

Computer Networking

A Top-Down Approach Sixth Edition

感觉学硬件从今年 2 月份到今天 3 月 6 号，满脑子都是操作系统，后面想深究一下网络基础和网络协议，毕竟后面是做关于网络当面的全栈工程师，估计基础部分要花一个月的时间。

一、计算机网络和因特网

1.1 什么是因特网

两种描述方式：

- 具体构成描述；
根据他的硬件和软件组件来描述。
- 服务描述；
跟据基础设施向分布式应用程序提供的服务来描述。

1) 主机 host/端系统 end system

端系统通过通信链路 communication link 和分组交换机 packet switch 连接到一起。通信链路 communication link 是由不同类型的物理媒体组成的，包括同轴电缆，铜线，光纤和无线电频谱。

端系统通过因特网服务提供商 Internet Service Provider, ISP 接入因特网，包括本地电缆或电话公司。

主机有时候有进一步划分为两类：客户 client 和服务器 server。客户机非正式地等同于桌面 PC，移动 PC 和智能手机等，而服务器非正式地等同于更为强大的及其，用于存储和发布 Web 页面，流视频，中继电子邮件。

2) 传输速率

传输速率的单位是比特/秒(bit/s, 或者 bps)

3) 分组 packet

当一台端系统向另外一台端系统传输数据的时候，发送端系统将数据分段，并为每段加上首部字节，由此形成的信息包称为分组。

4) 分组交换机

分组交换机从它的一条入通信链路接收到的分组，并从它大的一条出通信链路转发该分组。两种比较比较著名的类型是：

- 路由器 router

通常用于网络核心中。有一个特别的例子是边缘路由器，指的是端系统到任何其他远程端系统的路径上的第一台路由器。

- 链路层交换机 link-layer switch
通常用于接入网中
- 5) 路径
从发送端系统到接收端系统，一个分组所经历的一系列通信链路和分组交换机称为通过该网路的路径 route/path。
- 6) 协议 protocol
端系统，分组交换机和其他因特网部件都要运行一系列协议，这些协议控制因特网中信息的接收和发送。因特网的主要协议是 TCP/IP
TCP Transmission Control Protocol, 传输控制协议
IP Internet Protocol 网络协议
- 7) 分布式应用程序 distributed application
涉及到多台相互交换数据的端系统。
- 8) 应用程序编程接口 Application Programming Interface, API
该 API 规定了运行在一个端系统上的软件请求因特网基础设施向运行在另一个端系统尚的特定目的地软件交付数据的方式。
因特网 API 是一套发送软件必须遵循的规则集合，因此因特网能够将数据交付给目的地。
- 9) 协议
一个协议定义了在两个或者多个通信实体之间交换的报文格式和次序，以及报文发送/或者接收一条保温或其他时间所采取的动作；
其实在我看来关键在于“应答”，没有回应说明通信失败。
- 10) DSLAM
DSLAM 是 Digital Subscriber Line Access Multiplexer 的简称，中文称呼数字用户线路接入复用器。DSLAM 是各种 DSL 系统的局端设备，属于最后一公里接入设备 (the last mile)，其功能是接纳所有的 DSL 线路，汇聚流量，相当于一个二层交换机。其在 ADSL 系统中的位置如图所示。

1.2 网络边缘

- 1) 一个协议定义了在两个或者多个通信实体之间交换的报文格式和次序，以及报文发送/或者接收一条保温或其他时间所采取的动作；
其实在我看来关键在于“应答”，没有回应说明通信失败。
- 2) 主机/端系统
- 3) 接入网 access network
指将端系统连接到其边缘路由器的物理链路
- 4) 接入网的方式
 - 家庭接入：DSL，电缆，FTTH，拨号和卫星
宽带接入住宅最流行的两种类型：数字用户线 Digital Subscriber Line, DSL 和电缆。
 - 企业（和家庭）接入：以太网和 WiFi
 - 广域网无线接入：3G 和 LTE
- 5) 数字用户线 Digital Subscriber Line, DSL
家庭电话线同时承载了数据和传统的电话信号，他们编码为不同的频率：

- 高速下行信道，位于 50kHz 到 1MHz 频段；
- 中速上行信道，位于 4kHz 到 50kHz 频段；
- 普通的双向电话线，位于 0 到 4kHz 频段；

这种方法看起来有三个单独的线路，其实只要一个双绞铜线的电话线，关键在于电话公司的复用器 DSLAM 和家庭的 DSL 调制解调器，采用的正是复用技术：

在用户端，一个分频器把到达家庭的数据信号和电话信号分割开来，并将数据信号转发给 DSL 调制解调器。在本地电话公司一侧，DSLAM 把数据和电话信号分隔开，并将数据送往因特网中。

DSL 标准定义了 12Mbps 下行和 1.8Mbps 上行传输速率，以及 24Mbps 下行和 2.5Mbps 上行传输速率。因为这些下行和上行传输速率的速率是不同的，所以这种接入被称为“不对称的”。

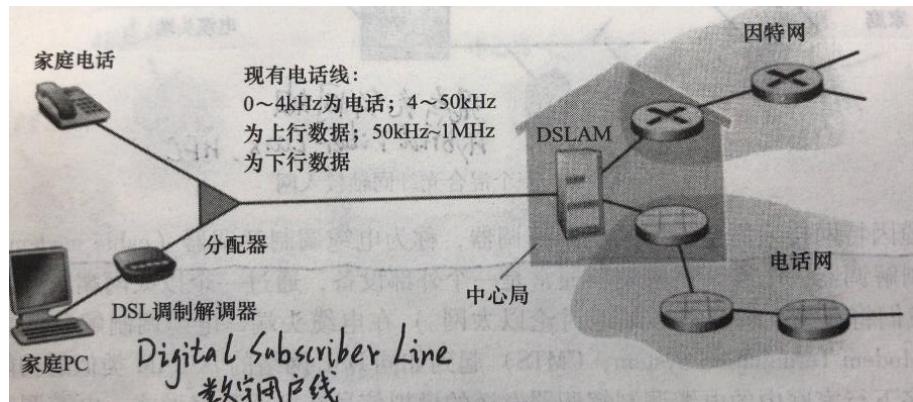


图 1-1 DSL 用户接入

6) 电缆

电缆因特网接入需要特殊的调制解调器，称为电缆调制解调器 cable modem。在电缆头端 Cable Modem Termination System, CMTS，起到 DSL 网络的 DSLAM 类似的功能，即将来自许多下行家庭中的电缆调制解调器发送的模拟信号转换为数字信号。

电缆因特网接入的一个重要特征是共享广播媒体，特别是从头端发送的每个分组向下行经每段链路到每个家庭；每个家庭发送的每个分组经上行信道向头端传输。因此，如果几个用户同时经下行信道下载一个视频文件，每个用户接收的视频文件的实际速率将大大低于电缆总计的下行速率。当然了，如果同一时段只有少些用户在上网的话，那么他们将独占全部网速。为了避免碰撞，需要一个分布式多路访问协议来协调。

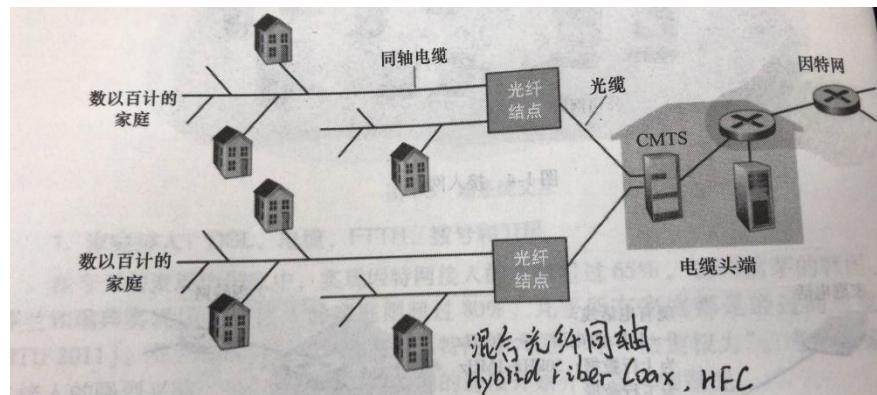


图 1-2 HFC 接入网接入

7) 光纤到户 Fiber To The Home, FTTH

直接从本地中心局拉一条光纤到您们家，但是实际上是到你们家附近然后分发一条给大家。分配的方案主要有两种：

- 主动光纤网络 Active Optical Network, AON
- 被动光纤网络 Passive Optical Network, PON

在 PON 的体系中，所有的从 OLT 发送到光纤分配器的分组都会在分配器处复制。

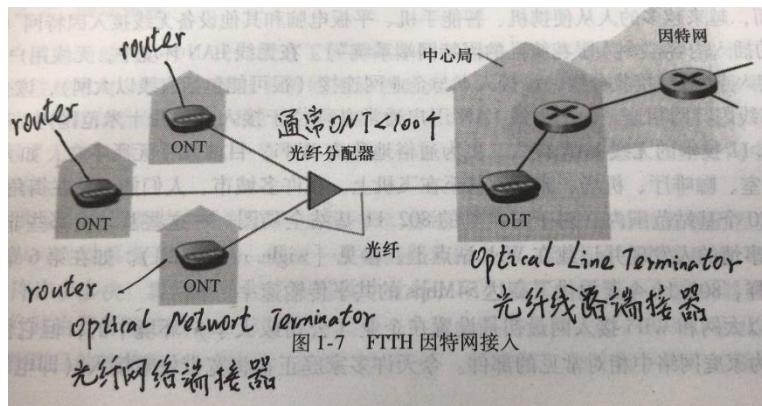


图 1-3 光纤接入网

8) 以太网接入

以太网是目前为止最为流行的接入方式

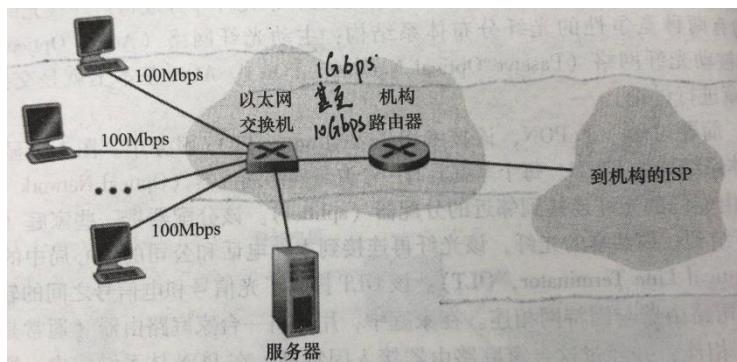


图 1-4 以太网接入网

1.3 物理媒介

1) 物理媒介分为导引型媒介 guided media 和非导引型媒介 unguided media

- 导引型媒介 guided media
双绞铜线，同轴电缆和光纤，成本比较低
- 非导引型媒介 unguided media
空气或其他外层空间

2) 双绞铜线

由两根隔离的铜线经过螺线形式排列着，以减少相邻铜线的电气干扰。一对铜线作为一个通信链路。许多对双绞铜线行程一根电缆。传输的速率取决于双绞铜线的粗细和发送方和接收方之间的距离长短。

3) 同轴电缆

由两根同心的铜线组成，中间有绝缘体。在电视系统中应用比较广泛。

4) 光纤

一个脉冲表示一个比特，传输速率极高，衰减率很低，而且不受电磁干扰，很难窃听，一般用作因特网的主干。

5) 陆地无线电通道和卫星无线电通道

1.4 网络核心

1) 分组交换

彼此交换的是报文 message，源将报文划分为一个个数据块，称为分组 packet。每个分组通过通信链路和分组交换机 packet switch（主要有路由器和链路层交换机）传送。

2) 存储转发传输

存储转发机制：分组交换机在发出分组第一个比特之前，必须接受完整个分组。

考虑由 N 条速率均为 R 的链路组成的路径（源和目的之间有 N-1 个分组交换机），一个分组共有 L 个比特，则端到端的时延为

$$d_{\text{端到端}} = N \frac{L}{R}$$

3) 排队时延和分组丢失

分组交换机都有一个用于存储分组的缓存，称为输出缓存 Output Buffer。如果分组比较多，网络比较拥挤，就会产生“排队时延”。更有甚者，假如分组交换机来不及发送分组，输出缓存就已经爆掉了，这时候在进入的分组就会丢失，这种现象称为“分组丢失”。

4) 转发表和路由选择协议

在因特网中，每个端系统都有一个 IP 地址，分组的头部包含目的地址的 IP，路由器内部有一个转发表 forward table，可以将 IP 地址或者地址的一部分映射成输出链路。

至于转发表怎么设置的，这由因特网中的路由选择协议 routing protocol 自动设置的。

5) 电路交换

跟分组交换网络不同的是，电路交换网络的资源（链路传输速率，缓存）是预留的，比如传统的电话网络。而分组交换网络是按需分配的。

➤ 频分复用 Frequency-Division Multiplexing, FDM

➤ 时分复用 Time-Division Multiplexing, TDM

每个间隙都是发送方与接收方一一对应的。

时分复用中每条链路的速率 = 帧速率 × 一个间隙中的比特数量

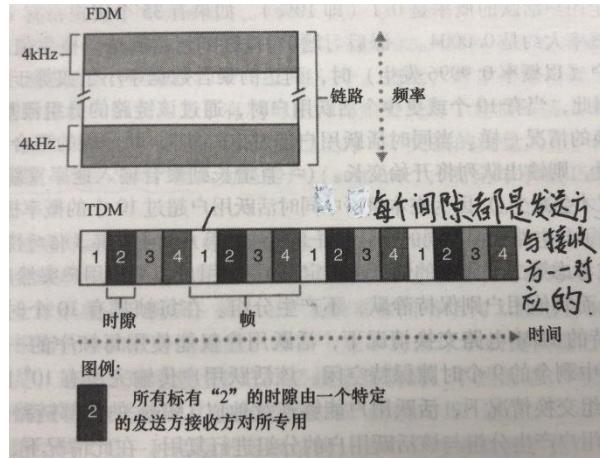


图 1-5 频分复用与时分复用

6) 分组交换和电路交换的优劣

有人说分组交换不适合实时服务，这是当网络拥挤的时候发生的，一般情况下分组交换的延时性比较低

分时交换提供比电路交换更好的带宽共享；

分时交换提供比电路交换更简单，更有效，实现成本更低；

虽然两者都是今天电信网络广泛采用的方式，但是毫无疑问的是分组交换是趋势。

1.5 分组交换网中的时延概述

1) 结点时延

➤ 结点处理时延 nodal processing delay

比如检查分组首部和它需要导向何处，微秒级

➤ 排队时延 queuing delay

毫秒级到微秒级

➤ 传输时延 transmission delay

L/R，毫秒级到微秒级

➤ 传播时延 propagation delay

已接近光速的速度传输，毫秒级

$$\text{节点总延迟: } d_{nodal} = d_{proc} + d_{queue} + d_{trans} + d_{prop}$$

$$\text{端到端总延迟: } d_{end-end}$$

$$= N(d_{nodal}) = N(d_{proc} + d_{queue} + d_{trans} + d_{prop})$$

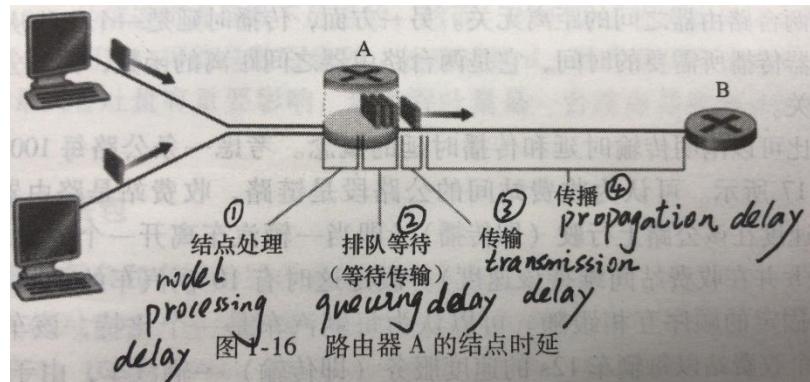


图 1-6 结点时延的组成

1.6 计算机网络中的吞吐量

- 1) 瞬时吞吐量 instantaneous throughput
- 2) 平均吞吐量 average throughput
- 3) 假设 N 条链路的传输速率分别是 $R_1, R_2, R_3 \dots R_N$, 则文件传输的吞吐量是

$$\min\{R_1, R_2, \dots, R_N\}$$

1.7 协议分层

- 1) 协议分层

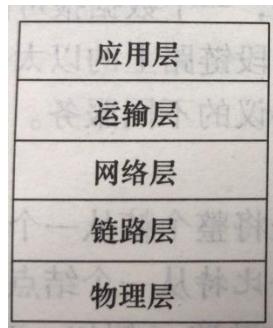


图 1-7 五层因特网协议栈

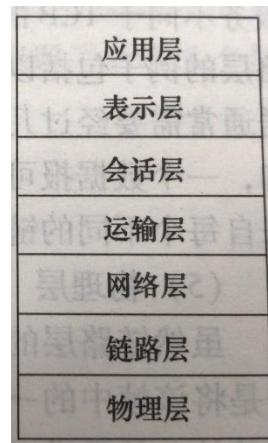


图 1-8 七层 ISO OSI 参考模型

- 2) 协议分层的缺点
 - 一层可能会冗余第一层的功能, 比如每一层都会提供一些容错恢复;
 - 某层的功能可能需要 仅在其他某层才出现的 信息。
- 3) 应用层

应用层是网络应用程序及他们的应用层协议存留的地方。这些协议包括:

 - HTTP 它提供了 Web 文档的请求和传送
 - SMTP 它提供了邮件的请求和传送

- FTP 它提供了文件的请求和传送
 - DNS 网址转换为 32 比特网络地址
- 位于应用层的信息称为报文 message

4) 运输层

运输层是在应用程序端点之间传送应用层报文

- TCP 向应用程序提供了面向连接的服务，包括确保应用层报文向目的地传递和流量控制（即发送方和接收方的速率匹配），将长报文划分为短报文，并提供拥塞控制机制；
- UDP 向应用程序提供无连接服务，这是一种不提供必要服务的服务，没有可靠性，没有流量控制，也没有拥塞控制机制。

位于运输层的分组称为报文段 segment。

5) 网络层

网络层负责将数据报分组从一个端系统移到另一个端系统，其中网络层的 TCP 协议或者 UDP 协议把报文段和目的地址递交给运输层，并决定路由的路由选择协议。

- IP 该协议定义了在数据包中的各个字段以及端系统和路由器如何作用于这些字段。

位于网络层的分组称为数据报 datagram

6) 链路层

将网路层的数据包从一个结点移动到下一个结点，然后下一个结点把数据包传给运输层。

不同的链路层可能有自己特定的链路层协议，如以太网， WiFi 等

链路层的分组称为帧 frame

7) 物理层

物理层的分组就变成是比特

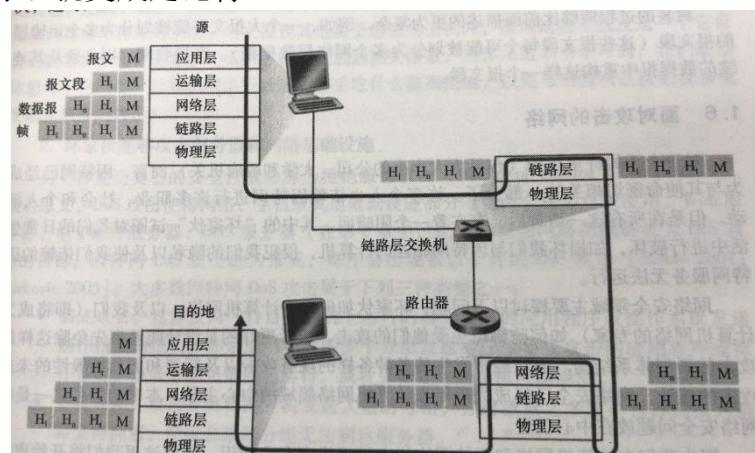


图 1-9 因特网信息传递模型

8) 封装

如图 1-9，链路层交换机只能到达链路层不能实现 IP 协议，而路由器可以达到网络层实现 IP 协议。

- H_t: 允许哪些程序可以接收报文等等运输层首部信息；
- H_n: 添加了源和目的端系统地址等网络层首部信息；
- H_l: 链路层首部信息；

每一层，一个分组具有两种类型的字段：首部字段和有效载荷字段 payload

field。

1.8 网络攻击

1) DoS——单一源

拒绝服务攻击 Denial-of-Service attack, DoS。分为以下三种情况

- 弱点攻击
- 带宽洪泛
- 链接洪泛

1) DDoS——多源

分布式拒绝服务攻击 Distributed Denial-of-Service attack

二、 应用层

2.1 网络应用程序体系结构

1) 由应用程序涉设计者设计

2) 客户-服务器体系结构 client-server architecture

- 服务器总是打开运行的；
- 服务器拥有固定的，周知的 IP 地址；

3) P2P 体系结构

对等方-对等方之间直接通讯，这些应用包括迅雷，Skype。

成本自担，大多数用户的带宽是非对称性的

2.2 进程通信

1) 无论是客户-服务器体系结构还是 P2P 体系结构，两个通信的进程总有一个是作为客户，另一个作为服务器。当然了，这两个进程的角色有时候可以互换。

2) 在给定的一对进程之间的通信会话场景中，发起通信（即在该会话开始时发起与其他进程的联系）的进程被标识为客户，在会话开始时等待联系的进程是服务器。

3) 套接字 socket

- 套接字是应用层和运输层之间接口，此话何解？即进程通过套接字的软件接口向网络发送报文和从网络接收报文，是在应用层与网络之间的应用程序可编程接口 API

- 应用开发者可以控制套接字在应用层的一切，但是对于该套接字在运输层的一切几乎没有控制权。其仅有的控制权旨在于：

- 选择运输层协议；
- 或许可以选择运输层几个参数，比如最大缓存和最大报文段长度；

2.3 进程寻址

需要定义两种信息

- 1) 主机的地址 (IP 地址, 32bit)
- 2) 定义在目的主机中的接收进程 (更具体的说, 应该是接收套接字) 的标识符 (端口号 port number)

2.4 可供应用程序使用的运输服务

- 1) 可靠数据传输

如果一个协议提供了这样的确保数据交付服务, 就认为提供了可靠数据传输; 否则, 当一个运输层协议不能提供可靠数据传输, 运输过程中可能存在数据丢失, 但能被容忍丢失的应用所接受, 比如交谈式视频和音频等多媒体。

- 2) 吞吐量

对吞吐量有要求的应用程序称为带宽敏感应用 bandwidth-sensitive application 能够根据情况或多或少利用可供使用的吞吐量叫弹性应用 elastic application

- 3) 时间

- 4) 安全

应用	数据丢失	带宽	时间敏感
文件传输	不能丢失	弹性	不
电子邮件	不能丢失	弹性	不
Web 文档	不能丢失	弹性 (几 kbps)	不
因特网电话/视频会议	容忍丢失	音频 (几 kbps ~ 1 Mbps)	是, 100ms
		视频 (10kbps ~ 5Mbps)	
存储音频/视频	容忍丢失	同上	是, 几秒
交互式游戏	容忍丢失	几 kbps ~ 10kbps	是, 100ms
即时讯息	不能丢失	弹性	是和不是

图 2-1 选择的网络应用的要求

- 5) TCP 服务

➤ 面向连接的服务

TCP 连接, 全双工, 即发送和接受双方可以同时进行报文收发; 经过握手阶段, 双方的连接建立。

➤ 可靠的数据传送服务

➤ 拥塞机制控制

当网络出现拥塞的时候, 可以抑制发送程序

➤ 安全性

TCP 的安全性比较差, 因为他都是用明文传送的, 容易在任何中间的链路被嗅探和发现。为此, 业界对 TCP 进行了加强, 叫 SSL, Secure Sockets Layer。SSL 不仅能做到 TCP 的一切, 还能提供进程到进程的安全性服务, 比如加密, 数据完整性和端点鉴别。这种强化是在应用层上实现的。

- 6) UDP 服务

UDP 是一种轻量级的不提供不必要的服务的运输协议

➤ 没有连接

- 双方通信之前不仅过握手的阶段
- 不可靠的数据传送服务
 - 没有拥塞控制机制
- 7) 可惜 TCP 和 UDP 都没有提供吞吐量和安全性的服务

2.5 应用层协议

设计成如果 UDP 通信失败机头从哪里开始重传		支撑的运输协议
应用	应用层协议	
电子邮件	SMTP [RFC 5321]	TCP
远程终端访问	Telnet [RFC 854]	TCP
Web	HTTP [RFC 2616] <i>HyperText Transfer Protocol</i>	TCP
文件传输	FTP [RFC 959]	TCP
流式多媒体	HTTP (如 YouTube) <i>HyperText Transfer Protocol</i>	TCP
因特网电话	SIP [RFC 3261]、RTP [RFC 3550] 或专用的 (如 Skype)	UDP 或 TCP

图 2-2 应用层协议的类型和用途

- 1) 应用层协议定义了
- 交换报文的类型
 - 各种报文的语法，如报文中的各个字段及这些字都拿是如何描述的
 - 字段的与意义，即这些字段包含着什么含义；
 - 一个进程何时以及如何发送报文，对报文的响应规则等。

2.6 Web 和 HTTP 协议

- 1) Web 和 HTTP
- HTTP 需要客户程序和服务器程序共同完成；
 - HTTP 定义了 Web 客户向 Web 服务器请求 Web 页面的方式
 - HTTP 使用 TCP 协议作为它的支撑；
 - HTTP 是一种无状态协议 (stateless protocol)，不会记录用户的信息，不会因为之前登陆过就不再相应；
 - 定义从客户端请求服务器相应再到客户端，称这样往返的时间为一个 RRT，Round-Trip Time。

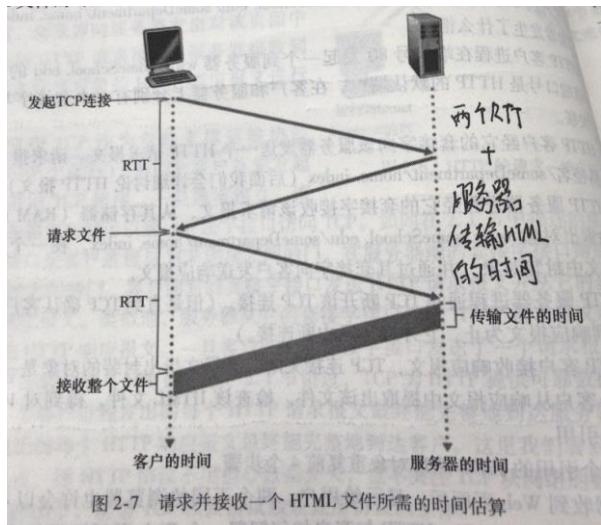


图 2-3 请求并接受一个 HTML 文件所需要的时间

➤ 默认方式下，大部分浏览器可以打开 5-10 个并行的 TCP 链接

- 持续连接（默认）

对于每一个请求/相应，只使用同一个 TCP；

- 非持续连接

对于每一个请求/相应，重新使用一个 TCP。缺点：每次建立连接，服务器都需要为其分配一个 TCP 的缓冲区和 TCP 变量，加重服务器的负担，同时，建立一次 TCP 连接需要两个 RTT，第一个 RTT 建立 TCP，第二个 RTT 用于请求和接收一个对象。

2) HTTP 报文格式

1. HTTP 请求报文

下面提供了一个典型的 HTTP 请求报文：

```
GET /somedir/page.html HTTP/1.1 请求行 request line 方法字段/URL字段/HTTP版本字段
Host: www.someschool.edu 对象所在的主机
Connection: close 持续与非持续连接
User-agent: Mozilla/5.0 指明用户代理，即发送请求的浏览器
Accept-language: fr 请求对象要求是法语版本
```

图 2-4 HTTP 报文格式

POST 时，则实体体中包含的就是用户提交的数据

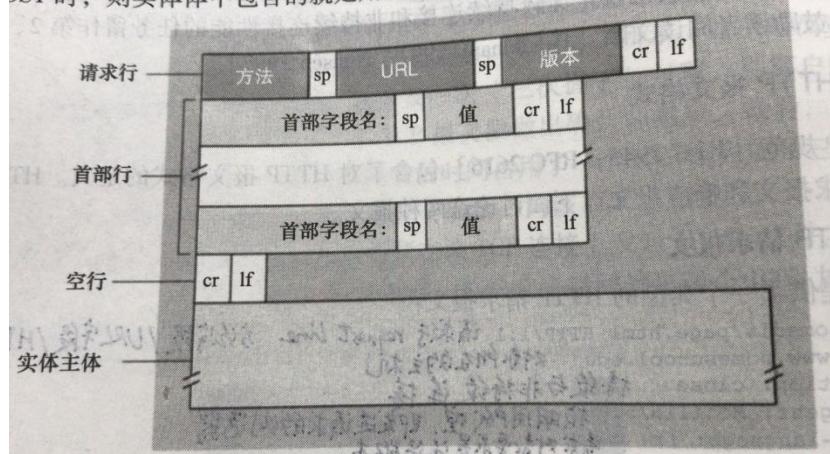


图 2-5 HTTP 请求报文的通用格式

如果是 get 的时候，实体主体为空，当方法为 post 的时候，实体主体是表单

请求的字段。

但是用 get 也可以包含表单请求的字段，实体主体必须为 0 空的情况下那是怎么做的呢？这时候使用拓展的 URL 地址，在地址的后面放入“? 字段”HEAD 方法跟 GET 差不多，会返回响应报文，但是不会返回对象，这是要用 来帮开发者调试跟踪的。

PUT 方法主要用来上传对象的；

DELETE 方法用来删除服务器对象的

3) 响应报文

```
HTTP/1.1 200 OK
Connection: close
Date: Tue, 09 Aug 2011 15:44:04 GMT
Server: Apache/2.2.3 (CentOS)
Last-Modified: Tue, 09 Aug 2011 15:11:03 GMT
Content-Length: 6821
Content-Type: text/html

(data data data data data ...)
```

图 2-6 HTTP 响应报文的格式

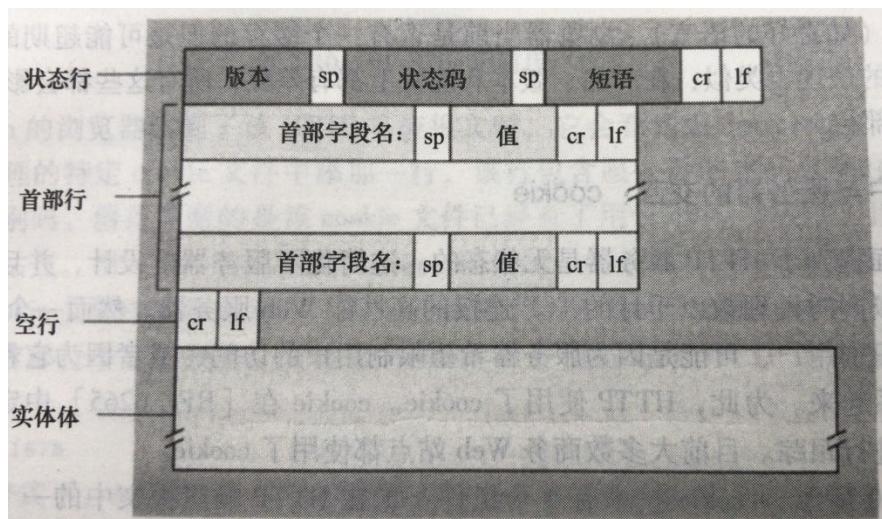


图 2-7 HTTP 响应报文的通用格式

相应状态码：

- 200: OK, 请求成功; (找到人了)
- 301: Moved Permanently: 请求的对象已经被永久转移了, 新的 URL 定义在相应报文的 Location: 首部行中。客户软件将自动获取新的 URL; (搬走了)
- 400: Bad Request, 一个通用的差错代码, 表示该请求不能被服务器理解; (遇到老外了, 无法沟通)
- 404: Not Found, 被请求的文档不在服务器上; (找的人不在)
- 505: HTTP Version Not Supported: 服务器不支持请求报文使用的 HTTP 协议版本。(钥匙打不开)

4) cookie

cookie 包括四个组件：

- 在 HTTP 响应报文中的一个 cookie 首部行;
- 在 HTTP 请求报文中的一个 cookie 首部行;
- 在用户端系统中保留有一个 cookie 组件，并由用户的浏览器进行管理;
- 位于 Web 站点中的一个后端数据库;

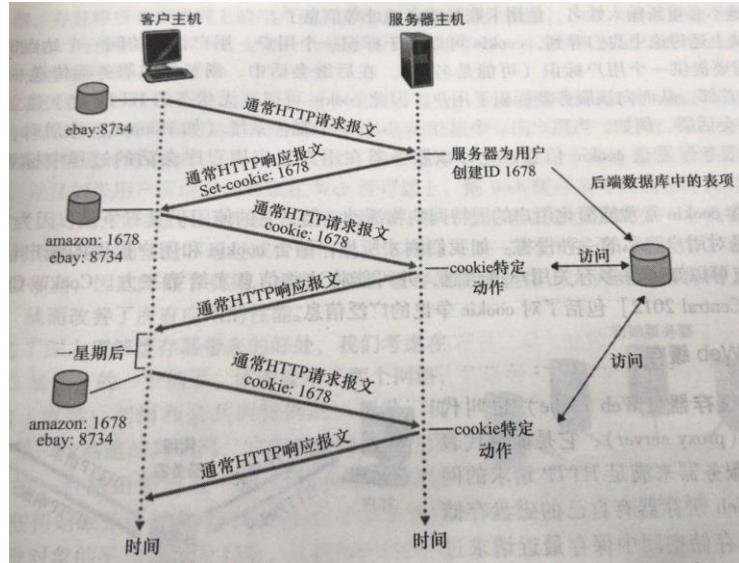


图 2-8 用 cookie 跟踪用户状态

2.7 Web 缓存

1) Web 缓存器

Web 缓存器，也叫代理服务器 proxy server。Web 缓存器是服务器的同时也是客户，当它接收浏览器的请求的时候，它是一个服务器。当它向初始服务器发出请求的并接收相应的时候，它是一个客户。

2) 好处

- 能大大减少相应的时间;
- 能大大降低流量，从而改善所有应用的性能;

3) 但是缓存有一个问题，就是缓存的对象可能是旧的，所谓旧的，就是自上次缓存过后，服务器那边的对象被服务器修改过。这时候如果从缓存中还是返回旧版本的对象，那就错了，为了避免这种问题，需要在请求报文中使用条件 get 的方法。

4) 条件 get 的方法

首先，浏览器向服务器发送请求报文，此报文会首先经过缓存器

```
GET /fruit/kiwi.gif HTTP/1.1
Host: www.exotiquecuisine.com
```

由于在第一次的时候，缓存器没有缓存该对象，于是从服务器中返回的对象中 copy 了一份，并给浏览器返回响应报文

```
HTTP/1.1 200 OK
Date: Sat, 8 Oct 2011 15:39:29
Server: Apache/1.3.0 (Unix)
Last-Modified: Wed, 7 Sep 2011 09:23:24
Content-Type: image/gif

(data data data data data ...)
```

一周后，假如浏览器再一次向服务器发送同一个请求，由于缓存器中有上一次缓存的对象，但是不确定那是不是最新的，或者说有没有被改动过，于是向服务器发送的请求报文中加入 If-Modified-Since：

```
GET /fruit/kiwi.gif HTTP/1.1
Host: www.exotiquecuisine.com
If-Modified-Since: Wed, 7 Sep 2011 09:23:24
```

服务器看到请求报文后，检查 If-Modified-Since 的时候如果确定在之后的时间里被请求的对象没有被改动过，则返回响应报文：

```
HTTP/1.1 304 Not Modified
Date: Sat, 15 Oct 2011 15:39:29
Server: Apache/1.3.0 (Unix)

(empty entity body)
```

2.8 文本传输协议：FTP

- 1) 在客户端和服务器端会话过程中，建立两条 TCP 连接，一条是控制连接，另一条是数据连接
- 2) 控制连接
 - 持续的，在整个会话过程都持续；
 - “存放 put” 和 “获取 get” 命令；
 - 控制信息是带外 out-of-band 传送的；
- 3) 数据连接
 - 等控制连接，发布命令后，服务器就发起一个到客户端的 TCP 连接
 - 每个文件传送都会建立一个 TCP 连接，传送完毕后就断开连接；
- 4) 状态
由于用户在使用 ftp 的时候，会持续在目录中进行操作，所以必须在会话期间保留用户的状态。而 HTTP 是无状态的，不必对任何用户的状态进行追踪。
- 5) 命令和状态码

较为常见的命令如下：

- USER username：用于向服务器传送用户标识。
- PASS password：用于向服务器发送用户口令。
- LIST：用于请求服务器回送当前远程目录中的所有文件列表。该文件列表是经一个（新建且非持续连接）数据连接传送的，而不是在控制 TCP 连接上传送。
- RETR filename：用于从远程主机当前目录检索（即 get）文件。该命令引起远程主机发起一个数据连接，并经该数据连接发送所请求的文件。
- STOR filename：用于在远程主机的当前目录上存放（即 put）文件。

图 2-9 FTP 相关命令

- 331 Username OK, Password required (用户名 OK, 需要口令)。
- 125 Data connection already open; transfer starting (数据连接已经打开, 开始传送)
- 425 Can't open data connection (无法打开数据连接)。
- 452 Error writing file (写文件差错)。

图 2-10 FTP 相关状态码

2.9 电子邮件传输协议：SMTP

1) 三个关键的部分

- 用户代理 user agent
- 邮件服务器 mail server
- 简单邮件传输协议 SMTP, simple mail transfer protocol

2) SMTP 传送的过程

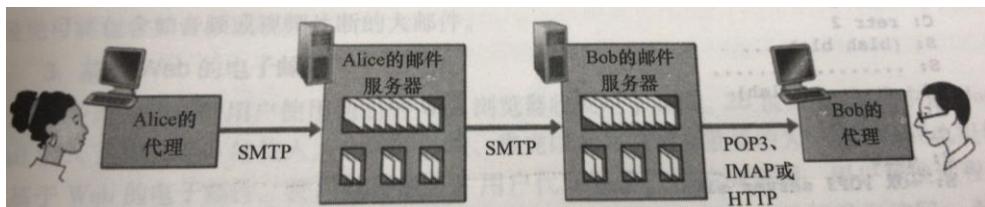


图 2-11 SMTP 传送的过程

3) 不经过中间邮件服务器

4) 运输层使用 TCP 协议

5) SMTP 与 HTTP 的比较

- HTTP 是拉协议 pull protocol, 而 SMTP 是推协议 push protocol;
- SMTP 协议要求报文使用 7 比特 ASCII 码格式, 如果没有用此格式编码, 则必须按照 7 比特 ASCII 码格式, 而 HTTP 协议则没有这要求
- HTTP 把每个报文对象都封装在自己的 HTTP 响应报文中, SMTP 则把所有报文对象都放在同一个报文中。

6) 典型的报文格式

```
From: alice@crepes.fr
To: bob@hamburger.edu
Subject: Searching for the meaning of life.
```

图 2-12 SMTP 典型的报文格式

7) 我们知道 SMTP 是一个推协议, 只负责推送邮件, 但是如何收邮件呢。POP 和 IMAP 协议就是一个专门的邮件访问协议, 用来收邮件的。

8) POP3 协议

- 简单
- 三个阶段
 - 特许，使用用户名和口令验明用户身份，登陆服务器
 - 事务处理：用户可以发出一些命令，包括取回报文 retr, 删除标记 dele, 取消报文删除标记和统计 list。而服务器可以有两种回答：+OK 和-ERR
 - 更新 用户执行 quit 命令后进入更新阶段
- 不保留用户状态

9) IMAP 协议

- 较为复杂；
- 允许用户拥有自己的云文件夹，进行创建和移动邮件的操作，故在会话阶段需要保留用户的状态；
- 允许只查看部分的报文部件，这在低带宽连接的时候比较有用；
- 保留用户状态

10) 基于 Web 的电子邮件

用户代理到邮件服务器之间的通信协议为 HTTP，而邮件服务器之间通信依然使用 SMTP

2.10 DNS：因特网的目录服务

1) DNS 是 Domain Name System，负责把域名转换为 IP 地址，它是一个分布式的数据库。

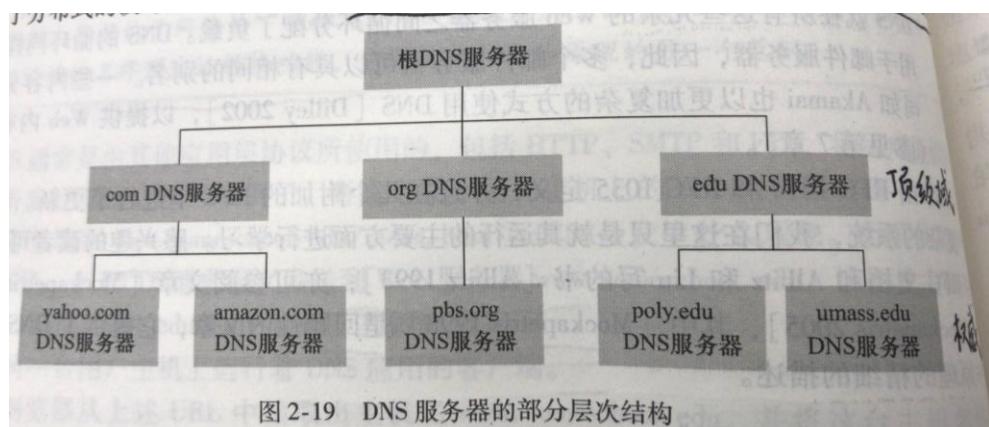


图 2-13 DNS 服务器的部分层次结构

根 DNS 服务器有 13 个，标号从 A-M，大部分放在美国；

另外还有一层比较草根的分层是本地服务器，最接近我们普通人

2) DNS 的任务

- 域名转换 IP 地址；
- 主机别名，其实就是一个 IP 地址对应多个别名，其中有一个叫规范主机名；
- 邮件服务器别名：就是把邮件服务器别名和用户代理的主机名共用一个 IP 地址；

- 负载分配：一个主机名对应多个 IP 地址，使得用户请求可以分散在不同的主机上。
- 3) 查询域名的步骤
- 同一台用户主机上运行着 DNS 应用的客户端；
 - 浏览器从 URL 中抽取主机名，并将主机名传送到 DNS 应用的客户端；
 - DNS 应用的客户端把主机名传送到 DNS 服务器进行查询，再把查询到的 IP 地址发回。

但是这样的查询相当花费时间，所以这时候附近的 DNS 服务器缓存就起到非常关键的作用了。

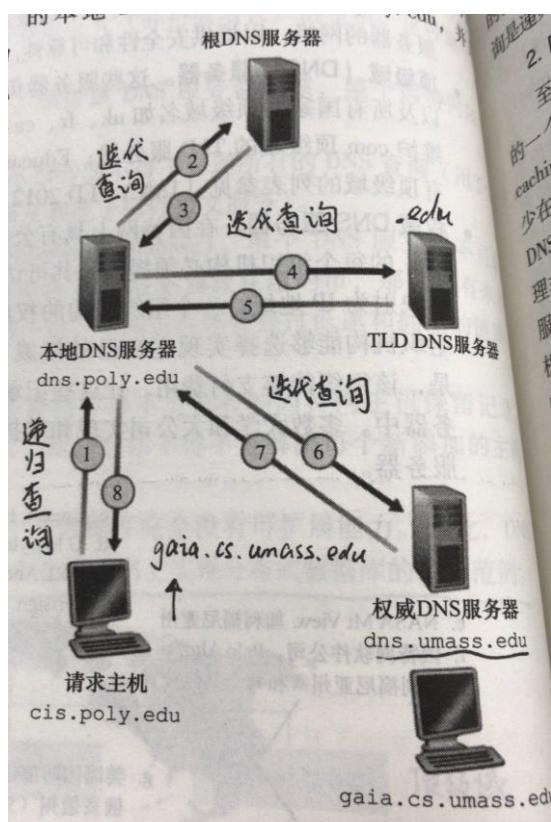


图 2-14 DNS 查询步骤和分类

DNS 查询分为递归查询和迭代查询，递归查询就是其中一个参与者必须为自己，迭代查询就是让别人帮你查。

4) DNS 缓存

通常保留两天的时间

5) DNS 记录和报文

DNS 记录，学名叫资源记录 Resource record，记录了主机到 IP 地址的映射。其格式为：

$$(Name, Value, Type, TTL)$$

其中 TTL：应当从缓存中删除的时间。

Type=A，A 记录，表示 Name 为主机名，Value 为 IP 地址；

Type=NS，NS 记录，表示 Name 为主机名的域，Value 为知道如何获取该域中 IP 地址的权威 DNS 服务器的主机名，但是只有 NS 记录是不行的，还必须有一条指向权威 DNS 服务器 IP 地址的 A 记录，要不然怎么找；

Type=CNAME, CNAME 记录，表示 Name 为主机名的别名，Value 为对应的规范主机名；

Type=MX, MX 记录，表示 Name 为邮箱服务器的主机名，Value 为对应的规范主机名；

6) 在 DNS 数据库中插入记录

- 首先先去注册登记机构注册域名；
- 该注册机构确保将跟你的域名 NS 记录，及其相关的一个基础和一个辅助的权威 DNS 服务器的 A 记录，插入 TLD (Top-level Domain) 顶级服务器中；
- 然后将你的域名 A 记录和 MX 资源记录插入到上一步的权威 DNS 服务器中；

2.11 P2P

1) 最小分发时间 D_{P2P}

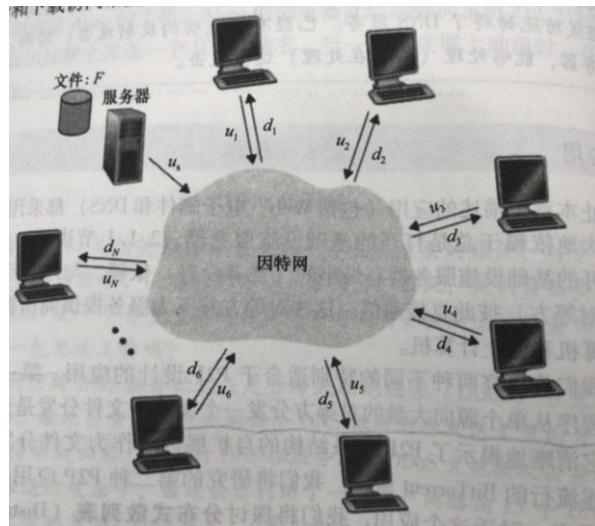


图 2-15 文件分发的实例

$$D_{P2P} \geq \max \left\{ \frac{F}{u_s}, \frac{F}{d_{min}}, \frac{NF}{u_s + \sum_{i=1}^N u_i} \right\}$$

当 N 趋于无穷的时候， $D_{P2P} \rightarrow \frac{F}{u_i}$ ，P2P 由于客户-服务器结构，因为当 N 足够大的时候，客户-服务器结构所花费的时间几乎是线性增长的，而 P2P 则 $D_{P2P} \rightarrow \frac{F}{u_i}$ ，原因是对等方除了是比特的消费者，还是比特的重新分发者。

2) BitTorrent

- 3) BitTorrent 是最出名的文件分发协议，Torrent 为洪流之意，分发的是文件块
4) 有一个管理服务器——追踪器，负责用户身份验证和追踪发送邻近对等方子集的 IP 地址；

- 邻近对等方子集的 TCP 并行连接

每个对等方会从追踪器那里取得邻近对等方的一个列表，而对等方自己就从中选取一个子集进行接下来的 TCP 并行连接；

➤ 最稀缺优先策略

即选择自己没有的文件块，而该文件块也正是邻近对等方子集中最稀缺的。这样做有一个好处，就是当自己拿到那个最稀缺文件块之后，利用 P2P 的特点对该稀缺文件进行上传（做种），使得稀缺文件变得不再稀缺，其目标是均衡每个块在洪流中的副本数量。

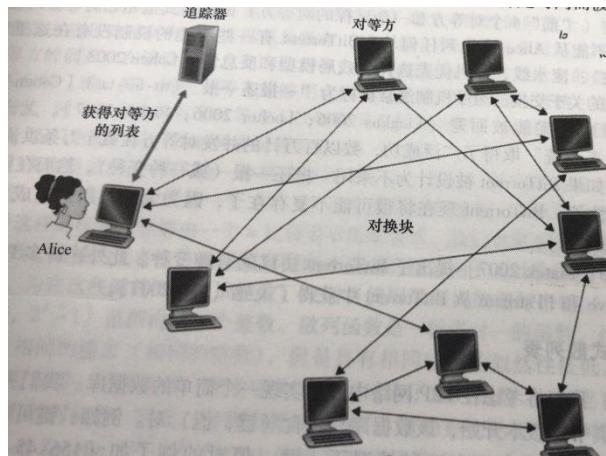


图 2-16 对等方与对换块

➤ 最高速率响应

在相应我方的请求的时候，我方并不“盲目”能得到所有的邻近对等方子集 TCP 连接，我们会利用最高速率策略，从中选择能给我们以最高速率传送的前四名选手，进行 TCP 连接，过一段时间后再重新挑选。

用 BitTorrent 的术语来讲，这四名选手称为“疏通 unchokedy ”

➤ 分布式散列表

- ◆ 这个主要用来查询用的，比如我方的对等方需要文件 Linux，那么发送 Linux 给服务器，那么服务器就以键值对的形式查询<Linux, 序号>，序号表示哪个对等方拥有 Linux 文件的资源，找到后就给我方对等方返回拥有 Linux 文件的资源的对等方的 IP 地址。而该服务器是客户-服务器结构的服务器。
- ◆ 插入的算法——最邻近插入
- ◆ Distributed Hash Table，对于环形的 DHT，每个对等方只知道自己直接前任和直接后继。然后通过查询最邻近的对等方再在其旁边插入。但是这样有一个不好的地方，就是查询比较慢。后来有了一种新的环形 DHT，就是每个对等方除了知道自己的直接前任和直接后继，还知道环上数量相对少的“捷径对等方”，这样能大大减少查询的次数，节省时间。
- ◆ 对等方扰动。所谓对等方扰动，其实就是在环内有对等方突然加入和退出。退出的问题比较严重。解决的关键在于直到第二个后继者。因为当第一个后继者掉线的时候，立马通知第二个后继者跟它连在一起，相当于把第二个备胎变成正备胎。

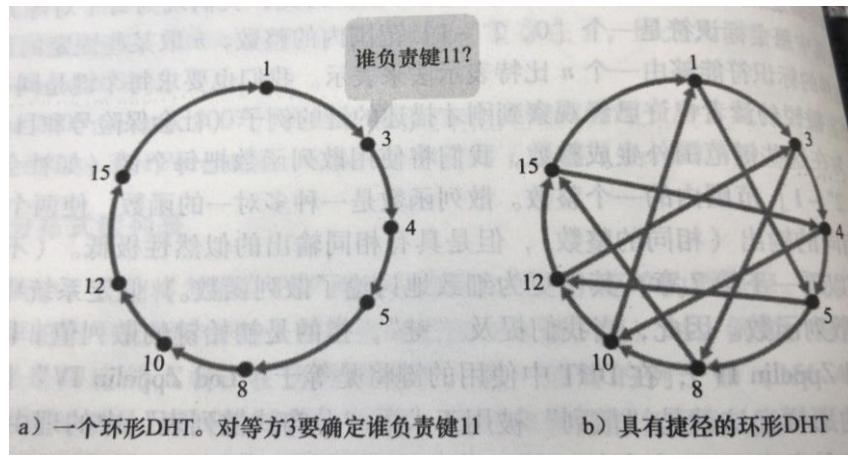


图 2-17 分布式散列表查询

三、运输层

3.1 运输层和网络层的关系

- 1) 运输层和网络层的关系

网络层保证端系统与端系统之间的逻辑通信;
运输层保证端系统进程与端系统进程之间的逻辑通信;
网络层为运输层提供时延和带宽的保证;
运输层能为网络层提供分组不被丢失，篡改和冗余，以及安全方面的服务
- 2) 网络层主要依靠的是 IP 协议，而 IP 协议模型是一种尽力而为交付服务(Best-Effort Delivery service)，这意味着这不是一种可靠的数据服务，但是使用运输层的 TCP 协议可以保证数据传输的可靠性。
- 3) 报文段从网络层到运输层，然后运输层再把报文段交给进程，但是这种交付并不是直接的，运输层和进程之间还隔了一个或者多个套接字 sockets。由于一个主机上不止有一个套接字 sockets，每个 sockets 都有唯一的标识符，该标识符的格式取决于它是 TCP 格式还是 UDP 格式。

3.2 多路复用 multiplexing 与多路分解 demultiplexing

- 1) 多路复用 multiplexing

源主机从不同的 sockets 中收集信息传到运输层，在运输层中加入首部信息（这以后便于分解）从而生成报文段，报文段在发送给网络层，这就叫做多路复用 multiplexing。

要求：

 - socket 需要有唯一的标识符；
 - 每个报文段有特殊字段来指示要交付到的 socket，一般这个特殊字段是源端口字段 source port number field 和目的端口字段 destination port number field。所以当我们新建一个进程的时候，需要给他分配一个唯一的端口号。

2) 多路分解 demultiplexing

识别运输层中的报文段中的有关 socket 的信息，将报文段分别发送给不同的 sockets，这就叫做多路分解 demultiplexing。

3) 端口号

16 比特 0~65535

0~1023，留给诸如 Http80 或者 ftp21 之类的熟知应用层协议来使用。

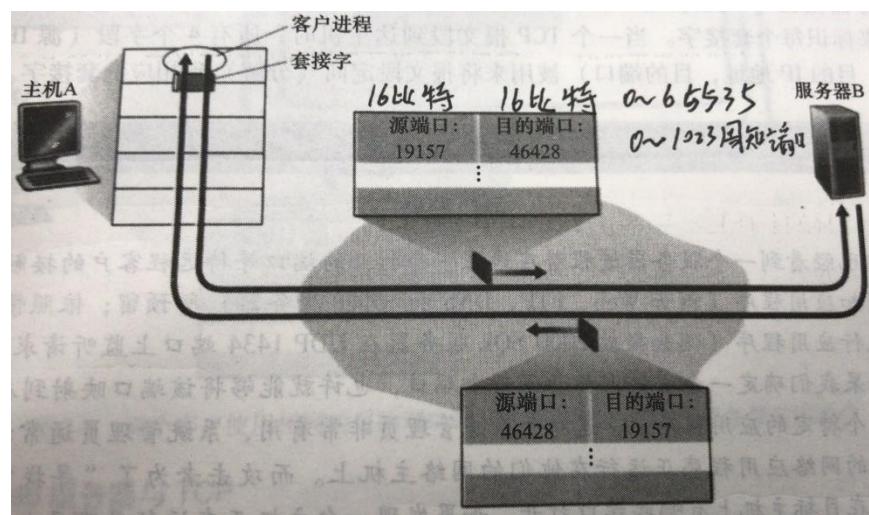


图 3-1 端口号

4) 无连接时的多路复用和多路分解

- 一个 UDP 套接字是由一个二元组来标识的，包括主机的目的 IP 地址和套接字的目的端口号。
- 如果源 IP 地址或源端口号不一致的报文段将进入相同的目的 IP 地址和目的端口号；
- 而报文段中的源 IP 地址和源端口号则作为返回地址；

5) 有连接时的多路复用和多路分解

一个 TCP 套接字是由一个四元组来标识的，包括主机的源 IP 地址和套接字的源端口号，目的 IP 地址和套接字的目的端口号。

3.3 UDP 协议

1) 相比 TCP 协议的优势

- 实时操作方面更加精细，因为没有拥塞控制机制；
- 无建立连接的开销；
- 无需维持连接状态；
- 分组首部报文开销小，因为 TCP 大概需要 20 个字节，而 UDP 之需要 8 字节。

2) 劣势

由于没有拥塞控制机制，发送方和接收方之间会存在高的丢包率，虽然 UDP 协议本身对这个问题无能为力，但是应用层可以自己建立一些可靠传输的措施，这样会使得 UDP 协议“左右逢源”，既没有拥塞控制，又可以有可靠的数据传输。

3) UDP 格式

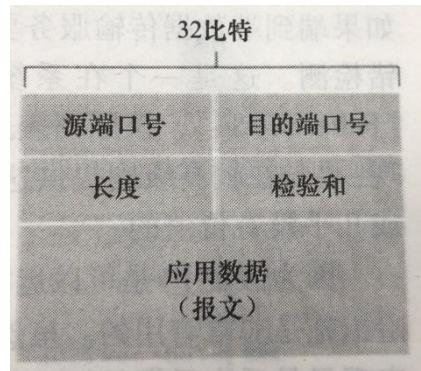


图 3-2 UDP 格式

长度指的是报文段的字节数（首部+数据，用两个字节表示）

4) 校验

- 源端口号，目的端口号，长度和校验和分别用 16bit 表示，总共 64bit 合计 8 字节。
- 将源端口号，目的端口号和长度一次相加，有溢出的需要“回卷”，所谓“回卷”，就是把溢出的最高位 1 和低 16 位做加法运算。
例如：原本是 (1) 0100101011000001，回卷就是 0100101011000001+1=0100101011000010，即把前面多出去的 1 加到最后。最后的结果在进行一次反码运算，最终的结果就是校验和的 16bit。
- 接收方把源端口号，目的端口号，长度和校验和全部加起来，如果结果等于 1111 1111，说明在传输过程中没有出错，如果某一位等于 0，说明出错了。

5) 为什么 UDP 要提供校验和的差错检测

原因是无法保证所有的链路层协议都有差错检测，唯有在运输层这里进行最终的检测，这就是被人歌颂的端到端原则 end-end principle.

虽然 UDP 要提供校验和的差错检测，但是它对已经发生的错误无法进行纠正，要么丢弃，要么报给应用层进行警告。

3.4 可靠的数据传输原理

1) rdt1.0 协议——最简单的协议

发送方发送分组，接收方接收分组；

2) rdt2.0 协议——解决分组出错的问题——停等协议

- 发送方发送分组，接收方接收分组，然后使用 ACK 表示分组完好，NAK 表示分组出错，要求发送方重发；
- 只有一个分组确认成功接收后，才进行下一个分组的发送，这叫做停等 Stop-and-Wait
- 注：这里有一个问题，就是确认回答 ACK/NAK 受损的情况，这里有三种解决方法：
 - 第一种：发送方要求接收方重发确认回答，这样有可能会陷入死胡同，确认回答一直受损；
 - 第二种：加入足够的检验和比特，可以进行纠错，那就一了百了了；
 - 第三种：不管三七二十一，发送方只要收到的来自接收方无法辨别的确认回答的时候，一律重发，不过这里会有一个问题，就是容易产生

冗余，接收方可能会接收到 1 条以上重复的分组。

- 3) rdt2.1 协议——解决分组出错后的冗余问题，添加分组序号
采用 rdt2.1 协议中第三种解决方案，每个分组分配一个序号 0 或者 1，0 和 1 交替添加，接收方收到相同分组，说明发生了冗余；
- 4) rdt2.2 协议——解决分组出错后的冗余问题，添加分组序号，并消除 NAK
就是不采用 ACK+NAK 回答，只是用 ACK，并在 ACK 后面再加上分组序号即可。
- 5) rdt3.0 协议——解决丢包的问题
关键：倒计数定时器
该协议有时又叫做“比特交替协议”
- 6) rdt 停等协议模型

图 3-3 rdt 停等协议模型

3.5 流水线可靠数据传输原理

- 1) 需要满足三个条件
 - 分组主键，扩大序号范围，使得序号跟分组建立一一对应的关系；
 - 缓存准备发送的分组和已到达的分组及其确认信息 ACK；
 - 需要纠错策略；
- 2) 回退 N 步策略 Go-Back-N GBN

图 3-4 GBN 模型介绍
基序号是最早没有被确认的分组序号；下一个序号是最小还没有被使用的待发送序号。

 - 发送方
 - 当 N 个分组已满，正等待接收方返回最早一个没有被确认分组的确认回复前的这段时间，socket 无法再向运输层传送报文；
 - 发送方的确认方式是累计确认，只有当最早的那个分组的确认回复收到后才开始发送下一个分组；
 - 接收方
- 3) 选择重传策略 SR
基本跟 GBN 策略差不多，区别在于失序的分组不丢弃，直至窗口期满。
而接收方将失序的分组缓存起来，直至一批的分组全部接受好，再交给 socket。
而且接收方要重新确认（而不是忽略）那些收到过的小于当前窗口基序列的分组，否则发送方的窗口永远无法向前进。
窗口长度必须小于或等于序号长度的一半（或者说发送方的窗口+接收方的窗口要小于或者等于序号长度），否则无法确定接收的分组是一个新的分组还是一次重传。

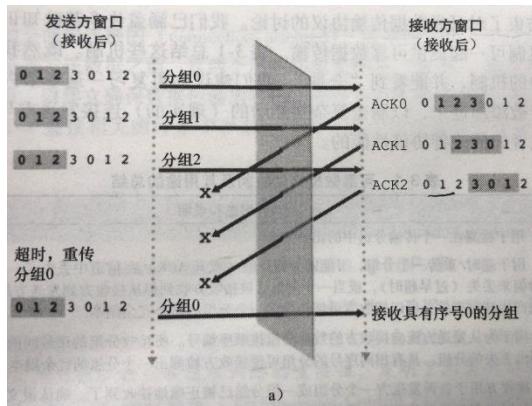


图 3-7 一次重传

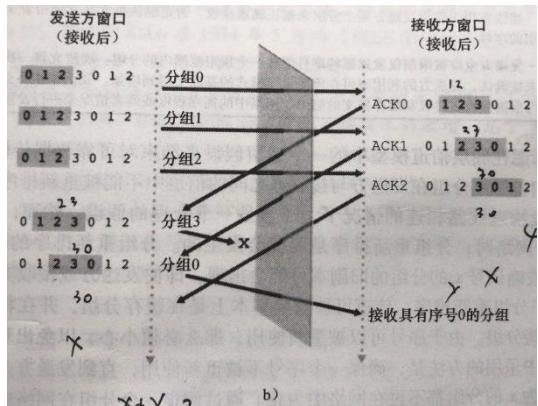


图 3-8 一个新的分组

3.6 TCP 协议

- 1) TCP 传输的报文段长度受限于最大报文段长度 Maximum Segment Size, MSS, 而 MSS 则受限于本地发送主机最大链路层帧长度
- 2) TCP 格式
 - 其中最重要的是 32bit 的序号和 32bit 的确认号;
 - 接收窗口: 用于流量的控制, 该字段用于指示接收方愿意接受的字节数量;
 - URG: 说明存在紧急数据;
 - ACK: 用于指示确认字段中的值是有效的;
 - PSH: 指示接收方应该立即把紧急数据传送给上层;
 - RST, SYN 和 FIN: 用于 TCP 连接的建立和拆除;

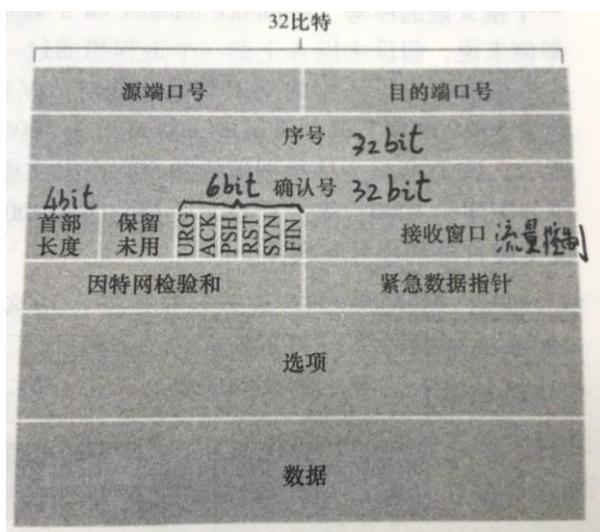


图 3-9 TCP 报文段格式

- 3) 序号

TCP 协议报文段中的序号不是基于报文段, 而是基于字节流的, 比如说要发送的字段共有 5000 个字节, 则序号从 0~49999。

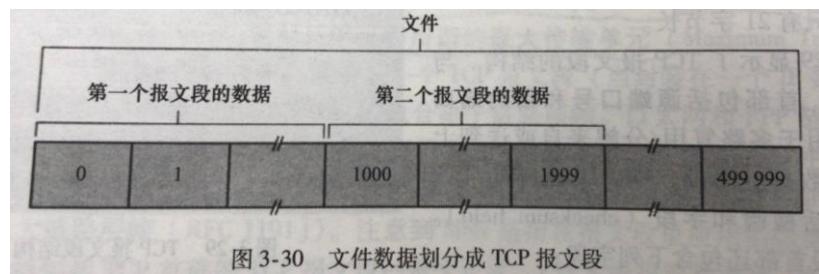


图 3-30 文件数据划分成 TCP 报文段

图 3-10 TCP 报文段序号

4) 确认号

假如主机 A 从主机 B 中拿到了 0-500 字节的报文段，那么由于 TCP 是全双工的，主机 A 返回主机 B 第 501 个字节序号，那么主机 B 下一次发的报文段的确认号就是 501

5) 累计确认

TCP 的接收方只会确认字节流从一开始到第一个丢失分组的报文段，然后向发送方重新发送丢失分组的确认号，这样可以确保确认号之前的报文段都顺利被接收，而接收方也不必重新确认和发送确认号之前的报文段。

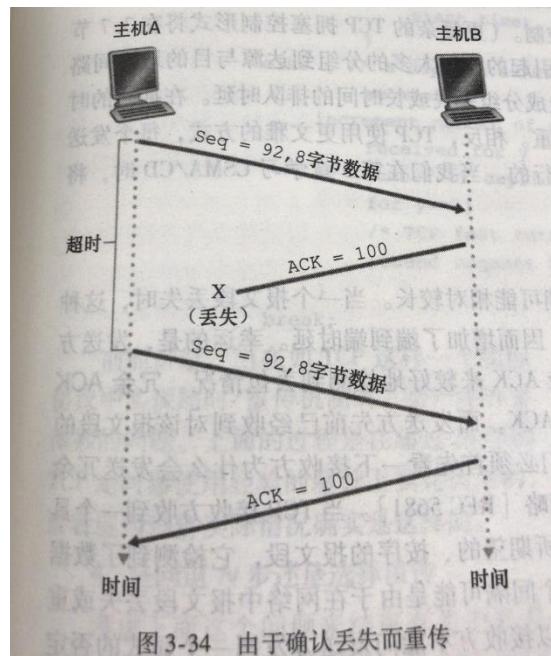


图 3-34 由于确认丢失而重传

图 3-11 由于确认丢失而重传

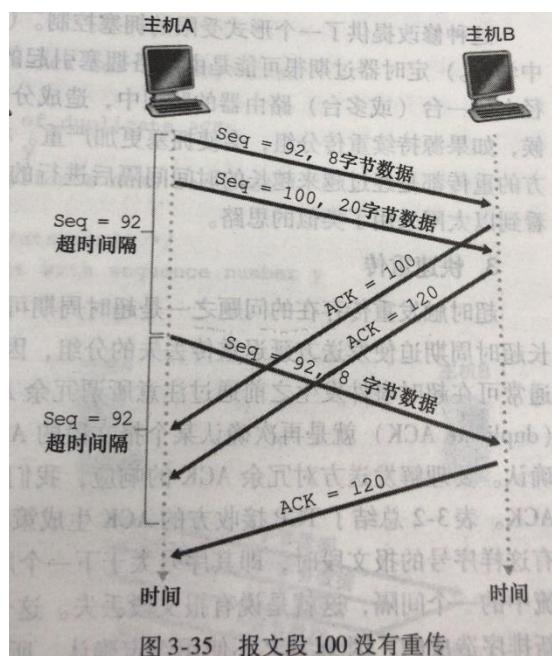


图 3-35 报文段 100 没有重传

图 3-12 第一个报文段超时重传，
而第二个没有超时，不需重传

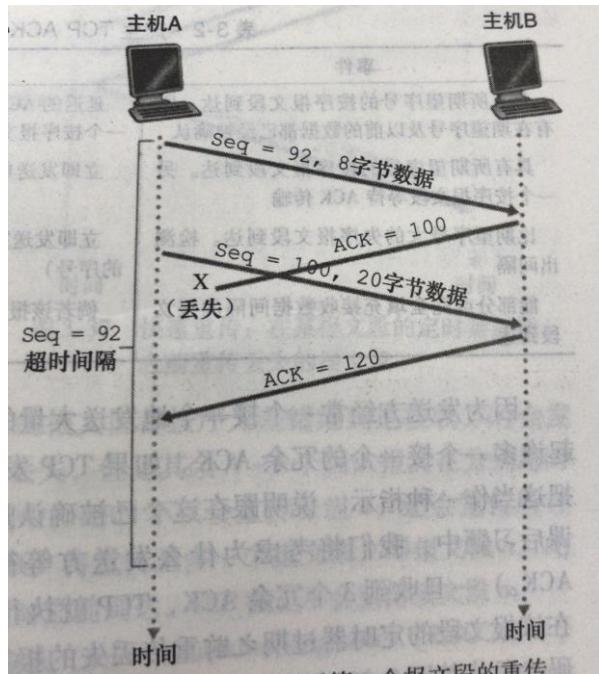


图 3-13 第一个确认丢失，第二个确认在超时前到达，由于 TCP 是累计确认的，说明第一个也已经顺利被接收了，不需再重传第一个。

6) 失序报文段

有趣的是，TCP 协议中并没有提到如何处理失序报文段，这需要留给编程人员去处理，通常来说有两种方法：

- 类似 GBN 模型，把失序报文段丢弃；
 - 类似 SR 模型，把失序报文段缓存起来；
- 显然后者优势更明显，是实际采用的方法。

7) telnet 例子

- 回显 echo back，用于确保用户发送的字段已经被服务器接收。
- 即使该报文段还没有数据，仍然会有序号这是应为 TCP 存有序号字段。

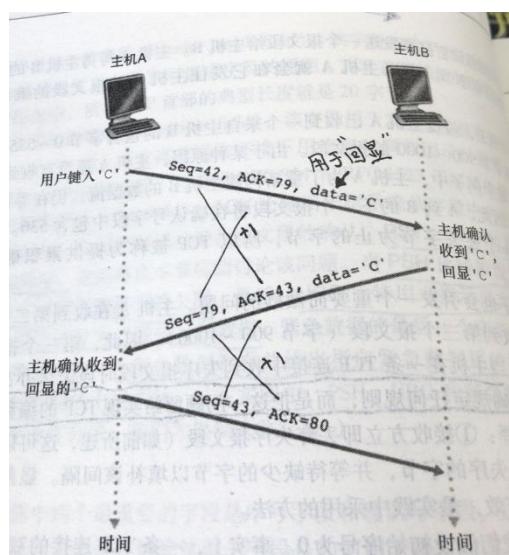


图 3-14 一个经 TCP 的简单 Telnet 应用的确认号和序号

图 3-14 一个 TCP 简单的应用

8) 往返时间 RTT 及其估计 Sample RTT

超时重传时间必须大于 RTT，否则会引起不必要的重传；

Sample RTT 只会测量最近一次发送方发送而且没有收到接收方确认回复的 RTT，最后我们使用指数加权移动平均法算出总平均：

$$\text{EstimatedRTT} = (1 - \alpha) \times \text{EstimatedRTT} + \alpha \times \text{Sample RTT}$$

$$\alpha_{\text{建议值}} = 0.125$$

实际Sample RTT和估计平均EstimatedRTT的偏差：

$$\text{DevRTT} = (1 - \beta) \times \text{DevRTT} + \beta \times |\text{Sample RTT} - \text{EstimatedRTT}|$$

$$\beta_{\text{建议值}} = 0.25$$

最后超时重传时间：

$$\text{TimeoutInterval} = \text{EstimatedRTT} + 4 \times \text{DevRTT}$$

注意，EstimatedRTT一开始的时候还没有稳定，TimeoutInterval初始化为 1 秒，后面假如出现超时的现象，就把TimeoutInterval加倍，当没有超时之后再使用该公式。

9) 超时间隔加倍

10) 快速重传

超时出发重传的缺点在于超时周期比较长，增加了端到端的时延。

当分组丢失的时候，接收方面对乱序的报文段，TCP 会先把乱序的报文段缓存起来，没接收一个乱序报文段，回复一个丢失报文段的确认以请求重传，当接收方接收到三个冗余的确认号的时候，应当早超时前立即重传丢失的报文段。

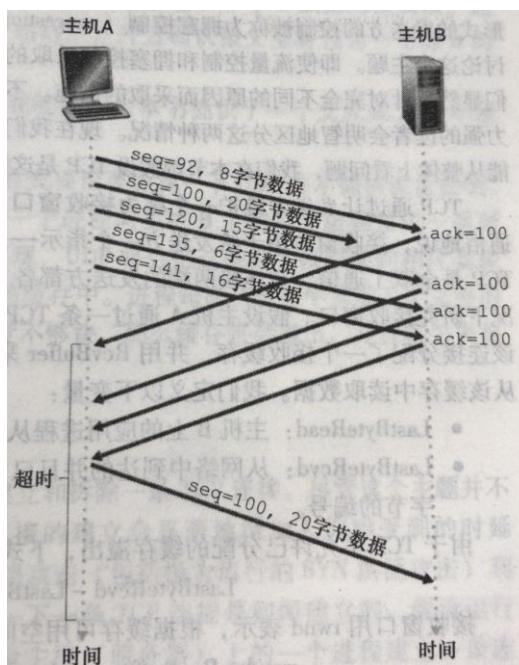


图 3-15 快速重传的例子

11) 差错控制：是选择 GBN 还是 SR？

首先我们知道 TCP 是累积确认的，接收方是不会确认乱序分组的，而发送方也只会维护已经发送还没有被确认的最小序号，从这个意义上讲，有点像 GBN 的风格；

另一方面，由于 TCP 是累积确认的，而且接收方会把乱序分组缓存起来，当超时后发送方会选择性地把丢失的分组重传，从这个意义上讲，这又有点想 SR

所以我们说 TCP 的纠错策略是 GBN 和 SR 的混合体。

12) 流量控制

对于接收方

接收缓存 (N 值) $RcvBuffer \geq LastByteRcvd - LastByteRead$

接收窗口 $rwnd = RcvBuffer - (LastByteRcvd - LastByteRead)$



图 3-38 接收窗口 (rwnd) 和接收缓存 (RcvBuffer)

图 3-16 接收窗口和接收缓存

对于发送方

发送缓存 (n 值) $SentBuffer \geq LastByteSent - LastByteAcked$

已发送待确认窗口 $swnd = LastByteSent - LastByteAcked$

需要满足接收方缓存不被溢出的条件：

$$rwnd \geq swnd$$

$$rwnd \geq LastByteSent - LastByteAcked$$

但这里会有一个问题，就是接收方只有在接收新数据或者要发送确认的时候才会给发送方发送消息，如果 $rwnd = 0$ 的时候，发送方无法发送报文段，而当接收方缓存被清空后，由于不能收到发送方新的报文段，这时候就会存在死锁。

为了解决这个问题，当 $rwnd = 0$ 的时候，发送方需要持续给接收方只有一个字节的报文段，好让接收方缓存被清空， $rwnd > 0$ 的时候给发送方消息，让它继续发送有效报文段。

13) 所以，由于 UDP 协议没有流量控制，当接收方缓存溢出的时候，报文段就会被丢失。

14) TCP 连接的建立和拆除

建立：三次握手，前两次握手 SYN 位置 1，seq 使用初始值，第三次握手时 SYN 位置 0，可以发送应用层数据。

拆除：FIN 位置 1，双方互发 FIN 和 ACK。

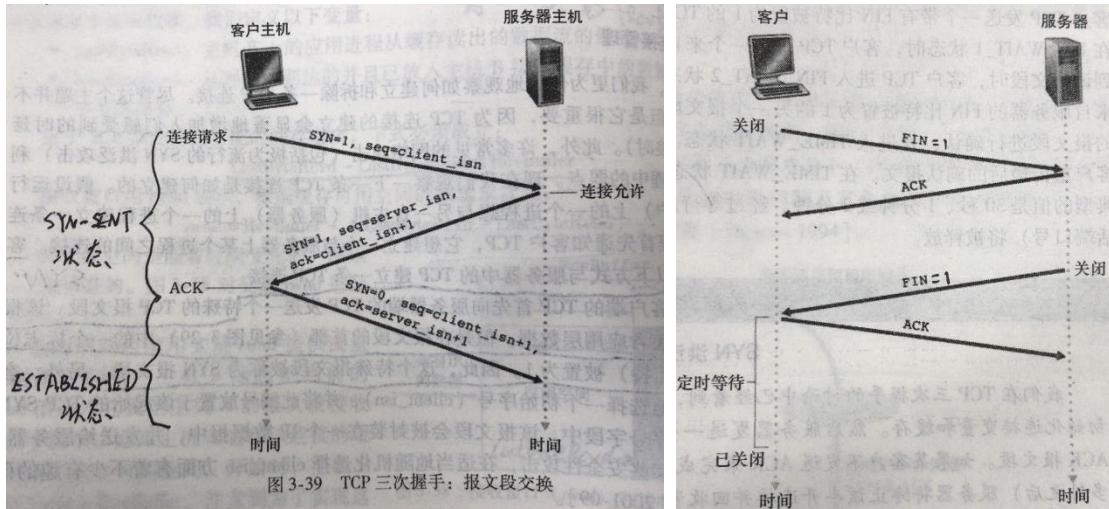


图 3-17 TCP 连接的建立和拆除

另外，在建立的连接的时候，假如客户主机收到 RST 的标志位等于 1，说明服务器端不存在接受该报文段的 socket，或者换种说法，服务器端没有该对应的端口号可用。

nmap 软件正是对使用这种原理对任意主机的端口号进行扫描：

nmap 给某主机发送 TCP SYN 报文段，

- 第一种结果：源主机从目标主机中收到 TCP SYNACK 报文段，说明该端口可用；
- 第二种结果：源主机从目标主机中收到 TCP RST 报文段，说明该端口不可用；
- 第三种结果：源主机从目标主机中什么也没有收到，说明中间被防火墙隔了。

3.7 拥塞控制原理

1) 拥塞的征兆

丢包一般是由于网络拥塞造成路由器缓存溢出所导致的，所以分组重传（或者三个冗余确认）可以看作是网络拥塞的征兆。

2) 拥塞所造成的代价

- 当分组的到达速率接近链路容量的时候，分组经历巨大的排队时延；
- 发送方需要执行不必要的重传操作；
- 由于大时延造成路由器需要利用其带宽转发不必要的重传副本；
- 当下游路由器由于拥塞丢弃分组是相当于浪费上游路由器转发该分组的带宽

3) 拥塞的控制方法

- 端到端的控制

网络层无法对端到端的控制任何显式的帮助，只能通过 TCP 协议自己的控制。

- 网络辅助(主要是路由器)的拥塞控制

路由器主要有两种方式进行控制：

- 直接控制：直接向发送方反馈；
- 间接控制：在发送方发给接收方的报文段中添加网络拥塞的信息，再经由接收方接收后，再传给发送方；

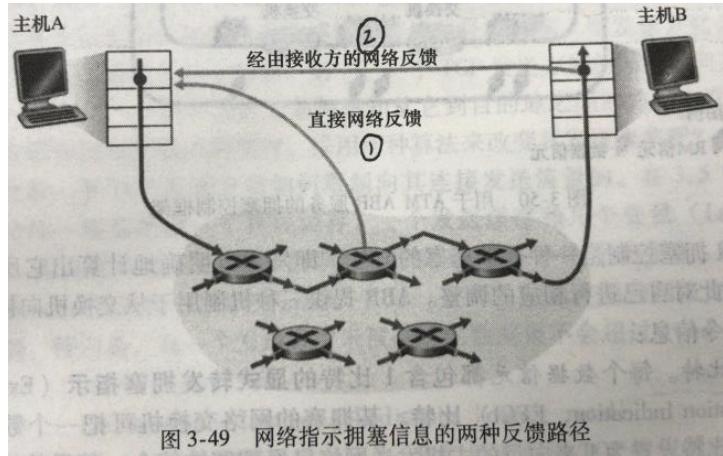


图 3-18 路由器控制的两种方式

- 4) ATM(异步传递方式)-ABR(可用比特率)服务框架
 - 基于速率的反馈方式
 - 分组在改服务框架下称为信元 cell，路由器称为交换机；
 - 信元主要有两类。一类是由发送方产生的，叫数据信元，另一类是由交换机产生的，叫 RM 信元 (Resource Management Cell)
 - 在数据信元上有一个比特叫来显示转发拥塞指示 Explicit Forward Congestion Indication，叫 EFCI 比特，数据信元在发送过程中，如果遇到拥塞，可由交换机修改 EFCI 比特为 1 表示拥塞。如果在接收方接到多个数据信元上 EFCI 比特为 1，则置 CI 比特为 1。
 - 每个 RM 信元上有一个 CI (Congestion Indication，拥塞指示) 比特，NI (No Increase，无增长) 比特和一个包含两字节的显式速率字段 ER(Explicit Rate)。
当网络轻微拥塞的时候设置 NI 比特为 1，当网络严重拥塞的时候设置 CI 比特为 1，当 RM 信元每经过一个交换机的时候，也许会降低 ER 字段所包含的值，通常会减 1，以使得能显示整条链路的瓶颈速率，即吞吐量。
最终 RM 信元会通过交给接收方再返回给发送方。

3.8 TCP 的拥塞控制

- 1) 总体策略
 - 一个丢失的报文段意味着网络拥塞，需要减慢发送速度；
 - 发送方收到一个确认的 ACK，意味着网络不拥塞，可以加快发送速度；
 - 贷款探测，其实就是尝试加快发送速度，当遇到丢包的时候再减慢发送速度；
- 2) 拥塞窗口和约束条件

之前我们推导出接收方缓存不被溢出的条件：

$$rwnd \geq \text{LastByteSent} - \text{LastByteAcked}$$

接下来我们加上拥塞的约束条件，设置 cwnd 为拥塞窗口，则

$$cwnd \geq \text{LastByteSent} - \text{LastByteAcked}$$

综合以上条件，得

$$\text{LastByteSent} - \text{LastByteAcked} \leq (\min(\text{cwnd}, \text{rwnd}))$$

3) 初始条件

cwnd初始值设置为一个比较小的值 MSS

丢包现象定义为超时或者接收方收到三个冗余的 ACK。

发送速度可以粗略定义为：

$$Velocity_{sent} = \frac{\text{cwnd}}{TRR}$$

4) 三个阶段

➤ 慢启动阶段

在每接收一个 ACK，则在下一个 RTT 阶段加倍报文数。

$$\text{cwnd} = 2^{\Delta RTT - 1} \times \text{MSS}$$

直至出现丢包，一旦出现丢包，则把阀值 ssthresh 设置为当前 cwnd 值的一半：

$$\text{阀值 ssthresh} = 0.5 \times \text{cwnd}$$

然后把 cwnd 置 1

$$\text{cwnd} = 1$$

再进入慢启动阶段，直至

$$\text{cwnd} \geq \text{ssthresh}$$

此时转入拥塞避免阶段。

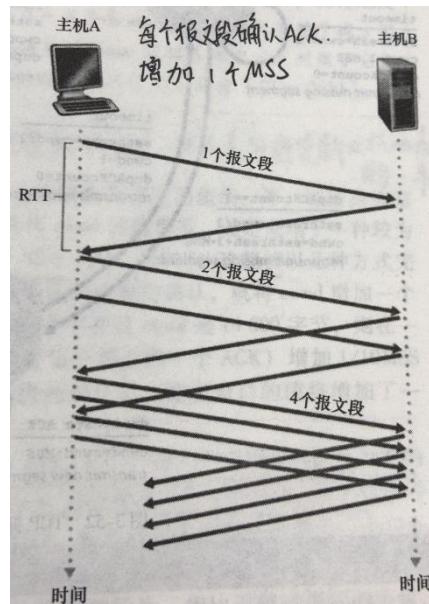


图 3-19 慢启动阶段

➤ 拥塞避免阶段

在此阶段，每一个 RTT 阶段增加一个 MSS

$$\text{cwnd} = \Delta RTT \times \text{MSS}$$

直至出现丢包，一旦出现丢包，则把阀值 ssthresh 设置为当前 cwnd 值的一半：

$$\text{阀值 ssthresh} = 0.5 \times \text{cwnd}$$

接着有两种思路：

第一种思路是 Tahoe 算法：把 cwnd 置 1

$$cwnd = 1$$

然后进入快速恢复阶段

第二种思路是 Reno 算法：把 cwnd 减 3

$$cwnd = cwnd - 3$$

其实我也不知道是否减 3，看图是这样的，书上没有写详细

然后进入拥塞避免阶段

➤ 快速恢复阶段

重传后再次进入慢启动阶段。

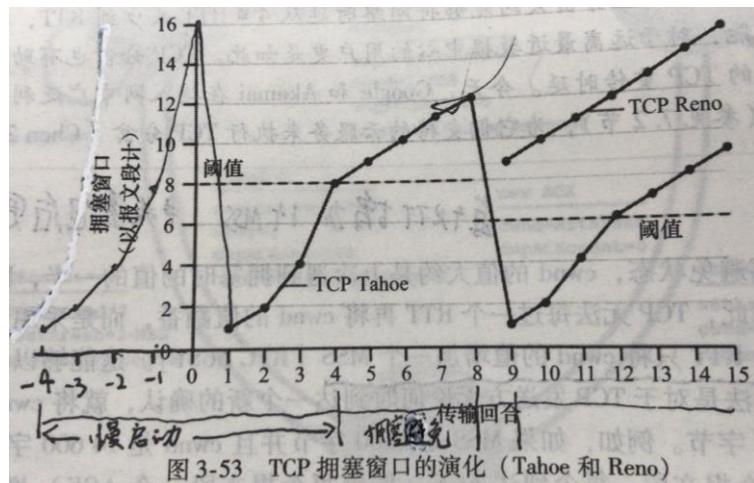


图 3-20 TCP 拥塞窗口的演化

3.9 TCP 吞吐量的宏观描述

1) 一条连接的平均吞吐量

$$\text{一条连接的平均吞吐量} = \frac{0.75 \times W}{RTT}$$

$$\text{一条连接的平均吞吐量} = \frac{1.22 \times W}{RTT\sqrt{L}}$$

其中： L 为丢包率

2) 公平性

四、 网络层

4.1 概述

1) 转发与路由选择的区别

- 转发 forward, 指的是分组从一条输入链路接口转移到适当的输出链路接口的路由器本地的动作；
- 路由选择：指网络范围内，已决定分组从源到目的地所采取的端到端的

路径；

- 2) 链路层交换机 VS 路由器
 - 链路层交换机是一台通用分组交换设备，他根据分组首部字段的值，从输入链路接口到输出链路接口转移分组，这是基于链路层字段中的值做转发决定的。
 - 而路由器则是基于网络层数据报的值做转发决定的。
- 3) 因特网服务模型是一种“尽力而为”的模型
对比其他的模型 ATM 的 CBR, Constant Bit Rate 恒定速率模型, ABR, Available Bit Rate, 可用比特率

表 4-1 因特网、ATM CBR 和 ATM ABR 服务模型

网络体系结构	服务模型	带宽保证	无丢包保证	有序	定时	拥塞指示
因特网	尽力而为	无	无	任何可能的顺序	不维护	无
ATM	CBR	保证恒定速率	是	有序	维护	不出现拥塞
ATM	ABR	保证最小速率	无	有序	不维护	提供拥塞指示

图 4-1 因特网和 ATM CBR 和 ATM ABR 的区别

其中，ATM CBR 主要用于音频和视频，因为可以保证恒定的比特率

- 4) 虚电路 Virtual-Circuit 和数据报网络 Datagram Network
VC 其实是面向端对端的有连接的服务；
而数据报网络 Datagram Network 则是无连接服务；
因特网的这服务其实跟运输层的 TCP 协议和 UDP 协议的有连接和无连接服务有点类似，但是又完全不一样。

4.2 数据报网络和虚电路网络

- 1) 数据报网络
转发表中转发的规则按照最长前缀匹配原则，前缀指的是目的 IP 地址的前缀，也就是前面的 N 个高位，一般来讲，一个 IP 地址由 32 个比特组成
- 2) 虚电路网络组成
 - 从源到目的主机的路径；
 - 沿着路径每条链路有一个 VC 值；
 - VC 号每经过一个路由器由其内部的转发表所更新

入接口	入 VC 号	出接口	出 VC 号
1	12	2	22
2	63	1	18
3	7	2	17
4	97	3	87
...

图 4-2 路由器的内部转发表

那问题来了，为什么同一条路径不使用同一个 VC 号呢？因为要维护一个唯一的 VC 号呢？这是因为这样做的维护成本太高了。

- 3) 虚电路网络三个不同的阶段
 - 建立阶段
网络层决定发送方和接收方时间的路径，然后沿着路径，每经过一个路由器，就会再其转发表中添加一个表项，该连接项更新下一条链路的 VC

号。

➤ 数据传输

一旦虚电路建立，分组便沿着虚电路的路径传输；

➤ 虚电路拆除

每释放一个链接，删除路径中的路由器转发表相应的表项；

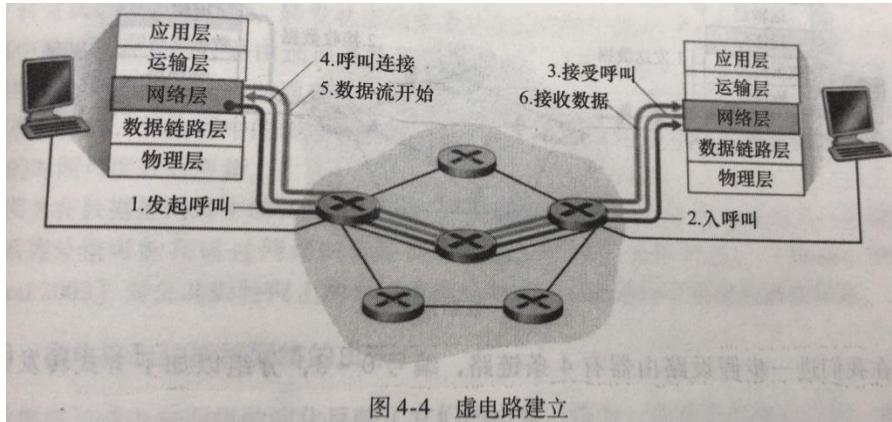


图 4-4 虚电路建立

图 4-3 虚电路的建立

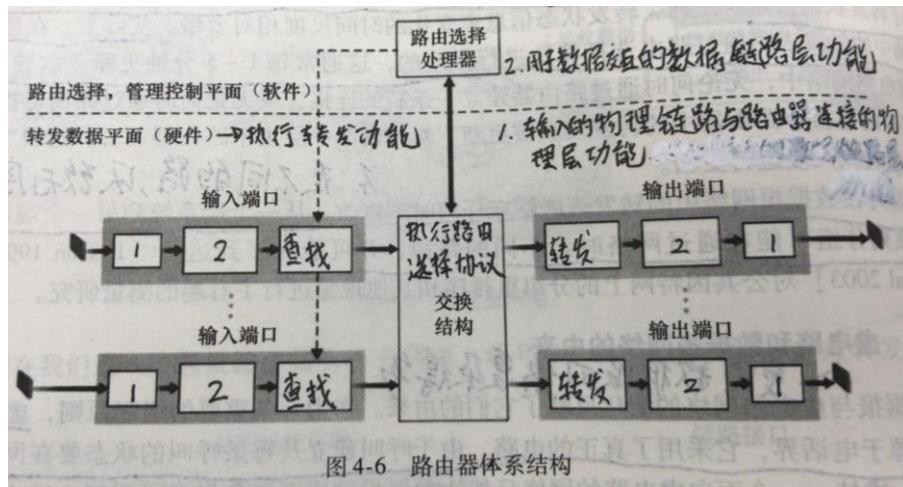
4) 注意事项

- 虚电路路径上的路由器都知道经过它的所有虚电路
- 端系统发起指示虚电路启动和终止的报文，和路由器之间传递的用于建立虚电路的报文，都叫做信令报文 *signaling message*，用于交换这些报文的协议叫做信令协议 *signaling protocol*.
- 虚电路网络要比数据报网络要复杂，因为虚电路网络要维持路径中所有经过的路由器的连接表状态，而数据报网络只需要维持转发表，转发表大概每隔 1~5 分钟更新一次，而连接表的状态每条连接的建立和释放都会更新；
- 虚电路的概念最初来自电话真实的有线电路。

4.3 路由器工作原理

1) 路由器的体系结构

我们常说的执行转发功能，专业的术语叫“转发数据平面”，值得重视的是，这种转发的执行，采用硬件去实现比采用软件去实现要快很多。
中间的交换结构执行路由选择协议。



2) 输入端

输入端前面三个框主要解决

- 输入链路与路由器连接的物理层功能；
- 用于数据交互的数据链路层功能；
- 查找，转发和排队。

正是在第三个框通过查找转发表找到转发的端口

影子副本：路由器的转发表由路由选择处理器负责计算和更新，改转发表在输入端有影子副本，为什么这样做呢？目的是避免所有的处理都集中在处理器，减轻处理器负担。而这个影子副本通过独立的总线如 PCI 总线复制到线路卡

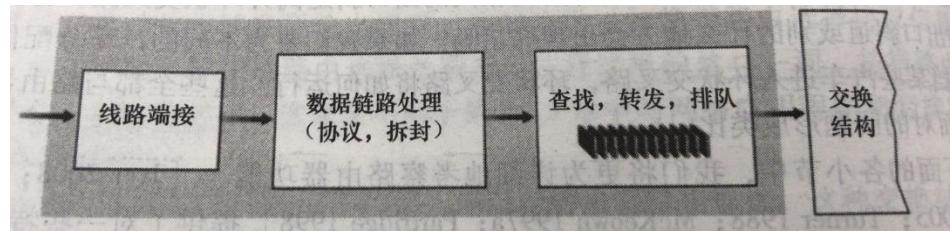


图 4-4 输入端模型

3) 交换结构

路由选择控制平面，采用分布式算法，网络中的所有路由器通过互相发送报文取得联系，及时根据协议更新自己手中的转发表。

三种交换技术：

- 内存交换
分组来到交换结构引发中断，内存复制分组，从分组中提取首部信息，计算输出端口号，将分组复制到输出端缓存，解除中断。
但是一次只能发送一个。

➤ 总线交换

分组预先添加一个标签，然后复制到所有的输出端，但是只有标签匹配的才能保存该分组。

但是一次只能发送一个。

➤ 经互联网络交换

纵横式的分布可以客服单一，共享式总线带宽的限制。但是面对不同输入端相同输出端的情况，也只能只有一个通过。

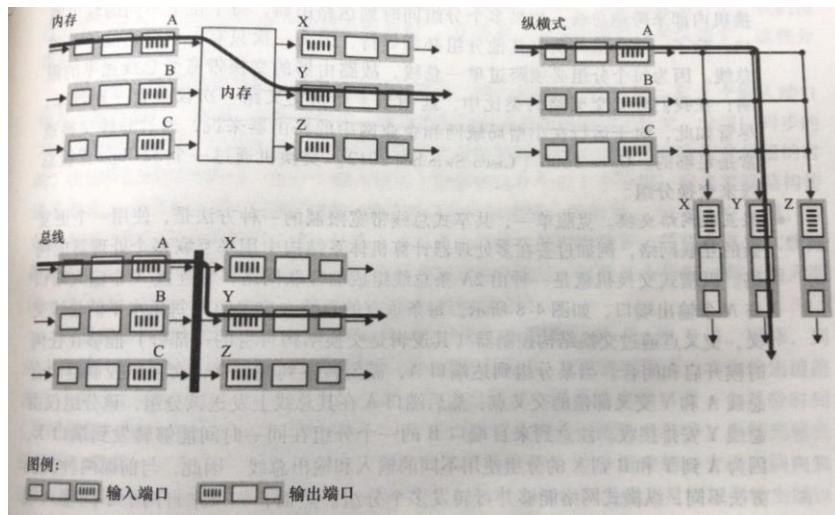


图 4-5 三种交换技术

4) 输出端

输出端处容易出现排队，一旦排队溢出缓存，就会出现丢包 packet loss 的现象。为了贯彻“尽力而为的服务”，在输出端的缓存管理必须使用合适的“分组调度算法”。比较出名的算法有 FCFS 先来先服务算法，加权平均算法，主动队列管理算法（这个比较特殊，它不是使用“弃尾”的做法，而是主动选择丢弃队列前面的分组，好让告知其发送方网络正处于拥塞状态），随机早期检测算法

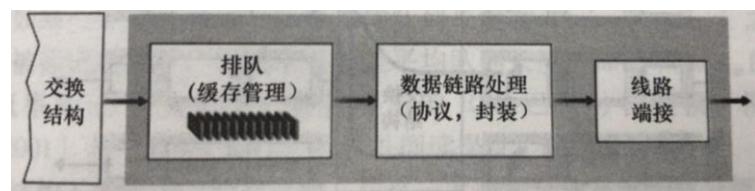


图 4-6 输出端口的处理

5) HOL 拥塞

head of line, 线路前部拥塞，意思是即便输出端口是空的，但是由于排在前面的分组收到阻塞了，自己也被阻塞了。

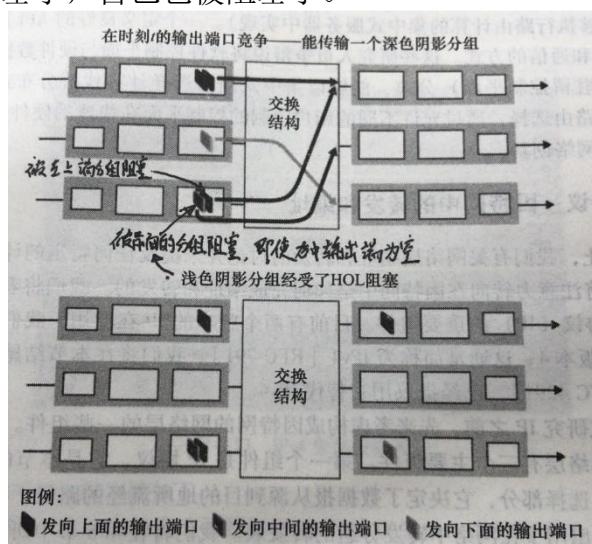


图 4-7 HOL 拥塞

4.4 IPv4 的数据报格式

1) IPv4 的数据报格式

- 服务类型：由路由器管理员的策略决定；
- 一般的数据报首部有 20 个字节；
- 首部长度：指的是数据实际从哪个字节开始，一般的数据包不包含这一个参数。
- 标识，标志和偏移：这是跟 IP 分片有关的。
- 数据包长度指的是首部加上数据的总长度，以字节计，因为该字节长 16bit，故数据包的最大长度为 65536 字节，即 64KB，但是数据包很少有超过 1500 字节的
- 选项：允许 IP 首部被拓展；
- 数据：其实就是运输层的报文段；

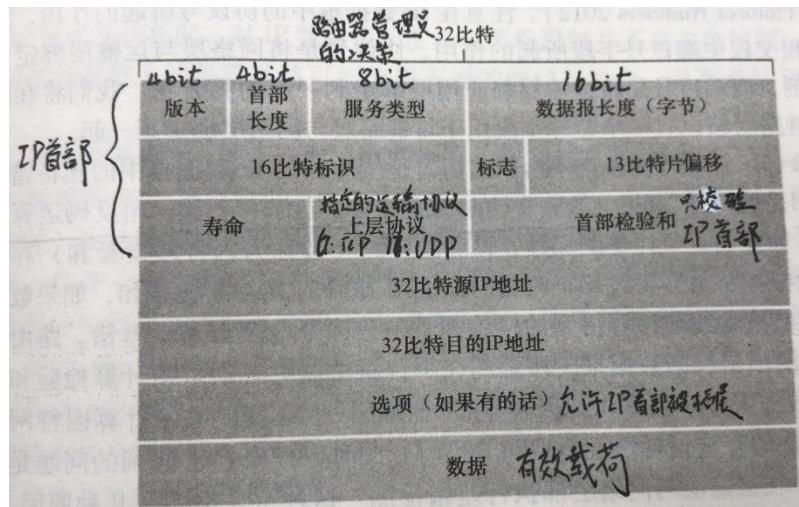


图 4-8 IPv4 的数据报格式

2) IP 校验和主要是用来保证数据 (IP 报头) 的完整性的，就是反码求和校验。需要注意的是反码求和又叫 1 的补码，而 2 的补码就是我们通常说的补码求和了。校验算法具体如下。

1、发送方

- i) 将校验和字段置为 0, 然后将 IP 报头按 16 比特分成多个单元，如包头长度不是 16 比特的倍数，则用 0 比特填充到 16 比特的倍数；
- ii) 对各个单元采用反码加法（也就是前面所说的“回卷”）运算(即高位溢出位会加到低位, 通常的补码运算是直接丢掉溢出的高位), 将得到的和的反码填入校验和字段；
- iii) 发送数据包。

2、接收方

- i) 将 IP 包头按 16 比特分成多个单元，如包头长度不是 16 比特的倍数，则用 0 比特填充到 16 比特的倍数；
- ii) 对各个单元采用反码加法运算，检查得到的和是否符合是全 1(有的实现可能对得到的和会取反码，然后判断最终值是不是全 0)；
- iii) 如果是全 1 则进行下步处理, 否则意味着包已变化从而丢弃之。

需要强调的是反码和是采用高位溢出加到低位的，如 3 比特的反码和运算：
 $100b + 101b = 010b$ (因为 $100b + 101b = 1001b$, 高位溢出 1, 其应该加到低位, 即 $001b + 1b$ (高位溢出位)= $010b$)。

4.5 IP 数据包的分片

1) 分片的原因

由于链路层的协议很多，每种协议它有限制最大传输单元 Maximum Transmission Unit, MTU，比如以太网协议中的 MTU 为 1500 个字节，而有的广域网协议的 MTU 只有 576 个字节。

为了解决数据报的长度大于 MTU 的长度的问题，需要把 IP 数据包分片

2) 执行分片操作的位置

在路由器上进行分片，在目标主机上进行组装。源主机在发送数据报的时候，给每个数据报的标识号+1，然后再在路由器上进行分片的操作。

3) 分片操作

- 标识：数据包的序号
- 标志：在每个数据包中，除了最后一个分片设置为 0，其他分片都设为 1。
那为什么这么做呢？就是为了确定收到最后一个分片了；
- 偏移量：每个数据包中该分片的起始位置，以 8 个字节为一个单位；

4) 注意

当接收数据报的分片不完整的时候，端系统会把它丢掉，如果运输层采用的是 TCP 协议的话，则报文段能够重传；

到了 IPv6 之后，这种分片的操作就不再采用了；

4.6 IPv4 编址

1) 点分十进制记法

通常一个 IP 地址是 32bit，比如 11000001 00100000 11011000 00001001，然后每八个比特用一个十进制数表示，每个十进制数之间用点“.”隔开，得：

193.32.216.9

2) 子网

以路由器接口或端系统为端点组成的链路段或者链路网叫做子网

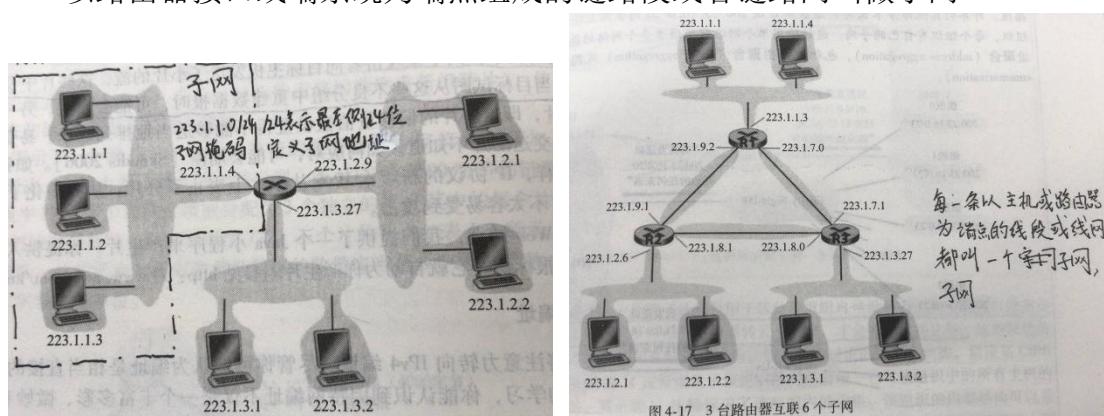


图 4-9 子网范例

特性：子网中的端系统或者路由器端口拥有相同的最大前缀匹配，即子网掩码 network mask，该匹配的比特数称为 x ；
端系统能知道子网中的子网掩码，默认网关（下一跳路由器的地址），地方 DNS 服务器地址

3) 地址分配策略

- 无类别域间路由选择（Classless Interdomain Routing, CIDR）
默认表示为

a. b. c. d/ x

- 分类编址（classful addressing）
A 类，子网地址有 8 位，即 $x=24$;
B 类，子网地址有 16 位，即 $x=16$;
C 类，子网地址有 24 位，即 $x=8$;
- 广播 IP 地址
225.225.225.255，通知子网中的所有主机

4) 组织

一个组织通常能获得一片连续的地址，内部可以包含有多个不同的子网。

4.7 DHCP 动态主机配置协议

1) DHCP 动态主机配置协议 Dynamic Host Configuration

当一个组织获得一块连续的区域之后，他可以动态为子网中的主机和路由器端口分配 IP 地址。DHCP 允许主机每次连接网络的时候自动获取 IP 地址，或者被网络管理员分配 IP 地址。而且，DHCP 还允许主机知道子网地址，子网掩码（下一跳路由器的地址）和地方 DNS 服务器地址。

DHCP 是一个客户-服务器协议，每个子网需要一台服务器，或者一个路由器中继代理（它知道该网络 DHCP 的地址）

2) DHCP 分配步骤

➤ 询问参数

新主机首先设置自己的 IP 地址为 0.0.0.0，然后使用 68 端口向 225.225.225.255 的 67 端口广播“寻人启示”，发送 DHCP 发现报文（其中包含有一个事务 ID）寻找子网中的 DHCP 服务器。

➤ 回答参数

附近的一个 DHCP 服务器接收到具有特定事务 ID 的发现报文后，向 225.225.225.255 的 68 端口广播，返回 DHCP 提供报文，里面包含按照 DHCP 协议生成的 IP 地址和有效期；

➤ 回显参数

接着新主机收到广播后，返回 DHCP 请求报文，事务 ID=事务 ID+1，实质上就是回显参数之用；

➤ 确认参数

DHCP 服务器接收后，就广播返回 DHCP ACK 报文，说这个 IP 地址可以用的。

经过四轮之后，新主机就可以使用新的 IP 地址了，但是这个 IP 地址一般都是有租用期限的，不过 DHCP 可以提供 IP 地址更新租用期限的机制。

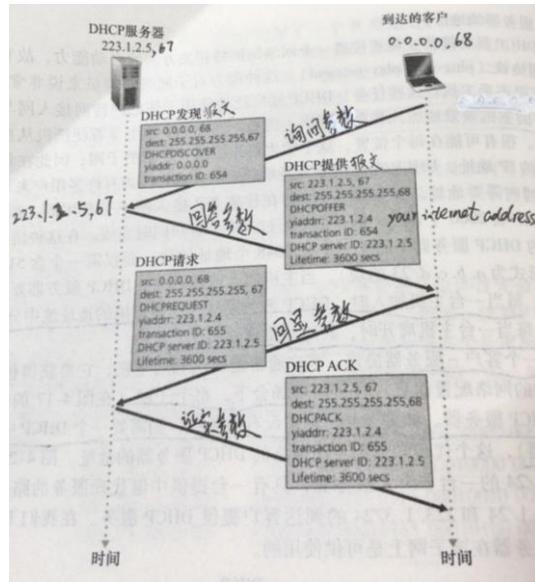


图 4-10 DHCP

4.8 NAT 网络地址转换

背景：全球唯一的 IP 地址不够用了

1) 实现机理

用一台路由器或者服务器作为中继代理，代理的一侧接入 WAN，拥有唯一的 IP 地址，然后另一侧接入 LAN 各个端系统。每个 LAN 端系统的进程想要访问 WAN 的时候，中继代理就会把端系统局域网 IP+端口号，映射成中继代理的广域网 IP 地址+端口号，而中继代理会把这张 NAT 转换表保存在自己那里。

其实质就是用端口号来表示局域网中的所有主机中的进程。由于端口号只有 16 位，所以共可以表示 65536 个不同主机上的不同进程。

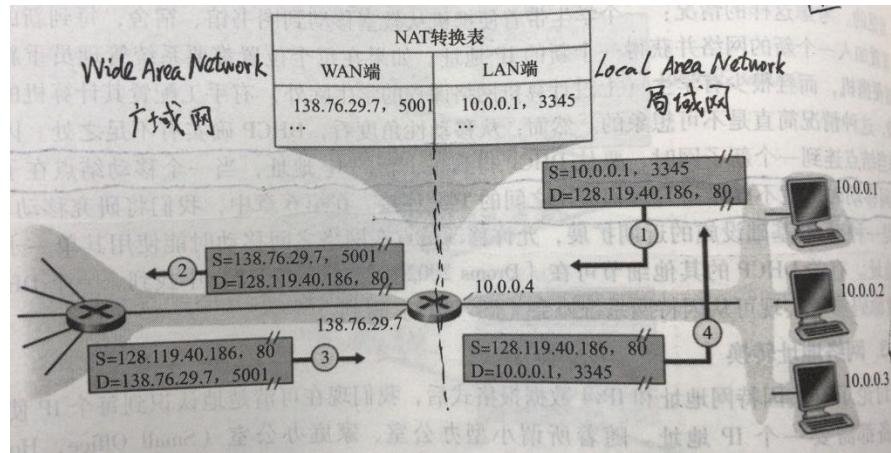


图 4-11 NAT

2) 质疑

- 端口号是用来为进程编址的，而不是用于主机编址的；
- 路由器最多只能在网络的第三层进行服务；
- NAT 协议违反了端到端的原则，不应在中间节点修改主机 IP 地址和端

口号；

➤ 他们认为应该使用 IPv6 解决 IP 地址不足的问题；

3) 主要问题

妨碍 TCP 协议；

比如主机 A 想要找找主机 B，于是主机 A 发起一个 TCP 连接，发送 SYN 报文，结果被主机 B 的中继代理给挡住了，因为中继代理没有为主机 B 事前分配特定的端口号，端口号都是被动分配的（即只有主机 B 有需求的时候才进行分配），所以主机 A 就没办法直接找到主机 B 了。

一种解决办法是通过“第三者”事先跟主机 B 建立通信，然后主机 A 再通过“第三者”跟主机 B 建立通信，这种方法又叫做 NAT 穿越（NAT traversal），这种雇佣关系又叫做连接反转（connection reversal）

4) 解决办法——UPnP

关键——追踪器

追踪器先会把主机 B 的某个进程映射到特定的中继代理端口号；接着当主机 A 去找主机 B 的时候，主机 A 就在中继代理那里的追踪器中寻找对应进程的端口号-进程地址的映射，从而找到主机 B，再建立 TCP 映射。这也叫作 UPnP 穿透。

4.9 ICMP 因特网控制报文协议

- 1) 在主机和路由器之间通信用的报文，典型的用处是差错报告。
- 2) ICMP 协议位于 IP 协议之上，证据是 ICMP 报文出现在 IP 数据包的有效载荷中
- 3) 结构
- 4) 典型应用：ping

源主机向目的主机发送一个 ping 请求，即 ICMP 类型=8，编码=0，这时如果目的主机接收到的话，就会返回 ICMP 类型=0，编码=0 的保温

ICMP 类型	编码	描述
0	0	回显回答（对 ping 的回答）
3	0	目的网络不可达
3	1	目的主机不可达
3	2	目的协议不可达
3	3	目的端口不可达
3	6	目的网络未知
3	7	目的主机未知
4	0	源抑制（拥塞控制）
8	0	回显请求 <i>ping 主机</i>
9	0	路由器通告
10	0	路由器发现
11	0	TTL 过期
12	0	IP 首部损坏

图 4-12 ICMP 类型和编码

5) 典型应用：源抑制报文

之前我们学运输层的时候，学过拥塞控制原理，除了利用进程控制反馈拥塞信息外，类似路由器自身返回的 RM 信元，虽然我不知道这两者有没有直接

的关系。这种报文可以强制减少源主机发生分组的速率。

6) 典型应用：Traceroute

进程有一个计时器，计时器每加 1，就发送一个可到达主机但不可到达主机端口号的 UDP 报文段，当计时器技术为 n 时报文段到达第 n 个路由器，这时候报文段上的 TTL 刚好过期（我猜那个 TTL 是没经过一个路由器都会减 1，所以当刚发出的 TTL 为 n 经过第 n 个路由器的时候才会刚好失效）。由于 TTL 过期，路由器会返回一个 ICMP=11，编码为 0 的报文，而且该报文还包含有该路由器的名字和 IP 地址到源主机，直至最后一个报文到达主机，但是由于这个端口号找不到，所以目标主机会返回一个目的端口不可到达的 ICMP 报文（ICMP=3，编码为 3）。这时候源主机就知道其与目标主机之间的往返时间 RTT 和路径的路由器数量和标识。

4.10 IPv6 编址

1) 格式

- 首部一共 40 字节，比 IPv4 要少；
- 流量类型：跟 IPv4 中的服务类型类似，指示路由器给数据报的服务优先级；
- 流标签：双方可以互相约定，流标签的类型是什么？新闻？视频？音频？还是优先级类型等
- 有效载荷长度：就是数据的字节数啦
- 下一个首部：指示交给那个运输层协议，TCP 或者 UDP 或者是其他；
- 跳限制：其实就是 IPv4 中的寿命 TTL Time-to-live，没经过一个路由器减 1，直到为 0 的话会被路由器抛弃。并返回 ICMP 报文指示该数据报以过期；
- 地址有原来的 32 比特扩充到 128 比特；
- 在中间路由器中不再执行分片和组装的工作，这一工作统一安排在端系统上执行，如果在链路中由于分组过大而无法传输的话，路由器会根据更新过的 ICMP 协议 ICMPv6 发送分组过大的警告 ICMP 报文。这个警告也是在 v6 版新增的。

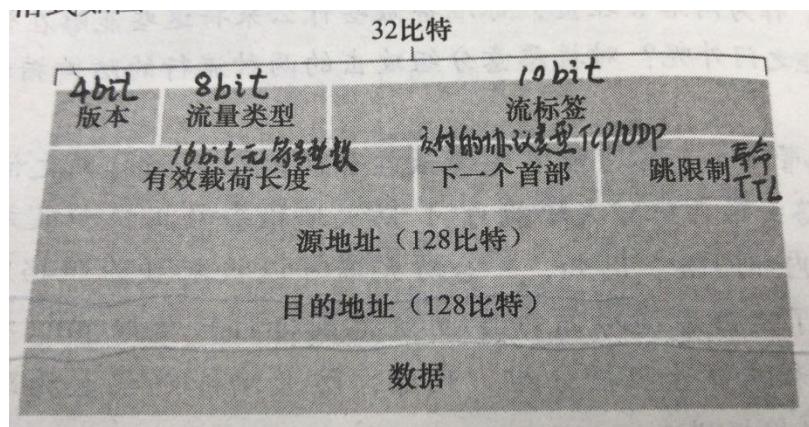


图 4-13 IPv6 格式

2) IPv4 到 IPv6 的迁移

其实迁移最大的问题就是兼容性的问题，这里采用的解决方案是双栈（dual-

stack), 意思是只要两个端系统或者路过的路由器存在只能执行 IPv4 协议的, 那么整条链路都采用 IPv4 协议, 否则会存在信息的丢失。

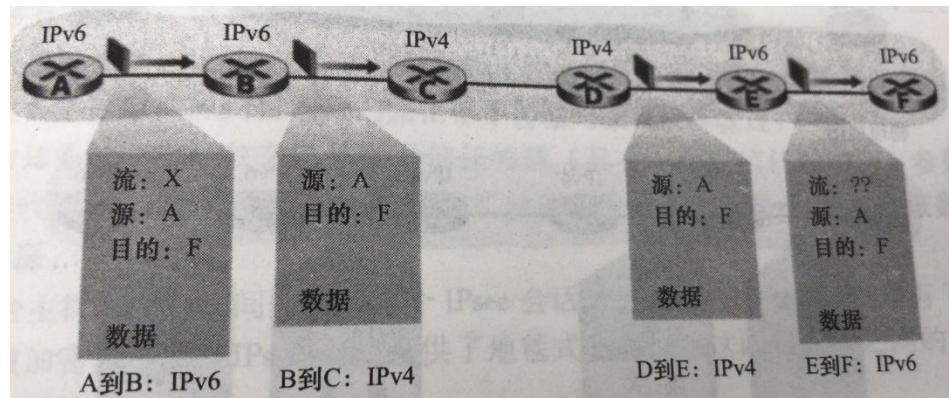


图 4-14 IPv6 转 IPv4 格式的信息丢失

为了避免这个问题, 人们又想出了“建隧道”的方法, 就是把整个 IPv6 的数据报作为有效载荷放在 IP4 的数据报上进行传输。

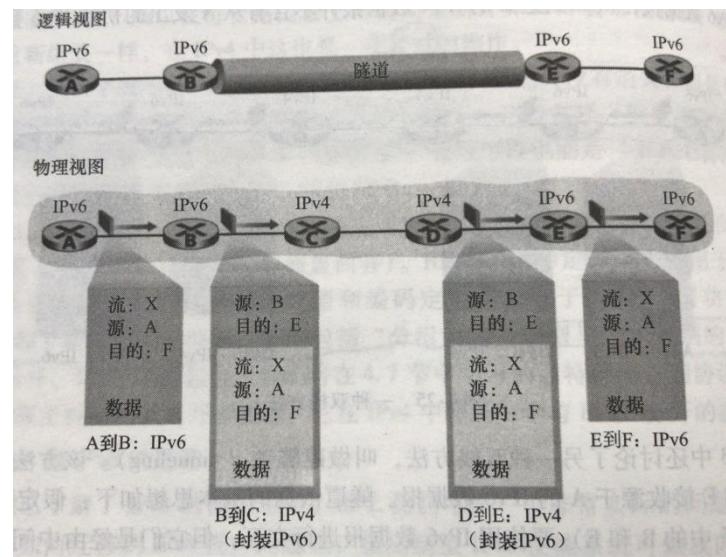


图 4-15 建隧道

4.11 路由选择算法

- 1) 端系统的第一跳路由器叫“默认路由器”
- 2) 算法
 - 全局式路由选择算法
 - 链路状态算法 Link State, LS 算法
 - 采用迭代, 从前一个节点最短距离到该节点的最短距离。
 - 分散式路由选择算法
 - DV 算法 Distance-Vector, DV
 - Bellman-Ford 方程
 - $$d_x(y) = \min_v \{c(x, v) + d_v(y)\}$$
- 3) 后面的算法就比较复杂了, 感觉用处不是很大, 有时间再看吧

五、链路层：链路，接入网和局域网

5.1 链路层提供的服务

1) 成帧服务

2) 链路接入服务

依靠媒体访问控制协议 Medium Access Control, MAC, 决定链路接入访问的规则，是一对一呢？还是一对多？多对一？多对多？还有怎么接入的问题。

3) 可靠服务

但是这里的可靠服务不是像运输层那样经过端对端的数据重传实现的，而是依靠数据在链路层传输的确认和重传，确保数据在链路层传输的可靠。这种服务在无线网这种容易出错的链路中应用有很大的好处，但是在物理导线这种低错率的链路传输时就会有点浪费。

4) 差错检测和纠正

➤ 校验和；

➤ 奇偶检验；

➤ 奇检验，就是在帧的最后再添加一位，如果前面的包含 1 的位数是偶数的时候，则最后一位为 1，否则为 0，保证含 1 的总位数为奇数；偶检验同理。但是这样只能保证出错奇数位时可以检验出来，而偶数位的时候就检验不出来。

➤ 循环冗余检测 CRC

<http://blog.csdn.net/wenqiang1208/article/details/71641414>

为什么引入 CRC

现实的通信链路都不会是理想的。这就是说，比特在传输的过程中可能会产生差错：1 可能会变成 0，0 可能会变成 1，这就叫做比特差错。在一段时间内，传输错误的比特占所传输比特总数的比率成为误码率 BER(Bit Error Rate)。误码率与信噪比有很大的关系，在实际通信中不可能使误码率下降到零。

因此，为了保证数据传输的可靠性，在计算机网络传输数据时，必须采用各种差错检测措施。

目前在数据链路层广泛使用了循环冗余检测 CRC 的检测技术

CRC 的原理

CRC 运算实际上就是在数据长为 k 的后面添加供差错检测用的 n 位冗余码，然后构成帧 k+n 位发送出去。

首先来介绍几个概念

(1) 模 2 运算：实际上是按位异或运算，即相同为 0，相异为 1，也就是不考虑进位、借位的二进制加减运算。如： $1111+1010=0101$

(2) FCS：其实就是冗余码，帧检验序列(Frame Check Sequence)

(3) 生成多项式：其实就是除数，比如下面将要用到的除数 $p = 1101$

$P(X) = X^3 + X^2 + 1$ 表示上面的除数 $P = 1101$ (最高位对应于 X^3 , 最低位对应于 X^0)。多项式 $P(X)$ 称为生成多项式。现在广泛使用的生成多项式 $P(X)$ 有以下几种:

$$\text{CRC-16} = X^{16} + X^{15} + X^2 + 1$$

$$\text{CRC-CCITT} = X^{16} + X^{12} + X^5 + 1$$

$$\text{CRC-32} = X^{32} + X^{26} + X^{23} + X^{22} + X^{16} + X^{12} + X^{11} + X^{10} + X^8 + X^7 + X^5 + X^4 + X^2 + X + 1$$

计算 n 位冗余码

现假定待传输的数据 $M = 101001(k = 6)$, 除数 $p = 1101(n = 3)$ 比 n 多一位

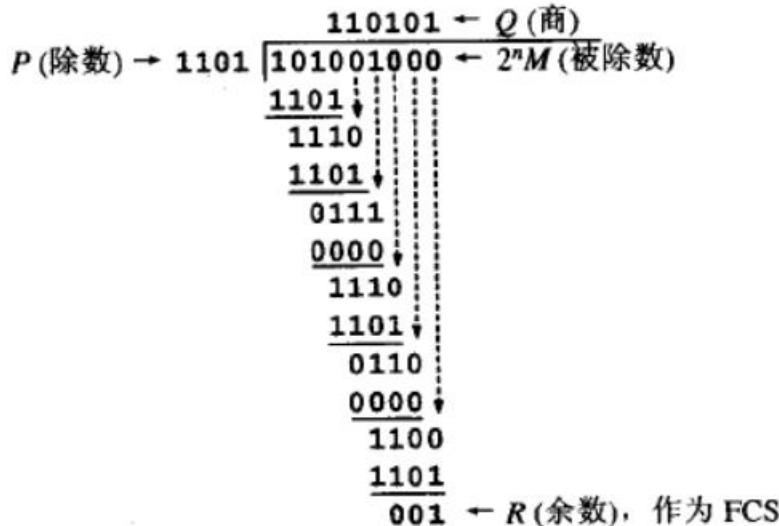
这 n 位冗余码可以用下面的方法得出。

(1) 用二进制的模 2 运算进行(2^n)乘 M 的运算, 相当于在 M 后面添加 n 个 0。

即 M 后面添加 3 个 0

(2) 现在得到 $M = 101001000(k+n = 9)$ 位的数除以除数 $p(n = 3)$ 位, 得到商是 Q (不关心), 余数 $R = 001$ (n 位) R 就是冗余码 FCS

现在加上 FCS 后发送的帧是 101001001



在接收端把接收到的数据 $M = 101001001$ 以帧为单位进行 CRC 检验: 把收到的每一个帧都除以相同的除数 p (模 2 运算), 然后检查得到的余数 R 。

如果在传输过程中没有差错, 那么经过检验后得到余数 R 肯定是 0。

(读者可以自己检验下, 被除数现在是 $M = 101001001$, 除数 $P = 1101$, 看余数是否为 0)

总之, 在接收端对接收到的每一个帧经过 CRC 检验后, 有两种情况:

(1) 余数 $R = 0$, 则判断这个帧没有问题, 就接受

(2) 余数 $R \neq 0$, 则判断这个帧有差错, 就丢弃。

总结一下:

在数据链路层若仅仅使用 CRC 差错检验技术, 则只能做到对帧的无差错接收。

5.2 多路访问协议

1) 碰撞 collide

节点同时收到几个帧，所接收的帧在接收方处信号被纠缠在一起，结果涉及到这次碰撞的所有帧都会失效。所以，如果频繁发生碰撞，链路中的带宽就会被浪费掉。

2) 信道划分协议

➤ 时分多路复用或者频分多路服用

这种绝对的平均主义会造成一个问题，当在场的所有发送方中只有一个真正在发送帧的时候，其他带宽就会被浪费掉。

➤ 随机接入协议

它的主要思想是，一个结点总想以全速发送帧，当遇上碰撞的时候，他会等待一个随机的时间在进行重发。

载波侦听多路访问 CSMA

说话前先听如果其他人在说话，就等他说完话为止。

如果与他人同时开始说话，停止说话（在网络领域这叫“碰撞检测”）；

➤ 轮流协议

5.3 地址解析协议 ARP

1) 链路层主要体现在网络适配器

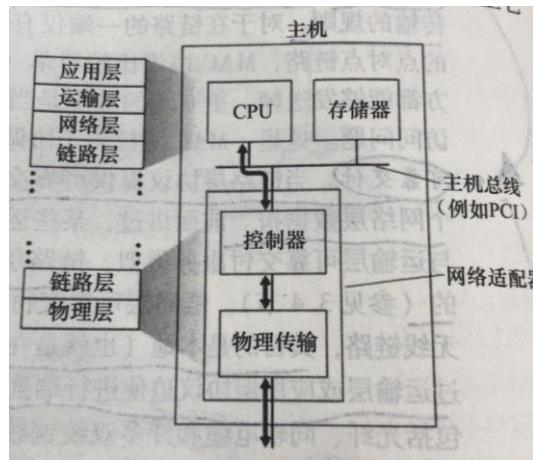


图 5-1 网络适配器

- 2) 端系统的链路层接口体现在网络适配器的接口，叫 MAC 地址。这个地址是有全球一个非盈利组织管理的，这个地址一共有 64bit，该组织分配高 32 bit 的连续地址给生产商，生产商自己可以决定低 32bit。同时采用 16 进制表示。如：

FF – FF – FF – FF – FF – FF

- 3) 当某源适配器想要向目的适配器发送一个帧的时候，会把目的适配器的 MAC 地址插入到该帧中，然后将帧发送到局域网上，广播该帧，看谁的适配器 MAC 地址与之相匹配，相匹配的话就会从封装的数据中取出数据报然后把数据往上传，如果不匹配的话就扔掉该帧。

这个用于试探的特殊帧我们叫做 ARP 分组，查询分组和相应分组用同样的格式。这个分组包含有源和目的的 IP 地址和 MAC 地址，因为一开始并不知道目的节点的 MAC 地址，所以用 FF – FF – FF – FF – FF – FF 替代，

4) ARP 表

这张表包含了 IP 地址到 MAC 地址的映射关系，这表位于交换机中。

5) ARP 和 DNS 协议异同

相同点：都用于解析地址；

不同点：DNS 协议是将主机名解析为 IP 地址，而且可以为全球因特网中的任何主机进行解析。而 ARP 协议用来把同一个子网中的主机和路由器接口解析 IP 地址。

6) 由于 ARP 既包含有 IP 地址，又包含有 MAC 地址，所以我们最好把它看成是对接网络层和链路层的双边协议。

5.4 以太网协议

1) 以太网帧结构

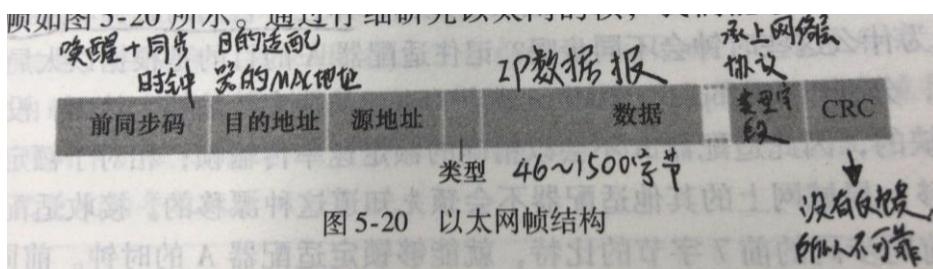


图 5-2 以太网帧结构

5.5 链路层交换机

1) 过滤

决定是否把数据转发给某个端口或者扔掉；

2) 转发

决定帧应该转发到哪个端口；

3) 如果链路层交换机没有目的 MAC 地址的表项，则需要广播该地址。

4) 自学习

其实就是老化算法，老化的时间大概是 60 分钟，即在老化时间之内，链路层交换机会保存相应的表项，超过时间之后就会清理掉。

5) 链路层交换机与路由器的比较

位于的五层因特网协议栈的层次不一样，链路层交换机位于第二层，而路由器位于第三层

链路层交换机对于没完没了的以太网广播帧流（广播风暴）不提供保护；

链路层交换机简单，即插即用，而路由器需要配置；

路由器提供防火墙的功能；

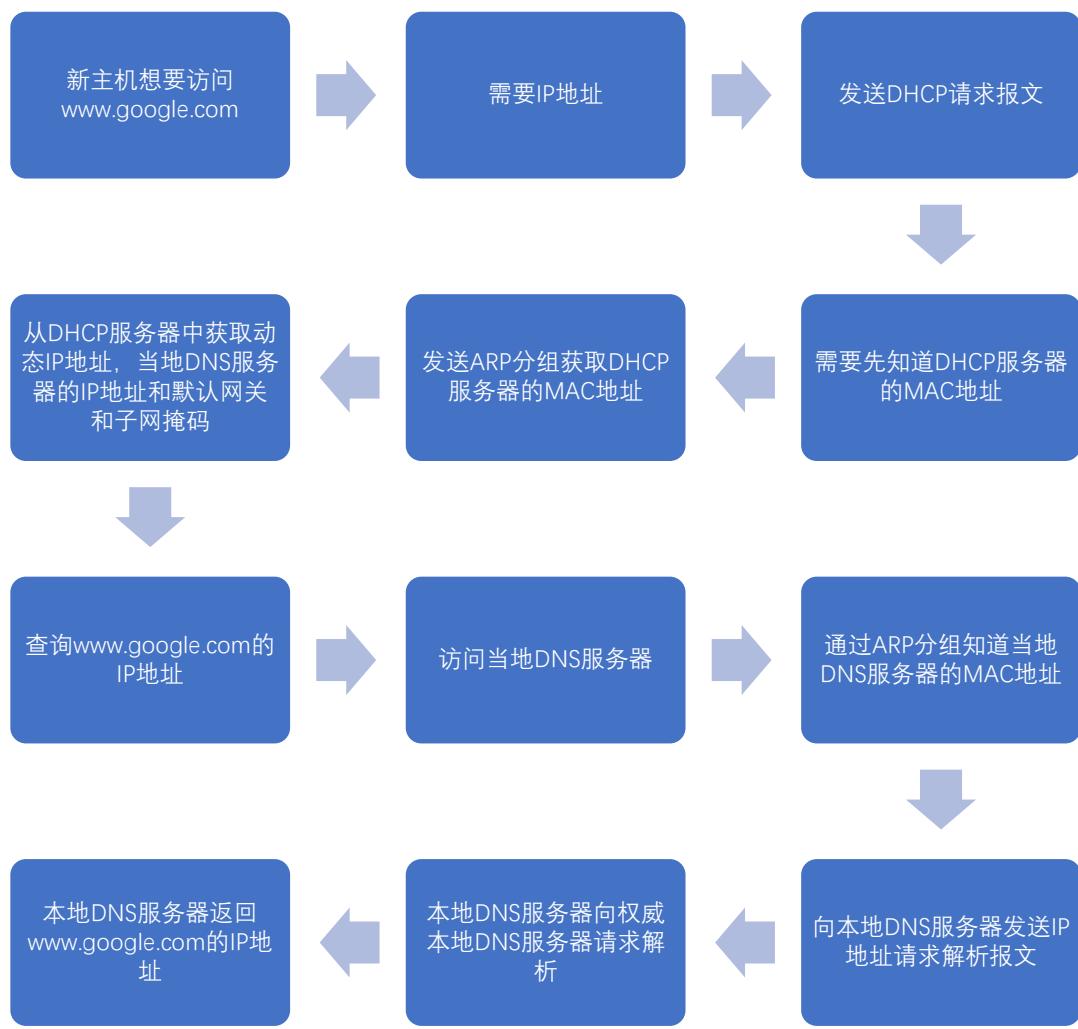
通常只有几百台主机的局域网用链路层交换机就可以了，因为它不要求 IP 地址的任何配置就能使流量局部化并增加总计吞吐量。如果达到上千台的规模

的话，建议还是用路由器；

表 5-1 流行的互联设备的典型特色的比较

	集线器	路由器	交换机
流量隔离	无	有	有
即插即用	有	无	有
优化路由	无	有	无

六、总结：访问 www.google.com



注意：广播请求的时候一般用 UDP 协议，而访问网站的时候一般使用 TCP 协议。后面还有四章的内容，由于时间不够了，以后有时间再学吧。