第一章 概述

互联网的组成以及几种网络传输方式

边缘部分：由所有连接在互联网上的主机构成。这部分是用户直接使用的，用来通讯和数据共享。计算机之间的通讯有两种，**一种是客户-服务器方式，一种是P2P方式。**

核心部分：**由大量网络和连接这些网络的路由器组成**。这部分是为边缘部分提供服务。**核心中起重要作用的是路由器，他的作用是分组交换，转发收到的分组。**

数据交换的几种方式和特点：

1.电路交换，需要进行建立连接、通话、释放连接三个过程。资源独占，线路传输的效率较低。

2.分组交换，需要将数据报文切分成较小的等长数据块，加入必要的控制信息，构成每一个分组，每一个分组独立传输。分组交换高效，灵活，迅速，可靠，但是也带来了时延和一些不必要的开销。

3.报文交换，整个报文传递到临近节点，全部存储下来后在查找转发表，然后转发到下一个节点。

电路交换——整个报文的比特流连续地从源点直达终点，好像在一个管道中传送。

报文交换——整个报文先传送到相邻结点，全部存储下来后查找转发表，转发到下一个结点。

分组交换——单个分组（这只是整个报文的一部分）传送到相邻结点，存储下来后查找转发表，转发到下一个结点。

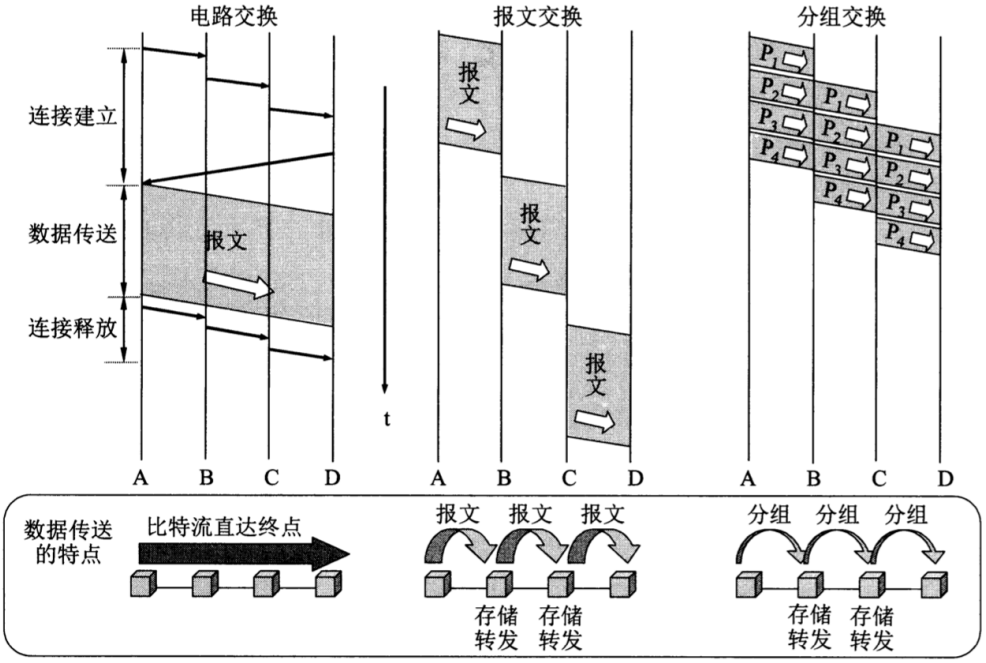


图 1-13　三种交换的比较。电路交换；报文交换；分组交换，P_1～P_4表示 4 个分组

分层

网络协议通常分不同层次进行开发，每一层分别负责不同的通信功能

TCP/IP通常被认为是一个**四层协议系统**，如下图所示

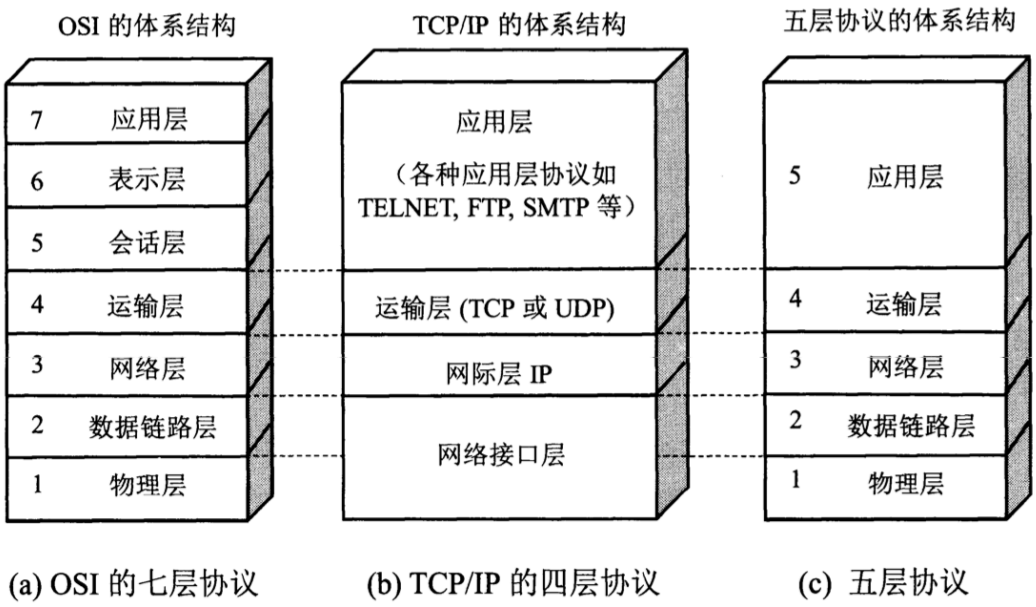


图 1-18　计算机网络体系结构

每一层负责不同的功能:

1.**数据链路层：有时候被称为网络接口层。**

2.**网络层：处理和分组相关，协议主要包含IP协议(网际协议)，ICMP协议(Internet互联网控制报文协议)，以及IGMP协议(Internet组管理协议)**

3.**运输层：为两台主机上的应用程序提供端到端的通信，主要的两个协议有TCP/UDPTCP为两台主机提供高可靠性的数据通信，UDP为应用层提供一种非常简单的服务，传输不可靠。**

4.**应用层：负责处理特定的应用程序细节。主要的应用有：Telnet、FTP、SMTP、SNMP（简单网络管理协议）**

网关的定义：一个链接两种不同协议簇的进程，为某一个特定的应用提供服务。

在TCP/IP协议簇中，**IP层使用不可靠的服务**，**TCP提供可靠的服务**，为了提供可靠到的服务，TCP采用了**超时重传，发送和接收端到端的确认分组机制**。

网桥是在**数据链路层**对网络进行互联的，**路由器**是在**网络层**对网络进行互联的。

数据的封装

当应用层使用TCP传输数据的时候，数据会被放入协议栈，每一层会被当作一串比特流送入网络，每一层会增加对应的首部。**TCP传递给IP的数据单元是TCP报文，IP传递给网络接口层的数据单元叫IP数据报，通过以太网传输的比特流称为帧。**

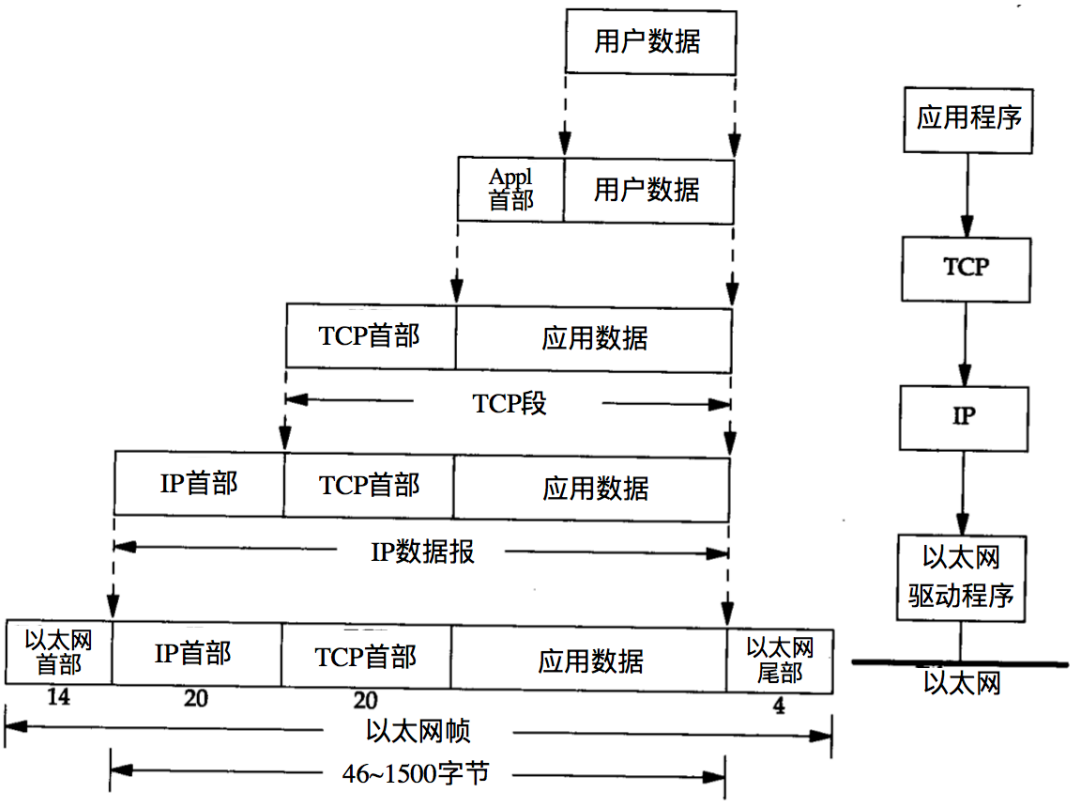


图1-7　数据进入协议栈时的封装过程

UDP数据与TCP数据基本一致，**唯一的不同点是数据首部封装的是UDP首部，长度为8字节，UDP给IP层传输的是UDP数据报。**

数据的分用与复用

当目的主机收到一个以太网数据帧时，数据就开始从协议栈中由底向上升，同时去掉各 层协议加上的报文首部。**每层协议盒都要去检查报文首部中的协议标识，以确定接收数据的 上 层 协 议 。 这 个 过 程 称 作 分 用**

端口号

TCP和UDP采用16bit的端口号进行识别应用层序。有一些知名的端口号，FTP的TCP端口号21，每一个Telnet对应的TCP端口号是23，TFTP对应的UDP端口号是69，**知名的端口号一般介于1-255之间，256-1023之间主要是Unix服务进程所占用，临时端口主要分布在1024-5000，其他保留的端口号是大于5000的。**

unix系统的端口号保存在 / etc/services中，使用一下命令可以查看端口号

grep “服务命令” / etc/services

第二章 数据链路层

数据链路层使用的信道主要有两种类型：

1.点对点信道

2.广播信道

数据链路层有三个基本问题：

1.封装成帧

2.透明传输

3.差错检测

封装成帧

就是在数据的前后添加尾部和首部，这样就构成了一个帧。

帧界定符：SOH（01）表示一个帧的开始，EOT（04）表示帧的结束。

透明传输

发送端的数据链路层在数据中出现控制字符"SOH"和“EOT”时候，需要在前面插入一个转义字符"ESC"，这种方式称之为字节填充或者字符填充。

差错控制

CRC校验

PPP协议组成

1.支持异步链路，也支持同步链路

2.建立、配置和测试数据链路连接的链路控制协议LCP

3.网络控制协议

字节填充

出现每一个0x7E字节的地方，插入0x7D，出现0x7D的地方后面插入0x5D

零比特填充

连续出现5个1，就在后面插入0

第三章 网络层

引言

IP层提供**不可靠、无连接的服务**，不可靠指的是不能保证IP数据包能够成功到达目的地；无连接表示IP数据报并不维护任何关于后续数据报的状态信息。

表 4-1　虚电路服务与数据报服务的对比

对比的方面	虚电路服务	数据报服务
思路	可靠通信应当由网络来保证	可靠通信应当由用户主机来保证
连接的建立	必须有	不需要
终点地址	仅在连接建立阶段使用，每个分组使用短的虚电路号	每个分组都有终点的完整地址
分组的转发	属于同一条虚电路的分组均按照同一路由进行转发	每个分组独立选择路由进行转发
当结点出故障时	所有通过出故障的结点的虚电路均不能工作	出故障的结点可能会丢失分组，一些路由可能会发生变化
分组的顺序	总是按发送顺序到达终点	到达终点的时间不一定按发送顺序
端到端的差错处理和流量控制	可以由网络负责，也可以由用户主机负责	由用户主机负责

与IP配套的协议还有三种

ARP：地址解析协议

ICMP：网际控制报文协议

IGMP：网际组管理协议

物理层使用的中间设备叫转发器

数据链路层使用的设备是网桥或者交换机

网络层使用的是路由器

网络层以上使用的是网关

IP首部

普通的**IP首部长度为20字节**。具体的图解如下：

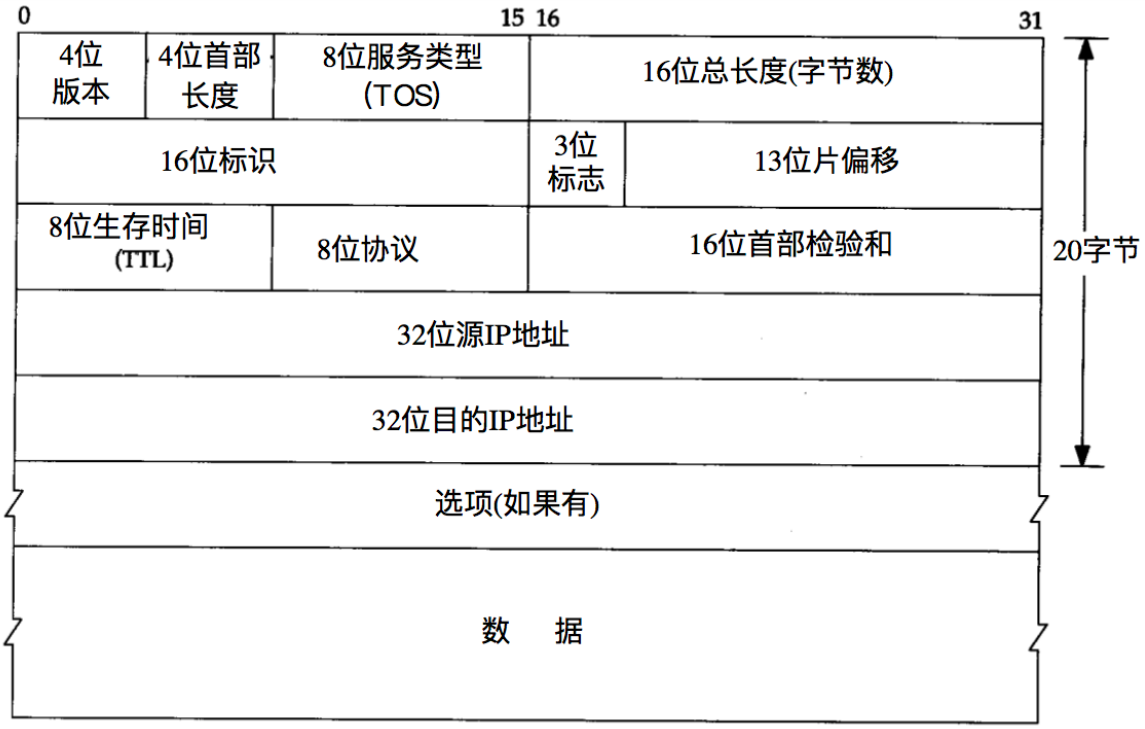


图3-1 IP数据报格式及首部中的各字段

首部长度指的是首部占 32 bit字的数目，包括任何选项。由于它是一个 4比特字段，因此**首部最长为 60 个字节**。

服务类型(TOS)字段包括一个3bit的优先权子字段(现在已被忽略)，4bit的TOS子字

段和1bit未用位但必须置0。**4bit的TOS分别代表:最小时延、最大吞吐量、最高可靠性和最小费用**。4 bit中只能置其中1 bit。如果所有4 bit均为0，那么就意味着是一般服务。

总长度字段是指整个IP数据报的长度，以字节为单位。利用首部长度字段和总长度字段，

就可以知道IP数据报中数据内容的起始位置和长度。**由于该字段长16比特，所以IP数据报最长可达65535字节**。

TTL(time-to-live)生存时间字段设置了数据报可以经过的最多路由器数。它指定了数据报的生存时间。TTL的初始值由源主机设置(通常为32或64)，一旦经过一个处理它的路由器，它的值就减去1。当该字段的值为0时，数据报就被丢弃，并发送ICMP报文通知源主机。

子网寻址

所有的主机都支持子网寻址，这是将主机号在划分成一个子网号和主机号；这样做是A类地址和B类地址为主机分配了太多的空间，在现实中，可能不需要那么多的空间来支持。

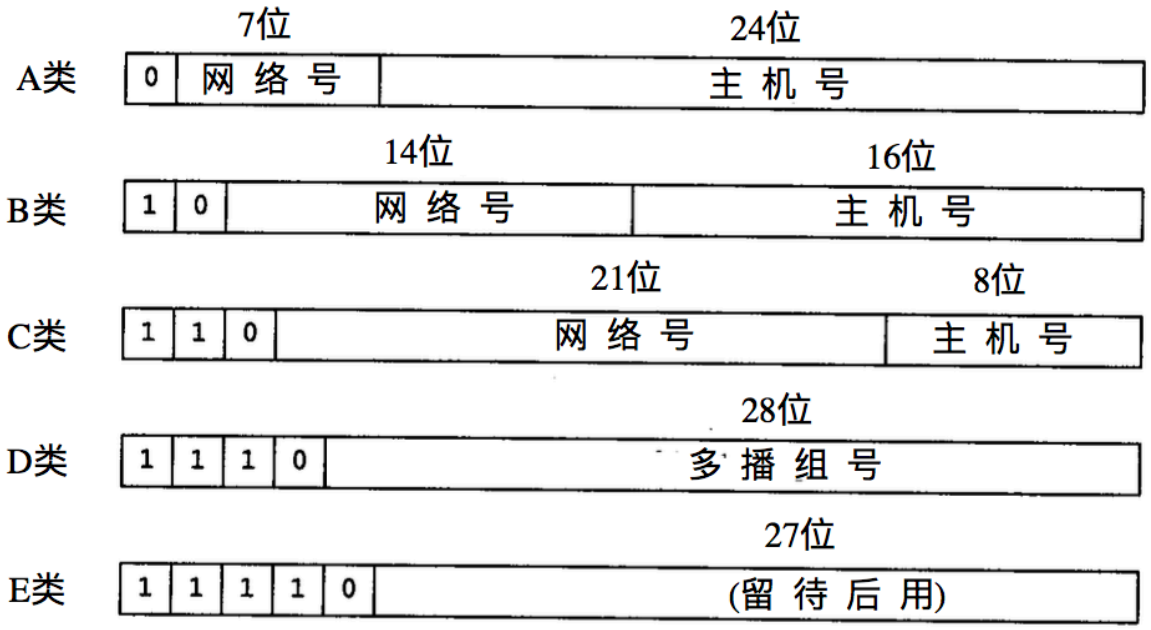


图1-5　五类互联网地址

IP地址的指派范围

表 4-2　IP 地址的指派范围

网络类别	最大可指派的网络数	第一个可指派的网络号	最后一个可指派的网络号	每个网络中的最大主机数
A	126 (2^7-2)	1	126	16777214
B	16383 ($2^{14}-1$)	128.1	191.255	65534
C	2097151 ($2^{21}-1$)	192.0.1	223.255.255	254

子网掩码

子网掩码的作用主要是来确定有多少位的主机段用于子网号，和主机号。在给定IP地址和子网掩码后，主机可以确定IP数据报的目的是：（1）本子网上的主机（2）本网络中其他子网的主机（3）其他网络的主机。

特殊的IP地址

经过子网的划分，可以确定7个特定的IP地址，

经过子网划分的描述，现在介绍 7个特殊的IP地址，如图3-9所示。在这个图中， 0表示所有的比特位全为0；-1表示所有的比特位全为 1；netid、subnetid和hostid分别表示不为全0或全1的对应字段。子网号栏为空表示该地址没有进行子网划分。

IP 地 址			可 以 为		描 述
网络号	子网号	主机号	源 端	目的端	
0		0	OK	不可能	网络上的主机（参见下面的限制）
0		主机号	OK	不可能	网络上的特定主机（参见下面的限制）
127		任何值	OK	OK	环回地址（2.7节）
−1		−1	不可能	OK	受限的广播（永远不被转发）
netid		−1	不可能	OK	以网络为目的向*netid*广播
netid	*subnetid*	−1	不可能	OK	以子网为目的向*netid*、*subnetid*广播
netid	−1	−1	不可能	OK	以所有子网为目的向*netid*广播

图3-9 特殊情况的IP地址

ifconfig

ifconfig是对网络接口进行配置和查询的命令，ifconfig命令一般在引导时运行，以配置主机上的每个接口。

netstat

netstat主要用于提供系统的接口命令。可以使用相关参数打印出每一个接口的MTU、输入分组数、输入错误、冲突以及当前的输出队列长度。

分组转发算法

1.从数据报文中取出目的地址D，然后得到其网络地址N

2.如果N就是与路由器直接相连的网络地址，那么就直接交付，不需要任何主机，如果不是，那么执行地3步骤

3.如果路由表中有到达特定D的主机路由，那么就把数据报交个所指明的吓一跳路由

4.如果路由表有到达网络N的路由，那么就把数据报交个所指明的吓一跳路由

5.如果路由表中有一个默认的路由，那么就直接交个指明的下一跳路由

6.报告分组出错

无分类编址CDIR

CDIR的两个特点：

1.消除了传统的A类、B类、C类地址以及子网的划分，将IP地址由三级分类重新分为两级

2.将网络前缀相同的连续的IP地址组成一个CDIR地址块

使用CDIR时，形成了超网，即地址聚合，在路由查找下一跳时候，采用最长匹配

ICMP

ICMP主要分为两种，ICMP差错报告报文和ICMP询问报文

ICMP差错报文有四种：

1.终点不可达

2.时间超过

3.参数问题

4.改变路由

ICMP询问报文：

1.回送请求和回答 用于是否可达以及了解相应状态

2.时间戳请求和回答 用于时间测量和时钟同步

traceroute(ICMP应用)

用于跟踪一个分组从源点到终点所经过的路径

Ping(ICMP应用)

主要用于测试主机之间的连通性。

ARP地址解析协议

ARP为IP地址到对应的硬件地址之间提供动态映射。ARP的功能主要是将逻辑的IP地址转化为对应的物理地址。

ARP 高效运作的关键是由于每个主机上都有 个 ARP 速缓存。这个高速缓存存放的是最近Internet地址到硬件地址之间的映射记录。高速缓存中每一项的存时间般为20分钟，起始时间从被创建时开始算起。

48 bit的以太地址 6个十 六进制的数来表示，中间以冒号隔开。

ARP地址解析过程：

1.当主机A需要向本局域网的机器B发送数据报文时候，需要在其ARP高速缓冲区查看是否有无主机B的IP地址，如果有，就将他的对应硬件地址取出，再将这个地址写入到MAC帧，然后通过局域网把该MAC帧发往此硬件地址。

如果没有需要进行广播请求：

1.广播发送一个ARP请求分组，然后写入A的IP地址和硬件地址，还有B的IP地址进行询问B的硬件地址

2.本局域网上的所有主机接收到ARP广播分组

3.主机B收到ARP分组，并向A返回自己的硬件地址和IP，局域网的其他主机不会做出反应

4.主机A收到返回结果后，将B的IP和硬件地址对应的写入高速缓存

RARP逆向地址解析协议

具有本地磁盘的系统引导时， 般是从磁盘上的配置文件中读取 IP地址。但是无盘机， 如X终端或 无盘作站，则需要采用RARP获得 IP地址。

对应于 ARP，RARP请求以广 播 式传送， RARP应答一般是单播 (unicast)传送的。

内部网关协议IGP

在一个自治系统内部使用的路由选择协议。

主要分为RIP和OSPF协议。

RIP适用于小型互联网

OSPF适用于大型互联网

外部网关协议BGP

IPv6

IPv6支持无连接，IPv6的主要特征有：

1.更大存址空间，IPv6的地址为128位

2.扩展的地址层次结构

3.灵活的首部格式

4.改进的选项

5.允许协议扩充

6.支持即插即用，IPv6不支持DHCP

7.支持资源的预分配

8.IPv6首部是8字节对齐

40字节的IPv6基本首部如下

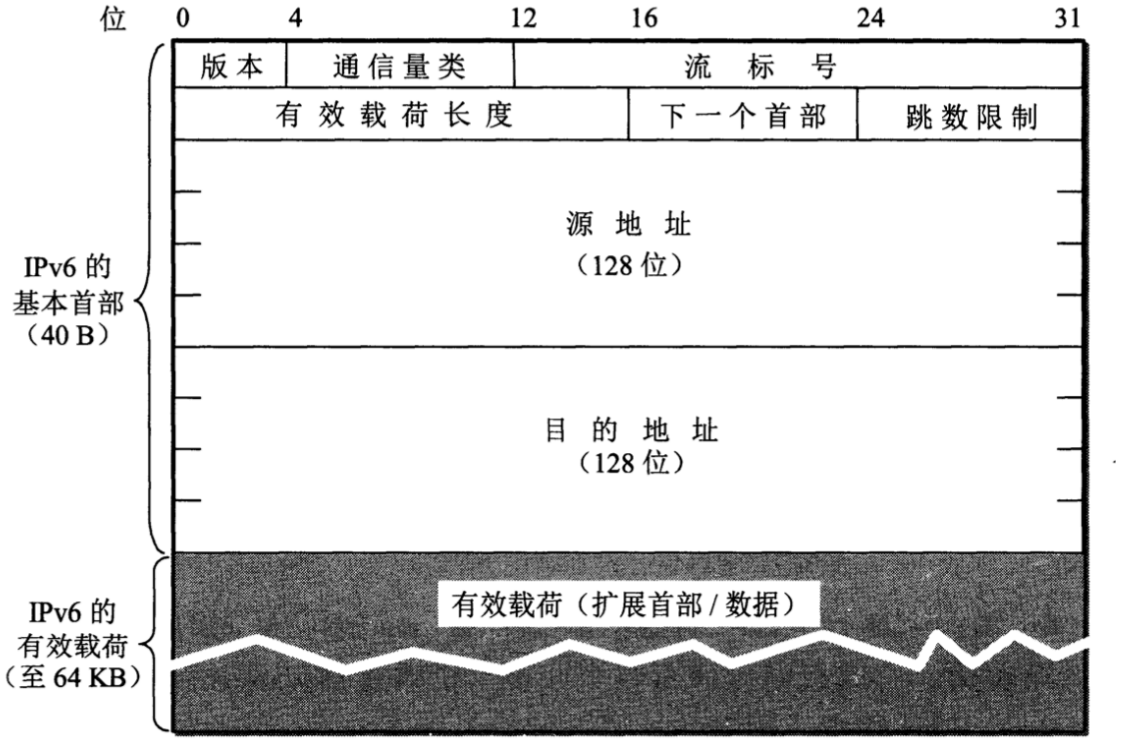


图 4-47　40 字节长的 IPv6 基本首部

一个IPv6数据报目的地址可以是有以下三种基本类型

单播：传统的点对点通信

多播：一对多点的通信

任播：终点是一组计算机，也叫做组播。

IPv使用的是零压缩法，也就是一串连续的零可以为一对冒号代替

```
FF05:0:0:0:0:0:0:B3
```

可压缩为：

```
FF05::B3
```

需要注意的是，在任一地址中只能使用一次零压缩。

表 4-10　IPv6 的地址分类

地址类型	二进制前缀
未指明地址	00…0（128 位），可记为::/128。
环回地址	00…1（128 位），可记为::1/128。
多播地址	11111111（8 位），可记为 FF00::/8。
本地链路单播地址	1111111010（10 位），可记为 FE80::/10。
全球单播地址	（除上述四种外，所有其他的二进制前缀）

第四章 运输层

网络层为主机之间提供逻辑通信，运输层为应用进程之间提供端到端的逻辑通信。

运输层的两个主要协议：

用户数据报协议UDP：

1.不建立连接

2.尽最大努力传输

3.面向报文的

4.UDP没有拥塞控制

5.首部开销少

6.可以一对一、一对多、多对多

传输控制协议TCP：

1.提供面向连接的服务

2.面向字节流

3.提供可靠服务

4.TCP只能是点对点

UDP和TCP协议各种应用

表 5-1　使用 UDP 和 TCP 协议的各种应用和应用层协议

应用	应用层协议	运输层协议
名字转换	DNS（域名系统）	UDP
文件传送	TFTP（简单文件传送协议）	UDP
路由选择协议	RIP（路由信息协议）	UDP
IP 地址配置	DHCP（动态主机配置协议）	UDP
网络管理	SNMP（简单网络管理协议）	UDP
远程文件服务器	NFS（网络文件系统）	UDP
IP 电话	专用协议	UDP
流式多媒体通信	专用协议	UDP
多播	IGMP（网际组管理协议）	UDP
电子邮件	SMTP（简单邮件传送协议）	TCP
远程终端接入	TELNET（远程终端协议）	TCP
万维网	HTTP（超文本传送协议）	TCP
文件传送	FTP（文件传送协议）	TCP

TCP端口

(1) **服务器端使用的端口号**　　这里又分为两类，最重要的一类叫做**熟知端口号**(well-known port number)或**系统端口号**，数值为 0~1023。这些数值可在网址 www.iana.org 查到。IANA 把这些端口号指派给了 TCP/IP 最重要的一些应用程序，让所有的用户都知道。当一种新的应用程序出现后，IANA 必须为它指派一个熟知端口，否则互联网上的其他应用进程就无法和它进行通信。表 5-2 给出了一些常用的熟知端口号。

表 5-2　常用的熟知端口号

应用程序	FTP	TELNET	SMTP	DNS	TFTP	HTTP	SNMP	SNMP (trap)	HTTPS
熟知端口号	21	23	25	53	69	80	161	162	443

另一类叫做**登记端口号**，数值为 1024~49151。这类端口号是为没有熟知端口号的应用程序使用的。使用这类端口号必须在 IANA 按照规定的手续登记，以防止重复。

(2) **客户端使用的端口号**　　数值为 49152~65535。由于这类端口号仅在客户进程运行时才动态选择，因此又叫做**短暂端口号**[①]。这类端口号留给客户进程选择暂时使用。当服务器进程收到客户进程的报文时，就知道了客户进程所使用的端口号，因而可以把数据发送给客户进程。通信结束后，刚才已使用过的客户端口号就不复存在，这个端口号就可以供其他客户进程使用。

TCP连接

TCP连接的端口叫做套接字或者插口

每一条TCP连接唯一地被通信两端的两个端点所确定：

在一个应用进程和另一个应用进程之间建立一个TCP连接。

TCP连接的端点是个抽象的套接字，即 (IP地址:端口)

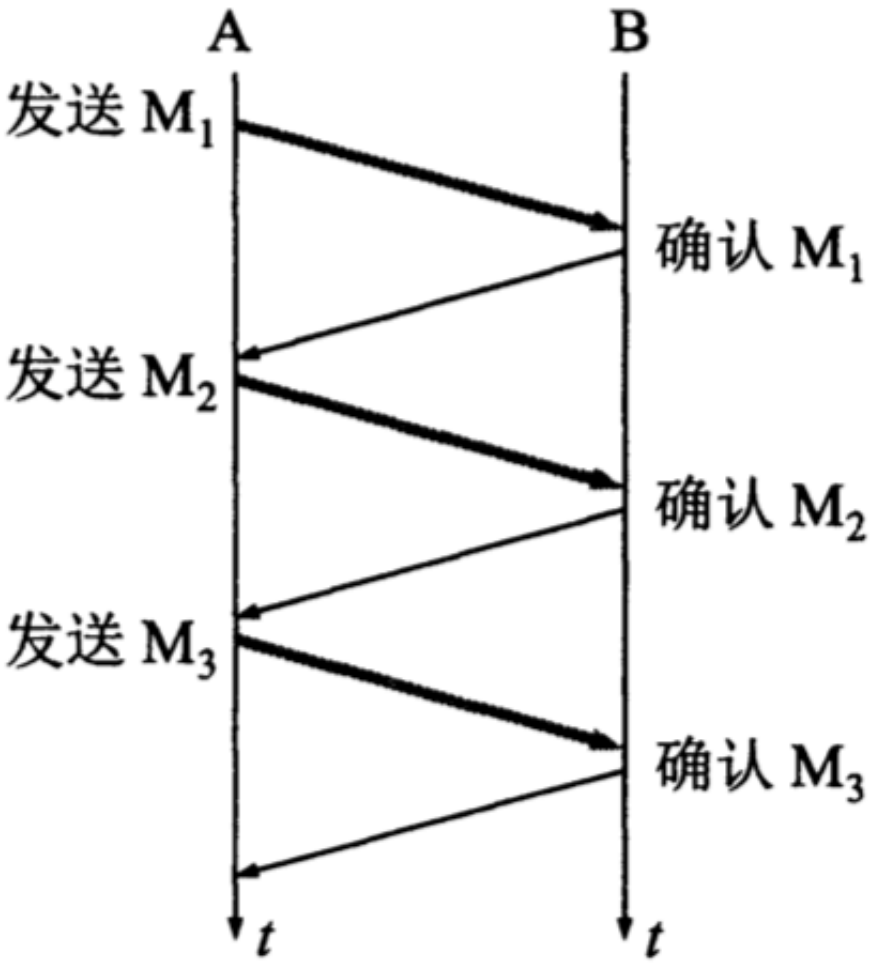
同一个IP地址可以有多个不同的TCP地址，而同一个端口号也可以在多个不同的TCP连接。

可靠传输的工作原理

停止等待协议：

停止等待就是每发送完一个分组就停止发送，等待对方的确认。

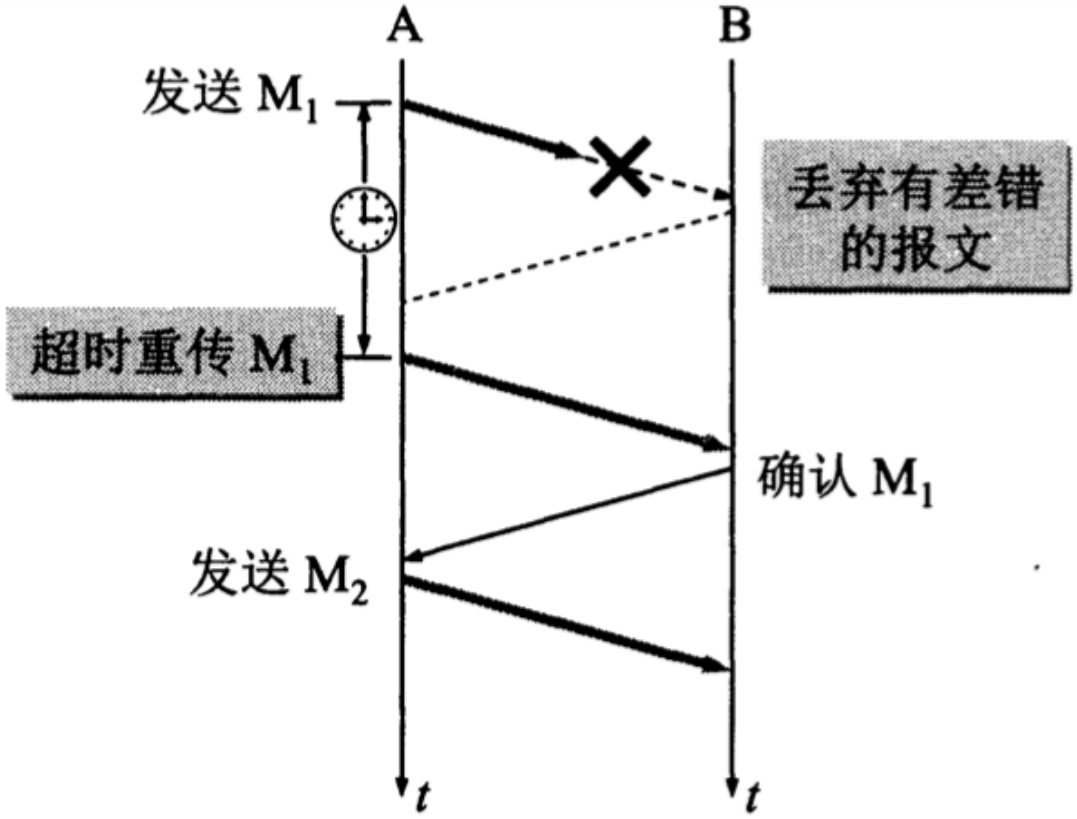
无差错情况



(a) 无差错情况

出现差错

A只要超过一段时间仍然没有受到确认，就认为刚才发送的分组丢失了，因而重传发送过的分组，这叫做超时重传。



(b) 超时重传

确认丢失和确认迟到

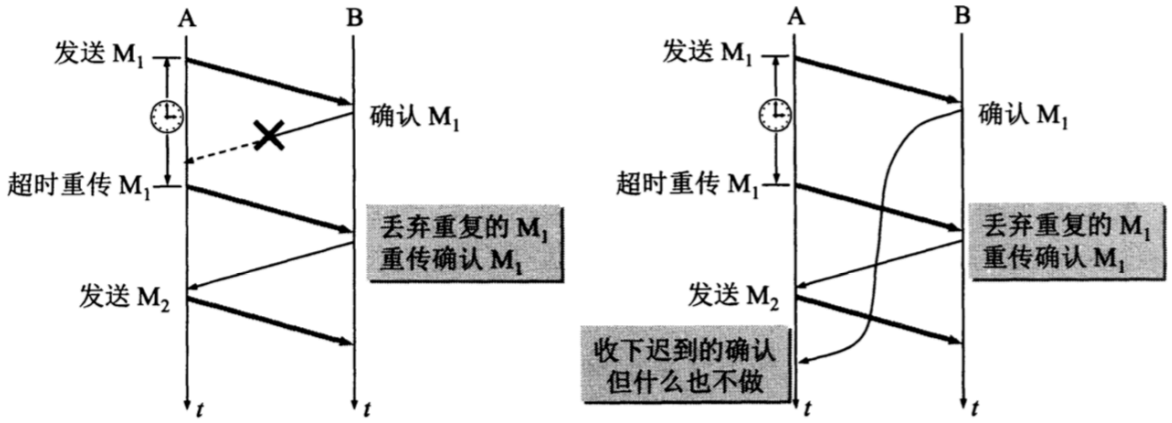


图 5-10　确认丢失和确认迟到

确认丢失采用两个行动：

1.丢失这个重复分组M1

2.向A发送确认

确认迟到采用的行动；

收到后丢弃

连续ARQ协议

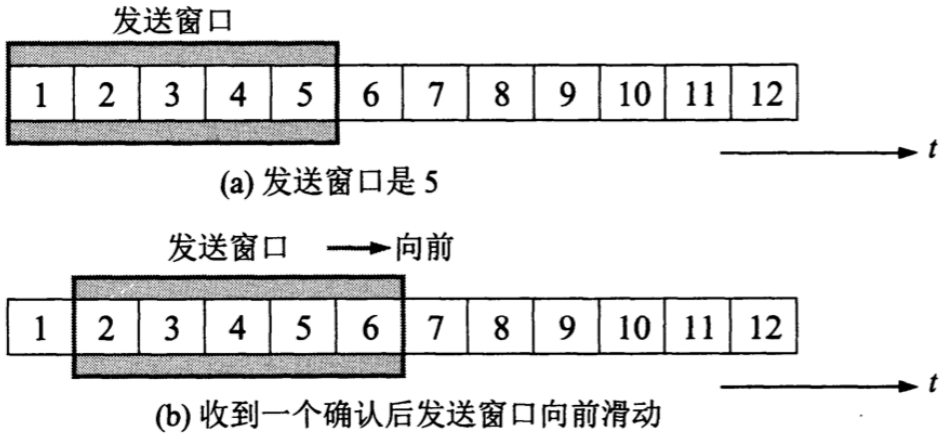


图 5-13　连续 ARQ 协议的工作原理

发送方收到一个确认，就将发送窗口向前滑动。

TCP报文段的首部格式

TCP报文首部前20个字节是固定的，TCP首部最小长度是20字节。

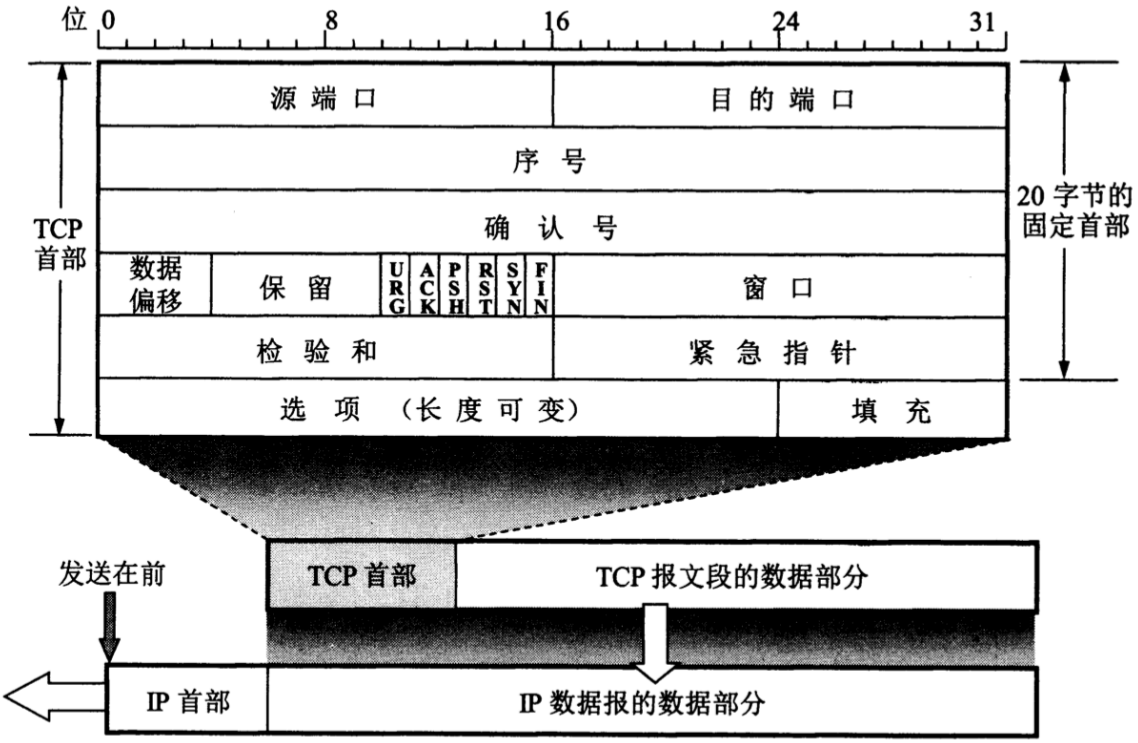


图 5-14　TCP 报文段的首部格式

各个字段意义：

源端口和目的端口：各占2个字节

序号：4字节，范围是[0,2^32-1]，可以看出最多4GB

确认号：4字节，期望收到对方下一个报文段的第一个数据字节的序号。

窗口：2字节

校验和：2字节

紧急指针；2字节

TCP拥塞控制

拥塞控制算法：慢开始、拥塞控制、快重传、快恢复

慢开始和拥塞避免

发送发维护一个拥塞窗口cwnd，开始是发送方让自己的发送窗口等于拥塞窗口。

还需要设置一个慢开始门限ssthresh：

当 cwnd < ssthresh 时，使用上述的慢开始算法。
当 cwnd > ssthresh 时，停止使用慢开始算法而改用拥塞避免算法。
当 cwnd = ssthresh 时，既可使用慢开始算法，也可使用拥塞避免算法。

慢开始使用指数增大，拥塞避免算法是使用加法增大

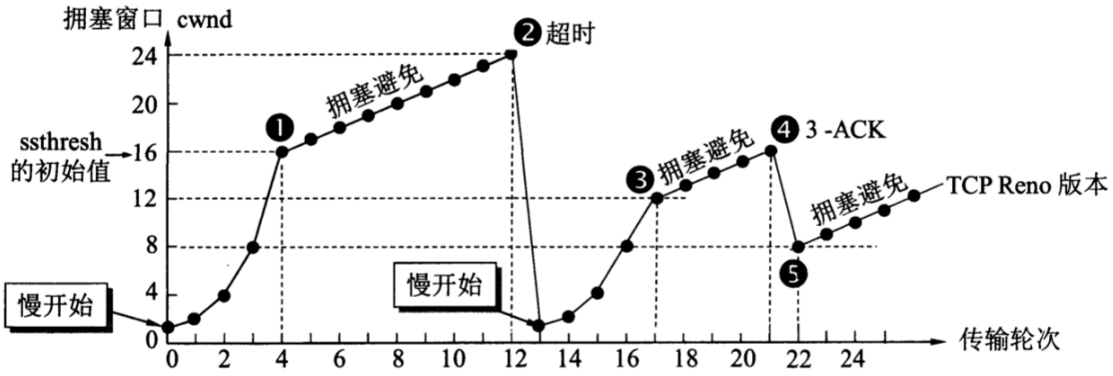


图 5-25　TCP 拥塞窗口 cwnd 在拥塞控制时的变化情况

开始时cwnd=1，ssthresh=16，使用慢开始算法

在传输轮次为4，使用拥塞避免算法

轮次为12，出现网络超时，ssthresh=cwnd / 2=12，cwnd=1 之后使用慢开始算法

在轮次为21时，发现重复确认，使用了快重传

采用快重传算法可以让发送方**尽早知道发生了个别报文段的丢失**。快重传算法首先要求接收方不要等待自己发送数据时才进行捎带确认，而是要**立即发送确认**，即使收到了**失序的报文段**也要立即发出对已收到的报文段的重复确认。如图 5-26 所示，接收方收到了 M_1 和 M_2 后都分别及时发出了确认。现假定接收方没有收到 M_3 但却收到了 M_4。本来接收方可以什么都不做。但按照快重传算法，接收方**必须立即发送对 M_2 的重复确认**，以便让发送方及早知道接收方没有收到报文段 M_3。发送方接着发送 M_5 和 M_6。接收方收到后也仍要再次分别发出对 M_2 的重复确认。这样，发送方共收到了接收方的 4 个对 M_2 的确认，其中后 3 个都是重复确认。快重传算法规定，发送方只要**一连收到 3 个重复确认**，就知道接收方确实没有收到报文段 M_3，因而应当**立即进行重传**（即“快重传”），这样就不会出现超时，发送方也不就会误认为出现了网络拥塞。使用快重传可以使整个网络的吞吐量提高约 20%。

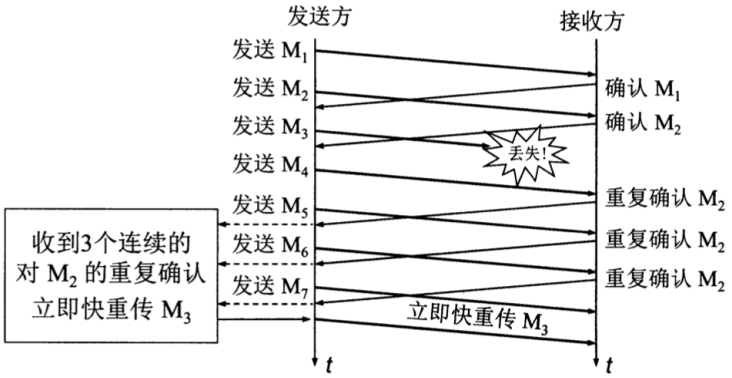


图 5-26　快重传的示意图

在4的时候，使用快恢复，将cwnd=ssthresh=cwnd/2=8然后执行拥塞避免

TCP连接管理

三次握手如下：

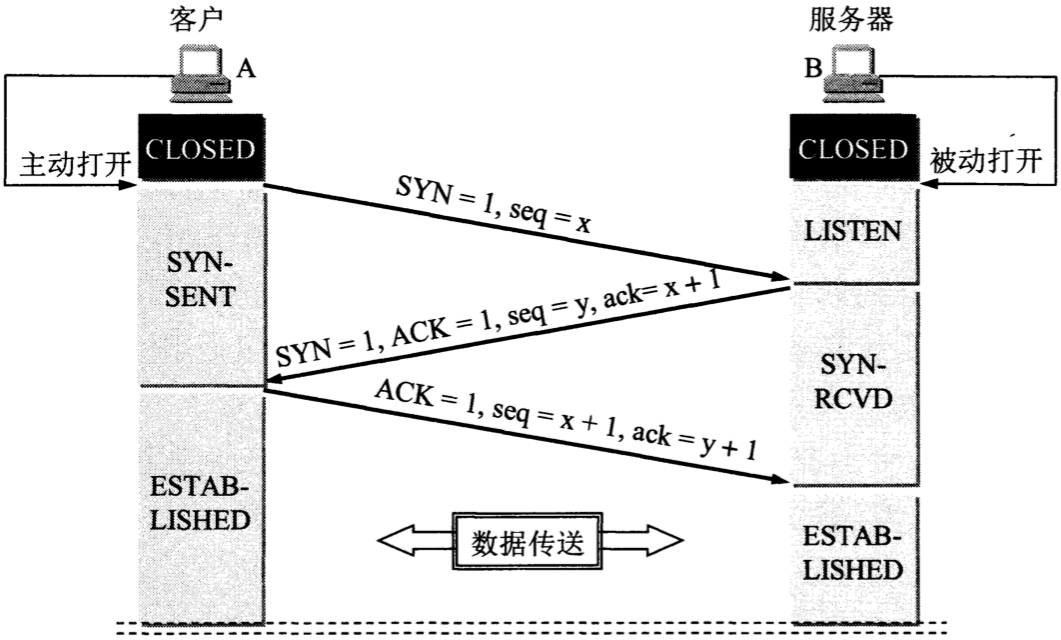


图 5-28 用三报文握手建立 TCP 连接

过程描述:

第一次握手：客户端发送syn包(syn=x)到服务器，并进入SYN_SEND状态，等待服务器确认；

第二次握手：服务器收到syn包，必须确认客户的SYN（ack=x+1），同时自己也发送一个SYN包（syn=y），即SYN+ACK包，此时服务器进入SYN_RECV状态；

第三次握手：客户端收到服务器的SYN＋ACK包，向服务器发送确认包ACK(ack=y+1)，此包发送完毕，客户端和服务器进入ESTABLISHED状态，完成三次握手。

握手过程中传送的包里不包含数据，三次握手完毕后，客户端与服务器才正式开始传送数据。理想状态下，TCP连接一旦建立，在通信双方中的任何一方主动关闭连接之前，TCP 连接都将被一直保持下去。

为什么需要使用3次握手呢?

防止已失效的连接请求报文段传输到B，因而产生错误。

现假定出现一种异常情况，即 A 发出的第一个连接请求报文段并没有丢失，而是在某些网络结点长时间滞留了，以致延误到连接释放以后的某个时间才到达 B。本来这是一个早已失效的报文段。但 B 收到此失效的连接请求报文段后，就误认为是 A 又发出一次新的连接请求。于是就向 A 发出确认报文段，同意建立连接。假定不采用报文握手，那么只要 B 发出确认，新的连接就建立了。

由于现在 A 并没有发出建立连接的请求，因此不会理睬 B 的确认，也不会向 B 发送数据。但 B 却以为新的运输连接已经建立了，并一直等待 A 发来数据。B 的许多资源就这样白白浪费了。

TCP连接释放

数据传输结束后，通信的双方都可释放连接。现在 A 和 B 都处于 ESTABLISHED 状态（图 5-29）。A 的应用进程先向其 TCP 发出连接释放报文段，并停止再发送数据，主动关闭 TCP 连接。A 把连接释放报文段首部的终止控制位 FIN 置 1，其序号 seq = u，它等于前面已传送过的数据的最后一个字节的序号加 1。这时 A 进入 FIN-WAIT-1（终止等待 1）状态，等待 B 的确认。请注意，TCP 规定，FIN 报文段即使不携带数据，它也消耗掉一个序号。

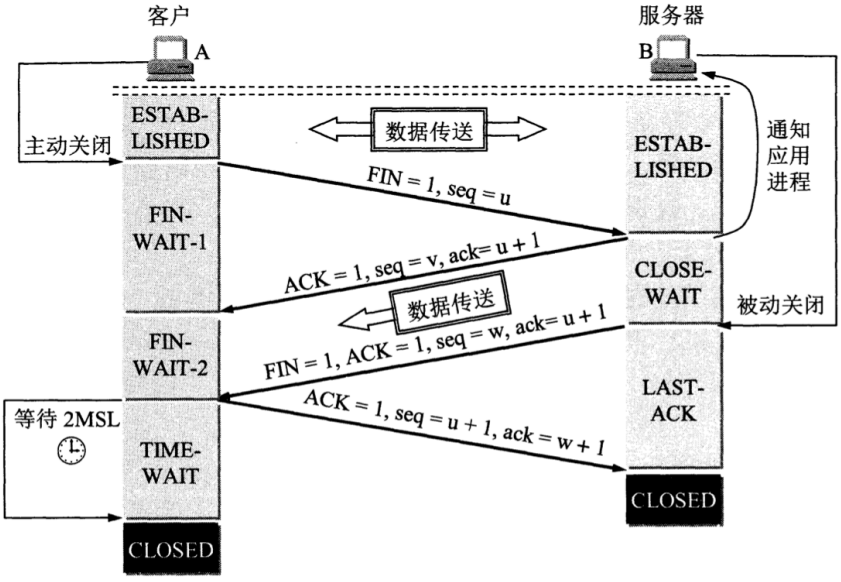


图 5-29　TCP 连接释放的过程

四次挥手:

与建立连接的“三次握手”类似，断开一个TCP连接则需要“四次握手”。

第一次挥手：主动关闭方发送一个FIN，用来关闭主动方到被动关闭方的数据传送，也就是主动关闭方告诉被动关闭方：我已经不会再给你发数据了(当然，在fin包之前发送出去的数据，如果没有收到对应的ack确认报文，主动关闭方依然会重发这些数据)，但是，此时主动关闭方还可 以接受数据。

第二次挥手：被动关闭方收到FIN包后，发送一个ACK给对方，确认序号为收到序号+1（与SYN相同，一个FIN占用一个序号）。 第三次挥手：被动关闭方发送一个FIN，用来关闭被动关闭方到主动关闭方的数据传送，也就是告诉主动关闭方，我的数据也发送完了，不会再给你发数据了。 第四次挥手：主动关闭方收到FIN后，发送一个ACK给被动关闭方，确认序号为收到序号+1，至此，完成四次挥手。

为什么 A 在 TIME-WAIT 状态必须等待 2MSL 的时间呢？这有两个理由。

第一，为了保证 A 发送的最后一个 ACK 报文段能够到达 B。这个 ACK 报文段有可能丢失，因而使处在 LAST-ACK 状态的 B 收不到对已发送的 FIN + ACK 报文段的确认。B 会超时重传这个 FIN + ACK 报文段，而 A 就能在 2MSL 时间内收到这个重传的 FIN + ACK 报文段。接着 A 重传一次确认，重新启动 2MSL 计时器。最后，A 和 B 都正常进入到 CLOSED 状态。如果 A 在 TIME-WAIT 状态不等待一段时间，而是在发送完 ACK 报文段后立即释放连接，那么就无法收到 B 重传的 FIN + ACK 报文段，因而也不会再发送一次确认报文段。这样，B 就无法按照正常步骤进入 CLOSED 状态。

第二，防止上一节提到的“已失效的连接请求报文段”出现在本连接中。A 在发送完最后一个 ACK 报文段后，再经过时间 2MSL，就可以使本连接持续的时间内所产生的所有报文段都从网络中消失。这样就可以使下一个新的连接中不会出现这种旧的连接请求报文段。

B 只要收到了 A 发出的确认，就进入 CLOSED 状态。同样，B 在撤销相应的传输控制块 TCB 后，就结束了这次的 TCP 连接。我们注意到，B 结束 TCP 连接的时间要比 A 早一些。

第五章 HTTP报文

报文流

HTTP报文是在HTTP应用程序之间发送的数据块，这些数据块以一些**文本形式**的元数据开头。

报文流入源端服务器

HTTP使用术语流入和流出来描述事务处理。

报文向下游流动

所有的报文都向下游流动，报文的发送者都在接收端者的上游。

报文的组成

报文主要由三个部分组成：起始行、包含属性的首部块、以及可选的、包含数据的主体部分。

起始行和首部都是由行分隔的ASCII文本，每行都是由两个字符组成的行终止序列作为结束，一个是**回车符**一个是**换行符**。

主体是一个可选的数据块，主要的是这里的数据可以是二进制数据和文本，或者是空。

报文段的语法

所有的报文都可以分为两类：请求报文和响应报文。

请求报文向Web服务器请求一个动作，响应报文会将请求的报文返回给客户端。

请求报文格式

```
<method> <request-URL> <version>
<headers>

<entity-body>
```

响应报文格式

```
<version> <status> <reason-phrase>
<headers>

<entity-body>
```

方法：例如GET、HEAD

请求URL：请求资源

版本：HTTP版本

状态码：发生的状态

原因短语：数字状态码的可读性版本

首部：可选字段，一般是报文长度，类型

主体：任意数据组层的数据块

起始行

1.请求行

请求方法和URL、版本号等，使用空格分隔

2.响应行

HTTP版本、数字状态码、原因短语、

3.方法

请求的起始行以方法作为开始

常用的方法

GET 从服务器获取文档

HEAD 从服务器获取文档首部

POST 向服务器发送需要处理的数据

PUT 请求的主体存储在服务器上

TRACE 对报文进行追踪

OPTIONS 决定可以在服务器上执行哪些方法

DELETE 从服务器删除文档

4.状态码

整体范围 定义范围 分类

100-199 100-101 信息提示

200-299 200-206 成功

300-399 300-305 重定向

400-499 400-415 客户端错误

500-599 500-505 服务器错误

5.短语原因

为状态码做解释

6.版本号

遵守的协议

首部

起始行后面是0或者多个HTTP首部字段，HTTP首部字段请求和响应报文添加了一些附加信息。本质上说是一些名/值对的列表。

1.首部分类

通用首部：可以出现在请求报文中，也可以出现在响应报文中

请求首部：提供能多相关请求的信息

响应首部:提供响应的信息

实体首部：主体的长度和内容，或者资源自身。

扩展首部：规范中没有定义的新首部

首部内容分行使用CRLF也就是\n\t

2.首部延续行

将长的内部行分为多行提高可读性，多出来的每行前面至少要有一个空格或者制表符。

实体的主体部分

主体主要是HTTP要传输的内容

版本0.9的报文

由请求和响应组成，请求中只包含方法和请求URL，响应中实体，没有版本信息，没有状态码或者短语，没有首部。

方法

如果一台服务器要与HTTP1.1兼容，只要为其资源实现GET方法和HEAD方法就可以。

安全方法

GET和HEAD方法被认为安全，这两个请求都不产生什么动作。

GET

用于请求服务器发送某个资源。

HEAD

服务器在响应中只返回首部。允许客户端在未获取实际资源的情况下，对资源的首部进行检查。

PUT

向服务器写入文档。让服务器用请求的主体部分来常见一个由所请求的URL命名的文档，如果存在，用这个主体代替它。

POST

向服务器输入数据，支持HTML表单

TRACE

追踪报文的路径，主要用于诊断。

OPTIONS

请求Web服务器告知支持的各种功能。可以询问服务器支持的那些方法，或者那些特殊资源支持方法。

DELETE

请求服务器删除钦地区URL所制定的资源。

扩展方法

COPY、LOCK、MOVE等

状态码

状态码分为五类

100-199 信息性状态码

100Continue状态码：说明请求的初始化部分，请求客户端继续。

101:根据客户端指定，将协议且 切换成Update首部

1.客户端与100Continue

客户端应用程序只有在避免向服务器发送一个服务器无法处理或者使用的大实体时，才应该使用100Continue。

2.服务器与100Continue

不主动向客户端发送这个状态码，除非出错

3.代理与100Continue

不知道下一跳服务器是否与HTTP/1.1兼容，封装Except向下转发，如果知道，返回417

200-209 成功状态码

200 请求成功

201 创建服务器对象请求

202 请求被接受，服务器没有执行任何动作

203 实体首部包含的信息不是来自源端服务器，而是来自资源的一份副本。

204 响应报文中国年包含若干首部和一个状态码，但是没有实体的主体部分

205 另一个主要用于浏览器的代码，晴空当前页面的HTML表单元素

206 成功执行一个部分的请求

300-399 重定向状态码

可以通过某些重定向状态码对资源的应用程序本地副本与远端服务器上的资源进行验证。

300 客户端请求一个实际指向多个资源的URL时会返回这个状态码

301 请求的URL以被移除时使用，响应返回资源所在的URL

302 与301类似，但是请求仍然应使用老的URL。

303 告知客户端应该用另一个URL来获取资源。

304 客户端可以通过包含的请求首部，使其请求变成有条件的。

305 说明必须通过一个代理来访问资源。

306 当前未使用

307 与301类似，客户端应该使用Location首部给出的URL来临时定位资源，将来的请求应该使用老的URL

HTTP/1.0发送POST请求，收到302，期望HTTP客户端使用GET请求返回的URL。

HTTP/1.1使用303，对于客户端使用307代替302临时重定向

400-499 客户单错误状态码

400 客户端发送错误请求

401 请求客户端认证

402 未使用

403 服务器拒绝服务

404 资源没找到

405 不支持请求方法

406 服务器没有与客户端可接受的URL相匹配的资源

407 要求资源进行认证的代理服务器

408 请求超时

409 请求可能资源上英法德一些冲突

410 服务器以前拥有过资源，用于Web站点维护

411 服务器要求在报文中包含Content-Length首部

412 客户端请求条件失败了

413 请求实体过大

414 请求URL过长

415 服务器无法理解请求内容类型

416 请求范围无效

417 请求的Expect请求首部包含一个期望，服务器无法满足此期望。

500-599 服务器错误代码

500 服务器内部错误

501 请求超出服务器能力范围

502 坏网关

503 现在无法提供服务

504 网关超时

505 请求无法支持此协议版本

首部

通用首部

提供与报文有关的最基本信息

通用缓存首部

请求首部

用于请求首部

Accept首部

条件请求首部

安全请求首部

代理请求首部

响应首部

用于响应

协商首部

安全响应首部

实体首部

包含实体

内容首部

实体缓存首部

持久连接

初始化对某服务器HTTP请求的应用程序很可能会在不久的将来对那台服务器发起更多的请求，这种性质被称为站点本地性。

在事务处理结束以后仍然保持在打开状态的TCP连接被称为持久连接。HTTP/1.1支持持久连接。

持久以及并行连接

持久连接有两种类型：比较老的HTTP/1.0+ "keep-alive"连接和现代HTTP/1.1 “persistent”连接

keep-alive连接

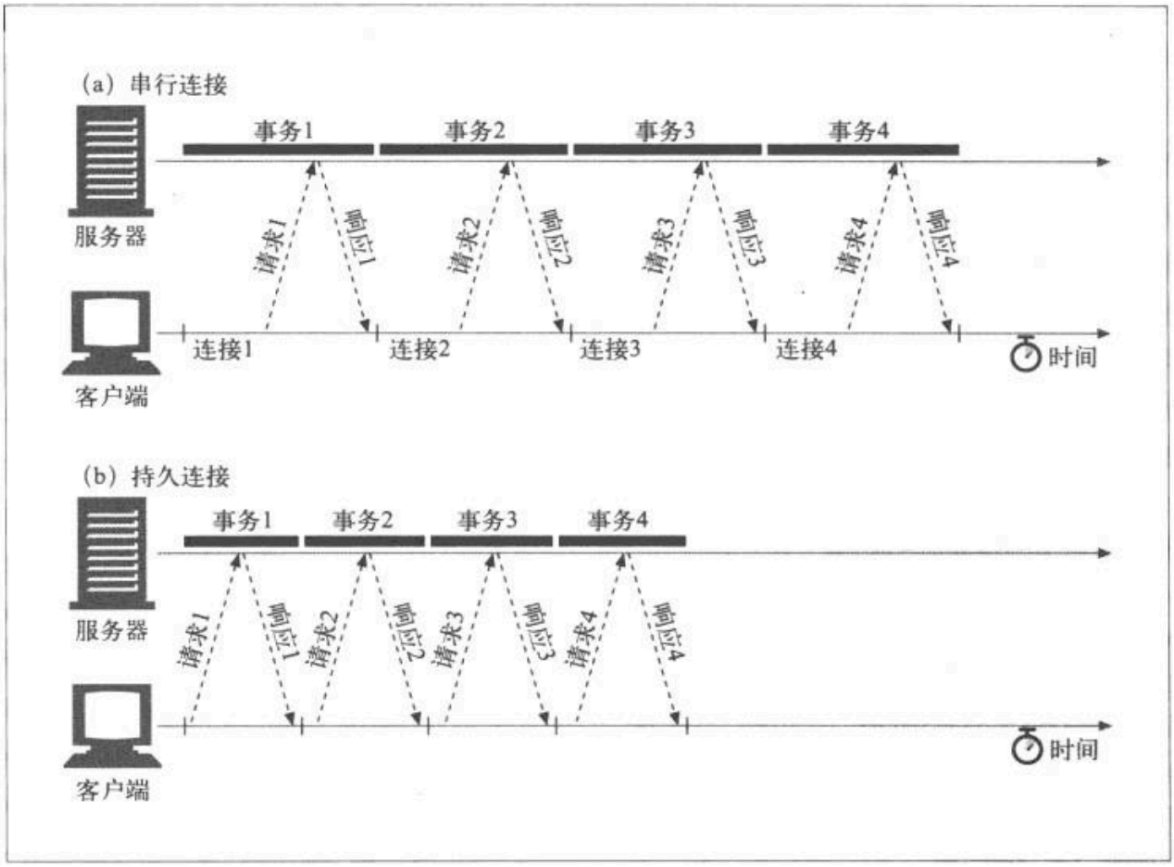


图 4-13　4 个事务（串行与持久连接）

Keep-Alive操作

实现keep-alive脸的客户端通过包含Connection：Keep-Alive首部请求将一条连接保持打开状态。

Keep-Alive选项

服务器客户端不一定同意keep-alive，需要设置逗号参数管理

timeout 连接保持时间

参数键max 估计服务器还有多少个事务保持此连接的活跃状态

支持未经出来的属性，用于诊断和调试。

Keep-Alive连接的限制和规则

HTTP/1.0中，keep-alive并不是默认使用的。

Connection:keep-alive首部必须随所有希望保持持久连接的报文一起发送。

客户端探明有没有持久连接首部，直到服务器响应之后是否关闭连接

keep-alive和哑代理

1.Connection首部和盲中继

不理解首部，直接转发，没有删除Connection首部，这就是盲中继。

2.代理和逐跳首部

现在的代理不转发含有Connection的首部

管道化连接

HTTP/1.1允许持久连接使用请求管道。

第六章 客户端识别于cookie机制

个性化接触

站点的个性化

1.个性化的问候

2.有的放矢的推荐

3.管理信息的存档

4.记录安全回话

HTTP首部

HTTP请求首部

表11-1 承载用户相关信息的HTTP首部

首部名称	首部类型	描 述
From	请求	用户的 E-mail 地址
User-Agent	请求	用户的浏览器软件
Referer	请求	用户是从这个页面上依照链接跳转过来的
Authorization	请求	用户名和密码（稍后讨论）
Client-IP	扩展（请求）	客户端的 IP 地址（稍后讨论）
X-Forwarded-For	扩展（请求）	客户端的 IP 地址（稍后讨论）
Cookie	扩展（请求）	服务器产生的 ID 标签（稍后讨论）

客户端IP地址

客户端IP地址识别用户的缺点：

1.客户端IP地址描述的是所有的机器。不是用户，有可能多个用户共享一台计算机

2.很多服务提供商都会在用户登录时动态分配IP地址

3。提高安全性，通过NAT来浏览网页

用户登录

HTTP中包含了一种内建机制，可以用www-Aythenticate首部和Authorization首部向Web站点传输用户的相关信息。

胖URL

有些Web站点会为每一个用户生成特定版本的URL来追踪用户的身份，通常是对真正的URL进行扩展，包含一些状态信息，改动后的URL成为胖URL。

技术问题：

1.丑陋的URL

2.无法共享URL

3.破坏缓存

4.额外的服务器负载

5.逃逸口

6.回话间非持久

cookie

cookie时当前识别用户，实现持久会话的最好方式。

cookie的类型

cookie分为两类：会话cookie和持久cookie

会话cookie时一种临时cookie，用户就会删除

持久cooki存储在看硬盘，计算机重启时仍然存在

唯一区别是过期时间

cookie是如何工作的

cookie中包含一个由名字=值这样的信息构成的任意列表，并通过Set-Cookie或者Set-Cookie2HTTP响应首部将其贴到用户身上去。

cookie罐：客户端的状态

cookie的基本思想是让浏览器积累一组服务器特有的信息，每次访问服务器都将这些信息提供给它。

1.网景公司的cookie，存储在cookie.txt中

2.ie存储在高速缓存目录下独立文本文件中

不同的站点使用不同的cookie

浏览器只向服务器发送服务器产生的那些cookie。

1.cookie的域属性

通过Set-Cookie响应首部添加一额Domain属性来控制哪些站点可以看到哪个cookie

2.cookie路径属性

cookie规范深知允许用户cookie与部分Web站点关联起来。可以通过Path属性来实现这一功能功能，在这个属性列出的URL路径前缀所有的cookie都是有效的。

cookie成分

cookie有两个版本，第一个是cookies版本0，第二个是cookie版本1

cookie的规范：

表11-2 cookie规范

标 题	描 述	位 置
持久客户端状态：HTTP cookies	最初的 Netscape cookie 标准	http://home.netscape.com/newsref/std/cookie_spec.html
RFC 2965：HTTP 状态管理机制	2000 年 10 月的 cookie 标准，废弃了 RFC 2109	http://www.ietf.org/rfc/rfc2965.txt

cookies版本0

定义了Set-Cookie响应首部、cookie请求首部以及用于控制cookie的字段。

```
Set-Cookie: name=value [; expires=date] [; path=path] [; domain=domain]
[; secure]

Cookie: name1=value1 [; name2=value2] ...
```

1.版本0的Set-cookie首部

有一个强制性的cookie名和cookie值，后面跟着可短的cookie属性，由分号隔开

2.版本0的cookie首部

将所有与域、路径和安全多虑期相匹配的国旗cookie都发送给这个站点。所有的cookie将会被 组合到一个cookie首部：

```
Cookie: session-id=002-1145265-8016838; session-id-time=1007884800
```

cookies版本1

引入set-Cookies2首部和Cookies首部，能与版本0系统进行互操作。

1.版本1的Set-cookie2首部

表11-4 版本1（RFC 2965）的Set-Cookie2属性

Set-Cookie2属性	描述及实例
NAME=VALUE	强制的。Web 服务器可以创建任意的 NAME=VALUE 关联，可以在后继对站点的访问中将其发回给 Web 服务器。“$”是保留字符，所以名字一定不能以它开头
Version	强制的。这个属性的值是一个整数，对应于 cookie 规范的版本。RFC 2965 为版本 1： Set-Cookie2: Part="Rocket_Launcher_0001"; Version="1"
Comment	可选。这个属性说明了服务器准备如何使用这个 cookie。用户可以通过检查此策略来确定是否允许使用带有这个 cookie 的会话。这个值必须采用 UTF-8 编码
CommentURL	可选。这个属性提供了一个 URL 指针，指向详细描述了 cookie 目的及策略的文档。用户可以通过查看此策略来判定是否允许使用带有这个 cookie 的会话
Discard	可选。如果提供了这个属性，就会在客户端程序终止时，指示客户端放弃这个 cookie
Domain	可选。浏览器只向指定域中的服务器主机名发送 cookie。这样服务器就可以将 cookie 限制在特定域中了。acme.com 域与主机名 anvil.acme.com 和 shipping.crate.acme.com 相匹配，但不匹配于 www.cnn.com。域名匹配的规则基本上与 Netscape cookie 一样，但有几条附加的规则。细节请参见 RFC 2965

2.版本1的cookie首部

带回与传送的每个cookie相关的附加信息，用来描述每个cookie途径的过滤器。

3.版本1的cookie2首部和版本协商

Cookie2请求首部负责能够理解不同cookie规范版本的客户端和服务器之间的进行互操作性的协商。

cookie与会话跟踪

可以用cookie在用户与某个web站点进行多项事务处理时对用户进行跟踪。

cookie与缓存

1.如果缓存文档，将其标示出来

2.缓存Set-Cookie首部要小心

3.小心处理带有cookie首部请求

cookie、安全性和隐私

常考题

在地址栏上输入了baidu.com或发生什么?

1、客户端浏览器通过DNS解析到www.baidu.com的IP地址220.181.27.48，通过这个IP地址找到客户端到服务器的路径。客户端浏览器发起一个HTTP会话到220.161.27.48，然后通过TCP进行封装数据包，输入到网络层。

2、在客户端的传输层，把HTTP会话请求分成报文段，添加源和目的端口，如服务器使用80端口监听客户端的请求，客户端由系统随机选择一个端口如5000，与服务器进行交换，服务器把相应的请求返回给客户端的5000端口。然后使用IP层的IP地址查找目的端。

3、客户端的网络层不用关心应用层或者传输层的东西，主要做的是通过查找路由表确定如何到达服务器，期间可能经过多个路由器，这些都是由路由器来完成的工作，我不作过多的描述，无非就是通过查找路由表决定通过那个路径到达服务器。

4、客户端的链路层，包通过链路层发送到路由器，通过邻居协议查找给定IP地址的MAC地址，然后发送ARP请求查找目的地址，如果得到回应后就可以使用ARP的请求应答交换的IP数据包现在就可以传输了，然后发送IP数据包到达服务器的地址。

HTTP1.0和HTTP1.1的区别

1、HTTP 1.1支持长连接（PersistentConnection）和请求的流水线（Pipelining）处理

在1.0时的会话方式：

1. 建立连接
2. 发出请求信息
3. 回送响应信息
4. 关掉连接

HTTP 1.1的持续连接，也需要增加新的请求头来帮助实现，例如，Connection请求头的值为Keep-Alive时，客户端通知服务器返回本次请求结果后保持连接；Connection请求头的值为close时，客户端通知服务器返回本次请求结果后关闭连接。HTTP 1.1还提供了与身份认证、状态管理和Cache缓存等机制相关的请求头和响应头。

2.HTTP 1.1增加host字段

在HTTP1.0中认为每台服务器都绑定一个唯一的IP地址，因此，请求消息中的URL并没有传递主机名（hostname）。但随着虚拟主机技术的发展，在一台物理服务器上可以存在多个虚拟主机（Multi-homed Web Servers），并且它们共享一个IP地址。

HTTP1.1的请求消息和响应消息都应支持Host头域，且请求消息中如果没有Host头域会报告一个错误（400 Bad Request）。此外，服务器应该接受以绝对路径标记的资源请求。

3、100(Continue) Status(节约带宽)

Session和Cookie的区别

ookie 和session 的区别：

1、cookie数据存放在客户的浏览器上，session数据放在服务器上。

2、cookie不是很安全，别人可以分析存放在本地的COOKIE并进行COOKIE欺骗 考虑到安全应当使用session。

3、session会在一定时间内保存在服务器上。当访问增多，会比较占用你服务器的性能 考虑到减轻服务器性能方面，应当使用COOKIE。

4、单个cookie保存的数据不能超过4K，很多浏览器都限制一个站点最多保存20个cookie。

5、所以个人建议： 将登陆信息等重要信息存放为SESSION 其他信息如果需要保留，可以放在COOKIE中

Post和Get的区别

- GET在浏览器回退时是无害的，而POST会再次提交请求。
- GET产生的URL地址可以被Bookmark，而POST不可以。
- GET请求会被浏览器主动cache，而POST不会，除非手动设置。
- GET请求只能进行url编码，而POST支持多种编码方式。
- GET请求参数会被完整保留在浏览器历史记录里，而POST中的参数不会被保留。
- GET请求在URL中传送的参数是有长度限制的，而POST么有。
- 对参数的数据类型，GET只接受ASCII字符，而POST没有限制。
- GET比POST更不安全，因为参数直接暴露在URL上，所以不能用来传递敏感信息。
- GET参数通过URL传递，POST放在Request body中。

HTTP2.0新特性

1.使用多路复用技术(Multiplexing),多路复用允许同时通过单一的 HTTP/2 连接发起多重的请求-响应消息。

2.HTTP/2新增首部压缩（Header Compression）:采用HPACK算法

3.HTTP/2新增服务端推送（Header Compression）

SYN攻击

TCP SYN泛洪发生在OSI第四层，这种方式利用TCP协议的特性，就是三次握手。攻击者发送TCP SYN，SYN是TCP三次握手中的第一个数据包，而当服务器返回ACK后，该攻击者就不对其进行再确认，那这个TCP连接就处于挂起状态，也就是所谓的半连接状态，服务器收不到再确认的话，还会重复发送ACK给攻击者。这样更加会浪费服务器的资源。攻击者就对服务器发送非常大量的这种TCP连接，由于每一个都没法完成三次握手，所以在服务器上，这些TCP连接会因为挂起状态而消耗CPU和内存，最后服务器可能死机，就无法为正常用户提供服务了。

TCP的流量控制和拥塞控制

TCP的流量控制

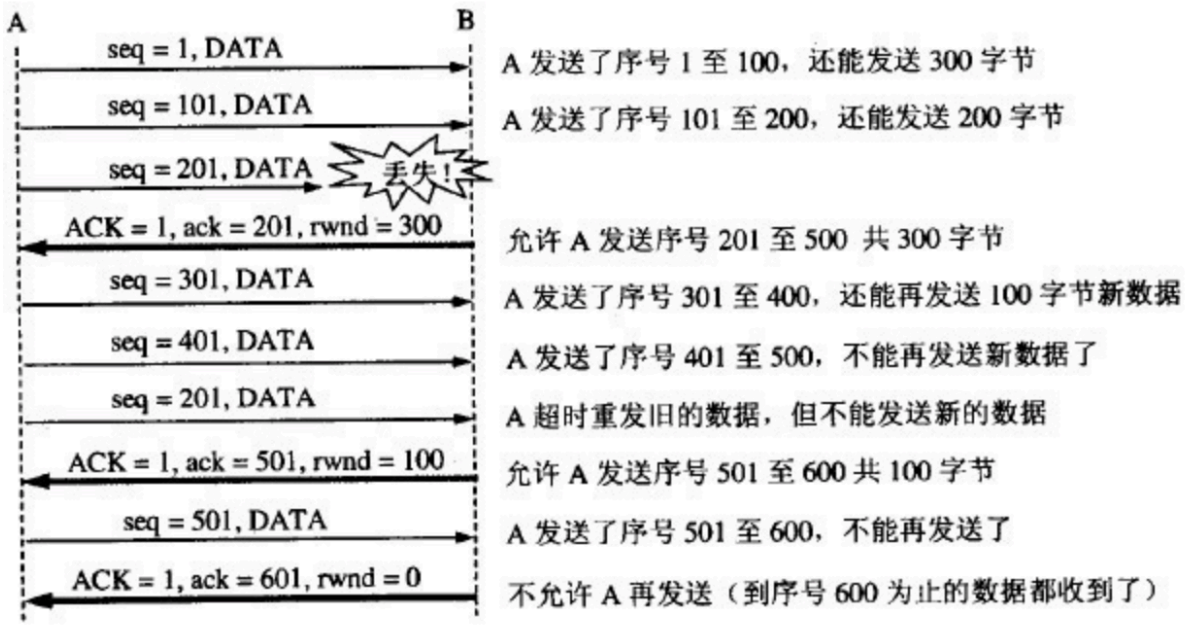
1. 利用滑动窗口实现流量控制

如果发送方把数据发送得过快，接收方可能会来不及接收，这就会造成数据的丢失。所谓**流量控制**就是让发送方的发送速率不要太快，要让接收方来得及接收。

利用滑动窗口机制可以很方便地在TCP连接上实现对发送方的流量控制。

设A向B发送数据。在连接建立时，B告诉了A：“我的接收窗口是 rwnd = 400 ”(这里的 rwnd 表示 receiver window) 。因此，发送方的发送窗口不能超过接收方给出的接收窗口的数值。请注意，TCP的窗口单位是字节，不是报文段。TCP连接建立时的窗口协商过程在图中没有显示出来。再设每一个报文段为100字节长，而数据报文段序号的初始值设为1。大写ACK表示首部中的确认位ACK，小写

ack表示确认字段的值ack。



从图中可以看出，B进行了三次流量控制。第一次把窗口减少到 rwnd = 300 ，第二次又减到了 rwnd = 100 ，最后减到 rwnd = 0 ，即不允许发送方再发送数据了。这种使发送方暂停发送的状态将持续到主机B重新发出一个新的窗口值为止。B向A发送的三个报文段都设置了 ACK = 1 ，只有在ACK=1时确认号字段才有意义。

TCP为每一个连接设有一个持续计时器(persistence timer)。只要TCP连接的一方收到对方的零窗口通知，就启动持续计时器。若持续计时器设置的时间到期，就发送一个零窗口控测报文段（携1字节的数据），那么收到这个报文段的一方就重新设置持续计时器。

2. 必须考虑传输速率

可以用不同的机制来控制TCP报文段的发送时机。如： <1>. TCP维持一个变量，它等于最大报文段长度MSS。只要缓存中存放的数据达到MSS字节时，就组装成一个TCP报文段发送出去。<2>. 由发送方的应用进程指明要求发送报文段，即TCP支持的推送(push)操作。<3>. 发送方的一个计时器期限到了，这时就把已有的缓存数据装入报文段(但长度不能超过MSS)发送出去。

Nagle算法：若发送应用进程把要发送的数据逐个字节地送到TCP的发送缓存，则发送方就把第一个数据字节先发送出去，把后面到达的数据字节都缓存起来。当发送方接收对第一个数据字符的确认后，再把发送缓存中的所有数据组装成一个报文段再发送出去，同时继续对随后到达的数据进行缓存。只有在收到对前一个报文段的确认后才继续发送下一个报文段。当数据到达较快而网络速率较慢时，用这样的方法可明显地减少所用的网络带宽。Nagle算法还规定：当到达的数据已达到 发送窗口大小的一半或已达到报文段的最大长度时，就立即发送一个报文段。

另，**滑动窗口综合证：**TCP接收方的缓存已满，而交互式的应用进程一次只从接收缓存中读取1字节（这样就使接收缓存空间仅腾出1字节），然后向发送方发送确认，并把窗口设置为1个字节（但发送的数据报为40字节的的话）。接收，发送方又发来1个字节的数据（发送方的IP数据报是41字节）。接收方发回确认，仍然将窗口设置为1个字节。这样，网络的效率很低。要解决这个问题，可让接收方等待一段时间，使得或者接收缓存已有足够空间容纳一个最长的报文段，或者等到接收方缓存已有一半空闲的空间。只要出现这两种情况，接收方就发回确认报文，并向发送方通知当前的窗口大小。此外，发送方也不要发送太小的报文段，而是把数据报积累成足够大的报文段，或达到接收方缓存的空间的一半大小。

4次挥手中，在什么情况下会出现大量的close_wait状态呢?

最近测试环境server由于需要与大量的后台server交互，今天突然发现有大量的close_wait产生，于是仔细研究了一下：

如果我们的服务器程序处于CLOSE_WAIT状态的话，说明套接字是被动关闭的！

因为如果是CLIENT端主动断掉当前连接的话，那么双方关闭这个TCP连接共需要四个packet：

1.Client -> FIN -> Server

2.Client <- ACK <- Server

这时候Client端处于FIN_WAIT_2状态；而Server 程序处于CLOSE_WAIT状态。

3.Client <- FIN <- Server

这时Server 发送FIN给Client，Server 就置为LAST_ACK状态。

4.Client -> ACK -> Server　Client回应了ACK，那么Server 的套接字才会真正置为CLOSED状态。

Server 程序处于CLOSE_WAIT状态，而不是LAST_ACK状态，说明还没有发FIN给Client，那么可能是在关闭连接之前还有许多数据要发送或者其他事要做，　导致没有发这个FIN packet。

通常来说，一个CLOSE_WAIT会维持至少2个小时的时间（这个时间外网服务器通常会做调整，要不然太危险了）。　如果有个流氓特地写了个程序，给你造成一堆的CLOSE_WAIT，消耗你的资源，那么通常是等不到释放那一刻，系统就已经解决崩溃了。　只能通过修改一下TCP/IP的参数，来缩短这个时间：修改tcp_keepalive_*系列参数有助于解决这个问题。