地址组成的集合  使用CIDER表示时，该局域网又称为地址块 | | |
| 类型 | A类 | 0+7bit网络号+24bit主机号，0.0.0.0~~ 127.255.255.255，其中10.0.0.0~10.255.255.255供内部使用，不允许出现在因特网上  127.xx.yy.zz 作为保留地址，用于回路测试 | | |
| B类 | 10+14bit网络号+16bit主机号，128.0.0.0~~ 191.255.255.255，其中172.16.0.0~172.31.255.255供内部使用，不允许出现在因特网上 | | |
| C类 | 110+21bit网络号+8bit主机号，192.0.0.0~~ 223.255.255.255，其中192.168.0.0~192.168.255.255供内部使用，不允许出现在因特网上 | | |
| D类 | 1110+多播组号：224.0.0.0~239.255.255.255 | | |
| E类 | 1111+保留，240.0.0.0~~ 247.255.255.255 | | |
| 端口号 | 16bit | | |  | |
|  | | |  | |
| 服务器端口号  (固定~) | | | 熟知端口号 | 0000~03FF，分派给一些重要的应用进程 |
| 登记端口号 | 0400~0BFF， |
| 客户机端口号  (短暂~) | | | 0C00~FFFF，客户机的进程运行时，获取一个端口号，通信结束后就会放弃该端口号；之后这个端口号可以供其他启动的的客户机进程使用。 | |
|  |  | | | | |
|  |  | | | | |
|  |  | | | | |
|  |  | | | | |

|  |  |
| --- | --- |
| 通信子网 | 物理层、链路层、网络层 |
| 中继系统 | 将多个计算机网络连接起来的中间设备： |

路由器工作在网络层，不转发广播包(255.255.255.255)，因此能够分隔广播域，抑制网络风暴。

交换机工作在链路层，能够分隔冲突域，但不能分隔广播域。

集线器和中继器时物理层设备，既不能分隔广播域也不能分隔冲突域

|  |  |  |
| --- | --- | --- |
| 网络层 |  | |
| 不可靠 | 不保证数据报成功到达目的地，发生错误则丢弃并报错  只提供最大努力交付服务 |
| 无连接 | 每个数据报的处理相互独立，不维护任何关于后续数据报的状态信息，即可以不按发送顺序接收 |
| 传输过程 | 可能中间经过多个路由器，最终到达目标主机；  数据报中dstIP始终不变  数据报中dstMAC不断发生变化，动态指向下一站链路层地址  每到达一个路由器，就要执行一次路由决策，来决定下一站的链路层地址，所有的路由决策都是基于目的IP |
|  |  |
| 网络  异构性 | 没有一种单一的网络可以适应所有用户的要求，因此全世界范围内有数以百万计的网络；  物理层与链路层的差异：寻址方案、网络接入机制、差错处理方法、路由选择机制等 | |
| 网络互联 | 在网络层采用标准化协议(IP协议)  多个网络连接起来，相连接的网络可以是异构的，对外作为一个统一的网络，称为虚拟互联网络(逻辑互联网络) | |

划分子网的好处是可以减小广播域

NAT的表项需要管理员添加，这样可以控制一个内网到外网的网络连接

ARP请求：单播

ARP响应：组播

路由器/网关： 一个路由器拥有多个接口，每个接口对应一个IP地址，对应连接一个网络

路由选择：路由选择处理机→构造、更新、维护路由表

分组转发：交换结构、输入端口、输出端口

路由表：表项称为路由，

每一个主机(包括路由器)都有一个路由表；路由表存储在IP协议内存中；

路由器转发时，要搜索其路由表：

主机发送时，IP层从某个上层收到数据报文，搜索路由表：

IP检查目的IP地址是否为本机的IP地址之一或者IP广播地址。

如果确实是这样，数据报文就被送到由IP首部协议字段所指定的协议模块进行处理。

如果数据报文的目的不是这些地址，那么

如果IP层被设置为路由器的功能，那么就对数据报文进行转发（也就是说，像下面对待发出的数据报文一样处理）；否则数据报文被丢弃。

IP路由选择是逐跳地（ hop- by- hop）进行的。

从这个路由表信息可以看出， IP并不知道到达任何目的的完整路径（当然，除了那些与主机直接相连的目的）。

所有的IP路由选择只为数据报传输提供下一站路由器的IP地址。它假定下一站路由器比发送数据报文的主机更接近目的，而且下一站路由器与该主机是直接相连的。

主机A有一份报文要传到主机B上，主机B的IP地址是140. 252. 13. 33；

主机A搜索自己的路由表，但是没有找到主机B的IP地址或网络地址相匹配的表目；因此只能用默认的表目，把数据报文传给下一站路由器，即主机sun；

主机sun要么就是一个路由器，要么是一个设置了路由功能的主机；

主机sun收到数据报文后，发现数据报文的目的IP地址并不是本机的IP地址，而sun已被设置成具有路由器的功能，因此它把数据报文进行转发给主机etb。

当etb收到数据报文后，它执行与sun主机相同的步骤：数据报文的目的地址不是本机地址，而etb也被设置成具有路由器的功能，于是它也对该数据报文进行转发，采用的也是默认路由表目，把数据报文送到下一站路由器gateway

主机gateway收到数据报文后，搜索自己的路由表，发现目的IP地址（140. 252. 13. 33）在一个和自己直接相连的网络上（以太网140. 252. 13. 0）。

于是，在表中找到匹配的网络地址，数据报文被送到主机B

# 物理层

集线器

中继器

# 链路层

|  |  |
| --- | --- |
|  |  |
| 局域网 | 局域网使用的协议在数据链路层：即结点要给发送的数据封装一层控制信息，用于检错纠错，这层控制信息是数据链路层给出的 |
| 广域网 | 各结点之间距离远超一个城市范围，使用高速链路连接  结点交换机：功能与路由器类似，在单个网络中转发分组，(路由器是在多个网络中转发分组)，执行**分组存储转发**的传输方式  广域网使用的协议在网络层：即结点要给发送的数据封装一层控制信息，用于检错纠错，这层控制信息是网络层给出的  通信子网主要使用分组交换技术，将分布在不同地区的局域网或计算机系统互联起来，达到资源共享的目的 |

/\*

设主机A将数据传输给主机E，但A和E不再同意个局域网中，不能直接传输中间经过了若干其他主机或路由器： A→B→C→D→E

每当一个主机收到报文后，不改变IP数据报，仅改变链路首部

因为链路首部中包含了下一个主机的物理地址dst MAC，而数据报中的dst IP，即E的IP地址，是始终不变的，dst MAC一直处于变化中

链路层收到来自IP层的IP数据报

ARP地址解析：dst IP→dst MAC

2)FTP客户端请求TCP用得到的IP地址建立连接。

3)TCP发送一个连接请求分段到远端的主机，即用上述IP地址发送一份IP数据报（在第18章我们将讨论完成这个过程的细节）。

4)如果目的主机在本地网络上（如以太网、令牌环网或点对点链接的另一端），那么IP数据报可以直接送到目的主机上。

如果目的主机在一个远程网络上，那么就通过IP选路函数来确定位于本地网络上的下一站路由器地址，并让它转发IP数据报。

在这两种情况下，IP数据报都是被送到位于本地网络上的一台主机或路由器。

5)假定是一个以太网，那么发送端主机必须把32bit的IP地址变换成48bit的以太网地址。从逻辑Internet地址到对应的物理硬件地址需要进行翻译。这就是ARP的功能。

ARP本来是用于广播网络的，有许多主机或路由器连在同一个网络上。

6)ARP发送一份称作ARP请求的以太网数据帧给以太网上的每个主机。这个过程称作广播，

ARP请求数据帧中包含目的主机的IP地址(主机名为bsdi)，其意思是“如果你是这个IP地址的拥有者，请回答你的硬件地址。”

目的主机的ARP层收到这份广播报文后，识别出这是发送端在寻问它的IP地址，于是发送一个ARP应答。这个ARP应答包含IP地址及对应的硬件地址。

8)收到ARP应答后，使ARP进行请求—应答交换的IP数据报现在就可以传送了。

9)发送IP数据报到目的主机。

|  |  |  |
| --- | --- | --- |
| 服务 | 以太网 | 无确认的无连接服务  接收方对接收的帧不进行确认  事先不建立逻辑连接，事后也不用释放逻辑连接  不检测帧丢失，检测任务交给上层  用于信道比较可靠并且对实时要求高的网络 |
| 无线通信 | 有确认的无连接服务 |
|  | 有确认的面向连接的服务 |
|  | 相邻主机之间直接的传输，没有中间设备 | |
| 信道 | 点对点 | 一对一， |
| 广播 | 一对多， |
|  | MTU-以太网帧：1500B  MTU-广域网帧：576B  每一种链路层协议规定了帧的**数据部分**的长度上限 | |
| 流量控制 | 限制发送方的发送速率，使接收方有足够的缓存来接收每一个帧  链路层流量控制：控制两相邻主机之间数据链路上的流量  TCP流量控制：控制源端口与目的端口之间数据链路上的流量  反馈机制： | |
| 丢失处理 | 超时重发 | |

|  |  |  |
| --- | --- | --- |
| 封装成帧 | 帧定界 | 发送方的链路层，对来自IP层的报文添加首部和尾部，封装成帧；以帧为单位进行传输，出错时只需重发出错的帧，而不必重发所有数据。 |
| 帧同步 | 接收方的链路层，对收到的二进制比特流，区分帧的起始和终止 |
| MTU | MTU-以太网帧：1500B  MTU-广域网帧：576B  每一种链路层协议规定了帧的**数据部分**的长度上限 |
| 透明传输 | 发送方发送5个连续的1时，后面插入一个0  接受方接收5个连续的1时，自动删除后面的一个0 | |
| 差错检测 |  | |

为防止在传输过程中帧丢失，在可靠链路层传输协议中，发送方对发送的每一个数据帧设计了一个定时器，当计时器到期而该帧的确认帧没有到达时

发送方重新发送该帧。

为保证接收方不会接收重复帧，需要对每个发送的帧编号

|  |  |
| --- | --- |
| 字符计数 | 第一个字节：帧长  帧长包含计数的字节 |
| 字符填充  首尾定界符 | 使用ASCII字符定界一帧的开始和结束  发送方发送的数据中含有特殊字符时，在特殊字符前面添加转移符DLE(0001 0000)  接收方接收的数据中收到转移符时，就知道后面紧跟的是数据信息而不是开始/结束标志 |
| 比特填充  首尾标志 |  |
| 曼彻斯特编码 | 违规编码，适配器接收端有电平变化表明有数据输入  因此MACFrame不需要帧界定符 |

差错控制

/\*

建立链路

用户拨号介入ISP

转发器：

适配器地址/接口地址/MAC地址/物理地址/硬件地址/EUI-48：

组织唯一标识符OUI：高24bit，由IEEE派发

扩展唯一标识符EUI：低24bit，适配器生产商设定

I/G(bit24)：I/G=0表示单站地址，I/G=1表示组地址(多播)

G/L(bit25)： G/L=1表示本地管理，G/L=0表示全球管理

地址总数：2^46，超过70万亿，目前是够用的

单播帧：

广播帧：

多播帧：

/\*

路由器收到一个带转发的数据报后，从路由表中得出下一跳路由器的IP地址，但并不把这个地址填入IP数据报，而是送交链路层的网络接口软件

网络接口软件把下一跳路由器的IP地址转换成MAC地址(ARP)，将此硬件地址放在链路层的MAC帧首部

网络层形成的IP地址，是数据传输的最终地址

实际传输过程中，数据可能不会直接传送到该最终地址，而是经过了多次路由器的跳转

使用MAC地址，记录下一个跳转地址： 源地址→MAC地址1→MAC地址2→MAC地址3→MAC地址4(IP地址)

\*/

MACAddr Create\_DA(IPAddr A,IPAddr B) // A→B

{

//由目标IP地址，得到目标网络地址D

//查看本路由器的路由表中的各个网络地址

//路由表中有该网络地址D，则直接交付该IP地址

//路由表中有该网络地址D的特定路由，交付给下一跳路由器

//路由表中有该网络地址D的路由，交付给下一跳路由器

//路由表中有该网络地址D的默认路由，交付给下一跳路由器

//出错

/\*

A发送IP报文到本局域网中的某台主机B

查看A的ARP分组，有无B的IP地址，

如果有，查看该IP地址对应的MAC地址，把该MAC地址装入MAC帧，然后发送

如果没有，原因可能是B刚入网，其IP地址还没存入A的ARP缓存，也可能是A刚上电，它的ARP缓存是空的

那么A运行ARP进程，寻找B的IP地址：

A在本局域网广播发送一个ARP请求分组信号，信号包含A的IP地址和MAC地址 ，以及B的IP地址

所有该局域网的主机收到信号后，都会比对IP地址

B的IP地址与该信号中的IP地址一致，于是发送它的MAC地址给A ，B在收到A的ARP请求时，也会将A的IP地址与MAC地址写入自己的ARP缓存，以后就就可以直接通信了

A收到B的ARP响应，将其MAC地址写入ARP缓存

A没有收到B的ARP响应，说明B不再本局域网中

A查看 ARP分组 找到本局域网的路由器 ，将路由器的MAC地址封装到MAC帧，接下来的工作交给该路由器完成

每个主机的ARP缓存保存ARP分组

分组中是其所在局域网的各个主机的IP地址以及对应的MAC地址

路由器收到报文后，去除MAC帧的首尾，即MAC源地址和MAC目的地址，查看该报文的目的IP地址

如果在本局域网中，就发送到该主机

如果不在本局域网，则根据该IP地址发送到本局域网的另一个路由器

特定路由：对特定的目的主机指明一个路由器

默认路由： 适用于小网络，网络中只有一个路由器的情况，

}

## PPP链路

|  |  |  |
| --- | --- | --- |
| PPP协议 | 点对点：一对一  全双工  广域网  使用串行线路通信  面向字节  用于：通过拨号或专线方式建立点对点连接发送数据，  同步线路、异步线路、Modem链路、租用的路由器到路由器 | |
| 封装成帧 | 帧定界 | 发送方的链路层，对来自IP层的报文添加首部和尾部，封装成帧；以帧为单位进行传输，出错时只需重发出错的帧，而不必重发所有数据。 |
| 帧同步 | 接收方的链路层，对收到的二进制比特流，区分帧的起始和终止 |
| MTU | 1500B |
| 透明传输 | 字节填充 | 异步传输：逐个字符传输，转义符ESC=0x7D  发送方发送0x7E时，前面插入0x7D  接收方收到0x7D时， |
| 零比特填充 | SONET/SDH链路，同步传输：一连串比特连续传输  硬件完成  发送方发送5个连续的1时，后面插入一个0  接受方接收5个连续的1时，自动删除后面的一个0 |
| 差错检测 | 接收端对收到的帧进行检测，立即丢弃错误帧  PPP提供差错检测但不提供纠错，只保证无差错接收，通过硬件进行CRC检验  不是可靠的传输协议，因此不使用序号确认机制  PPP两端可以运行不同的网络层协议，但仍然使用同一个PPP进行通信 | |

|  |  |
| --- | --- |
| 链路控制协议  LCP | 建立、配置、测试、管理数据链路  PPP支持多种链路，串行并行、同步异步、低俗告诉、电、光、交换、非交换  当用户拨号接入ISP后，就建立了一条从用户个人电脑到ISP的物理连接，这时，用户个人电脑想ISP发送一系列ISP分组(PPP帧)，来建立LCP连接，这些分组和响应选择了将要使用的一些PPP参数 |
| 网络控制协议  NCP | 为网络层协议建立和配置逻辑连接  PPP允许同时采用多种网络层协议，每个不同的网络层协议要用一个相应的NCP来配置  NCP给新接入的用户个人电脑分配一个临时的IP地址，用户通信完毕时，NCP释放网络层连接，收回之前分配给用户主机的IP复制  接着：LCP释放数据链路层连接  最后：释放物理层连接 |

PPP协议点对点，并不是总线型，所以不采用CSMA/CD协议，没有最短帧，所以信息占用0~1500B，而不是46~1500B，

当数据部分出现和标志位一样的比特组合时，需要采用一些措施实现透明传输

|  |  |
| --- | --- |
| 初始状态 | 链路静止，不存在物理层连接 |
|  | 检测到有载波信号时，建立物理连接，线路变为建立状态 |
|  | LCP开始选项商定，商定成功后进入身份验证状态，鉴别 |
| 鉴别状态 | 只允许传送LCP分组、鉴别协议分组、检测链路质量的分组  若使用口令鉴别协议PAP，则需要发起通信的一方发送身份标识符合口令，允许重试多次  若使用口令握手鉴别协议CHAP，具有更好的安全性  若鉴别失败，直接进入链路终止状态，若鉴别成功，进入网络状态 |
| 网络状态 | 采用NCP配置网络层，  PPP链路两端的网络控制协议NCP，根据网络层的不同协议互相交换网络层特定的网络控制分组，因为路由器能够同时支持多种网络层协议  即PPP两端可以运行不同的网络层协议，但仍然可以使用同一个PPP协议通信  如果是IP协议，则对PPP链路的一端配置IP协议模块时，就要使用NCP中支持IP的协议，IPCP  IPCP分组封装成PPP帧，在PPP链路上传送  配置成功后，进入打开状态 |
| 链路打开  状态 | 进行数据传输  两端都可以发送 LCP回送请求分组和LCP回送回答分组，以检查链路状态  数据传输完成后，一端发送LCP终止请求分组，另一端收到后回复 终止确认分组，进入链路终止状态  如果链路出现故障，也会进入链路终止状态 |
| 链路终止  状态 | 调制解调器的载波停止后回到静止状态 |
|  |  |

## 以太网

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 介绍 | 局域网，基带总线局域网  以争用方式接入到共享信道，经过指数退避指示的时间，重发  无连接：不必建立连接，直接发送数据；  不可靠交付：目的站收到错误帧直接丢弃，适配器对发送的数据帧不进行编号，也不要求对方发挥确认，工作简单，局域网信道质量好  传输速率：1km/5us | | | |
| 发送 | 准备发送 | 接收来自网络层的分组，加上以太网的首部和尾部，放入适配器的缓存 | | |
| 检测信道 | 占用：持续检测，直到信道空闲，进入下一步  空闲：等待帧间最小间隔，信道仍为空闲，进入下一步  帧间最小间隔=9.6us，使刚刚收到数据帧的站的接收缓存来得及清理，做好接收下一帧的准备 | | |
| 发送数据 | 保留发送 | 每发送完一帧，暂时保留，如果检测到碰撞，则要重传 | |
| 最大传输时延 | 以太网作为局域网，覆盖范围远小于5km，再考虑转发器时延等因素取=25.6us | |
| 碰撞冒险期 | 发送端发送第一个bit，到，接收端收到该bit，持续时间=≤  期间存在遭遇碰撞的可能性：在此期间接收端未收到任何信息，认为信道空闲而发送数据，于是发生碰撞 | |
| 碰撞窗口  (争用期) | 争用期：发送端发送第一个bit，到检测到碰撞的时间最大值  取接收端在最远的距离处，当第一个bit即将到达接收端时，接收端发送数据，发生碰撞，碰撞信号再返回到发送端，得争用期=2=51.2us=512bit时间 | |
| 在此期间发送端边发送边检测(信道上电压变化情况)：信号电压变化幅度因叠加而增大，超过门限值时认为发生碰撞(多个站同时发送数据) | |
| 最小帧长 | 考虑发送一个很短的帧，发送完毕后经过一段时间收到一个碰撞信号，那么发送端无法将这个碰撞信号与相应的帧对应起来，因而也不会重传对应的那个帧，因此：  设置最小帧长：保证发送端在发送数据的过程中收到碰撞信号，  最小帧长=10Mbit/s\*争用期=512bit=64B，如果发送的数据很少，则加入填充字节 | |
| 检测碰撞 | 无  碰  撞 | 发送数据后，经过争用期的时间还没有检测到碰撞，就可以确定本次本次发送不会发生碰撞，即如果发生碰撞，一定能在争用期间内检测到，即发送前512bit的时间内，本数据发送完毕后，回到“准备发送” |
| 有碰撞 | 立即停止发送数据：那么发生冲突时，已经发送的数据肯定都小于64B |
| 发送人为干扰信号(32bit或48bit)：以便让所有的用户都知道发生了碰撞 |
| 截断二进制指数退避：用来确定碰撞后重传的时机  如果发生碰撞的站停止发送后等待信道空闲又立即同时重传，必然再次发生碰撞  因此发送端停止发送后，推迟**重传退避时间**，然后回到“检测信道”  取：k=min[10,重传次数]，r=random[0,1,…-1]，重传退避时间=r\*争用期  初始：k=1，r=0或1，退避时间=0或  网络拥塞：重传达16次扔不成功，丢弃此帧，报错  总线占用时间：发送数据到检测到碰撞并停止发送+人为干扰信号发送时间+ |
| 接收 | 因此接收方收到的凡是小于64B的帧都是由于冲突而异常终止的无效帧，只要收到就立即丢弃 | | | |
| 信道  利用率 | 总线占用时间=+  a=/  理想情况：没有碰撞，且充分利用总线资源(总线始终被占用)，  S=/(+)=1/(a+1)  a越小，信道利用率越高 | | | |
| 逻辑拓扑 | 总线型结构：许多计算机多点接入，连接在一根总线上，同一时间只能允许一台计算机发送数据(半双工通信) | | | |
| 物理拓扑 | 集线器  星形拓扑 |  | | |
|  | 星形结构、拓展星形结构 | | |
| 传输介质 | 粗缆、细缆、双绞线、光纤 | | | |

|  |  |
| --- | --- |
| 网络适配器  (网卡) | 连接计算机和传输介质  帧的发送、接收、封装、拆封，介质访问控制  数据编码、解码、缓存 |
| MAC地址  (硬件地址)  (物理地址) | 全世界每块网卡对应一个MAC地址(48bit)  只知道dst IP并不能发送数据给它对应的主机，必须知道目的端的硬件地址才能发送数据 ，数据传输需要MAC地址 |
| 介质访问  控制 | CSMA/CD  争用期=51.2us，对于10Mb/s的以太网，最小帧长=512bit=64B |
| 曼彻斯特  编码 | 高电平信号=高电平+低电平  低电平信号=低电平+高电平  缺点：所占频带宽度比原始的基带信号增加了一倍 |

## ARP

|  |  |  |  |
| --- | --- | --- | --- |
|  | 表项 | IP |  |
|  | MAC |  |
|  | TTL | 生存时间，超过生存时间，将从分组中删除 |
|  |  |  |  |

由于不知道目标设备在哪里，ARP请求使用广播发送

广播MAC：ff ff ff ff ff ff

ARP应答使用单播

ARP高速缓存：

ARP高效运行的关键，存放了最近IP地址到MAC地址的映射记录。高速缓存中每一项的一般为20分钟，起始时间

生存时间：从表项被创建时开始算起，一般，完整表项20分钟，不完整表项(上对一个不存在的主机发出ARP请求)3分钟

发送IP数据报时，查看ARP缓存(ARP表)，

根据该IP地址查看是否有匹配的MAC地址

如果没有则发送ARP请求

收到应答后，将新的匹配表项写入ARP表，再发送IP报文

如果没有收到应答，说明目的主机不再本网络，则将该IP报文发送给本局域网的路由器，接下来交给路由器

\*/

## 帧发送

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 帧编号 | |  | | | | |
| 发送帧  缓冲区 | | 发送完数据后保留该数据在缓冲区，如果发生超时就可以执行重传  知道收到确认才可以从缓冲区中移除 | | | | |
| 发送窗口  WT | | 一组可以连续发送的帧的序号  发送窗口大小：表示可以在不接受确认信息的情况下最多还可以发送的帧的个数，  WT≤窗口总数-1  窗口大小在传输过程中是固定的，这和TCP中的滑动窗口不同 | | | | |
| 信道  利用率 | | 发送速率P=单位时间内发送方可以发送的数据量  发送周期T：发送方发送某一帧数据，到收到该帧数据的确认帧  发送时延T1：在一个发送周期中，发送方持续发送的时间，受发送窗口的影响，发送方往往不能持续不断地发送数据，  传输时延R：单向传播延迟  T=(确认帧长度/P)+2R，默认确认帧与数据帧长度相同  信道利用率=T1/T  传输速率(吞吐率)Tp=P\*T2/T | | | | |
| 发送  方法 | 停止  等待 | | 发送窗口大小=1，接收窗口大小=1  发送方：每发送一帧，都要等待收到接收方的应答信号，之后才能发送下一帧 | | | |
| 退N帧  GBN | | 发送窗口大小>1，接收窗口大小=1  不需要收到上一帧的确认才能发送下一帧，而是可以连续发送  若发送n帧后，该n帧的前一个帧发生超时，那么接下来重发这n+1个帧，即接收方只能顺序接收 | | | |
| 选择  重传  SR | | 发送窗口大小>1，接收窗口大小>1  为了保证接收方，前移窗口后，新窗口的序号和旧窗口的序号没有重叠，要求满足  接收窗口大小+接收窗口大小≤  接收窗口大小≤  不然接收方无法确定收到的帧是重传的帧还是新发的帧  收到NAK时，对NAK指定的帧进行重传 | | | |
| 差错  控制 | 差错  来源 | | 热噪声 | 信道固有，随机差错  措施：提高信噪比 | | |
| 冲击  噪声 | 外界电磁干扰，突发差错，是传输差错的主要原因  无法通过提高信噪比避免 | | |
| 差错  控制 | | 自动重传  请求  ARQ | 接收方检测到差错时，通知发送方重发，知道接收到正确的数据为止 | | |
| 前向纠错  FEC | 接收方发现差错，并确定二进制数码的错误位置 | | |
| 检错  编码 | 奇偶  校验 |  |
| CRC |  |
| 纠错  编码 |  | |
|  |  | |

|  |  |  |
| --- | --- | --- |
| 超时重传 | 为每个发送的帧设置一个计数器，当计时器超时，该帧就会重传  发送方每发送一帧，启动定时器，  有三种引起超时重传的情况：  1.数据帧丢失  2.数据帧出错  3.确认帧丢失 | |
| 自动重传  请求ARQ | 停止等待 |  |
| 退N帧 |  |
| 选择重传 |  |

## 帧接受

|  |  |
| --- | --- |
| 接收帧  缓冲区 |  |

|  |  |  |
| --- | --- | --- |
| 确认  机制 | 确认帧 | 不含数据，控制帧 |
| 逐个确认 | 每接收一帧，都要回复一个应答信号 |
| 累积确认 | 收到多个帧后，一起回复确认  发送的确认帧接收窗口大小≤发送窗口大小，一般就相等  确认对应的数据帧，以及前面的数据帧都正确接收 |
| 捎带确认 |  |
|  | 一旦发现帧出错，发送一个否定帧NAK，要求发送方对NAK中指定的帧进行重传 |
| 选择重传 | 逐个确认正确接收的分组，不管接收到的分组是否有序，只要正确接收就发送ACK分组进行确认，ACK分组不具有累积确认的功能，即是一对一的确认 |
| 错误  处理 |  | 差错检测错误后，简单地讲该帧丢弃 |
| 乱序  处理 |  | 当接收窗口=1时，可以保证帧的有序接收 |
|  | 若收到乱序的帧，则直接丢弃  然后重复发送已经发送过的最后一个确认帧，这么做是为了防止该确认帧丢失引起的差错 |
|  | 为提高信道的利用率，设法让发送方只重传发生超时的数据帧，则必须增加接收窗口 |

## 介质访问控制

多个信号在一个物理信道上传输：

信道划分：多路复用技术

随机访问：争用型，每个主机根据自己的意愿发送信息，为防止碰撞引起的丢失，主机要按一定的规则重复发送直到发送成功，胜利者通过争用获得信道，获得信息的发送权，既不共享时间，也不共享空间

多路复用

|  |  |
| --- | --- |
| FDM | 频分多路复用，共享时间(同时传输)，分割带宽，  多路信号调制到不同频率载波上，叠加成复合信号；  带宽分配给各个信号  保护频带：插入在在相邻频率的信号之间，防止信号之间的干扰  优点：充分利用传输介质的带宽，系统效率高，技术成熟  用于：模拟信号 |
| TDM | 时分多路复用，共享带宽，分割时间(分时传输)  传输时间等分成时隙，多个信号轮流占用时隙，使用信道，每个信号占用固定序号的时隙  不会发生冲突  介质位速率大于单个信号的位速率  用于：数字信号 |
| WDM | 波分多路复用  对光的频分多路复用，在一根光纤中传输多种波长的光信号，由于波长不同，所以各路光信号互不干扰，最后再用波长分解复用器将各路波长分解出来 |
| CDM | 码分多路复用，共享时间(同时传输)，共享带宽，  多路信号混合传输，到达目的地后再分开  任意一个主机配置一个唯一的标签序列，任意两个主机的标签序列正交，如主机A：-1,-1,-1,+1,+1,-1,+1,+1；主机B：-1,-1,+1,-1,+1,+1,+1,-1  发送：1→发送该主机的标签序列；0→发送该主机的标签序列的反码  某一时刻：A向C发送1，B向C发送0，  信道发送：序列线性相加→0,0,-2,2,0,-2,0,2  信道接收：A：(0,0,-2,2,0,-2,0,2)·(-1,-1,-1,+1,+1,-1,+1,+1)/8=1  B：(0,0,-2,2,0,-2,0,2)·(-1,-1,+1,-1,+1,+1,+1,-1)/8=-1→0 |
| 纯  ALOHA | 不进行任何检测直接发送，如果一段时间内没有收到确认，认为发生冲突，  等待一段随机时间后重发，以避免再次碰撞，直至成功 |
| 间隙  ALOHA | 等间隔发送  数据到达后，进入发送缓冲区，等到发送间隔来临  若间隔来临时，缓冲区有多个数据，则会产生冲突，若发生冲突，等待一段随机时间后重发，以避免再次碰撞，直至成功 |

|  |  |  |
| --- | --- | --- |
| CSMA | 载波侦听装置： | |
| 1  persistent | 要发送数据时，先侦听信道，如果空闲直接发送，如果忙则等待并继续侦听  延迟问题： |
| Non  persistent | 要发送数据时，先侦听信道，如果空闲直接发送，如果忙则延迟一个随机时间后重发 |
| P  persistent | 用于时分信道  要发送数据时，先侦听信道，如果忙则等待下一个时间间隔到来后再侦听  空闲以概率P直接发送，概率1-P不发送，推迟到下一个时间间隔，重发 |
| CSMA  /CA | 确认机制：对正确接收的数据帧进行确认 | |

|  |  |
| --- | --- |
| 轮询访问  令牌传递 | 用户不能随机发送信息，而是通过一个集中控制的监控站，以循环的方式轮询每个结点，再决定信道的分配。  某结点使用信道时，其他结点都不能使用信道  用于令牌环局域网中  一个令牌再各个结点之间以某个固定的次序交换，令牌是由一组特殊的比特组合而成的帧，  当环上的一个站要发送数据时，等待令牌，一旦受到令牌，就发送；  帧中包含目的站地址，  主机收到某一帧时，不管是不是发给自己的，都进行转发，直到该帧回到发送方，并由发送方终止其传送  目的主机受到帧后，保存一个副本，然后再转发，对转发的帧的尾部设置响应比特，表示已经接收  主机发送完一帧后，释放令牌  由于令牌只有一个，因此不会发生冲突  令牌再设备之间传递的通路是一个环  不共享时间，不共享空间  适用：负载高，广播信道 |

## 令牌环

## FDDI

## HDLC

|  |  |  |
| --- | --- | --- |
|  | 广域网  面向比特  HDLC协议不依赖于任何一种字符编码集，  数据报文可以透明传输，用于实现透明传输的“0比特插入法”易于硬件实现，  全双工通信  所有帧采用CRC校验，对信息帧进行顺序编号，可防止漏收或重发，可靠性高  传输控制功能与处理功能分离，灵活性大。  适用于：  1.非平衡控制：一个主站控制整个链路  2.平衡控制：链路两端的两个站都是复合站，每个复合站都可以平等地发起数据传输，而不需要得到对方复合站的允许。 | |
|  | 主站 | 负责控制链路的操作，主站发出的帧称为命令帧 |
| 从站 | 受控于主站，按主站的命令进行操作，发出响应帧 |
| 复合站 | 兼具主站与从站的功能 |
| 数据操作  方式 | 正常  响应 | 非平衡结构操作  主站向从站传输数据，从站进行响应传输，从站只有收到主站的许可后，才可以进行响应 |
| 异步  平衡 | 平衡结构操作  每一个复合站都可以进行对另一站的数据传输 |
| 异步  响应 | 非平衡结构操作  从站在没有接到主站的允许下就可以进行传输 |

使用序号确认机制

## 网桥

多个以太网通过网桥连接起来，成为一个更大的以太网，原来的以太网可以称为这个大以太网的一个网段，

设网络1和网络2通过网桥连接

网桥接收网络1的数据帧，检查数据帧中的地址，

如果是网络2的地址，则发送给网络2，

如果是网络1的地址，则丢弃，即发送给同一个网络的帧，不需要经过网桥转发

3个网段使用两个网桥，组成新的局域网，设每个网段吞吐率是10Mb/s，那么，整个局域网的吞吐率是30Mb/s，因为各个网段内部传输，相互之间不会发生冲突

1.网桥具备寻址和路径选择能力，以确定帧的传输方向

2.从源网络接收帧，以目的网络的介质访问控制协议向目的网络转发该帧

3.网桥可以连接不同的LAN，即接收某LAN(协议1)的帧，发送给另一个LAN(协议2)

4.存储并转发，有足够大的缓存空间

优点：

过滤通信量

扩大物理范围

可使用不同的物理层

互联不同类型的局域网

提高可靠性

改善性能

缺点：

增加时延

没有流量控制功能

互联不同局域网时需要转换帧的格式

适用：用户数不多，同信量不打的局域网，

|  |  |
| --- | --- |
| 透明网桥 |  |
| 源路由网桥 |  |

## 交换机

检测从以太网端口来的数据帧的srcMAC和dstMAC

与系统内部的动态查找表进行比较，若数据帧的MAC地址不在查找表中，则将该MAC地址加入查找表，并将数据帧发送给相应的目的端口。

以太网交换机的每个端口都直接与单个主机相连

全双工

能同时联通多对端口，使每一对相互通信的主机都能够像独占通信媒体那样，无碰撞地传输数据

即插即用，其内部帧的转发通过自学习算法自动逐渐建立起来

使用专用的交换结构芯片，交换速率高

独占传输媒体的带宽

对于10Mb/s的共享式以太网，共N个用户，每个用户占有带看10/N

若使用交换机，则每个用户还是拥有10Mb/s，交换机总容量N×10

|  |  |  |
| --- | --- | --- |
| 交换模式 | 直通式 | 只检查dstMAC，帧接收后可以马上发送出去，  速度快，但缺乏智能性和安全性，无法支持不同速率端口的交换 |
| 存储转发式 | 现将接收到的帧存储到告诉缓存器，检查数据是否正确，确认无误后通过查找表换成输出端口，发送该帧，  如果有错误则丢弃该帧  可靠性好但延迟大 |

# 网络层

## 拥塞控制

轻度拥塞：随着网络负载的增加，网络吞吐量明显小于正常的吞吐量

拥塞：随着网络负载的增加，网络吞吐量反而下降

死锁：网络吞吐量为零

注意流量控制与拥塞控制的区别：

流量控制：点对点，发送端和接收端之间的通信量控制，抑制发送端的发送速率，一般接收端来得及接收

拥塞控制：全局控制，涉及网络中所有主机，路由器等所有因素

方法1：开环控制

静态预防：事先考虑到有关发生拥塞的因素，力求在工作时避免不发生拥塞

确定何时可接受新流量、何时丢弃分组、丢弃哪些分组、确定调度决策等

在做决定时不考虑当前网络状态

方法2：闭环控制

事先不考虑有关发生有色的各种因素，不断监测网络，发生拥塞时，动态调整

## IP地址

|  |  |  |  |
| --- | --- | --- | --- |
| 主机 | 每一台主机的每一个接口，对应一个全世界唯一的IP地址(32bit)  IP地址=网络前缀+主机号=000.000.000.000~255.255.255.255  网络号：全0→本网络，全1→环回测试  主机号：全0:本主机，全1→该网络所有主机  具有本地磁盘的主机，从磁盘上的配置文件中读取其IP地址,无盘机则需要采用其他方法 | | |
| 局域网  LAN |  |  | |
| 网络号 | 地址  掩码 | IP地址&掩码=网络前缀→地址块，高位部分(11…1)网络前缀，低位部分(00…0)主机号  如果随IP地址还给出了子网掩码，那么该IP地址的子网编址就是确定的：  子网掩码是一个32bit的值，其中值为1的比特留给网络号和子网号，值为为0的比特留给主机号 |
| CIDR  记法 | IP地址/前缀位数，如128.14.35.7/20，CIDER技术把小的网络汇聚成大的超网  地址块：相同网络前缀的IP地址组成的集合  使用CIDER表示时，该局域网又称为地址块 |
| 类型 | A类 | 0+7bit网络号+24bit主机号，0.0.0.0~~ 127.255.255.255，其中10.0.0.0~10.255.255.255供内部使用，不允许出现在因特网上  127.xx.yy.zz 作为保留地址，用于回路测试 |
| B类 | 10+14bit网络号+16bit主机号，128.0.0.0~~ 191.255.255.255，其中172.16.0.0~172.31.255.255供内部使用，不允许出现在因特网上 |
| C类 | 110+21bit网络号+8bit主机号，192.0.0.0~~ 223.255.255.255，其中192.168.0.0~192.168.255.255供内部使用，不允许出现在因特网上 |
| D类 | 1110+多播组号：224.0.0.0~239.255.255.255 |
| E类 | 1111+保留，240.0.0.0~~ 247.255.255.255 |
| 自治  系统  AS |  | 同一个自治系统内使用相同的路由选择协议，不同的自治系统间可能使用不同的路由选择协议  内部网关协议IGP：统一个自治系统内路由器之间的传输使用  外部网关协议EGP：不同自治系统 | |
|  | 划分区域：泛洪法交换链路状态信息的范围局限在每一个区域内，而不是整个自治系统  一个区域内部的路由器只知道本区域的网络拓扑，而不知道其他区域的网络拓扑的情况  采用分层次划分区域的方法虽然使交换信息的种类增多了，同时也使OSPF协议更加复杂了。但这样做却能使每一个区域内部交换路由信息的通信量大大减少，因而使OSPF能用于规模大的自治系统中。 | |
|  |  | |
| 传输 | 直接  交付 | 主机发送报文：比较dstIP与srcIP，查看目的主机与本主机是否在同一网络，若在同一网络，即标主机与源主机直接相连（如点对点链路），则直接交付，无需经过路由器，否则执行，下一步 | |
| 间接  交付 |  | |

：

1.2.主机将报文发送给本网络的路由器：主机通过ARP协议获得本网络的路由器的MAC地址，发送数据报给该路由器

目标主机与源主机都在一个共享网络上（如以太网或令牌环网）：报文直接送到目标主机上

目标主机与源主机不在同一个网络上：源主机先把数据发给 管理目标主机所在的网络的路由器，再由该路由器将数据发送给目标主机

主机发送自己产生的报文，路由器执行分组转发

|  |  |
| --- | --- |
| 最大地址块 | 整个互联网，包含所有IP地址：000.000.000.000~255.255.255.255 |
| 地址块申请 | 向ICANN申请地址块即得到一个网络前缀，申请一个20bit网络前缀的地址块就拥有4094个IP地址 |
| 地址块划分 | 主机号划出一部分高位到网络前缀中  地址块划分是地址块内部事宜，其对外仍表现为一个网络  52.196.0.0/14→日本亚马逊公司数据中心：52.196.0.0/14~52.199.255.255/14  202.117.0.0/18→西安交通大学：202.117.0.0/18~202.117.63.255/18  分配网络前缀时，应先分配地址数较多的前缀  每个路由器接口也要占用一个IP， |
| 地址块包含 | A包含B，如果某IP是在B中，那么必然也在A中，随该IP一起给出的掩码， 可以是指示地址块A的掩码，也可以是指示B的掩码，显然指示B的掩码更加详细 |
| 地址块聚合 | 若干个地址块，找到包含它们的大地址块 |
| 子网编址 | 主机号再分成一个子网号(高位)和一个主机号(低位)  这样做的原因是因为A类和B类地址为主机号分配了太多的空间，  可分别容纳的主机数为224-2和216-2(全0或全1的主机号无效，因此总数减去2)，实际并不需要这么多；  在InterNIC获得某类IP网络号后，就由当地的系统管理员来进行分配，由他来决定是否建立子网，以及分配多少比特给子网号和主机号。 |
| 子网掩码 |  |

在像Internet这样的系统中，目前采用了许多不同的选路协议。Internet是以一组自治系统(AS，AutonomousSystem)的方式组织的，每个自治系统通常由单个实体管理。

常常将一个公司或大学校园定义为一个自治系统。NSFNET的Internet骨干网形成一个自治系统，这是因为骨干网中的所有路由器都在单个的管理控制之下。

每个自治系统可以选择该自治系统中各个路由器之间的选路协议-内部网关协议IGP，最常用的IGP是选路信息协议RIP

一种新的IGP是开放最短路径优先OSPF，它意在取代RIP。

一种较早的IGP协议—HELLO，现在已经不用了。

实现任何动态选路协议的路由器必须同时支持OSPF和RIP，还可以支持其他IGP协议。

外部网关协议EGP用于不同自治系统之间的路由器。在历史上，（令人容易混淆）改进的EGP有着一个与它名称相同的协议：EGP。

新EGP是当前在NSFNET骨干网和一些连接到骨干网的区域性网络上使用的是边界网关协议BGP，BGP意在取代EGP。

Unix常常运行名为routed路由守护程序。几乎在所有的TCP/IP实现中都提供该程序。该程序只使用RIP进行通信，这是一种用于小型到中型网络中的协议。

另一个程序是gated。IGP和EGP都支持它。

大多数运行路由守护程序的系统都可以运行routed，除非它们需要支持gated所支持的其他协议。

## 路由策略(路由选择)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 路由表 | 每个路由器维护一个路由表  路由表根据路由策略(路由选择算法)得出 | | | | |
| 表项  (路由) | dstNet | 目标主机所在的网络的网络号  主机号非0→某一特定的主机，主机号全0→某网络中的所有主机  路由器仅根据目的主机所连接的网络号来转发分组，而不考虑主机号，从而减少了路由表所占的存储空间 | | | |
| metric | 单位:hop，相邻的跳数为1，最大值15；  RIP只能用在主机间最大跳数值为15的AS内。  度量为16表示到无路由到达该IP地址 | | | |
| next | direct | | | dstNet与本路由器直接相连  将报文转发给IP=dstIP的主机 |
| IP地址 | | | dstNet与本路由器不直接相连  将报文转发给IP=next的路由器 |
| flags | U | U=1本表项有效； U=0本表项无效 | | |
| G | G=1：间接路由，nextHost→路由器的IP  G=0：直接路由，nextHost→某直接相连的主机的IP | | |
| H | H=1：主机路由，dstIP→主机地址，用于主机地址匹配  H=0：网络路由，dstIP→网络地址，用于网络地址匹配 | | |
| D | D=1该路由由重定向报文创建 | | |
| M | M=1该路由已被重定向报文修改 | | |
| refcnt | 使用该路由的TCP连接的个数 | | | |
| use | 通过该路由传输的报文计数 | | | |
| interface | 本地接口的名字 | | | |
| 路由  类型 | 环回路由 | UH | | 主机与自己本身通信，使用特殊的接口-环回接口 | |
| 默认路由 |  | | 目标IP和子网掩码都是0.0.0.0  每个主机都有一个或多个默认路由，如果路由选择失败，则使用默认路由  大多数主机和一些路由器可以用默认路由来处理任何目的，除非它在本地局域网上 | |
| 局域网  路由 |  | | 主机连在一个局域网上，只能访问局域网上的主机 | |

路由选择协议：获取网络拓扑信息，构建路由表，更新路由表，选择到达每个目的网络的最优路径，识别一个网络的无环通路

收敛：当路由环境发生变化后，各路由器调整自己的路由表以适应变化，最终所有路由器的路由表与网络拓扑状态保持一致

收敛越快，表示路由器越快适应网络变化

结构应当使查找过程最优化，对网络拓扑变化的计算最优化

## 路由机制(分组转发)

路由器

一个路由器管理一个子网：

子网中的设备(包括该路由器)子网号相同，但主机号各不相同

某主机要发送一条报文，不管是发送到其他子网还是本子网的设备，都得先传输给该子网的路由器

外部网络想要访问某主机，该主机由路由器X管理，那么只需要知道通往X的路径就可以了，接下来的工作交给X

路由器是特殊的主机

路由器有多个接口，每个接口有一个IP地址，每个接口连接一个网络

与一般主机的区别在于主机发送的IP报文是自己生产的，路由器发送的IP报文是转发的

检测到拥塞时丢弃分组

1.比较dstIP与srcIP，查看目的主机与本主机是否在同一网络，若在同一网络，即标主机与源主机直接相连（如点对点链路），则直接交付，无需经过路由器，否则执行间接交付，下一步

2.主机将报文发送给本网络的路由器：主机通过ARP协议获得本网络的路由器的MAC地址，发送数据报给该路由器

3.

目标主机与源主机都在一个共享网络上（如以太网或令牌环网）：报文直接送到目标主机上

目标主机与源主机不在同一个网络上：源主机先把数据发给 管理目标主机所在的网络的路由器，再由该路由器将数据发送给目标主机

主机发送自己产生的报文，路由器执行分组转发

|  |  |
| --- | --- |
| 路由器接收报文，执行路由机制  获取报文中的dstIP(设为IPx)，计算下一跳路由器或主机的IP，从合适的端口发送出去 | |
| 主机  地址  匹配 | 数据发给某特定主机  遍历路由表，routet[i].dstIP=IPx & route[i].H=1  匹配成功：返回routet[i].nextHost，  routet[i].nextHost就是目的主机→直接交付  routet[i].nextHost是路由器→间接交付  匹配失败：进入下一步  最长地址匹配：由IPx查找路由表得到多个匹配的路由，则选择具有最长网络前缀的路由 |
| 网络  地址  匹配 | 数据发给该网络所有主机  遍历路由表  routetable[i].dstIP\*地址掩码==IPx\*地址掩码 && routetable[i].H==0  routet[i].nextHost就是目的网络→直接交付  routet[i].nextHost是路由器→间接交付 |
| 默认  地址  匹配 | 默认表项一般在路由表中被指定为一个网络表项，其网络号为0 |
| 选路  失败 | 没有匹配项也没有默认项  本地数据：给发送该数据报文的应用程序返回一个差错，“主机不可达”或“网络不可达”。  转发数据：给原始端发送一份ICMP主机不可达的差错报文 |

主机A发送一个IP数据报给主机B

1.A查找自己的路由表，若主机B处于同一个网络，则直接交付，否则交付给与本网络相连的路由器(间接交付)

2.间接交付时，可能面对多个路由器，则要使用路由选择协议

3.路由器收到一个数据报， 查看其路由表，

4.遍历路由表每个表项

5.每个表项的子网掩码和IP地址相与，结果与目的网络比较

\*/

|  |  |
| --- | --- |
| 交换结构 | 存储器 |
| 总线 |
| 互联网络 |

静态路由策略

|  |  |  |
| --- | --- | --- |
| 静态 | 在路由器启动时配置路由表，也可以在启动后由用户随时配置路由表  优点：实现简单，开销小，维护整个网络的拓扑结构信息，  缺点：不能及时适应网络状态的变化，非自适应，不能估计流量和结构来调整路由决策 | |
| 默认方式 |  |
| route命令 |  |
| ICMP重定向 |  |

每当初始化一个接口时（通常是用i f c o n f i g命令设置接口地址），就为接口自动创建一个直接路由。对于点对点链路和环回接口来说，路由是到达主机（例如，设置H标志）。对于广播接口来说，如以太网，路由是到达网络。

到达主机或网络的路由如果不是直接相连的，那么就必须加入路由表

初始化路由表

初始化一个接口：接口创建一个直接路由。

点对点链路和环回接口：H=1

广播接口：如以太网，H=0

到达主机或网络的路由如果不是直接相连的，那么就必须加入路由表。

在配置接口时，以默认方式生成路由表项（对于

直接连接的接口），并通过r o u t e命令增加表项（通常从系统自引导程序文件），或是通过I C M P

重定向生成表项（通常是在默认方式出错的情况下

|  |  |  |
| --- | --- | --- |
| ICMP路由器  发现报文 | 通告报文：可以通告多个地址。  地址数指的是报文中所含的地址数。  地址项大小指的是每个路由器地址32bit字的数目，始终为2。  生存期：是通告地址有效的时间，单位:秒数  一对或多对IP地址：发送路由器的某个地址。  路由器发现报文一般由守护程序用户进程创建和处理 | |
|  | 优先级 | 有符号32bit整数，值越大优先级越高  表示该IP地址作为默认路由器地址的优先等级，这是与子网上的其他路由器相比较而言的。  优先级=8000 0000说明对应的地址不能作为默认路由器地址使用，尽管它也包含中通告报文中。  默认值：0  如果子网上有多台路由器，由系统管理员为每个路由器设置优先等级。例如，主默认路由器就要比备份路由器具有更高的优先级 |
| 启动 | 广播或多播传送接口  主机在引导期间一般发送三份路由器请求报文，每三秒钟发送一次。一旦接收到一个有效的通告报文，就停止发送请求报文。  主机也监听来自相邻路由器的请求报文。这些通告报文可以改变主机的默认路由器。另外，如果没有接收到来自当前默认路由器的通告报文，那么默认路由器会超时。  只要有一般的默认路由器，该路由器就会每隔10分钟发送通告报文，报文的生命周期是30分钟。这说明主机的默认表项是不会超时的，即使错过一份或两份通告报文。 |
| 发送通告报文 | 广播或多播传送接口  不定期发送，以减小与子网上其他路由器发生冲突的概率。一般每两次通告间隔450秒和600秒。一份给定的通告报文默认生命周期是30分钟。  使用生命周期域的另一个时机是当路由器上的某个接口被关闭时。在这种情况下，路由器可以在该接口上发送最后一份通告报文，并把生命周期值设为0 |
| 监听请求报文 | ，并发送路由器通告报文以响应这些请求报文。 |

在一般的体制中，IP在本地生成的数据报并进行发送，或者从一个网络接口接收数据报进行转发。

IP层在内存中有一个路由表。当收到一份数据报并进行发送时，它都要对该表搜索一次。

当数据报来自某个网络接口时，IP首先检查目的IP地址是否为本机的IP地址之一或者IP广播地址。

如果是，数据报就被送到由IP首部协议字段所指定的协议模块进行处理。

如果数据报的目的不是这些地址，那么

（1）如果IP层被设置为路由器的功能，那么就对数据报进行转发（也就是说，像下面对待发出的数据报一样处理）；否则（2）数据报被丢弃。

如果不能传送的数据报来自本机，那么一般会向生成数据报的应用程序返回一个“主机不可达”或“网络不可达”的错误。

完整主机地址匹配在网络号匹配之前执行。只有当它们都失败后才选择默认路由。

当IP数据报应该被发送到另一个路由器时，收到数据报的路由器就要发送ICMP重定向差错报文给IP数据报的发送端。

只有当主机可以选择路由器发送分组的情况下，我们才可能看到ICMP重定向报文（回忆我们在图7-6中看过的例子）。

1)我们假定主机发送一份IP数据报给R1。这种选路决策经常发生，因为R1是该主机的默认路由。

2)R1收到数据报并且检查它的路由表，发现R2是发送该数据报的下一站。

当它把数据报发送给R2时，R1检测到它正在发送的接口与数据报到达接口是相同的（即主机和两个路由器所在的LAN）。

这样就给路由器发送重定向报文给原始发送端提供了线索。

3)R1发送一份ICMP重定向报文给主机，告诉它以后把数据报发送给R2而不是R1。

重定向一般用来让具有很少选路信息的主机逐渐建立更完善的路由表。

路由器启动时路由表中可以只有一个默认表项，一旦默认路由发生差错，对应的默认路由器将通知它进行重定向，并对路由表作相应的改动。

ICMP重定向允许主机在进行选路时不需要具备智能特性，而把所有的智能特性放在路由器端，连在LAN上的所有主机在启动时只需一个默认路由，通过接收重定向报文来逐步学习。

路由器转发分组：存储转发机制

最长前缀匹配：路由表项指示的网络是包含关系

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | | | |
| RIP  距离-向量  路由算法 | 仅相邻路由器间交换各自的路由表信息，按固定时间间隔交换  经过多次更新后，任意一个路由器都包含最新的网络情况  RIP属于内部网关协议IGP，RIP是应用层协议，使用UDP | | | |
| 发送  RIP请求 | 初始化 | | 路由器启动时路由表为空，启动路由守护程序，发送RIP请求报文至相邻路由器 |
| 更新发送 | | 每次更新路由表，发送发生变化的路由至相邻路由器 |
| 发送  RIP应答 | 监听回复 | | 收到RIP请求后，回复RIP应答报文，完整的路由表 |
| 定期发送 | | 每隔30秒，发送完整路由表给相邻路由器 |
| 更新  路由表 | 收到RIP应答报文(来自IP=A的路由器)，根据报文内容更新路由表，对报文中的每个路由，都执行以下操作： | | |
| 匹配  dstNet | 失败：配置新路由metric+1、next=A  成功：匹配nextHost | |
| 匹配  next | 相同：不更新  不同：比较metric | |
| 比较  metric | 等长或原路由更短：不更新  新路由更短：配置新路由metric+1、next=A | |
| 超时  删除 | 每条路由配置一个定时器，溢出值3min，  溢出：设置路由metric=16，60秒后删除  复位：该路由得到更新 | | |
| 构建网络拓扑结构 | 路由器通过交换每个结点到邻居结点的延迟或开销来构建一个完整的网络拓扑结构。得到了完整的拓扑结构后，路由器就使用Dijkstra最短路径算法，来计算到所有结点的最短路径 | | |
| 路由回路  (慢收敛)  问题 | 1.路由器A发生故障，  路由变化{net1,2,direct}→{net1,16,direct}  发送变化报文a至邻接路由器B  2.B发送完整路由表至A，其中包含{net1,2,A}  3.B收到报文更新路由：{net1,16,A}  4.A收到报文更新路由：{net1,16,direct}→{net1,3,B}，发送变化报文  5. | | |

问题

1.RIP没有子网地址的概念。例如，如果标准的B类地址中16bit的主机号不为0，那么RIP无法区分非零部分是一个子网号，或者是一个主机地址。有一些实现中通过接收到的RIP信息，来使用接口的网络掩码，而这有可能出错。

2.

。

|  |  |  |  |
| --- | --- | --- | --- |
|  | RIP | OSPF  链路状态  路由算法 | BGP |
|  | 应用层协议 | 网络层协议 |  |
|  |  | IGP | EGP，不同的自制系统之间交换路由信息 |
|  |  |  | 只交换可达性信息，告诉相邻路由器，  在每个AS中，选择至少一个路由器作为BGP speaker  各AS之间交换信息是通过各自的 speaker 来交换信息的，需要先建立TCP连接，在此连接上建立BGP session，    BGP要交换整个路由表和更新信息，TCP提供可靠交付，因此不需要不断交换信息 |

不是传送整个路由表，而是传送受影响的路由更新报文

每个主机都有完全的网络拓扑信息

主机主动测试所有邻接主机的状态

主机定期将链路状态传播给所有其他主机结点

主机收到链路状态报文后，更新自己的路由表

划分自治系统为多个区域，一个主干区域+若干分支区域

主干路由器

区域边界路由器

AS边界路由器

使用IP报文传送，协议字段=89

问候(Hello)： 保持与邻接路由器的连接

数据库描述(Database Description)：

链路状态请求(Link State Request)：

链路状态更新(Link State Update)：

链路状态确认(Link State Acknowledgement)：

任意相邻路由器每隔10s交换一次问候报文，由此确认邻站是可以到达的

另外4种用来进行链路状态库的同步

一旦某个路由器的链路状态发生变化，该路由器发送链路状态更新报文，用泛洪法向全网发送

向本自治系统中的所有路由器发送信息，泛洪法：本路由器向所有相邻路由器发送信息， 每一个相邻路由器收到后在将此信息发送给其所有相邻路由器

发送的信息内容是是与本路由器相邻的所有路由器的链路状态(和那些路由器相邻，以及链路的)

仅在链路发生变化时，执行发送

OSPF使用IP，因为其使用可靠的泛洪法，IP报文灵活性好，开销小

## 广播与多播(组播)

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

单播(unicast)： 每个以太网帧包含源主机和目的主机的以 太网地址（48bit）。通常每个以太网帧仅发往单个目的主机，目的地址指明单个接收接口

在这种方式下，任意两个主机的通信不会干扰网内其他主机（可能引起争夺共享信道的情况除外）。

\*/

void Broadcast()

{

/\*

广播：一个主机要向某网上的所有其他主机发送帧

1.传输层：UDP

2.应用层：将报文同时传往多个接收者的应用

3.发送主机通过环回机制收到自己发送的广播报文

受限广播地址：255.255.255.255，用于主机配置过程中IP数据报的目的地址，此时，主机可能还不知道它所在网络的网络掩码，甚至连它的IP地址也不知道。

路由器不转发dst IP=受限广播地址的数据报，这样的数据报仅出现在本地网络中。

一个未解的问题是：如果一个主机是多接口的，当一个进程向本网广播地址发送数据报时，为实现广播，是否应该将数据报发送到每个相连的接口上？

如果不是这样，想对主机所有接口广播的应用必须确定主机中支持广播的所有接口，然后向每个接口发送一个数据报复制。

大多数BSD系统将255.255.255.255看作是配置后第一个接口的广播地址，并且不提供向所属具备广播能力的接口传送数据报的功能。

不过，routed（见10.3节）和rwhod（BSDrwho客户的服务器）是向每个接口发送UDP数据报的两个应用程序。

这两个应用程序均用相似的启动过程来确定主机中的所有接口，并了解哪些接口具备广播能力。同时，将对应于那

种接口的指向网络的广播地址作为发往该接口的数据报的目的地址。

HostRequirementsRFC没有进一步涉及多接口主机是否应当向其所有的接口发送受限的广播。

指向网络的广播地址：主机号为全1，如A类网络广播地址=xxx.255.255.255，一个路由器必须转发指向网络的广播，但它也必须有一个不进行转发的选择。

指向子网的广播地址：主机号为全1且有特定子网号，作为子网直接广播地址的IP地址需要了解子网的掩码。

如路由器收到发往128.1.2.255的数据报，当B类网络128.1的子网掩码为255.255.255.0时，该地址就是指向子网的广播地址；

但如果该子网的掩码为255.255.254.0，该地址就不是指向子网的广播地址。

指向所有子网的广播

指向所有子网的广播也需要了解目的网络的子网掩码，以便与指向网络的广播地址区分开。指向所有子网的广播地址的子网号及主机号为全1。

如目的子网掩码为255.255.255.0，那么IP地址128.1.255.255是一个指向所有子网的广播地址。

然而，如果网络没有划分子网，这就是一个指向网络的广播。

当前的看法[Almquist1993]是这种广播是陈旧过时的，更好的方式是使用多播而不是对所有子网的广播。

[Almquist1993]指出RFC922要求将一个指向所有子网的广播传送给所有子网，但当前的路由器没有这么做。

这很幸运，因为一个因错误配置而没有子网掩码的主机会把它的本地广播传送到所有子网。

例如，如果IP地址为128.1.2.3的主机没有设置子网掩码，它的广播地址在正常情况下的默认值是128.1.255.255。

但如果子网掩码被设置为255.255.255.0，那么由错误配置的主机发出的广播将指向所有的子网。

将主机号全1作为广播地址。

广播的例子

广播是怎样传送依赖于广播的类型、应用的类型、TCP/IP实现方法以及有关路由器的配置。

首先，应用程序必须支持广播。如果执行

sun%ping255.255.255.255

/usr/etc/ping:unknownhost255.255.255.255

打算在本地电缆上进行广播。但它无法进行，原因在于该应用程序（ping）中存在一个程序设计上的问题。

大多数应用程序收到点分十进制的IP地址或主机名后，会调用函数inetaddr(3)来把它们转化为32bit的二进制IP地址。

假定要转化的是一个主机名，如果转化失败，该库函数将返回-1来表明存在某种差错（例如是字符而不是数字或串中有小数点）。

但本网广播地址（255.255.255.255）也被当作存在差错而返回-1。

大多数程序均假定接收到的字符串是主机名，然后查找DNS（第14章），失败后输出差错信息如“未知主机”。

如果我们修复ping程序中这个欠缺，结果也并不总是令人满意的。在6个不同系统的测试中，仅有一个像预期的那样产生了一个本网广播数据报。

大多数则在路由表中查找IP地址255.255.255.255，而该地址被用作默认路由器地址，因此向默认路由器单播一个数据报。 最终该数据报被丢弃。

指向子网的广播是我们应该使用的。在6.3节中，我们向测试网络（见扉页前图）中IP地址为140.252.13.63的以太网发送数据报，并接收以太网中所有主机的应答。

与子网广播地址关联的每个接口是用于命令ifconfig（见3.8节）的值。

如果我们ping那个地址，预期的结果是：IP通过目的地址（140.252.13.63）来确定，这是指向子网的广播地址，然后向链路层的广播地址发送该数据报。

在6.3节提到的这种广播类型的接收对象为局域网中包括发送主机在内的所有主机，因此可以看到除了收到网内其他主机的答复外，还收到来自发送主机（sun）的答复。

在这个例子中，我们也显示了执行ping广播地址前后ARP缓存的内容。这可以显示广播与ARP之间的相互作用。执行ping命令前ARP缓存是空的，而执行后是满的（也就是说，对

网内其他每个响应回显请求的主机在ARP缓存中均有一个条目）。我们提到的该以太网数据帧被传送到链路层的广播地址（0xffffffff）是如何发生的呢？由sun主机发送的数据帧不需要ARP。

如果使用tcpdump来观察ping的执行过程，可以看到广播数据帧的接收者在发送它的响应之前，首先产生一个对sun主机的ARP请求，因为它的应答是单播的。

在4.5节我们介绍了一个ARP请求的接收者（该例中是sun）通常在发送ARP应答外，还将请求主机的IP地址和物理地址加入到ARP缓存中去。

这基于这样一个假定：如果请求者向我们发送一个数据报，我们也很可能想向它发回什么。

我们使用的ping程序有些特殊，原因在于它使用的编程接口（在大多数Unix实现中是低级插口(rawsocket)）通常允许向一个广播地址发送数据报。

如果使用不支持广播的应用如TFTP，情况又如何呢？（TFTP将在第15章详细介绍。）

bsdi%tftp启动客户程序

tftp>connect140.252.13.63说明服务器的IP地址

tftp>gettemp.foo试图从服务器或获取一个文件

tftp:sendto:Permissiondenied

tftp>quit终止客户程序

在这个例子中，程序立即产生了一个差错，但不向网络发送任何信息。

产生这一切的原因在于，

插口提供的应用程序接口API只有在进程明确打算进行广播时才允许它向广播地址发送UDP数据报。

这主要是为了防止用户错误地采用了广播地址（正如此例）而应用程序却不打算广播。

在广播UDP数据报之前，使用插口中API的应用程序必须设置SOBROADCAST插口选项。

并非所有系统均强制使用这个限制。某些系统中无需进程进行这个说明就能广播UDP数据报。而某些系统则有更多的限制，需要有超级用户权限的进程才能广播。

下一个问题是是否转发广播数据。有些系统内核和路由器有一选项来控制允许或禁止这一特性（见附录E）。

如果让路由器bsdi能够转发广播数据，然后在主机slip上运行ping程序，就能够观察到由路由器bsdi转发的子网广播数据报。

转发广播数据报意味着路由器接收广播数据，确定该目的地址是对哪个接口的广播，然后用链路层广播向对应的网络转发数据报。

我们观察到它的确正常工作了，同时也看到BSD系统中的ping程序检查重复的数据报序列号。

如果出现重复序列号的数据报就显示DUP!，这意味着一个数据报已经在某处重复了，

然而它正是我们所期望看到的，因为我们正向一个广播地址发送数据。

我们还可以从远离广播所指向的网络上的主机上来进行这个试验。

在主机angogh.cx.berkeley.edu（和我们的网络距离14跳）上运行ping程序，如果路由器sun被设置为能够转发所指向的广播，它还能正常工作。

在这种情况下，这个IP数据报（传送ICMP回显请求）

被路径上的每个路由器像正常的数据报一样转发，它们均不知道传送的实际上是广播数据。接着最后一个路由器netb看到主机号为63，就将其转发给路由器sun。

路由器sun觉察到该目的IP地址事实上是一个相连子网接口上的广播地址，就将该数据报以链路层广播传往相应网络。

广播是一种应该谨慎使用的功能。在许多情况下，IP多播被证明是一个更好的解决办法。

\*/

}

void Multicast()

{

/\*

多播：传送给属于多播组的多个主机。

1.传输层：UDP

2.应用层：将报文同时传往多个接收者的应用

主机组：能够接收发往一个特定多播组地址数据的主机集合，一个主机组可跨越多个网络。主机组中成员可随时加入或离开主机组。主机组中对主机的数量没有限制，同时不属于某一主机组的主机可以向该组发送信息。

永久组(知名组)：注意组成员不是永久的

224.0.0.1：该子网内的所有系统组

224.0.0.2：该子网内的所有路由器组

224.0.1.1：用作网络时间协议NTP，

224.0.0.9：用作RIP-2

224.0.1.2：用作SGI公司的dogfight应用。

IP多播提供两类服务：

1)向多个目的地址传送数据。有许多向多个接收者传送信息的应用：例如交互式会议系统和向多个接收者分发邮件或新闻。

如果不采用多播，目前这些应用大多采用TCP来完成（向每个目的地址传送一个单独的数据复制）。然而，即使使用多播，某些应用可能继续采用TCP来保证它的可靠性。

2)客户对服务器的请求。例如，无盘工作站需要确定启动引导服务器。

目前，这项服务是通过广播来提供的（正如第16章的BOOTP），但是使用多播可降低不提供这项服务主机的负担。

主机对由信道传送过来帧的过滤过程：

1.网卡查看由信道传送过来的帧，确定是否接收该帧，若接收后就将它传往设备驱动程序。通常网卡仅接收那些目的地址为网卡物理地址或广播地址的帧，(如果帧检验和错，网卡将丢弃该帧)

另外，多数接口均被设置为混合模式，这种模式能接收每个帧的一个复制。目前，大多数的网卡经过配置都能接收目的地址为多播地址或某些子网多播地址的帧。

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

2.网卡→设备驱动程序：首先，帧类型中必须指定要使用的协议（IP、ARP等等）。其次，进行多播过滤来检测该主机是否属于多播地址说明的多播组。

3. 设备驱动程序→IP层，IP根据IP地址中的源地址和目的地址进行更多的过滤检测。如果正常，就将数据报传送给下一层

4. IP层→UDP，UDP根据目的端口号，有时还有源端口号进行数据报过滤。

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

如果UDP数据报存在检验和错，将被丢弃。使用广播的问题在于它增加了对广播数据不感兴趣主机的处理负荷。

发送UDP广播数据时，其余主机不得不处理这些广播数据报，一直到UDP层，收到的UDP广播数据报才会被丢弃

多播的出现减少了对应用不感兴趣主机的处理负荷。使用多播，主机可加入一个或多个多播组。

这样，网卡将获悉该主机属于哪个多播组，然后仅接收主机所在多播组的那些多播帧。

多播组地址到以太网地址的转换

IANA拥有一个以太网地址块，即高位24bit为00:00:5e（十六进制表示），这意味着该地址块所拥有的地址范围从00:00:5e:00:00:00到00:00:5e:ff:ff:ff。

IANA将其中的一半分配为多播地址。

为了指明一个多播地址，任何一个以太网地址的首字节必须是01，这意味着与IP多播相对应的以太网地址范围从01:00:5e:00:00:00到01:00:5e:7f:ff:ff。

这里对CSMA/CD或令牌网使用的是Internet标准比特顺序，和在内存中出现的比特顺序一样。这也是大多数程序设计员和系统管理员采用的顺序。

IEEE文档采用了这种比特传输顺序。AssignedNumbersRFC给出了这些表示的差别。

这种地址分配将使以太网多播地址中的23bit与IP多播组号对应起来，通过将多播组号中的低位23bit映射到以太网地址中的低位23bit实现，这个过程如图12-3所示。

由于多播组号中的最高5bit在映射过程中被忽略，因此每个以太网多播地址对应的多播组是不唯一的。32个不同的多播组号被映射为一个以太网地址。

例如，多播地址224.128.64.32（十六进制e0.80.40.20）和224.0.64.32（十六进制e0.00.40.20）都映射为同一以太网地址01:00:5e:00:40:20。

既然地址映射是不唯一的，那么设备驱动程序或IP层（见图12-1）就必须对数据报进行过滤。 因为网卡可能接收到主机不想接收的多播数据帧。

另外，如果网卡不提供足够的多播数据帧过滤功能，设备驱动程序就必须接收所有多播数据帧，然后对它们进行过滤。

局域网网卡趋向两种处理类型：

1.网卡根据对多播地址的散列值实行多播过滤，这意味仍会接收到不想接收的多播数据

2.网卡只接收一些固定数目的多播地址，这意味着当主机想接收超过网卡预先支持多播地址以外的多播地址时，必须将网卡设置为“多播混杂(multicastpromiscuous)”模式。

因此，这两种类型的网卡仍需要设备驱动程序检查收到的帧是否真是主机所需要的。

即使网卡实现了完美的多播过滤（基于48bit的硬件地址），由于从D类IP地址到48bit的硬件地址的映射不是一对一的，过滤过程仍是必要的。

尽管存在地址映射不完美和需要硬件过滤的不足，多播仍然比广播好。

单个物理网络的多播是简单的。多播进程将目的IP地址指明为多播地址，设备驱动程序将它转换为相应的以太网地址，然后把数据发送出去。

这些接收进程必须通知它们的IP层，它们想接收的发往给定多播地址的数据报，并且设备驱动程序必须能够接收这些多播帧。

这个过程就是“加入一个多播组”（使用“接收进程”复数形式的原因在于对一确定的多播信息，在同一主机或多个主机上存在多个接收者，这也是为什么要首先使用多播的原因）。

当一个主机收到多播数据报时，它必须向属于那个多播组的每个进程均传送一个复制。

这和单个进程收到单播UDP数据报的UDP不同。使用多播，一个主机上可能存在多个属于同一多播组的进程。

当把多播扩展到单个物理网络以外需要通过路由器转发多播数据时，复杂性就增加了。

需要有一个协议让多播路由器了解确定网络中属于确定多播组的任何一个主机。

这个协议就是Internet组管理协议（IGMP），也是下一章介绍的内容。

12.4.3FDDI和令牌环网络中的多播FDDI网络使用相同的D类IP地址到48bitFDDI地址的映射过程[Katz1990]。

令牌环网络通常使用不同的地址映射方法，这是因为大多数令牌控制中的限制。

\*/

}

## 分片

物理层限制每次发送数据帧的最大长度，发送时查看发送接口的MTU，与数据报长度进行比较，如果需要则进行分片。

分片和重新组装对传输层是透明的，除了某些可能的越级操作外。

分片报文可能会再次进行分片

分片报文的数据长度取最大，并且使数据部分是8B的倍数

|  |  |
| --- | --- |
| 分片 | 源主机或中间路由器  分片后，每个片的总长度值要改为该片的长度值。  当IP数据报被分片后，每一片都成为一个分组，具有自己的IP首部，并在选择路由时与其他分组独立。  这样，当数据报的这些片到达目的端时有可能会失序，但是在IP首部中有足够的信息让接收端能正确组装这些数据报片。  如果对数据报分片的是中间路由器，而不是起始端系统，那么起始端系统就无法知道数据报是如何被分片的。就这个原因，经常要避免分片 |
| 重新组装 | 目的主机IP层 |
|  | 分片丢失：重传整个数据报，因为IP层没有超时重传机制  即使只丢失一个分片报文，TCP收到的报文时错误的，执行超时重传，TCP在超时后会重发整个TCP报文段，没有办法只重传数据报中的一个数据报片。 |

分片过程

我们可以用SOCk程序来增加数据报的长度，直到分片发生。在一个以太网上，数据帧的最大长度是1500字节（见图2- 1），其中1472字节留给数据，假定IP首部为20字节， UDP首部为8字节。我们分别以数据长度为1471, 1472, 1473和1474字节运行SOCk程序。最后两次应该发生分片：

bsdI% socK-U-I-nL-w1471svr4discard

bsdI% socK-U-I-nL-w1472svr4discard

bsdI% socK-U-I-nL-w1473svr4discard

bsdI% socK-U-I-nL-w1474svr4discard

相应的TCPDUMp输出如图11- 7所示。

前两份UDP数据报（第1行和第2行）能装入以太网数据帧，没有被分片。

但是对应于写1473字节的IP数据报长度为1501，就必须进行分片（第3行和第4行）。

同理，写1474字节产生的数据报长度为1502，它也需要进行分片（第5行和第6行）。

当IP数据报被分片后， TCPDUMP打印出其他的信息。

首先，fraG26304（第3行和第4行）和fraG26313（第5行和第6行）指的是IP首部中标识字段的值。分片信息中的下一个数字，即第3行中位于冒号和@号之间的1480，是除IP首部外的片长。

两份数据报第一片的长度均为1480：UDP首部占8字节，用户数据占1472字节（加上IP首部的20字节分组长度正好为1500字节）。第1份数据报的第2片（第4行）只包含1字节数据—剩下的用户数据。

第2份数据报的第2片（第6行）包含剩下的2字节用户数据。

在分片时，除最后一片外，其他每一片中的数据部分（除IP首部外的其余部分）必须是8字节的整数倍。在本例中， 1480是8的整数倍。位于@符号后的数字是从数据报开始处计算的片偏移值。两份数据报第1片的偏移值均为0(第3行和第5行)，第2片的偏移值为1480（第4行和第6行）。跟在偏移值后面的加号对应于IP首部中3bit标志字段中的“更多片”比特。设置这一比特的目的是让接收端知道在什么时候完成所有的分片组装。

最后，注意第4行和第6行（不是第1片）省略了协议名（ UDP）、源端口号和目的端口号。协议名是可以打印出来的，因为它在IP首部并被复制到各个片中。但是，端口号在UDP首部，只能在第1片中被发现。发送的第3份数据报（用户数据为1473字节）分片情况如图11- 8所示。需要重申的是，任何运输层首部只出现在第1片数据中。

另外需要解释几个术语：

IP数据报是指IP层端到端的传输单元（在分片之前和重新组装之后），

分组是指在IP层和链路层之间传送的数据单元。

一个分组可以是一个完整的IP数据报，也可以是IP数据报的一个分片。

\*/

}

## NAT

NAT路由器维护一个NAT转换表

该路由器所管理的局域网中的主机发送报文至外部网络时，经过该路由器，路由器改变报文的srcIP

来自外部网络的报文发送给该局域网中的主机时，经过该路由器，路由器改变报文的dstIP

NAT表项由管理员添加，控制一个内网到外网的网络连接

主机发送的分组在NAT表中匹配失败，则不转发，丢弃该分组

需要源端口与源IP都匹配

私有IP地址：

|  |  |
| --- | --- |
| A类网络 | 10.0.0.0~10.255.255.255 |
| B类网络 | 172.16.0.0~172.31.255.255 |
| C类网络 | 192.168.0.0~192.168.255.255 |

## IPv6

128bit

首部包含8个字段，简化了首部，首部必须是8B的整数倍

首部长度固定，因此不需要首部长度字段

改进选项

尽在源主机可以分片，并在目的主机重组，路由器不能分片

支持QoS，

没有校验和

# 传输层

## UDP

|  |  |
| --- | --- |
|  |  |
|  | 1.没有连接  2.不可靠交付：接收端收到错误UDP时直接丢弃  3.面向报文：对应用报文添加首部交付IP层，对IP报文去除首部交给指定应用进程，都是交付一个完整的报文  4.没有拥塞控制  5.支持一对一，一对多，多对一，多对多的交互通信  6.首部开销小 |
|  | 什么样的应用程序可以使用UDP？  应用程序必须关心IP数据报的长度。如果它超过网络的MTU（2.8节），那么就要对IP数  据报进行分片。如果需要，源端到目的端之间的每个网络都要进行分片，并不只是发送端主  机连接第一个网络才这样做（我们在2.9节中已定义了路径MTU的概念）。在11.5节中，我们  将讨论IP分片机制。  UDP适合实时通信，因为若使用TCP，只要出现差错或丢失就会重传而导致额外的延迟  而UDP通信是不可靠传输，并且开销很小，传输时若有错误或丢失，使通话质量下降，但满足实时要求，  可靠性由应用层负责 |
|  |  |

计算结果=0→CheckSum=FF FF

/\*

端口号表示发送进程和接收进程。在图1-8中，我们画出了TCP和UDP用目的端口号来分用来自IP层的数据的过程。

由于IP层已经把IP数据报分配给TCP或UDP（根据IP首部中协议字段值），因此TCP端口号由TCP来查看，而UDP端口号由UDP来查看。TCP端口号与UDP端口

号是相互独立的。

\*/

/\*

应用进程的每个输出操作都正好产生一个UDP数据报，并组装成一份待发送的IP数据报。

这与TCP不同，应用程序产生的全体数据与真正发送的单个IP数据报可能没有什么联系。

应用程序产生的数据报长度将保持不变

服务器的设计

典型的服务器与操作系统进行交互作用，大多数需要同时处理多个客户。

通常一个客户启动后直接与单个服务器通信，然后就结束了。而对于服务器来说，它启动后处于休眠状态，等待客户请求的到来。

客户数据报到达时，服务器苏醒，数据报中可能包含来自客户的某种形式的请求消息。

客户IP地址及端口号

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

当一个应用程序接收到UDP数据报时，操作系统必须告诉它是谁发送了这份消息，即源IP地址和端口号。

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

目的IP地址

一些应用程序需要知道数据报是发送给谁的，即目的IP地址。例如，TFTP服务器忽略接收到的发往广播地址的数据报

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

UDP输入队列

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

通常程序所使用的每个UDP端口都与一个有限大小的输入队列相联系。这意味着，来自不同客户的差不多同时到达的请求将由UDP自动排队。

接收到的UDP数据报以其接收顺序交给服务进程（在应用程序要求交送下一个数据报时）。

然而，排队溢出造成内核中的UDP模块丢弃数据报的可能性是存在的。

服务进程并不知道其输入队列何时溢出。只是由UDP对超出数据报进行丢弃处理。

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

UDP输出队列是FIFO（先进先出）的，而我们在11.9节中所看到的ARP输入却是LIFO（后进先出）的。

限制本地IP地址

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

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

例如，我们以端口号777启动一个UDP服务器：

sun%sock-u-s7777

然后，用netstat命令观察端点的状态：

sun%netstat-a-n-finet

ActiveInternetconnections(includingservers)

ProtoRecv-QSend-QLocalAddressForeignAddress(state)

udp00\*.7777\*.\*

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

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

而不是服务名称。-finet选项表示只报告TCP和UDP端点。

本地地址以\*.7777格式打印，星号表示任何本地IP地址。

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

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

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

本地IP地址。例如：

sun%sock-u-s140.252.1.297777

就限制服务器在SLIP接口(140.252.1.29)处接收数据报。netstat输出结果显示如下：

ProtoRecv-QSend-QLocalAddressForeignAddress(state)

udp00140.252.1.29.7777\*.\*

如果我们试图在以太网上的主机bsdi以地址140.252.13.35向该服务器发送一份数据报，

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

所示。

图11-21服务器本地地址绑定导致拒绝接收UDP数据报

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

一般必须告诉系统应用程序重用相同的端口号没有问题。

使用socketsAPI时，必须指定SOREUSEADDRsocket选项。在sock程序中是通过-A

选项来完成的。

在主机sun上，可以在同一个端口号（8888）上启动5个不同的服务器除了第一个以外，其他的服务器都必须以-A选项启动，告诉系统可以重用同一个端口号。

5个服务器的netstat输出结果如下所示：

在这种情况下，到达服务器的数据报中，只有带星号的本地IP地址，其目的地址为

140.252.1.255，因为其他4个服务器占用了其他所有可能的IP地址。

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

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

的端点。

限制远端IP地址

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

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

进行限制。

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

端IP地址和端口号：

sun%sock-u-s-f140.252.13.35.44445555

这样就设置了远端IP地址140.252.13.35（即主机bsdi）和远端端口号4444。服务器的有名端

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

ProtoRecv-QSend-QLocalAddressForeignAddress(state)

udp00140.252.13.33.5555140.252.13.35.4444

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

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

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

140.252.13.33相连。

图11-22总结了UDP服务器本身可以创建的三类地址绑定。

图11-22为UDP服务器指定本地和远端IP地址及端口号

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

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

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

每个端口有多个接收者

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

一份传给该端点。端点的IP地址可以含星号，正如我们前面讨论的那样。

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

sun%sock-u-s9999

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

们指定了-A选项：

sun%sock-u-s9999我们预计它会失败

can'tbindlocaladdress:Addressalreadyinuse

sun%sock-u-s-A9999因此，这次尝试-A参数

can'tbindlocaladdress:Addressalreadyinuse

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

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

SOREUSEADDRsocket选项）。

4.4BSD支持多播传送，需要应用程序设置一个不同的socket选项（SOREUSEPORT）

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

的第一个端点。

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

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

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

端点传送一份数据报的复制。选择哪个端点传送数据取决于各个不同的系统实现。

\*/

## TCP

传输层接收应用层所有应用进程的信息，传输到网络层；接收网络层的数据，传输到应用层中指定的各应用进程

全双工：

面向连接：

1.网络通信实际是主机的应用进程间的相互通信，使用socket表示某个进程，socket=IP:port，IP→唯一主机接口，port→主机中唯一进程

2.两个使用TCP的应用(client-server)，交换数据前先建立一个TCP连接，两个socket确定一个唯一的TCP连接：{socket1,socket2}={(IP1:port1),(IP2:port2)}

3.端口号(16bit：0000~FFFF)

0000~03FF：服务器进程，固定分配，well-known，查看www.iana.org

0400~BFFF：服务器进程，固定分配，后来登记的

C000~FFFF：客户端进程，临时分配，进程运行时申请该临时端口号，运行完毕后返还

可靠性：

1.TCP将应用数据分割成最适合发送的数据块，形成若干个TCP报文，分别装入IP报文

2.超时重传

3.检验和：这是一个端到端的检验和，目的是检测数据在传输过程中的任何变化。如果收到段的检验和有差错， TCP将丢弃这个报文段和不确认收到此报文段（希望发端超时并重发）。

4.确认机制，

5.流量控制。TCP连接的每一方都有固定大小的缓冲空间。TCP的接收端只允许另一端发送接收端缓冲区所能接纳的数据。这将防止较快主机致使较慢主机的缓冲区溢出。

6.失序处理：TCP报文被封装入IP数据报来传输，IP数据报可能会失序到达，因此TCP报文段也会随之失序到达。如果必要， 接收端对收到的数据进行重新排序，以正确的顺序交给应用层。

7.重复处理：TCP报文被封装入IP数据报来传输，IP数据报可能会重复到达，因此TCP报文段也会随之重复到达， 接收端必须丢弃重复的数据。

\*/

void TCP()

{

/\*

首部固定部分：

1~2，xx xx，src port：报文发送端进程端口号

3~4，xx xx，dst port：报文接收端进程端口号

5~8，xx xx xx xx，seq：本报文第一个字节的序号，若数据段为空，值为发送的下一字节的序号 ；发送端对发送的每一个字节进行计数，溢出回零；全双工通信时两端都可作为发送端，对各自发送的字节进行独立计数

9~12，xx xx xx xx，ack：接收端确认已收到的字节，ack表示期望收到的下一个序号(ack=已成功收到字节序号加1)，ACK=1时ack才有效

13~14，x0+[00+URG+ACK+PSH+RST+SYN+FIN]：首部长度=x\*4B(5≤x≤F)，即TCP首部最少15B最多60B，

15~16，xx xx，rwnd：本主机接收窗口大小，单位:B

17~18，xx xx，CheckSum：首部+数据，必需，发送端计算和存储，接收端验证，

19~20，xx xx，Urgent Pointer： 在数据报文中使用到，用于表示紧急数据报文

首部选项部分：

type+length+其他，每项都是4B的倍数，不足右补NOP

选项表结束： 00

NOP： 01，用来填充字节，因为每个首选项都必须是4B的倍数

MMS： 02 04 [MMS,2B] ，接收一个报文包含的应用数据的最大字节数，两端建立连接时在SYN报文中通告各自的MSS，在不导致分片的前提下，MMS越大越好，以提高网络利用率

Window Scale： 03 03 0x，x=0~E，接收窗口从16bit增加为(16+x)bit

SACK Permitted：04 02

TIME： 08 0A [时间戳,4B] [回显应答,4B]

数据：

应用层报文，可选项

如：1.建立/终止TCP连接，2.处理超时的许多情况，3.没有数据要发送而仅使用没有任何数据的首部来确认收到的数据

-------------------------------------------------------------------------------------------------------------------------------------------------------

SYN：同步序号用来发起一个连接，当建立一个新的连接时，SYN标志变1。序号字段包含由这个主机选择的该连接的初始序号ISN（InitialSequenceNumber）。

该主机要发送数据的第一个字节序号为这个ISN加1，因为SYN标志消耗了一个序号（将在下章详细介绍如何建立和终止连接，届时我们将看到FIN标志也要占用一个序号）。

ACK：发送ACK无需任何代价，因为32bit的确认序号字段和ACK标志一样，总是TCP首部的一部分。因此，我们看到一旦一个连接建立起来，这个字段总是被设置，ACK标志也总是被设置为1。

URG：URG=1时紧急指针才有效。紧急指针是一个正的偏移量，和序号字段中的值相加表示紧急数据最后一个字节的序号。TCP的紧急方式是发送端向另一端发送紧急数据的一种方式。我们将在20.8节介绍它。

ACK：确认序号有效。

PSH：接收方应该尽快将这个报文段交给应用层。

RST：重建连接。

FIN：发端完成发送任务。

以太网MSS值最大1460B，默认值-536B，BSD-1024B，目的地址非本地时使用默认值

检验和： 0~79bit，以2B分组，相加，取反

1.数据右补0填充至偶数字节

2.设置12B伪首部。伪首部包含IP首部一些字段。其目的是让UDP两次检查数据是否已经正确到达目的地（例如，IP没有接受地址不是本主机的数据报，以及IP没有把应传给另一高层的数据报传给UDP）

计算结果=0→CheckSum=FF FF

CheckSum=00 00→发送端没有计算检验和。如果发送端没有计算检验和而接收端检测到检验和有差错，那么UDP数据报就要被悄悄地丢弃。不产生任何差错报文（当IP层检测到IP首部检验和有差错时也这样做）

\*/

}

/\*

PSH

最初由应用层设置PUSH，该段数据传输至TCP后直接发送，而不要在发送缓存中滞留

目前由TCP自动设置PUSH标志

服务器收到一个设置了PUSH标志的报文段，立即将该报文数据递交给服务器进程而不等待

TCP报文的总长度≤65535B

TCP报文的总长度≤建立连接时对方给出的MMS+首部长度

MMS：最大报文段长度，相对TCP连接固定，发送缓存中的数据达到MMS个字节时，讲组装成一个TCP发送

字节服务：对收到的数据报文中每个字节进行确认，确认号表示该数值之前的所有数据字节都已确认完毕，

应答信息：每确认一个数据报文，产生一个应答信息(一一对应)

接收缓存：数据来自发送端，分为三部分：【rwnd:接收窗口(空闲区)】【RC1:未确认区】【RC2:已确认但尚未发送确认区】；

发送缓存：数据来自应用层，分为四部分：【SC1:空闲区】【SC2:不允许发送区】【swnd:发送窗口】【SC3:已发送尚未收到确认区】；

发送数据报文时的调整：1.发送的数据：swnd→SC3；

收到确认报文时的调整：1.获取ack，将确认的数据从SC3中移出；2.获取rwnd，更新cwnd，更新swnd=min{rwnd,cwnd}，调整数据swnd↔SC2

cwnd：拥塞窗口，预测可以发送的字节数，不致使网络发生拥塞，值取决于网络的拥塞程度，初始值：2MMS(MSS>2190)、3MMS(MSS>1095)、4MMS(MSS≤1095)

发送方法：对swnd数据执行的发送方法，停止等待协议、滑动窗口协议、Nagle算法、慢启动、拥塞避免算法、 快恢复

确认方法：对RC1数据执行的确认方法，捎带确认、快重传

\*/

void Sender()

{

/\*

停止等待协议： 一次只发送一个报文，收到对该报文的所有数据的确认后才能发送下一个报文

滑动窗口协议：可以连续发送多个报文，不必每发一个就停下来等待确认

Nagle算法：停止等待，等待期间收集已准备好的小分组报文，组合成大分组报文，规模不断扩大，一旦收到确认，就发送这个大分组报文

慢启动算法：滑动窗口，每收到一个ACK，cwnd等比递增(\*2)

拥塞避免算法：滑动窗口，每收到一个ACK，cwnd等差递增(+1)

Nagle算法特点：

自适应：确认达得越快，数据也就发送得越快。而在希望减少微小分组数目的低速广域网上，则会发送更少的分组

适用：小分组报文，报文往返时间很长(广域网)，可用于解决糊涂窗口综合症

不适用：(1)局域网：报文的往返时间短；(2)无时延地传输，如鼠标键盘等交互式

超时处理：

重传定时器：(溢出值:超时重传时间RTO)(启动:发送报文)(归零:收到确认或溢出 )(溢出:重传报文)

RTT：往返时间(发送数据→收到确认)，传播时间(固定)+末端处理时间(固定)+路由器处理时间(随网络拥塞程度的变化而变化)，固定是相对于该TCP连接，初始用一个较大的估计值

SRTT：加权平均往返时间(平滑往返时间)，初始SRTT=RTT

DRTT：RTT偏差的加权平均值，初始DRTT= RTT/2

每收到一个ACK，进行更新：

RTT=当前时间-数据包中Timestamp选项的回显时间，回显时间：该数据包的发出时间

S=0.875\*S+0.125\*RTT，

D=0.75\*D+0.25\*|S-RTT| ， 初始D=RTT/2

RTO=S+4\*D

Karn算法：发送报文，发生超时重传，然后收到确认报文 ，那么该确认报文是对原先发送的报文的确认还是对重传的报文的确认？ 如果发生超时重传，计算SRTT时

丢失处理：

在重传计时器溢出前收到重复的ACK，表明某个报文没有到达接收端，但后续的报文却到达了，情况1，这个报文段在途中耽搁了一些时间，情况2：:这个报文已经丢失

丢失：收到3个连续重复确认：

拥塞处理：

拥塞：网络中出现大量数据，接收端进程来不及处理，路由器或链路过载，路由器会丢弃分组，

判断：随着通信子网的负载增加，吞吐量反而降低，发生超时，每次发生超时时的拥塞窗口可能不同，即网络的拥塞程度是处于变化的

ssthresh：慢启动门限，初始值65535B，

(1)执行慢启动直至cwnd=ssthresh

(2)执行拥塞避免直至发生以下两种情况之一

(3)发生拥塞：ssthresh=ssthresh/2，cwnd=1，→(1)

(3)发生丢失：ssthresh=ssthresh/2，cwnd=ssthresh，重传丢失报文，→(2)

死锁处理

窗口死锁：B发送中断报文，A收到后不再发送报文，之后B发送窗口报文(rwnd=400)但该报文丢失，于是A仍不能发送，B等待接收，双方互相等待

窗口报文：不确认任何数据，包含本端的接收窗口大小； 该报文传输不可靠，丢失时发送方和接收方都不知情

中断报文：特殊的窗口报文，rwnd=0 ，发送方收到中断ACK时，并不是把发送窗口缩回到零，而是立即停止发送

坚持计时器：(溢出值:1.5,3,6,12,24,48,60,60…)(启动:收到中断报文)(归零:收到窗口报文)(溢出:发送窗口探测报文WindowProbe)

窗口探测报文：接收端收到窗口探测报文后回复窗口报文，发送端收到后获取rwcd，0→重启计时器，非0→死锁 )

糊涂窗口综合症：

1. 发送端应用层→发送缓存速度慢，发送缓存总处于很空的状态，总是产生小分组报文

2. 接收端应用层←接收缓存速度慢，接收缓存总处于很满的状态，回复rwnd总是都很小

两种情况都会导致发送方每次只能发送少量的数据，效率很低

处理方法：Nagle算法

只有以下条件之一满足时才发送数据：

(1)可以发送一个满长度的报文段；

(2)可以发送至少是接收方通告窗口大小一半的报文段；

(3)可以发送任何数据并且不希望接收ACK（也就是说，我们没有还未被确认的数据）或者该连接上不能使用Nagle算法。

条件(1)主要对付那些总是通告小窗口（也许比1个报文段还小）的主机，

条件(3)使我们在有尚未被确认的数据（正在等待被确认）以及在不能使用Nagle算法的情况下，避免发送小的报文段。

如果应用进程在进行小数据的写操作（例如比该报文段还小），条件(3)可以避免出现糊涂窗口综合症。

这三个条件也可以让我们回答这样一个问题：在有尚未被确认数据的情况下，如果Nagle算法阻止我们发送小的报文段，那多小才算是小呢？

从条件( 1)中可以看出所谓“小”就是指字节数小于报文段的大小。

条件(2)仅用来对付较老的、原始的主机。

步骤2中的条件( B)要求发送方始终监视另一方通告的最大窗口大小，这是一种发送方猜测对方接收缓存大小的企图。

虽然在连接建立时接收缓存的大小可能会减小，但在实际中这种情况很少见。

\*/

}

void Receiver()

{

/\*

确认机制：

1.捎带确认：发送数据时才捎带确认

2.延时确认：收到数据并不立即发送确认报文，每隔固定时间发送一个确认报文，一个确认报文可能包含多个应答信息

经受时延定时器：(溢出值:一般200ms)(启动:TCP连接建立)(溢出:发送确认报文，包含当前所有应答信息，重启定时器)

回复延迟：接收端为接收的每条数据报文设定一个延迟时间：收到该数据报文→发送包含该数据报文应答信息的确认报文，范围：0~~主机的经受时延

快重传：

接收方每收到一个失序的报文，立即发出重复确认，为使发送及早知道报文段没有到达，而不要等待自己发送数据时才进行捎带确认

失序处理：

1.将数据保存到接收缓存，作为[尚未确认的数据]

2.快重传：立即发送一个ACK，确认号是已确认的数据中最大序号加1, 显然这是一个重复的ACK

由于我们不知道一个重复的ACK是由一个丢失的报文段引起的，还是由于仅仅出现了几个报文段的重新排序，因此我们等待少量重复的ACK到来。

假如这只是一些报文段的重新排序，则在重新排序的报文段被处理并产生一个新的A C K之前，只可能产生1 ~ 2个重复的ACK。

报文段丢失：一连串收到3个或3个以上的重复ACK，重传丢失的数据报文段，而无需等待超时定时器溢出。这就是快速重传算法。

接下来执行的不是慢启动算法而是拥塞避免算法。这就是快速恢复算法。

重复处理：

糊涂窗口综合症：

TCP避免通告小的窗口大小或发送小的报文段来避免出现糊涂窗口综合症

Clark算法：接收方发送ACK通告一个零窗口，直到接收缓存空闲空间足够大

延迟法：延迟一段时间后再发送ACK，优点是减少了通信量，缺点是可能会导致重传

退N帧协议：

发送方不需要收到上一帧的ACK后才能开始发送下一帧 ，可以连续发送

接收方检测出失序的信息帧，要求发送方重发最后一个正确接收的信息帧之后的所有未被确认的帧

或者当发送方发送了N个帧后，若发现该N个帧的前一个帧在计时器超时后仍未返回其确认信息，则该帧被判为出错或丢失 ，此时发送方不得不又重传该出错帧即随后的N个帧，

即接收方只允许按顺序接收帧

使用累积确认，比如发送方收到了对第5个帧的确认，但没有第3个帧的确认 ，还是认为前5帧都收到了

序列号个数≥max\_seq+1

\*/

}

void TCPLink()

{

}

/\*

保活计时器：(溢出值2小时 )(启动:服务器收到客户端的信息)(归零:服务器收到客户端的信息)(溢出:每隔75秒发送探测报文到客户端，若还是没有响应，则认为客户端出现故障，自动终止TCP连接)

为防止在两个TCP之间的连接出现长时间的空闲

TCP连接：建立一个连接需3个握手报文，终止一个连接要经过4个握手报文。

SYN报文和FIN报文，不含数据，但仍然消耗一个序号，因为很重要，不允许丢失

初始序号：ISN是一个32bit的计数器，每4ms加1，由此保证相近的若干个连接都具有不同的ISN，防止在网络中被延迟的分组在以后又被传送，而导致某个连接的一方对它作错误的解释。

C→S：SYN=1，seq=ISNc， 客户端发送第一个SYN报文，指明服务器的端口，初始序号ISNc

S→C：SYN=1，ACK=1，seq=ISNs，ack=ISNc+1，服务端回复第二个SYN报文，初始序号SYNs，

C→S：ACK=1，seq=ISNc+1，ack=ISNs+1，客户端回复，客户必须将确认序号设置为服务器的ISNy加1以对服务器的SYN报文段进行确认

……DATA……

C→S：FIN=1，seq=u //客户端单方面释放连接，不再发送数据，但可以接收数据

S→C：ACK=1，seq=v，ack=u+1

S→C：DATA

……

S→C：DATA

S→C：FIN=1，ACK=1，seq=w，ack=u+1

C→S：ACK=1，seq=u+1，ack=w+1

客户端和服务器都可以提出释放连接的请求

使用三个握手报文建立连接是为了避免偶然性连接

第三个握手报文不需要得到回复，这个报文即使丢失， 客户端发送第一个握手报文后(不管有没有到达服务器)，就紧接着开始发送数据，服务端收到数据报文同样能开启连接

主动打开：发送第一个SYN报文

被动打开：接收第一个SYN报文并回复第二个SYN报文

主动关闭：发送FIN报文，执行后将不再使用该TCP连接发送数据，一般由客户端执行

被动关闭：接收FIN报文并回复一个ACK报文，通知应用层另一端已终止数据传送。

完全关闭：双方相继执行主动关闭和被动关闭，即一共传输4个终止握手报文

半关闭：仅一方执行主动关闭(之后只接收数据不发送数据)，另一方执行被动关闭(之后只发送数据不接收数据) ，实际应用中很少这样做

建立连接超时： 有很多情况导致无法建立连接。一种情况是服务器主机没有处于正常状态

最大生存时间MSL：任何报文段被丢弃前在网络内的最长时间，常用值是30秒，1分钟，或2分钟

socket状态：

使用状态：不能主动打开，不能被动打开，可以主动关闭，可以被动关闭

等待状态：不能主动打开，但可以被动打开

空闲状态：主动打开，被动打开

2MSL定时器：(溢出值2MSL)(启动:主动关闭)(归零:)(溢出:)

计时器溢出之前，连接处于中间过渡状态，保证FIN报文可以达到目的站，从而将其丢弃，

一个关于主动关闭的问题： 一端发送FIN报文，执行主动关闭，这时不能马上关闭TCP连接，因为如果这个FIN报文丢失，该端还需要重传

如果当初已经关闭了TCP连接，那么就不能发送重传报文了

因此一端执行主动关闭后，TCP连接还是会保持一段时间，该时间取2MSL，该时间段称为TIMEWAIT状态(2MSL等待状态)

对一个具体实现所给定的MSL值，处理的原则是：当TCP执行一个主动关闭，并发回最后一个ACK，该连接必须在TIMEWAIT状态停留的时间为2倍的MSL。

这样可让TCP再次发送最后的ACK以防这个ACK丢失（另一端超时并重发最后的FIN）。

这个TCP连接在2MSL等待期间，定义这个连接的socket不能再被使用。这个连接只能在2MSL结束后才能再被使用。

遗憾的是，大多数TCP实现（如伯克利版）强加了更为严格的限制。在2MSL等待期间，插口中使用的本地端口在默认情况下不能再被使用。 我们将在下面看到这个限制的例子。

某些实现和API提供了一种避开这个限制的方法。使用插口API时，可说明其中的SOREUSEADDR选项。

它将让调用者对处于2MSL等待的本地端口进行赋值，但我们将看到TCP原则上仍将避免使用仍处于2MSL连接中的端口。

在连接处于2MSL等待时，任何迟到的报文段将被丢弃。

因为处于2MSL等待的、由该插口对(socketpair)定义的连接在这段时间内不能被再用，因此当要建立一个有效的连接时，

来自该连接的一个较早替身（incarnation）的迟到报文段作为新连接的一部分不可能不被曲解

（一个连接由一个插口对来定义。一个连接的新的实例（instance）称为该连接的替身）。

执行被动关闭，不会进入TIMEWAIT状态。

客户端执行主动关闭，终止客户端进程后又立即重新启动，此时还处于2MSL等待状态，刚使用的端口号不能用，但该客户端可以正常运行，因为这时客户端使用了一个新的端口号

服务器进程执行主动关闭，终止服务器进程后又立即重新启动，此时还处于2MSL等待状态，程序报告一个差错信息说明不能绑定它的端口，因为该端口此时还处于2MSL等待状态，仍处于使用中

如果一直试图重新启动服务器程序，测量它直到成功所需的时间，就能确定出2MSL值。

如果一个客户程序试图申请一个处于2MSL等待的端口（客户程序通常不会这么做），就会出现同样的差错。

和以前介绍的一样，一个插口对（即包含本地IP地址、本地端口、远端IP地址和远端端口的4元组）在它处于2MSL等待时，将不能再被使用。

尽管许多具体的实现中允许一个进程重新使用仍处于2MSL等待的端口（通常是设置选项SOREUSEADDR），不允许一个新的连接建立在相同的插口对上。

处于2MSL等待状态的socket不能主动打开，但可以被动打开，这违反了TCP规范，但被大多数的伯克利版实现所支持。

这些实现允许一个新的连接请求到达仍处于TIMEWAIT状态的连接，只要新的序号ISN大于该连接前一个替身的最后序号。

这项技术仍可能存在缺陷。对于同一连接的前一个替身，这个具体实现中的特性让客户程序和服务器程序能连续地重用每一端的相同端口号，但这只有在服务器执行主动关闭才有效。

平静时间

TCP在主机重启动后的MSL秒内不能建立任何连接。这就称为平静时间(quiettime)。

为了防止重启前的某个TCP连接传输的报文被错误地当作属于重启后新连接的报文段

只有极少的实现版遵守这一原则，因为大多数主机重启动的时间都比MSL秒要长。

FIN\_WAIT\_2状态

在FINWAIT2状态我们已经发出了FIN，并且另一端也已对它进行确认。

除非我们在实行半关闭，否则将等待另一端的应用层意识到它已收到一个文件结束符说明，并向我们发一个FIN来关闭另一方向的连接。

只有当另一端的进程完成这个关闭，我们这端才会从FINWAIT2状态进入TIMEWAIT状态。

这意味着我们这端可能永远保持这个状态。另一端也将处于CLOSEWAIT状态，并一直保持这个状态直到应用层决定进行关闭。

许多伯克利实现采用如下方式来防止这种在FINWAIT2状态的无限等待。如果执行主动关闭的应用层将进行全关闭，而不是半关闭来说明它还想接收数据，就设置一

个定时器。如果这个连接空闲10分钟75秒，TCP将进入CLOSED状态。在实现代码的注释中确认这个实现代码违背协议的规范。

18.7复位报文段

无论何时一个报文段发往基准的连接（referencedconnection）出现错误，TCP都会发出一个复位报文段

基准的连接：由目的IP地址和目的端口号以及源IP地址和源端口号指明的连接。这就是为什么称之为插口

1.目的端口没有进程：

对于UDP，当一个数据报到达目的端口时，该端口没在使用，它将产生一个ICMP端口不可达的信息

对于TCP，则使用复位

注意复位报文段中的seq和ack。因为ACK比特在到达的报文段中没有被设置为1，复位报文段中的序号被置为0，确认序号被置为进入的ISN加上数据字节数。

尽管在到达的报文段中没有真正的数据，但SYN比特从逻辑上占用了1字节的序号空间；

因此，在这个例子中复位报文段中确认序号被置为ISN与数据长度（0）、SYN比特所占的1的总和。

异常终止abortiverelease

发送一个复位报文段释放连接

优点：(1)丢弃任何待发数据并立即发送复位报文段；(2)RST的接收方会区分另一端执行的是异常关闭还是正常关闭。

应用程序使用的API必须提供产生异常关闭而不是正常关闭的手段。

一端发送一个RST而不是通常的FIN。

RST报文段中包含一个序号和确认序号。需要注意的是RST报文段不会导致另一端产生任何响应，另一端根本不进行确认。

收到RST的一方将终止该连接，并通知应用层连接复位。

另一端收到RST时，产生一个差错。这个差错正是我们所期待的：连接被对方复位了。

半打开Half-Open：

如果一方已经关闭或异常终止连接而另一方却还不知道，我们将这样的TCP连接称为半打开的。

只要不打算在半打开连接上传输数据，仍处于连接状态的一方就不会检测另一方已经出现异常。

产生原因： 主机异常、主机突然掉电而不是正常的结束客户应用程序后再关机。

务器服将永远不知道客户程序已经消失了。 用户重新启动客户程序，在服务器主机上会启动一个新的服务器程序，这样会导致服务器主机中产生许多半打开的TCP连接

能很容易地建立半打开连接

断开服务器主机与以太网的电缆，并重启服务器主机，模拟服务器主机出现异常（在重启服务器之前断开以太网电缆是为了防止它向打开的连接发送FIN，某些TCP在关机时会这么做）。

服务器主机重启后，我们重新接上电缆，并从客户向服务器发送另一行字符。

由于服务器的TCP已经重新启动，它将丢失复位前连接的所有信息，因此它不知道数据报文段中提到的连接。

处理原则：接收方以复位作为应答。

注意由于服务器主机经过重新启动，它的ARP高速缓存为空，因此需要一个ARP请求和应答。

客户收到复位报文段后显示连接已被另一端的主机终止(所以客户发送的信息没有什么价值)

同时打开simultaneous open

两个应用程序同时彼此执行主动打开：

每一方必须发送一个SYN，且这些SYN必须传递给对方。每一方使用一个对方熟知的端口作为本地端口

如主机A中的一个应用程序使用本地端口7777，并向主机B的端口8888执行主动打开，主机B中的应用程序则使用本地端口8888，并与主机A的端口7777执行主动打开。

对于同时打开要仅建立一条连接而不是两条连接(其他的协议族，最突出的是OSI运输层，在这种情况下将建立两条连接而不是一条连接)。

两端几乎在同时发送SYN，并进入SYNSENT状态。当每一端收到SYN时，状态变为SYNRCVD，同时它们都再发SYN并对收到的SYN进行确认，当双方都收到SYN及相应的ACK时，状态都变迁为ESTABLISHED。

一个同时打开的连接需要交换4个报文段，比正常的三次握手多一个。

每一端既是客户又是服务器。

同时关闭simultaneous close

双方都执行主动关闭

当应用层发出关闭命令时，两端均从ESTABLISHED变为FIN\_WAIT\_1。

双方各发送一个FIN，双方都收到FIN，状态由FIN\_WAIT\_1变迁到CLOSING，并发送最后的ACK。当收到最后的ACK时，状态变化为TIMEWAIT。

同时关闭与正常关闭使用的段交换数目相同。

TCP服务器的设计

大多数的TCP服务器进程是并发的。

当一个新的连接请求到达服务器时，服务器接受这个请求，并调用一个新进程来处理这个新的客户请求。

不同的操作系统使用不同的技术来调用新的服务器进程。在Unix系统下，常用的技术是使用fork函数来创建新的进程。如果系统支持，也可使用轻型进程，即线程（thread）。

1.TCP服务器端口号： 当一个服务器进程接受一来自客户进程的服务请求时是如何处理端口的？如果多个连接请求几乎同时到达会发生什么情况？

2.限定的本地IP地址：服务器不能任选其本地IP地址而必须使用特定的IP地址时的情况。

3.限定的远端IP地址：服务器有时会指定远端IP地址和远端端口，但大多数不这么做，服务器不指明远端插口，而等待连接请求的到来，然后检查客户端的IP地址和端口号。

4.呼入连接请求队列：

服务器在创建一个新的进程时，或操作系统正忙于处理优先级更高的进程时，到达多个连接请求。当服务器正处于忙时，TCP是如何处理这些呼入的连接请求？

在伯克利的TCP实现中采用以下规则：

1)连接队列：队列中的连接已被TCP接受(已完成三次握手)，但还没有被应用层接受。TCP接受一个连接是将其放入这个队列，而应用层接受连接是将其从该队列中移出。

2)积压值(backlog)：队列的最大长度，范围是0~5，应用层指定

3)最大连接数：允许接受连接的最大数目，积压值对系统所允许的最大连接数，或者并发服务器所能并发处理的客户数，并无影响

4)队列未满时，TCP接受新的SYN请求，但并不让应用层了解该连接源于何处（即不告知源IP地址和源端口）。 应用层只有在三次握手中的第三个报文段收到后才会知道这个新连接时。

这不是TCP所要求的，而只是共同的实现技术（如伯克利源代码通常都这么做）。

另外，当客户进程的主动打开成功但服务器的应用层还不知道这个新的连接时，它可能会认为服务器进程已经准备好接收数据了（如果发生这种情况，服务器的TCP仅将接收的数据放入缓冲队列)。

5)队列已满时，TCP不理会新的SYN请求，也不发回RST作为应答，因为这是一个软错误，而不是一个硬错误

如果应用层不能及时接受已被TCP接受的连接，这些连接可能占满整个连接队列，客户的主动打开最终将超时。

通常队列已满是由于应用程序或操作系统忙造成的，这样可防止应用程序对传入的连接进行服务。这个条件在一个很短的时间内可以改变。

但如果服务器的TCP以系统复位作为响应，客户进程的主动打开将被废弃（如果服务器程序没有启动我们就会遇到）。

由于不应答SYN，服务器程序迫使客户TCP随后重传SYN，以等待连接队列有空间接受新的连接。

如果一个API如TLI（见1.15节）向应用程序提供了解连接请求的到来的方法，并允许应用程序选择是否接受连接。当应用程序假定被告知连接请求已经到来时，

TCP的三次握手已经结束！其他运输层的实现可能将连接请求的到达与接受分开（如OSI的运输层），但TCP不是这样。

Solaris2.2提供了一个选项使TCP只有在应用程序说可以接受（tcpeagerlisteners见E.4），才允许接受传入的连接请求。

这种行为也意味着TCP服务器无法使客户进程的主动打开失效。

当一个新的客户连接传递给服务器的应用程序时，TCP的三次握手就结束了，客户的主动打开已经完全成功。

如果服务器的应用程序此时看到客户的IP地址和端口号，并决定是否为该客户进行服务，服务器所能做的就是关闭连接（发送FIN），或者复位连接（发送RST）。无论哪种情况，客户进程都认为一切正常，因为它的主动打开已经完成，并且已经向服务器程序发送过请求

## TCP发送

|  |  |  |
| --- | --- | --- |
| 发送  缓存 | 数据来自应用层，分为四部分：【SC1:空闲区】【SC2:不允许发送区】【swnd:发送窗口】【SC3:已发送尚未收到确认区】 | |
| 发送数据报文时的调整：1.发送的数据：swnd→SC3；  收到确认报文时的调整：1.获取ack，将确认的数据从SC3中移出；2.获取rwnd，更新cwnd，更新swnd=min{rwnd,cwnd}，调整数据swnd↔SC2 | |
| 拥塞  窗口  cwnd | 预测可以发送的字节数，不致使网络发生拥塞，值取决于网络的拥塞程度，初始值：2MMS(MSS>2190)、3MMS(MSS>1095)、4MMS(MSS≤1095) | |
| 滑动  窗口 |  | |
| 发送  机制 | 对swnd数据执行的发送方法 | |
| 停止  等待 | 一次只发送一个报文，收到对该报文的所有数据的确认后才能发送下一个报文  滑动窗口协议：可以连续发送多个报文，不必每发一个就停下来等待确认 |
|  |  |
| Nagle  算法 | 停止等待，等待期间收集已准备好的小分组报文，组合成大分组报文，规模不断扩大，一旦收到确认，就发送这个大分组报文  Nagle算法特点：  自适应：确认达得越快，数据也就发送得越快。而在希望减少微小分组数目的低速广域网上，则会发送更少的分组  适用：小分组报文，报文往返时间很长(广域网)，可用于解决糊涂窗口综合症  不适用：(1)局域网：报文的往返时间短；(2)无时延地传输，如鼠标键盘等交互式 |
| 慢启动 | 滑动窗口，每收到一个ACK，cwnd等比递增(\*2) |
| 拥塞  避免 | 滑动窗口，每收到一个ACK，cwnd等差递增(+1) |
| 快恢复 |  |
| 超时  处理 | 重传定时器：(溢出值:超时重传时间RTO)(启动:发送报文)(归零:收到确认或溢出 )(溢出:重传报文)  RTT：往返时间(发送数据→收到确认)，传播时间(固定)+末端处理时间(固定)+路由器处理时间(随网络拥塞程度的变化而变化)，固定是相对于该TCP连接，初始用一个较大的估计值  SRTT：加权平均往返时间(平滑往返时间)，初始SRTT=RTT  DRTT：RTT偏差的加权平均值，初始DRTT= RTT/2  每收到一个ACK，进行更新：  RTT=当前时间-数据包中Timestamp选项的回显时间，回显时间：该数据包的发出时间  S=0.875\*S+0.125\*RTT，  D=0.75\*D+0.25\*|S-RTT| ， 初始D=RTT/2  RTO=S+4\*D  Karn算法：发送报文，发生超时重传，然后收到确认报文 ，那么该确认报文是对原先发送的报文的确认还是对重传的报文的确认？ 如果发生超时重传，计算SRTT时 | |
| 丢失  处理 | 在重传计时器溢出前收到重复的ACK，表明某个报文没有到达接收端，但后续的报文却到达了，情况1，这个报文段在途中耽搁了一些时间，情况2：:这个报文已经丢失  丢失：收到3个连续重复确认 | |
| 拥塞  处理 | 拥塞：网络中出现大量数据，接收端进程来不及处理，路由器或链路过载，路由器会丢弃分组，  判断：随着通信子网的负载增加，吞吐量反而降低，发生超时，每次发生超时时的拥塞窗口可能不同，即网络的拥塞程度是处于变化的  ssthresh：慢启动门限，初始值65535B，  (1)执行慢启动直至cwnd=ssthresh  (2)执行拥塞避免直至发生以下两种情况之一  (3)发生拥塞：ssthresh=ssthresh/2，cwnd=1，→(1)  (3)发生丢失：ssthresh=ssthresh/2，cwnd=ssthresh，重传丢失报文，→(2) | |
| 死锁  处理 | 窗口死锁：B发送中断报文，A收到后不再发送报文，之后B发送窗口报文(rwnd=400)但该报文丢失，于是A仍不能发送，B等待接收，双方互相等待  窗口报文：不确认任何数据，包含本端的接收窗口大小； 该报文传输不可靠，丢失时发送方和接收方都不知情  中断报文：特殊的窗口报文，rwnd=0 ，发送方收到中断ACK时，并不是把发送窗口缩回到零，而是立即停止发送  坚持计时器：(溢出值:1.5,3,6,12,24,48,60,60…)(启动:收到中断报文)(归零:收到窗口报文)(溢出:发送窗口探测报文WindowProbe)  窗口探测报文：接收端收到窗口探测报文后回复窗口报文，发送端收到后获取rwcd，0→重启计时器，非0→死锁 ) | |
| 糊涂  窗口  综合症 | 1. 发送端应用层→发送缓存速度慢，发送缓存总处于很空的状态，总是产生小分组报文  2. 接收端应用层←接收缓存速度慢，接收缓存总处于很满的状态，回复rwnd总是都很小  两种情况都会导致发送方每次只能发送少量的数据，效率很低  处理方法：Nagle算法  只有以下条件之一满足时才发送数据：  (1)可以发送一个满长度的报文段；  (2)可以发送至少是接收方通告窗口大小一半的报文段；  (3)可以发送任何数据并且不希望接收ACK（也就是说，我们没有还未被确认的数据）或者该连接上不能使用Nagle算法。  条件(1)主要对付那些总是通告小窗口（也许比1个报文段还小）的主机，  条件(3)使我们在有尚未被确认的数据（正在等待被确认）以及在不能使用Nagle算法的情况下，避免发送小的报文段。  如果应用进程在进行小数据的写操作（例如比该报文段还小），条件(3)可以避免出现糊涂窗口综合症。  这三个条件也可以让我们回答这样一个问题：在有尚未被确认数据的情况下，如果Nagle算法阻止我们发送小的报文段，那多小才算是小呢？  从条件( 1)中可以看出所谓“小”就是指字节数小于报文段的大小。  条件(2)仅用来对付较老的、原始的主机。  步骤2中的条件( B)要求发送方始终监视另一方通告的最大窗口大小，这是一种发送方猜测对方接收缓存大小的企图。  虽然在连接建立时接收缓存的大小可能会减小，但在实际中这种情况很少见。 | |

## TCP接收

|  |  |
| --- | --- |
| 字节服务 | 对收到的数据报文中每个字节进行确认，确认号表示该数值之前的所有数据字节都已确认完毕， |
| 应答信息 | 每确认一个数据报文，产生一个应答信息(一一对应) |
| 接收缓存 | 数据来自发送端，分为三部分：【rwnd:接收窗口(空闲区)】【RC1:未确认区】【RC2:已确认但尚未发送确认区】； |
| 确认方法 | 对RC1数据执行的确认方法，捎带确认、快重传 |
| 确认机制 | 1.捎带确认：发送数据时才捎带确认  2.延时确认：收到数据并不立即发送确认报文，每隔固定时间发送一个确认报文，一个确认报文可能包含多个应答信息  经受时延定时器：(溢出值:一般200ms)(启动:TCP连接建立)(溢出:发送确认报文，包含当前所有应答信息，重启定时器)  回复延迟：接收端为接收的每条数据报文设定一个延迟时间：收到该数据报文→发送包含该数据报文应答信息的确认报文，范围：0~~主机的经受时延  快重传：  接收方每收到一个失序的报文，立即发出重复确认，为使发送及早知道报文段没有到达，而不要等待自己发送数据时才进行捎带确认 |
| 失序处理 | 1.将数据保存到接收缓存，作为[尚未确认的数据]  2.快重传：立即发送一个ACK，确认号是已确认的数据中最大序号加1, 显然这是一个重复的ACK  由于我们不知道一个重复的ACK是由一个丢失的报文段引起的，还是由于仅仅出现了几个报文段的重新排序，因此我们等待少量重复的ACK到来。  假如这只是一些报文段的重新排序，则在重新排序的报文段被处理并产生一个新的A C K之前，只可能产生1 ~ 2个重复的ACK。  报文段丢失：一连串收到3个或3个以上的重复ACK，重传丢失的数据报文段，而无需等待超时定时器溢出。这就是快速重传算法。  接下来执行的不是慢启动算法而是拥塞避免算法。这就是快速恢复算法。 |
| 重复处理 |  |
| 糊涂窗口综合症 | TCP避免通告小的窗口大小或发送小的报文段来避免出现糊涂窗口综合症  Clark算法：接收方发送ACK通告一个零窗口，直到接收缓存空闲空间足够大  延迟法：延迟一段时间后再发送ACK，优点是减少了通信量，缺点是可能会导致重传 |
| 退N帧协议 | 发送方不需要收到上一帧的ACK后才能开始发送下一帧 ，可以连续发送  接收方检测出失序的信息帧，要求发送方重发最后一个正确接收的信息帧之后的所有未被确认的帧  或者当发送方发送了N个帧后，若发现该N个帧的前一个帧在计时器超时后仍未返回其确认信息，则该帧被判为出错或丢失 ，此时发送方不得不又重传该出错帧即随后的N个帧，  即接收方只允许按顺序接收帧  使用累积确认，比如发送方收到了对第5个帧的确认，但没有第3个帧的确认 ，还是认为前5帧都收到了  序列号个数≥max\_seq+1 |

## TCP连接

|  |  |
| --- | --- |
| TCP连接 | 建立一个连接需3个握手报文，终止一个连接要经过4个握手报文。  SYN报文和FIN报文，不含数据，但仍然消耗一个序号，因为很重要，不允许丢失  初始序号：ISN是一个32bit的计数器，每4ms加1，由此保证相近的若干个连接都具有不同的ISN，防止在网络中被延迟的分组在以后又被传送，而导致某个连接的一方对它作错误的解释。 |
|  | C→S：SYN=1，seq=ISNc， 客户端发送第一个SYN报文，指明服务器的端口，初始序号ISNc  S→C：SYN=1，ACK=1，seq=ISNs，ack=ISNc+1，服务端回复第二个SYN报文，初始序号SYNs，  C→S：ACK=1，seq=ISNc+1，ack=ISNs+1，客户端回复，客户必须将确认序号设置为服务器的ISNy加1以对服务器的SYN报文段进行确认  ……DATA……  C→S：FIN=1，seq=u //客户端单方面释放连接，不再发送数据，但可以接收数据  S→C：ACK=1，seq=v，ack=u+1  S→C：DATA  ……  S→C：DATA  S→C：FIN=1，ACK=1，seq=w，ack=u+1  C→S：ACK=1，seq=u+1，ack=w+1  同步序号用来发起一个连接，当建立一个新的连接时，SYN标志变1。  序号字段包含由这个主机选择的该连接的初始序号ISN  该主机要发送数据的第一个字节序号为这个ISN加1，因为SYN标志消耗了一个序号 |
|  | 客户端和服务器都可以提出释放连接的请求  使用三个握手报文建立连接是为了避免偶然性连接  第三个握手报文不需要得到回复，这个报文即使丢失， 客户端发送第一个握手报文后(不管有没有到达服务器)，就紧接着开始发送数据，服务端收到数据报文同样能开启连接    主动打开：发送第一个SYN报文  被动打开：接收第一个SYN报文并回复第二个SYN报文  主动关闭：发送FIN报文，执行后将不再使用该TCP连接发送数据，一般由客户端执行  被动关闭：接收FIN报文并回复一个ACK报文，通知应用层另一端已终止数据传送。  完全关闭：双方相继执行主动关闭和被动关闭，即一共传输4个终止握手报文  半关闭：仅一方执行主动关闭(之后只接收数据不发送数据)，另一方执行被动关闭(之后只发送数据不接收数据) ，实际应用中很少这样做 |
| 建立连接超时 | 有很多情况导致无法建立连接。一种情况是服务器主机没有处于正常状态 |
| 最大生存时间MSL | 任何报文段被丢弃前在网络内的最长时间，常用值是30秒，1分钟，或2分钟 |
| 保活计时器 | (溢出值2小时 )(启动:服务器收到客户端的信息)(归零:服务器收到客户端的信息)(溢出:每隔75秒发送探测报文到客户端，若还是没有响应，则认为客户端出现故障，自动终止TCP连接)  为防止在两个TCP之间的连接出现长时间的空闲 |
| socket状态 | 使用状态：不能主动打开，不能被动打开，可以主动关闭，可以被动关闭  等待状态：不能主动打开，但可以被动打开  空闲状态：主动打开，被动打开 |
| 2MSL定时器 | (溢出值2MSL)(启动:主动关闭)(归零:)(溢出:)  计时器溢出之前，连接处于中间过渡状态，保证FIN报文可以达到目的站，从而将其丢弃， |

/\*

：

：

一个关于主动关闭的问题： 一端发送FIN报文，执行主动关闭，这时不能马上关闭TCP连接，因为如果这个FIN报文丢失，该端还需要重传

如果当初已经关闭了TCP连接，那么就不能发送重传报文了

因此一端执行主动关闭后，TCP连接还是会保持一段时间，该时间取2MSL，该时间段称为TIMEWAIT状态(2MSL等待状态)

对一个具体实现所给定的MSL值，处理的原则是：当TCP执行一个主动关闭，并发回最后一个ACK，该连接必须在TIMEWAIT状态停留的时间为2倍的MSL。

这样可让TCP再次发送最后的ACK以防这个ACK丢失（另一端超时并重发最后的FIN）。

这个TCP连接在2MSL等待期间，定义这个连接的socket不能再被使用。这个连接只能在2MSL结束后才能再被使用。

遗憾的是，大多数TCP实现（如伯克利版）强加了更为严格的限制。在2MSL等待期间，插口中使用的本地端口在默认情况下不能再被使用。 我们将在下面看到这个限制的例子。

某些实现和API提供了一种避开这个限制的方法。使用插口API时，可说明其中的SOREUSEADDR选项。

它将让调用者对处于2MSL等待的本地端口进行赋值，但我们将看到TCP原则上仍将避免使用仍处于2MSL连接中的端口。

在连接处于2MSL等待时，任何迟到的报文段将被丢弃。

因为处于2MSL等待的、由该插口对(socketpair)定义的连接在这段时间内不能被再用，因此当要建立一个有效的连接时，

来自该连接的一个较早替身（incarnation）的迟到报文段作为新连接的一部分不可能不被曲解

（一个连接由一个插口对来定义。一个连接的新的实例（instance）称为该连接的替身）。

执行被动关闭，不会进入TIMEWAIT状态。

客户端执行主动关闭，终止客户端进程后又立即重新启动，此时还处于2MSL等待状态，刚使用的端口号不能用，但该客户端可以正常运行，因为这时客户端使用了一个新的端口号

服务器进程执行主动关闭，终止服务器进程后又立即重新启动，此时还处于2MSL等待状态，程序报告一个差错信息说明不能绑定它的端口，因为该端口此时还处于2MSL等待状态，仍处于使用中

如果一直试图重新启动服务器程序，测量它直到成功所需的时间，就能确定出2MSL值。

如果一个客户程序试图申请一个处于2MSL等待的端口（客户程序通常不会这么做），就会出现同样的差错。

和以前介绍的一样，一个插口对（即包含本地IP地址、本地端口、远端IP地址和远端端口的4元组）在它处于2MSL等待时，将不能再被使用。

尽管许多具体的实现中允许一个进程重新使用仍处于2MSL等待的端口（通常是设置选项SOREUSEADDR），不允许一个新的连接建立在相同的插口对上。

处于2MSL等待状态的socket不能主动打开，但可以被动打开，这违反了TCP规范，但被大多数的伯克利版实现所支持。

这些实现允许一个新的连接请求到达仍处于TIMEWAIT状态的连接，只要新的序号ISN大于该连接前一个替身的最后序号。

这项技术仍可能存在缺陷。对于同一连接的前一个替身，这个具体实现中的特性让客户程序和服务器程序能连续地重用每一端的相同端口号，但这只有在服务器执行主动关闭才有效。

平静时间

TCP在主机重启动后的MSL秒内不能建立任何连接。这就称为平静时间(quiettime)。

为了防止重启前的某个TCP连接传输的报文被错误地当作属于重启后新连接的报文段

只有极少的实现版遵守这一原则，因为大多数主机重启动的时间都比MSL秒要长。

FIN\_WAIT\_2状态

在FINWAIT2状态我们已经发出了FIN，并且另一端也已对它进行确认。

除非我们在实行半关闭，否则将等待另一端的应用层意识到它已收到一个文件结束符说明，并向我们发一个FIN来关闭另一方向的连接。

只有当另一端的进程完成这个关闭，我们这端才会从FINWAIT2状态进入TIMEWAIT状态。

这意味着我们这端可能永远保持这个状态。另一端也将处于CLOSEWAIT状态，并一直保持这个状态直到应用层决定进行关闭。

许多伯克利实现采用如下方式来防止这种在FINWAIT2状态的无限等待。如果执行主动关闭的应用层将进行全关闭，而不是半关闭来说明它还想接收数据，就设置一

个定时器。如果这个连接空闲10分钟75秒，TCP将进入CLOSED状态。在实现代码的注释中确认这个实现代码违背协议的规范。

18.7复位报文段

无论何时一个报文段发往基准的连接（referencedconnection）出现错误，TCP都会发出一个复位报文段

基准的连接：由目的IP地址和目的端口号以及源IP地址和源端口号指明的连接。这就是为什么称之为插口

1.目的端口没有进程：

对于UDP，当一个数据报到达目的端口时，该端口没在使用，它将产生一个ICMP端口不可达的信息

对于TCP，则使用复位

注意复位报文段中的seq和ack。因为ACK比特在到达的报文段中没有被设置为1，复位报文段中的序号被置为0，确认序号被置为进入的ISN加上数据字节数。

尽管在到达的报文段中没有真正的数据，但SYN比特从逻辑上占用了1字节的序号空间；

因此，在这个例子中复位报文段中确认序号被置为ISN与数据长度（0）、SYN比特所占的1的总和。

异常终止abortiverelease

发送一个复位报文段释放连接

优点：(1)丢弃任何待发数据并立即发送复位报文段；(2)RST的接收方会区分另一端执行的是异常关闭还是正常关闭。

应用程序使用的API必须提供产生异常关闭而不是正常关闭的手段。

一端发送一个RST而不是通常的FIN。

RST报文段中包含一个序号和确认序号。需要注意的是RST报文段不会导致另一端产生任何响应，另一端根本不进行确认。

收到RST的一方将终止该连接，并通知应用层连接复位。

另一端收到RST时，产生一个差错。这个差错正是我们所期待的：连接被对方复位了。

半打开Half-Open：

如果一方已经关闭或异常终止连接而另一方却还不知道，我们将这样的TCP连接称为半打开的。

只要不打算在半打开连接上传输数据，仍处于连接状态的一方就不会检测另一方已经出现异常。

产生原因： 主机异常、主机突然掉电而不是正常的结束客户应用程序后再关机。

务器服将永远不知道客户程序已经消失了。 用户重新启动客户程序，在服务器主机上会启动一个新的服务器程序，这样会导致服务器主机中产生许多半打开的TCP连接

能很容易地建立半打开连接

断开服务器主机与以太网的电缆，并重启服务器主机，模拟服务器主机出现异常（在重启服务器之前断开以太网电缆是为了防止它向打开的连接发送FIN，某些TCP在关机时会这么做）。

服务器主机重启后，我们重新接上电缆，并从客户向服务器发送另一行字符。

由于服务器的TCP已经重新启动，它将丢失复位前连接的所有信息，因此它不知道数据报文段中提到的连接。

处理原则：接收方以复位作为应答。

注意由于服务器主机经过重新启动，它的ARP高速缓存为空，因此需要一个ARP请求和应答。

客户收到复位报文段后显示连接已被另一端的主机终止(所以客户发送的信息没有什么价值)

同时打开simultaneous open

两个应用程序同时彼此执行主动打开：

每一方必须发送一个SYN，且这些SYN必须传递给对方。每一方使用一个对方熟知的端口作为本地端口

如主机A中的一个应用程序使用本地端口7777，并向主机B的端口8888执行主动打开，主机B中的应用程序则使用本地端口8888，并与主机A的端口7777执行主动打开。

对于同时打开要仅建立一条连接而不是两条连接(其他的协议族，最突出的是OSI运输层，在这种情况下将建立两条连接而不是一条连接)。

两端几乎在同时发送SYN，并进入SYNSENT状态。当每一端收到SYN时，状态变为SYNRCVD，同时它们都再发SYN并对收到的SYN进行确认，当双方都收到SYN及相应的ACK时，状态都变迁为ESTABLISHED。

一个同时打开的连接需要交换4个报文段，比正常的三次握手多一个。

每一端既是客户又是服务器。

同时关闭simultaneous close

双方都执行主动关闭

当应用层发出关闭命令时，两端均从ESTABLISHED变为FIN\_WAIT\_1。

双方各发送一个FIN，双方都收到FIN，状态由FIN\_WAIT\_1变迁到CLOSING，并发送最后的ACK。当收到最后的ACK时，状态变化为TIMEWAIT。

同时关闭与正常关闭使用的段交换数目相同。

TCP服务器的设计

大多数的TCP服务器进程是并发的。

当一个新的连接请求到达服务器时，服务器接受这个请求，并调用一个新进程来处理这个新的客户请求。

不同的操作系统使用不同的技术来调用新的服务器进程。在Unix系统下，常用的技术是使用fork函数来创建新的进程。如果系统支持，也可使用轻型进程，即线程（thread）。

1.TCP服务器端口号： 当一个服务器进程接受一来自客户进程的服务请求时是如何处理端口的？如果多个连接请求几乎同时到达会发生什么情况？

2.限定的本地IP地址：服务器不能任选其本地IP地址而必须使用特定的IP地址时的情况。

3.限定的远端IP地址：服务器有时会指定远端IP地址和远端端口，但大多数不这么做，服务器不指明远端插口，而等待连接请求的到来，然后检查客户端的IP地址和端口号。

4.呼入连接请求队列：

服务器在创建一个新的进程时，或操作系统正忙于处理优先级更高的进程时，到达多个连接请求。当服务器正处于忙时，TCP是如何处理这些呼入的连接请求？

在伯克利的TCP实现中采用以下规则：

1)连接队列：队列中的连接已被TCP接受(已完成三次握手)，但还没有被应用层接受。TCP接受一个连接是将其放入这个队列，而应用层接受连接是将其从该队列中移出。

2)积压值(backlog)：队列的最大长度，范围是0~5，应用层指定

3)最大连接数：允许接受连接的最大数目，积压值对系统所允许的最大连接数，或者并发服务器所能并发处理的客户数，并无影响

4)队列未满时，TCP接受新的SYN请求，但并不让应用层了解该连接源于何处（即不告知源IP地址和源端口）。 应用层只有在三次握手中的第三个报文段收到后才会知道这个新连接时。

这不是TCP所要求的，而只是共同的实现技术（如伯克利源代码通常都这么做）。

另外，当客户进程的主动打开成功但服务器的应用层还不知道这个新的连接时，它可能会认为服务器进程已经准备好接收数据了（如果发生这种情况，服务器的TCP仅将接收的数据放入缓冲队列)。

5)队列已满时，TCP不理会新的SYN请求，也不发回RST作为应答，因为这是一个软错误，而不是一个硬错误

如果应用层不能及时接受已被TCP接受的连接，这些连接可能占满整个连接队列，客户的主动打开最终将超时。

通常队列已满是由于应用程序或操作系统忙造成的，这样可防止应用程序对传入的连接进行服务。这个条件在一个很短的时间内可以改变。

但如果服务器的TCP以系统复位作为响应，客户进程的主动打开将被废弃（如果服务器程序没有启动我们就会遇到）。

由于不应答SYN，服务器程序迫使客户TCP随后重传SYN，以等待连接队列有空间接受新的连接。

如果一个API如TLI（见1.15节）向应用程序提供了解连接请求的到来的方法，并允许应用程序选择是否接受连接。当应用程序假定被告知连接请求已经到来时，

TCP的三次握手已经结束！其他运输层的实现可能将连接请求的到达与接受分开（如OSI的运输层），但TCP不是这样。

Solaris2.2提供了一个选项使TCP只有在应用程序说可以接受（tcpeagerlisteners见E.4），才允许接受传入的连接请求。

这种行为也意味着TCP服务器无法使客户进程的主动打开失效。

当一个新的客户连接传递给服务器的应用程序时，TCP的三次握手就结束了，客户的主动打开已经完全成功。

如果服务器的应用程序此时看到客户的IP地址和端口号，并决定是否为该客户进行服务，服务器所能做的就是关闭连接（发送FIN），或者复位连接（发送RST）。无论哪种情况，客户进程都认为一切正常，因为它的主动打开已经完成，并且已经向服务器程序发送过请求

\*/

## ICMP

ICMP是网络层协议，但ICMP报文封装在IP报文中

ICMP报文本身出错不再处理

|  |  |
| --- | --- |
| 终点不可达 |  |
| 源点抑制 |  |
| 超时问题 |  |
| 参数问题 |  |
| 改变路由 | 拥塞造成的报文丢失，主机或路由器向源点发送该报文，使源点知道应当把数据报的发送速率放慢 |

## IGMP

/\*

用于支持主机和路由器进行多播，它让一个物理网络上的所有系统知道主机当前所在的多播组。

多播路由器需要这些信息以便知道多播数据报应该向哪些接口转发。

加入一个多播组

多播的基础就是一个进程的概念（使用的术语进程是指操作系统执行的一个程序），该进程在一个主机的给定接口上加入了一个多播组。

在一个给定接口上的多播组中的成员是动态的—它随时因进程加入和离开多播组而变化。

这里所指的进程必须以某种方式在给定的接口上加入某个多播组。进程也能离开先前加入的多播组。

这些是一个支持多播主机中任何API所必需的部分。使用限定词“接口”是因为多播组中的成员是与接口相关联的。

一个进程可以在多个接口上加入同一多播组。

一个主机通过组地址和接口来识别一个多播组。主机必须保留一个表，此表中包含所有至少含有一个进程的多播组以及多播组中的进程数量。

实现细节

为改善该协议的效率，有许多实现的细节要考虑。

1.某主机首次发送IGMP报告(第一个进程加入一个多播组)，并不保证该报告被可靠接收(因为使用IP)，下一个报告将在0~10秒后发送。

2.主机收到IGMP\_Inquire，并不立即响应，而是经过一定的时间间隔后才发出一些响应

采用“响应”的复数形式是因为该主机必须对它参加的每个组均置为随机时延。在一个物理网络中的所有主机将收到同组其他主机发送的所有报告，

因为如图13-3所示的报告中的目的地址是那个组地址。这意味着如果一个主机在等待发送报告的过程中，

却收到了发自其他主机的相同报告，则该主机的响应就可以不必发送了。

因为多播路由器并不关心有多少主机属于该组，而只关心该组是否还至少拥有一个主机。

的确，一个多播路由器甚至不关心哪个主机属于一个多播组。它仅仅想知道在给定的接口上的多播组中是否还至少有一个主机。

在没有任何多播路由器的单个物理网络中，仅有的IGMP通信量就是在主机加入一个新的多播组时，支持IP多播的主机所发出的报告。

生存时间：IGMP报告和查询的IP数据报，TTL=1，

一个初始TTL为0的多播数据报将被限制在同一主机。在默认情况下，待传多播数据报的TTL被设置为1，这将使多播数据报仅局限在同一子网内传送，更大的TTL值能被多播路由器转发。

回顾6.2节，对发往一个多播地址的数据报从不会产生ICMP差错。当TTL值为0时，多播路由器也不产生ICMP“超时”差错。

在正常情况下，用户进程不关心传出数据报的TTL。然而，一个例外是Traceroute程序（第8章），它主要依据设置TTL值来完成。既然多播应用必须能够设置要传送数

据报的TTL值，这意味着程序设计接口必须为用户进程提供这种能力。

扩展环搜索：通过增加TTL值的方法，一个应用程序可实现对一个特定服务器的扩展环搜索

第一个多播数据报以TTL等于1发送。如果没有响应，就尝试将TTL设置为2，然后3，等等。

在这种方式下，该应用能找到以跳数来度量的最近的服务器。从224.0.0.0到224.0.0.255的特殊地址空间是打算用于多播范围不超过1跳的应用。

不管TTL值是多少，多播路由器均不转发目的地址为这些地址中的任何一个地址的数据报。

所有主机组

在图13-3中，我们看到了路由器的IGMP查询被送到目的IP地址224.0.0.1。

该地址被称为所有主机组地址。它涉及在一个物理网络中的所有具备多播能力的主机和路由器。

当接口初始化后，所有具备多播能力接口上的主机均自动加入这个多播组。

这个组的成员无需发送IGMP报告。\_\_

\*/

/\*

多播路由器使用IGMP报文来记录与该路由器相连网络中组成员的变化情况。使用规则如下：

1)当第一个进程加入一个组时，主机发送一个IGMP报告。如果一个主机的多个进程加入同一组，只发送一个IGMP报告。这个报告被发送到进程加入组所在的同一接口上。

2)进程离开一个组时，主机不发送IGMP报告，即便是组中的最后一个进程离开。主机知道在确定的组中已不再有组成员后，在随后收到的IGMP查询中就不再发送报告报文。

3)多播路由器定时发送IGMP查询来了解是否还有任何主机包含有属于多播组的进程。多播路由器必须向每个接口发送一个IGMP查询。因为路由器希望主机对它加入的每个多播组均发回一个报告，因此IGMP查询报文中的组地址被设置为0。

4)主机通过发送IGMP报告来响应一个IGMP查询，对每个至少还包含一个进程的组均要发回IGMP报告。

使用这些查询和报告报文，多播路由器对每个接口保持一个表，表中记录接口上至少还

包含一个主机的多播组。当路由器收到要转发的多播数据报时，它只将该数据报转发到（使

用相应的多播链路层地址）还拥有属于那个组主机的接口上。

图13-3显示了两个IGMP报文，一个是主机发送的报告，另一个是路由器发送的查询。该

路由器正在要求那个接口上的每个主机说明它加入的每个多播组

\*/

void IGMP\_Inquire()

{

/\*

1，【0001：版本】 【0001：类型→多播路由器发出的查询报文】

2，00

3~4，检验和

5~8，00 00 00 00

\*/

}

void IGMP\_Report()

{

/\*

1，【0001：版本】 【0002：类型→主机发出的报告报文】

2，00

3~4，检验和

5~8，组地址，D类I P地址

\*/

}

# 应用层

## 模型

C/S模型：

服务器：总是处于打开状态，接收请求的状态

服务器运行专门提供某种服务的程序，可同时处理多个远程或本地客户机的要求

服务器系统启动后，自动调用并一直运行，被动地等待并接受来自各个客户的请求

服务器收到客户机的请求，分析请求，进行处理，得到结果发送给客户机

客户机需要知道服务器的地址

网络中的各计算机地位不平等，服务器可以铜鼓对用户权限的限制来达到管理客户机的目的，使他们不能随意存储/删除数据，或进行其他受限的网络活动

整个网络的管理工作由少数的服务器担当，古网络的管理集中和方便

客户机相互之间不直接通信，如web应用中的两个浏览器之间不直接通信

可扩展性不佳，收服务器硬件和网络带宽的限制，一个服务器支持的客户机有限

P2P模型：

各主机没有固定的客户机/服务器的划分，任意一对主机，可以直接相互通信，即可以作为客户端也可以作为服务器

网络中的各计算机地位平等

## DNS

|  |  |  |
| --- | --- | --- |
| 客户端 |  | |
| 服务端 | port=53 | |
| 服务器 | 根域名~ | 存储所有顶级域名， 遍布世界各地，分为13组，同组使用相同域名：<a~m>.rootservers.net |
| 顶级域名~ | 存储在该顶级域名下注册的所有二级域名，每个顶级域名对应一组顶级域名服务器如cn域名服务器、org域名服务器、com域名服务器 |
| 权限域名~ | 存储管辖区中所有主机的域名  某单位申请得到一个域名abc.cn，可以将它的域再分成多个区，如分成abc.cn、x.abc.cn、y.abc.cn  需要为每一个区配置一个权限域名服务器  一个域名服务器负责一个或多个区域 |
| 主、辅域名~ | 一个区域的管理者必须为该区域提供一个主域名服务器和至少一个辅助域名服务器。  主、辅名字服务器必须是独立和冗余的，以便当某个名字服务器发生故障时不会影响该区域的名字服务。  主、辅名字服务器的主要区别在于主名字服务器从磁盘文件中调入该区域的所有信息，而辅名字服务器则从主服务器调入所有信息。  我们将辅名字服务器从主服务器调入信息称为区域传送。 |
| 高速缓存 | 存放最近查询过的域名以及相应的记录 ，以后若遇到相同的映射请求，就直接使用缓存中的结果而无需通过其他服务器查询 |
| UDP |  | |
| IP |  | |

|  |  |  |
| --- | --- | --- |
| 域名 |  | 代替IP地址用于访问主机 |
| 格式 | ASCII码，最多63个字符；  ……<三级域名>.<二级域名>.<顶级域名>  顶级域名：arpa；com、edu、gov、int、mil、net、org；cn、us、uk、hk… |
| 分配 | cn由中国管理，中国分配出一个二级域edu.cn，授权给中国教育科研计算机网CERNET管理，CERNET分配出一个三级域xjtu.edu.cn，授权给西安交通大学管理 |
|  |  | |
| 域名解析 | 域名→IP地址  用户主机要进行域名解析时，只需将目标域名发送给最近的本地域名服务器  本地域名服务器与上述三种域名服务器无关， 每个几个路由器的范围就会设置一个本地域名服务器  目标域名必定是在某个权限域名服务器内，所以要找到该权限域名服务器 | |
| 递归查询 | 若本地服务器中有B的域名，则返回B的IP地址给A，若没有，本地域名服务器将B的域名放入DNS请求报文中 | |
| 迭代查询 | 本地服务器作为客户机：DNS请求报文→最近的根域名服务器  根域名服务器根据收到的DNS报文中包含的域名的顶级域名，将该顶级域名对应的顶级域名服务器(最近)IP地址发送给本地域名服务器  本地服务器作为客户机：DNS请求报文→顶级域名服务器  顶级域名服务器根据收到的DNS报文中包含的域名的二级域名，将该二级域名对应的权限域名服务器的IP地址发送给本地域名服务器  本地服务器作为客户机：DNS请求报文→权限域名服务器  权限域名服务器根据收到的DNS报文中包含的域名：  1.该域名没有三级域名，那么必定是在该权限域名服务器  2.该域名包含三级域名，那么是在子权限域名服务器中 | |

主机A访问主机B，A将B的域名放入DNS请求报文中

主机A作为客户机：DNS请求报文→本地服务器

1)Rlogin客户程序启动后，调用它的名字解析器函数将用户键入的域名转换为一个IP地址。一个A类型的查询请求被送往一个根服务器。

2)由根服务器返回的响应中包含为该服务器所在域服务的名字服务器名。

3)客户端的名字解析器将向该服务器的名字服务器重发上述A类型查询，这个查询通常是将期望递归标志设置为1。

4)返回的应答中包含Rlogin服务器的IP地址。

5)Rlogin客户和Rlogin服务器建立一个TCP连接（第18章将提供该步骤的细节）。客户和服务器的TCP模块间将交换3个分组。

6)Rlogin服务器收到来自客户的连接请求后，调用它的名字解析器通过TCP连接请求中的IP地址获得客户主机名。

这是一个PTR查询请求，由一个根名字服务器处理。这个根名字服务器可以不同于步骤1中客户使用的根名字服务器。

7)这个根名字服务器的响应中含有为客户的in-addr.arpa域的名字服务器。

8)服务器上的名字解析器将向客户的名字服务器重传上述PTR查询。

9)返回的PTR应答中含有客户主机的FQDN。

10)服务器的名字解析器向客户的名字服务器发送一个A类型查询请求，查找前一步返回的名字对应的IP地址。

这可能由服务器中的gethostbyaddr函数自动完成，正如我们在14.5节中介绍的那样，否则Rlogin服务器将完成这一步。

此外，客户的名字服务器常常就是客户的in-addr.arpa名字服务器，但这不是必需的。

11)从客户的名字服务器返回的响应含有客户主机的A记录。Rlogin服务器将客户的TCP连接请求中的IP地址与A记录作比较。

高速缓存将减少这个图中交换的分组数目。

|  |  |  |
| --- | --- | --- |
| 多个IP  对应  一个域名 | 可靠工作 | 同一个服务点配置多台，一台主，多台辅，主服务器故障时，辅助服务器顶上 |
| 扩大分布 | 根服务器的13个域名要供全世界使用，每个域名对应多个服务器，分配到世界各地，使各地用户能快速访问到目标域名 |
| 负载均衡 | 访问量巨大的服务器，分成多台，每台内容相同，来缓解访问压力；IP地址不同，但使用相同的域名 |
| 双线接入 | 一个主机只是用两块网卡接到两个网络，具有两个IP地址，但他们映射到同一个域名(因为只有一台服务器) |
| 多个域名  对应  一个IP |  |  |

## FTP

|  |  |  |
| --- | --- | --- |
| 客户端 | 控制进程  数据传输进程  用户界面进程 | |
| 服务端 | control-port=21，data-port=20  一个主进程：负责接收新的请求  若干从进程：处理单个请求  控制从进程  数据传送从进程 | |
| TCP | 两个的TCP连接**并行工作**  控制连接用于传输各种FTP命令，数据连接用于文件传送。  两条独立的连接可以使FTP更加简单，高效  在文件传输中，可以利用控制连接控制传输，如客户可以请求终止、暂停等 | |
| 控制链接 | 建立方式：服务器以被动方式打开用于FTP的端口，等待客户的连接。客户则以主动方式打开TCP端口21，来建立连接。  传输内容：连接将命令从客户传给服务器，并传回服务器的应答，  持续时间：一直保持到客户-服务器连接的全过程，始终等待客户与服务器之间的通信  控制信息是带外传送的 |
| 数据连接 | 建立方式：客户端选择一个临时端口号，从该端口发布一个被动的打开，使用PORT命令从控制连接上把端口号发向服务器。  服务器在控制连接上接收端口号，并向客户端发布一个主动的打开。  传输内容：文件(服务器→客户端、客户端→服务器)、目录列表(服务器→客户端)。  持续时间：一次数据连接的建立只传输一个文件或一个目录列表，要为每一个文件或目录列表建立一个新的数据连接  通常服务器也执行数据连接的主动关闭，除非当客户向服务器发送流形式的文件时，需要客户来关闭连接（它给服务器一个文件结束的通知）。 |
| IP | 控制连接：最大限度地减小迟延，由于命令通常是由用户键入的  数据连接：最大限度提高吞吐量 | |

传输

提供交互式访问，允许客户指明文件的类型和格式，允许文件具有存取权限

FTP用于文件传输，注意分清文件传送( file transfer)和文件存取(file access) 之间的区别，前者FTP提供，后者是NFS等应用系统提供

FTP将一个完整的文件从一个系统复制到另一个系统中。

使用FTP，需要有登录服务器的注册帐号，或者通过允许匿名FTP的服务器来使用

与Telne t类似，FTP最早用于可能运行在不同的操作系统下、使用不同的文件结构、并可能使用不同字符集的两台主机

与Telnet不同，Telnet获得异构性是强制两端都采用同一个标准：使用7比特ASCII码的NVT。FTP是采用另一种方法来处理不同系统间的差异。

FTP支持有限数量的文件类型（ASCII，二进制，等等）和文件结构（面向字节流或记录）。

|  |  |  |
| --- | --- | --- |
| 方向 | TCP | 报文 |
| / | / |  |
| 客户→服务器 | 控制连接 | TCP连接请求 |
|  |  |  |

|  |
| --- |
| 服务器打开主控制进程，等待客户端进程的连接请求 |
| 客户通过控制连接发送FTP请求，给服务器的控制进程 |
| 服务器收到FTP客户机发来的FTP请求后，创建数据传送进程和数据连接 |
| 启动从属进程处理客户进程发来的请求，从属进程对客户进程的请求处理完毕后即终止  但从属进程在运行期间根据需要还可能创建其他一些子进程 |
| 数据传送完毕，关闭数据连接  关闭控制连接  4.回到等待状态 |

主机下载文件：

建立控制连接，客户发送登录账号和密码

服务器返回登录成功的信息

客户打开一个随机端口x，讲端口号发送给服务器

客户发送读取文件命令，内容为get file，服务器用端口20建立一个TCP连接到客户机的端口x

服务器把文件内容通过第二个连接发送给主机，传输完毕关闭数据连接

控制连接在整个会话期间一直保持打开

传输时，客户端和服务器之间建立两个TCP链接，

控制连接：整个会话期间保持打开，

数据连接：传输文件，传送完成后，连接关闭

匿名FTP：anonymous

ftp://<文档所在主机的IP地址对应的域名>:20/文件路径

修改服务器上的文件：先将文件传送到本地主机，修改后，再发送到原服务器

Unix实现的FTP：

文件类型：ASCII文件、二进制图像文件

格式：非打印→文件中不含有垂直格式信息

结构：文件→连续的字节流，不存在内部的文件结构。

传输方式：流方式→字节流，传输文件时在文件尾提示关闭数据连接，传输记录时有专用的两字节序列码标志记录结束和文件结束。

非Unix的实现提供了处理它们自己文件格式的FTP功能。

\*/

## SMTP、POP3

传输

邮箱地址：用户名@邮箱所在主机的域名

邮件服务器：发送和接收邮件，同时还要向发送方报告邮件传送情况

邮件发送协议SMTP：

邮件读取协议POP3：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 客户端 | 用户主机 | 用户代理进程(UA)  (电子邮件系统程序)  Outlook、 Foxmail、Thunderbird |  | |
| 服务端 | 发送方  邮件服务器 | 服务进程 | 25 |  |
| TCP |  |  |  | |
| IP |  |  |  | |

|  |  |  |
| --- | --- | --- |
| 客户端 | 发送方邮件服务器  SMTP客户进程 | |
| 服务端 | 服务器 | 接收方邮件服务器 |
| 服务进程 | port=25 |
|  |  |
| TCP | SMTP客户进程发现在邮件缓存中有待发送的邮件， 就向运行在接收方邮件服务器的SMTP服务器进程发起建立TCP连接  4.TCP连接建立后,SMTP客户进程开始向远程的SMTP服务器进程发送邮件  当所有的待发送邮件发送完，SMTP就关闭所建立的TCP连接  5.运行在接收方邮件服务器中的SMTP服务器进程收到邮件后，将邮件放入收信人的用户邮箱中  6.收信人调用用户代理，使用POP3或IMAP协议将自己的邮件从接收方邮件服务器的用户邮箱中取回 | |
| IP |  | |

发信人调用UA，编辑邮件内容

1.发信主机运行用户代理进程，作为SMTP客户端， 发送邮件给

3.

MIME：对现有邮件主体的结构 进行扩充

STMP进程：port=25

1.连接建立：

发件人的邮件发送到发送方邮件服务器的邮件缓存后，SMTP客户每隔一定时间对邮件缓存扫描一次

如发现有邮件，就是用SMTP端口号25，与接收方邮件服务器的SMTP服务器建立TCP连接，连接建立后，接收方SMTP服务器发送220 Service ready

邮件服务器A(SMTP客户机)→邮件服务器B(SMTP服务器)：HELO <发信人主机名>

SMTP不使用中间的邮件服务器，TCP连接总是在发送方和接收方这两个邮件服务器之间直接建立，不管相隔多远

当接收方的邮件服务器因故障暂时不能建立连接时，发送方的邮件服务器只能等待一段时间后再尝试连接

2.邮件传送

连接建立后传送邮件：

C→S：MAIL FROM:<abc@hust.edu.cn>

S→C：250 OK //SMTP服务器已准备接收

C→S：RCPT TO:<收件人地址> //RCPT命令作用：先弄清接收方系统是否已做好接收邮件的准备，然后才发送邮件，不至于发送了很长的邮件以后才知道是地址错误

S→C：250 OK

C→S：RCPT TO:<收件人地址>

S→C：550 No such user here

C→S：DATA //客户机获得OK的回答后，使用DATA命令，表示要开始传输邮件的内容了

S→C：354 Start mail input;end with <CRLF>.<CRLF> //<CRLF>是"回车换行"的意思 ，<CRLF>.<CRLF>表示邮件内容的结束

……

S→C：OK

C→S：QUIT //邮件发送完毕，连接释放

S→C：221 //同意释放TCP连接

SMTP只支持传输7bit的ASCII码内容

支持在邮件服务器之间发送邮件

支持从用户代理向邮件服务器发送邮件

电子邮件的内容包括邮件头与邮件体两部分

邮件体就是实际要传送的信函内容

MIME协议允许电子邮件系统传输文字、图像、语音、视频等多种信息

SMTP邮件系统是基于因特网的

目前使用的较多的是基于www的电子邮件：Hotmail、Gmail等 ，使用HTTP

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 客户端 | 用户主机 | POP客户进程 |  | |
| 服务端 | 接收方  邮件服务器 | POP服务进程 | 110 |  |
| TCP |  |  |  | |
| IP |  |  |  | |
| 传输 | 下载并保留模式 | 主机从邮件服务器上读取了邮件后，邮件依然保存在邮件服务器上，用户下次可以再次从服务器上读取该邮件 | | |
| 下载并删除模式 | 邮件一旦被读取之后，就从邮件服务器上删除，用户不能再次从服务器上读取 | | |

POP进程，port=110

传输层使用TCP

接收方主机的用户代理必须运行，在接收方邮件服务器上运行POP服务进程

由客户端选择接收后是否将邮件保存在服务器上，而不是服务器决定

POP协议基于ASCII，不能发送二进制数据

## HTTP

|  |  |
| --- | --- |
| 客户端 | 用户主机，浏览器 |
| 服务端 | 服务进程，port=80或443(带安全协议，https) |
| 服务器 | 存储互联网资源 |
| TCP |  |
| IP |  |

1.一台主机运行一个HTTP服务器程序。

2.一台主机运行多个HTTP服务器程序.

单个主机为多个组织提供HTTP服务。每一个组织的域名映射一个不同的I P地址(都在同一子网上)，

单个以太网接口分别对每一个不同的I P地址赋予别名

第6 . 6节中描述了Net / 3怎样允许单个网络接口上的多个I P地址。在主IP地址之后指派给网络接口的IP地址均称为别名)。

22个服务器实例中的每一个都只使用一个I P地址。当服务器程序启动时，它把本地的I P地址绑定到它的监听T C P插口上，

因此它只收到那些目的地址是它的I P地址的连接。

3.多台主机，每台均提供服务器程序的一个副本。用于繁忙的组织在多个主机上分布输入负载(即负载平衡)。

对应组织的域名指派多个I P地址，每一个提供H T T P服务的主机有不同的I P地址(卷1的第1 4章，D N S中的多条A记录)。

这种组织的D N S服务器响应D N S客户请求时，必须能以不同的顺序返回多个不同的I P地址。

D N S中把这个称为循环使用( r o u n d - r o b i n )，例如，在通常的D N S服务器程序当前版本中均支持这种功能。

例如，N C S A提供9个H T T P服务器。我们第一次查询它们的域名服务器时，返回如下

www文档： 互联网资源，文件目录、文件、文档、图像、声音等

Web客户端：某台主机上运行的浏览器程序

URL：http://<服务器域名>:80/<路径> ，页面地址，路径为空表示该服务器的主页

Web客户端与Web服务器使用一个或多个T C P连接进行通信。

Web服务器之间可以通过超文本链接传输数据

这些链接并不是只可以指向Web服务器，还可以是其他类型的服务器，例如：一台FTP或是Telnet服务器。

每一个次元对应一个URL

Web客户端建立一条同Web服务进程的TCP连接，然后发出请求并读取服务器进程的响应。

服务进程关闭连接表示本次响应结束。服务进程返回的文件通常含有指向其他服务器上文件的指针(超文本链接)。

用户显然可以很轻松地沿着这些链接从一个服务器到下一个服务器。

浏览器提供简单、漂亮的图形界面。服务进程只是简单返回客户进程所请求的文档，因此H T T P服务器软件比H T T P客户软件要小得多。

例如，N C S A版本1 . 3的U n i x服务器由大约6 500行C代码写成，而X Wi n d o w环境下的U n i x，Mosaic 2.5浏览器有约80 000行C代码。

使用URL并非仅仅能够访问www页面而且还能通过URL使用其他的互联网应用程序，如FTP或USENET等

更重要的是，用户在使用这些应用程序时，只是用一个程序，即浏览器，这是很方便的

http定义了浏览器(即万维网客户端进程)怎样想万维网服务器请求www文档，以及服务器怎样把文档传给浏览器

HTTP不仅传送完成超文本跳转所必需的信息，而且也传送任何从互联网上得到的信息，如文本、超文本、声音、图像等

每个，不断监听TCP端口80，一遍发现是否有浏览器(即www客户)向它发出连接请求

一旦坚挺到连接建立请求并建立了TCP连接后，浏览器就想www服务器发出浏览某个页面的请求，服务器就返回请求的页面作为回应

浏览器和服务器之间的请求和响应的交互，必须按照规定的格式和遵循一定的规则，这些规定和规则就是HTTP协议

HTTP协议：

客户端与服务器至今啊的每次交互，都有一个ASCII码串构成的请求和一个类似的通用互联网扩充

即：MIME-like 响应组成，

用户浏览页面的方法：在浏览器输入URL， 或页面中的超链接 ，但两种方法的本质是一样的，都是

应用层协议

请求/响应

无状态：同一个客户第二次访问同一个服务器上的页面时，服务器的响应与第一次相同，因为服务器不会记住之前访问的客户

双向传输

能力协商

高速缓存

代理服务器

GET http://www.xyz.edu.cn/dir/index.htm HTTP/1.1 //请求读取URL

Host:www.xyz.edu.cn //客户主机域名

Connection:close //告诉服务器发送完请求的文档就可释放连接，keep-alive

User-Agent:Mozilla/5.0 //浏览器

Accept-Language:cn //优先语言版本

空行

H T T P连接通常由服务器关闭，服务器经过T I M E WA I T时延后关闭连接，导致在繁忙的服务器上许多控制块停留在该状态。

建议客户与服务器保持一个打开的T C P连接，而不是服务器在发出响应后关闭连接。当服务器知道生成的响应报文的长度时才可以这样做，

回想前面1 3 . 3 . 4节中我们提到的例子， C o n t e n t - L e n g t h报文首部中指出G I F图象的大小。

否则，服务器必须通过关闭连接来为客户程序指出响应的结尾。对协议作这样的修改必须同时修改客户端和服务器端。

客户端规定 Pragma: hold-connecti报on文首部，提供向后

兼容的能力。如果服务器不能识别这种P r a g m a，就会忽略它，然后在发送完响应后关闭连接。

这种P r a g m a允许新客户程序在尽可能情况下保持连接，同时访问新的服务器，还允许现有所

有客户和服务器交互操作。

H T P P协议的下一版本(版本1 . 1 )中可能会支持持续的连接

在最后一个响应报文首部的后面，服务器程序紧跟着图象后发送了一个空行(一个回车/换行对)

因为H T T P协议交换8 bit字节的数据，所以可以通过T C P连接发送二进制数据。

这点不同于其他的I n t e r n e t应用，特别是S M T P协议(卷1的第2 8章)，它通过T C P连接传输7 bit的A S C I I

字符，显式地将每字节的高位设置为0，阻止了二进制数据的交换。

第一条TCP连接是访问主页(GET/)，主页的HTML文档共涉及了7个GIF图象。

客户程序收到这个主页后，马上并行地打开4条T C P连接去获取前4个GIF图象。这是Netscape程序为了减少打开主页总时间的一种方法(

大多数Web客户程序并不像这样，而是只能一次下载一个图象。

并行连接数量可由用户来配置，默认是4个。当这些连接中有一条结束时，客户程序会立即打开一条新的连接来获取下一个图象，直到客户程序取得全部7个图 象。

8条TCP连接都由客户端程序发起，依次使用1114~1 121的8个客户端口号。

8条TCP连接都由服务器程序关闭。

把客户程序发送最初的SYN 看作连接的开始，客户程序收到服务器程序的FIN后发送FIN 认为是连接的结束

取得这个主页以及它所涉及的所有7个图象共需要约12秒的时间。

下一章的图1 4 - 2 2中给出了由客户程序发起的第一条连接(端口1 11 4 )的Tc p d u m p分组跟踪情况。

注意，端口号为1115，111 6，111 7的三条连接是在第一条连接(端口号为111 4 )结束之前建立的，

这是因为N e t s c a p e的客户程序在读到第一条连接上的文件结束标志以后，并在关闭第一条连接之前发起三条无阻塞的连接。

实际上，客户程序在收到FIN标志后约半秒钟才发出FIN分组同时使用多条T C P连接是否真的能减少交互式用户所需的处理时间呢？

为了测试这一点，我们在主机s u n 上运行N e t s c a p e客户程序(图1 - 1 3 )，还是来获取A d d i s o n - We s l e y的主页。

但这台主机是采用如今常用的方式连接I n t e r n e t，即通过拔号调制解调器以28.8 Kb/s的速度连接I n t e r n e t。

修改客户程序的首选文件，对客户程序最大的连接数从1至7都进行了测试。测试时关闭了客户程序的硬盘缓存功能。

在每一种最大连接数下客户程序均运行三次，取结果的平均值。

结果可以看出，从1到4，随着连接数增加，总时间在减少

但是如果用Tc p d u m p来跟踪这种交换，我们会发现，虽然用户可能把连接数设成超过4，但是程序的极限是4。

不管怎么说，超过4条连接后增加连接数对总时间即便有影响也是很小，不如从1 ~ 2，2 ~ 3，3 ~ 4那么明显。

图1 3 - 5所示的总时间比图1 3 - 6所示的最短时间( 1 0 . 2秒)要多约2秒，这是因为客户主机的显示硬件速度有差异。

图1 3 - 6所示的测试是客户程序运行在一台工作站上，而图1 3 - 5所示的测试客户程序运行在一台显示速度和运行速度均较慢的个人计算机上。

[Padmanabhan 1995]指出了多连接方法的两个问题：

1.这样做对其他协议不公平。例如，FTP协议获取多个文件时每次只能使用一条连接(不包括控制连接)。

2.当在一条连接上遇到拥塞并执行拥塞避免(在卷1的第2 1 . 6节中有描述)时，拥塞避免信息不会传递到其他连接上去。

对客户程序来说，同时对同一主机使用多条连接实际上使用的可能是同一条路径。

如果处于瓶颈的路由器因为拥塞而丢弃某条连接的分组，那么其他连接的分组通过该路由器时也同样可能会被丢弃。

客户程序同时使用多个连接带来的另一个问题是容易造成服务器程序未完成的连接队列溢出，

这样会使得客户主机重传它的S Y N分组而造成较大的时延。下一章我们讨论We b服务器时，

将在1 4 . 5节中详细讨论服务器程序的未完成连接队列

## DHCP

自动获得IP信息而无需手工干预

|  |  |
| --- | --- |
| 客户端 |  |
| 服务端 |  |
| 服务器 |  |
| UDP |  |
| IP |  |

## SNMP

|  |  |
| --- | --- |
| 客户端 |  |
| 服务端 |  |
| 服务器 |  |
| UDP |  |
| IP |  |

## TELNET

|  |  |
| --- | --- |
| 客户端 |  |
| 服务端 |  |
| 服务器 |  |
| UDP |  |
| IP |  |