计算机网络

一、计算机网络体系结构

OSI 体系结构	TCP/IP 协议集	
应用层	应用层	TELNET、FTP、HTTP、SMTP、DNS 等
表示层		
会话层		
传输层	传输层	TCP、UDP
网络层	网络层	IP、ICMP、ARP、RARP
数据链路层	网络接口层	各种物理通信网络接口
物理层		

1、五层协议

应用层：为特定应用程序提供数据传输服务，例如 HTTP、DNS 等协议。数据单位为报文。

传输层：为进程提供通用数据传输服务。由于应用层协议很多，定义通用的传输层协议就可以支持不断增多的应用层协议。运输层包括两种协议：传输控制协议 TCP，提供面向连接、可靠的数据传输服务，数据单位为报文段；用户数据报协议 UDP，提供无连接、尽最大努力的数据传输服务，数据单位为用户数据报。TCP 主要提供完整性服务，UDP 主要提供及时性服务。

网络层：为主机提供数据传输服务。而传输层协议是为主机中的进程提供数据传输服务。网络层把传输层传递下来的报文段或者用户数据报封装成分组。

数据链路层：网络层针对的还是主机之间的数据传输服务，而主机之间可以有很多链路，链路层协议就是为同一链路的主机提供数据传输服务。数据链路层把网络层传下来的分组封装成帧。

物理层：考虑的是怎样在传输媒体上传输数据比特流，而不是指具体的传输媒体。物理层的作用是尽可能屏蔽传输媒体和通信手段的差异，使数据链路层感觉不到这些差异。

2、OSI

其中表示层和会话层用途如下：

- 表示层：数据压缩、加密以及数据描述，这使得应用程序不必关心在各台主机中数据内部格式不同的问题。
- 会话层：建立及管理会话。

五层协议没有表示层和会话层，而是将这些功能留给应用程序开发者处理。

一、应用层：TELNET、FTP、TFTP、SMTP、SNMP、HTTP、BOOTP、DHCP、DNS

二、表示层：

- 文本：ASCII，EBCDIC
- 图形：TIFF，JPEG，GIF，PICT
- 声音：MIDI，MPEG，QUICKTIME

三、会话层：NFS、SQL、RPC 、X-WINDOWS、ASP（APPTALK会话协议）、SCP

四、传输层：TCP、UDP、SPX

五、网络层：IP、IPX、ICMP、RIP、OSPF(Open Shortest Path First开放式最短路径优先)

六、数据链路层：SDLC、HDLC、PPP、STP（Spanning Tree Protocol）、帧中继

七、物理层：EIA/TIA RS-232、EIA/TIA RS-449、V.35、RJ-45

3、TCP/IP

TCP/IP协议是一个协议集合。所以统称为TCP/IP。TCP/IP协议族中有一个重要的概念是分层，TCP/IP协议按照层次分为以下四层：应用层、传输层、网络层、数据链路层。为什么要分层？这就如同邓小平1978年的大包干，责任到人。一个层只负责一个层次的问题，如果出问题了，和其他的层次无关，只要维护这个层次也就好了。其实编程语言里也能体现这个分层理论，即封装性、隔离。

4、数据在各层之间的传递过程

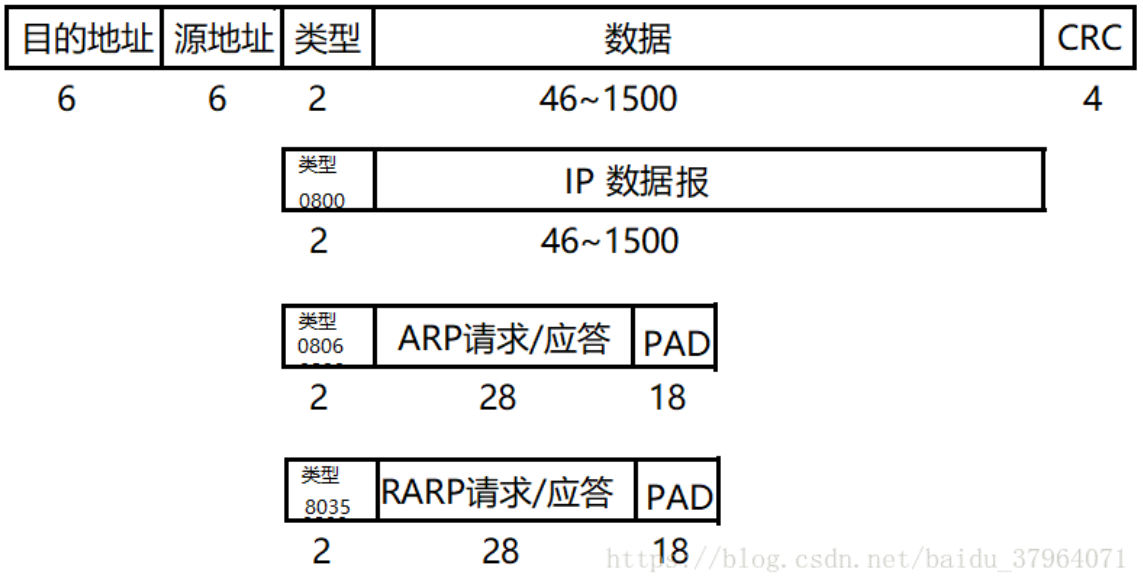
在向下的过程中，需要添加下层协议所需要的首部或者尾部，而在向上的过程中不断拆开首部和尾部。

路由器只有下面三层协议，因为路由器位于网络核心中，不需要为进程或者应用程序提供服务，因此也就不需要传输层和应用层。

二、数据链路层

负责帧数据的传递。（经过数据链路层封装的数据称为帧）

1、以太网帧格式



1. 源地址和目的地址是指网卡的硬件地址(也叫MAC地址)，长度是48位，是在网卡出厂时固化的
2. 帧协议类型字段有三种值,分别对应IP、ARP、RARP ，此变量决定了在分用过程中，网络层该利用哪个协议。
3. 帧末尾是CRC校验码，循环冗余校验字段（FCS）提供了一种错误检测机制。该字段长度为4个字节

2、MTU

从上图可以看出，数据段最长是1500字节，也就是说单个帧中，IP数据包必须小于1500字节，这个1500就是MTU能达到的最大值，它是数据链路层允许的最大IP包。MTU的定义就是：数据链路层允许的最大IP包长（其最大值是1500字节）。

1. 不同网络类型的MTU是不同的
2. 以太网帧中的数据长度规定最小46字节，最大1500字节
3. ARP数据包的长度不够46字节,要在后面补填充位

那如果数据包的长度大于1500字节呢?
如果一个数据包从以太网路由到拨号链路上,数据包长度大于拨号链路的MTU了，则需要对数据包进行**分片**。

由于以太网传输电气方面的限制，每个以太网帧最小64字节，最大不能超过1518字节，对于小于或者大于这个限制的以太网帧，以太网都可以视之为错误的数据

帧。一般的以太网转发设备会丢弃这些数据帧。（注：小于64字节的数据帧一般是由于以太网冲突产生的“碎片”或者线路干扰或者坏的以太网接口产生的，对于大于1518字节的数据帧我们一般把它叫做Giant帧，这种一般是由于线路干扰或者坏的以太网口产生）。

最早的以太网工作方式：在剥夺路复用/冲突检测（CSMA/CD），因为网络是共享的，即任何一个节点发送数据之前，先要侦听线路上是否有数据在传输，如果有，需要等待，如果线路可用，才可以发送。

CDMA/CD

CSMA/CD 表示载波监听多点接入 / 碰撞检测。

多点接入：说明这是总线型网络，许多主机以多点的方式连接到总线上。
载波监听：每个主机都必须不停地监听信道。在发送前，如果监听到信道正在使用，就必须等待。
碰撞检测：在发送中，如果监听到信道已有其它主机正在发送数据，就表示发生了碰撞。虽然每个主机在发送数据之前都已经监听到信道为空闲，但是由于电磁波的传播时延的存在，还是有可能会发生碰撞。

记端到端的传播时延为 τ，最先发送的站点最多经过 2τ 就可以知道是否发生了碰撞，称 2τ 为 争用期 。只有经过争用期之后还没有检测到碰撞，才能肯定这次发送不会发生碰撞。

假设A发送第一个bit位到达B，而B也正在传输第一个bit位，于是产生冲突，冲突信号让A在完成最后一个bit位之前到达A，这个一来一回的时间间隙slot time是57.6us。

在10Mbps的网络中，在57.6us的时间内，能够传输576个bit，所以要求以太网帧最小长度为576个bit，从而让最极端的碰撞都能够被检测到。这576个bit换算一下就是72个字节，去掉8个字节的前导符和帧开始符，以太网的最小长度为64个字节。定义最小帧长度是为了使发送方能在一个帧的传输时间内检测到此帧是否在链路上产生冲突，如发生冲突，退避重发，若帧长小于最小帧，则无法检测帧传输中是否发生冲突。

所以说以太网的最小长度64byte是因为CSMS/CD限制所导致的，那最大长度1500byte的原因是什么?

IP头total length为两个byte，理论上IP packet 可以有65535byte，加上以太网帧头和尾，可以有65535+14+4=65553byte。如果在10Mbps的以太网上，将会占用共享链路长达50ms，这将严重影响其他主机的通信，特别是对延迟敏感的应用是无法接受的。

由于线路质量差引起的丢包，发生在大包的概率也比小包的概率大得多，所以大包在丢包率较高的线路上不是一个好的选择。

但是如果选择一个较小的长度，传输效率又不高，拿TCP应用来说，如果选择以太网的长度为218byte，TCP数据包 = 218-以太网帧头和尾-IP头-TCP头 = 218-18-20-20=160byte。此时有效传输效率=160/218-73%。

而如果以太网长度为1518，那么有效传输效率=1460/1518=96%。

通过比较，选择较长的帧长度，有效传输效率更高，而更大的帧长度同时也会造成上面的问题，于是选择一个折中的长度：1518byte，对应的IP数据包长度为1500byte，这也是最大传输单元MTU的由来。

3、ARP协议

ARP（Address Resolution Protocol，地址解析协议）是将IP地址解析为以太网MAC地址（物理地址）的协议。

在局域网中，当主机或其他网络设备有数据要发送给另一个主机或设备时，它必须知道对方的网络层地址（即IP地址）。但是仅仅有IP地址是不够的，因为IP数据报文必须封装成帧才能通过物理网络发送。因此发送方还需要有接收方的物理地址，也就需要一个从IP地址到物理地址的映射，ARP就是事先这么功能的协议。

3.1工作原理

两个主机在同一个网段中

1、主机A首先查看自己的ARP表（一个IP地址与MAC地址的映射表），确定其中是否包含有主机B的IP地址和对应的MAC地址。如果找到了对应的MAC地址，则主机A直接利用ARP表中的MAC地址对IP数据报进行帧封装，并将数据报发送给主机B。类似于计算机组成中高速缓存的命中。

2、如果主机A在ARP表中找不到对应的MAC地址，则先缓存该数据报文，然后以广播方式（参见数据链路层物理地址寻址部分，目的MAC地址为广播MAC地址——FFFFFF，任一同网段的节点均可收到，该网络上的所有主机）发送一个ARP请求报文，ARP请求报文中的发送端（源）IP地址和发送端MAC地址分别为主机A的IP地址和MAC地址，目的IP地址和目的MAC地址为主机B的IP地址和全0的MAC地址。ARP请求报文是以广播的形式发送，所以该网段上的所有主机都会接收到这个请求包，但只有其IP地址与目的IP地址一致的主机B会对该请求进行处理。

3、主机B将ARP请求报文中的发送端（主机A）的IP地址和MAC地址存入自己的ARP表中，然后以单播方式（一对一，点对点形式）向主机A发送一个ARP响应报文，应答报文中就包含了自己的MAC地址，即原来请求报文中要请求的MAC地址。

4、主机A在收到来自主机B的ARP响应报文后，将主机B的MAC地址加入到自己的ARP表中以用于后续报文的转发，同时将原来缓存的IP数据报再次修改（在目的MAC地址子弹填上已获得的主机B的的MAC地址）后发送出去。

当两主机不在同一个网段中，其ARP地址解析过程如下

1、如果主机A不知道网关（一个网络连接另一个网络的关口）的MAC地址（即主机A的ARP表没有命中），则主机A现在本网段中发出一个ARP请求广播报文，ARP请求报文中的目的IP地址为网关IP地址，代表其目的就是想获得网关的MAC地址。如果主机A已知网关MAC地址，则略过此步。

2、如果网关的ARP表（网关也有ARP映射表项）中已有主机B对应的MAC地址，则网关直接将在来自主机A的报文中的目的MAC地址字段填上主机B的MAC地址后转发给主机B。

3、如果网关ARP中没有，那么网关会再次向主机B所在网段发送ARP广播请求报文，此时目的IP地址为主机B的IP地址，其后续处理同前。

3.2ARP数据包消息格式

硬件类型：表示ARP报文可以在哪种类型的网络上传输

上层协议类型：表示硬件地址要映射的协议地址类型，若为IP协议，其值为4

操作类型：指定本次ARP报文类型，例如，1表示ARP请求，2表示ARP应答

ARP报文并不是直接在网络层上发送的，它还是需要向下传输到数据链路层，所以当ARP报文传输到数据链路层后，需要再次进行封装。在以太网中，ARP传输到数据链路层后会封装成ARP帧。其格式如下图所示：

可以看出，其帧封装都是一样的，网络层的直接作为数据链路层帧的数据部分。

目的MAC地址：如果是ARP请求帧，因为它是一个广播帧，所以要填上广播MAC地址——FF-FF-FF-FF，其目标为网络上的所有主机。

源MAC地址：发送ARP帧的节点MAC地址

帧类型：标识帧封装的上层协议

4、RARP协议

RARP分组的格式与ARP分组基本一致。RARP为逆地址解析协议，作用与ARP相反，用于将MAC地址转换为IP地址。

具有本地磁盘的系统引导时，一般是从磁盘上的配置文件中读取IP地址，但是无盘机，如X中断或无盘工作站，则需要采用其他方法来获得IP地址。

总得说来就是，网络上的每个系统都具有唯一的硬件地址，它是由网络接口生产厂家配置的。无盘系统的RARP实现过程是从接口卡上读取唯一的硬件地址，然后发送一份RARP请求，请求某个主机响应该无盘系统的IP地址。

RARP的工作过程如下：
1、网络上每台设备都会有一个独一无二的硬件地址，一般是由设备厂商分配的MAC地址。发送主机从网卡上读取MAC地址，然后在网络上发送一个RARP请求的广播数据包，请求任何收到此请求的RARP服务器分配一个IP地址；

2、RARP服务器收到此请求后，检查其RARP表项，查找该MAC地址对应的IP地址；

3、如果存在，RARP服务器就给发送主机回复一个响应数据包，并将此IP地址提供给对方主机使用；

4、如果不存在，RARP服务器对此不做任何的响应；

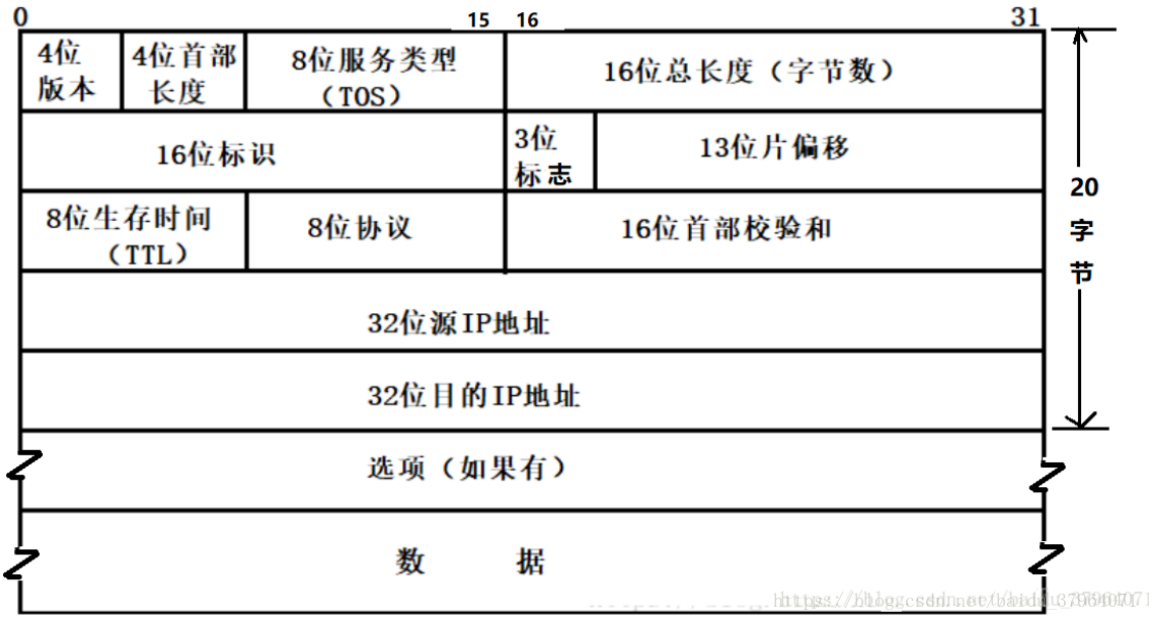
5、发送主机收到从RARP服务器的响应信息，就利用得到的IP地址进行通讯，如果一直没有收到RARP服务器的响应消息，表示初始化失败。
与ARP不同的是RARP是主机向RARP服务器获取自己的IP地址。

三、网络层

1、IP

IP提供**不可靠、无连接、无状态**的数据报传输服务。
不可靠：它不保证数据报能成功的到达目的地（如果发生某些错误，比如路由器用完了缓冲区，IP处理方法就是丢弃，然后发送ICMP差错报文给信源端。所有的可靠性都由上层来保证）
无状态：IP并不维护后续数据报的状态信息，每个数据报处理时都相互独立、没有上下文关系。因此缺点是无法处理乱序和重复的IP数据报。（IP数据报头部的标识字段是唯一标识一个IP数据报的，但它是用来处理IP分片和重组的，而不是用来指示接收顺序的
无连接：指IP通信双方不长时间维持对方的任何信息，这样，上层协议每次发送数据的时候，都必须明确指定对方的IP地址。

1.1IP首部格式



版本：有 4（IPv4）和 6（IPv6）两个值；

**首部长度 **: 占 4 位，标识该IP头部有多少个32bit字（4字节），因为4位最大能表示15，所以IP头部最长是60字节。因为固定部分长度为 20 字节，因此该值最小为 5。如果可选字段的长度不是 4 字节的整数倍，就用尾部的填充部分来填充。

服务类型：包括3位优先级字段、1位保留字段和4位TOS字段。4为TOS分别表示：最小时延、最大吞吐量、最高可靠性和最小费用。其中最多有一个能置为1。

总长度：指整个IP数据报的长度，以字节为单位，因此IP数据报的最大长度为65535（2^16-1）字节。但是由于MTU限制，长度超过MTU的数据报将会分片传输，所以实际传输的IP数据报长度远远没有达到最大值。以下3个字段描述了如何分片；

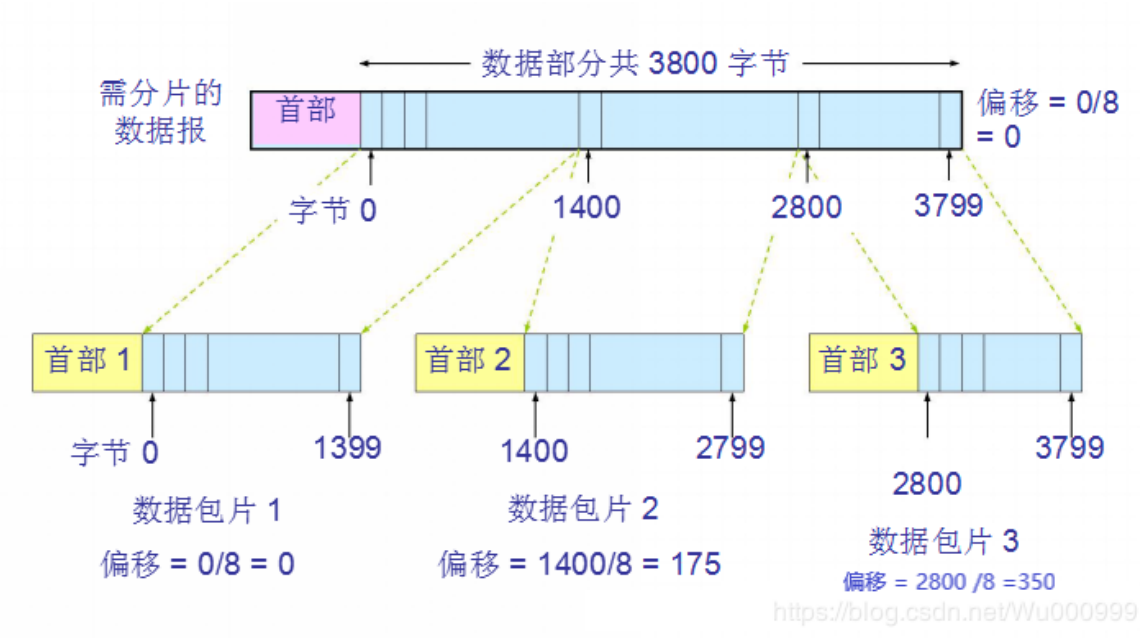
16位标识：唯一标识主机发送的每一个数据报，其初始值是系统随机生成：每发送一个数据报，其值加1.该值在分片时被复制到每一个分片中，因此同一个数据报的所有分片都有相同的标识值。

3位标志：这三位同一时刻也是只能有一个位的值能设置为1
R：标志字段中的第一位是一个保留位，现在还没有使用，可能将来会用到这位
D：标志字段中间的一位是 DF (Don’ t fragment)，表示传输的数据不允许分片。一般DF = 1的话，表示数据一次性传输过去，不允许分片。超过MTU的数据将会丢弃并返回一个ICMP差错报文。
M：标志字段的最低位是 MF (More fragment)。代表数据是否分片，如果MF位

值为1，表示后面还有数据，还没有传输完毕，相当于数据分片，分批次传输，如果MF = 0表示最后一个分片或者只有一个分片。

13位分片偏移：是分片相对原始IP数据报开始处的偏移（仅指数据部分）。实际偏移值是该值左移3位得到的（以8字节为单位）。所以除最后一个分片，其他分片的数据部分长度必须是8的整数倍。

为什么是以8字节为单位?
它是由IP头部格式中的“总长度（16bit）”和“偏移（13bit）”两个字段所决定的。总长度定义了IP包的最大长度为2^16 =64KB,偏移说明了IP分片时它最多能表示2^13 个偏移单位，这样偏移单位就是总长度除以偏移量得出片偏移单位（即2^16 / 2^13=2^3，即为8字节了）。



因为偏移量是以8字节为偏移单位，对于分片1来说从0字节开始算，偏移量为0 / 8 = 0，对于分片2来说从1400字节开始算，偏移量为1400 / 8 = 175，对于分片3来说是从2800开始算，偏移量为2800 / 8 = 350。

片偏移为0表示这可能是第一个分片，也有可能是这个数据报文不支持分片。

另外，这些数据包分片的ip首部大部分都是一样的，因为它们都属于同一数据报文，对于数据进行重组就需要参考片偏移和标识这两个重要的字段，需要注意的是，片偏移是根据某一数据片的开始位置来计算的。

8位生存时间（TTL）：设置了数据可以经过的最多的路由器数（一般是64），每经过一次路由器，该值减1，如果该值减为0依旧没有到达目的主机，就丢弃改数

据报，发送ICMP差错报文（目标不可达）

8位协议：用于区分上层协议。其中ICMP为1，TCP为6，UDP为17等等。

协议名	ICMP	IGMP	TCP	EGP	IGP	UDP	IPv6	OSPF
协议字段值	1	2	6	8	9	17	41	89

16位首部校验和：由发送端填充，接收端对其使用CRC算法检验IP数据报头部在传输过程中是否损坏（只检查头部，不管数据部分）

来看看IP数据报是**如何检验**的：

1、在发送方，先把IP数据报首部划分为许多的16位字的序列，并把检验和字段置零。

2、在用反码算数运算把所有16位字相加后，将得到的和的反码写进检验和字段。

3、接收方收到数据报之后，将首部的所有16位字在使用反码算数运算相加一次，将得到的和取反码，即得出接收方检验和的计算结果。

4、我们根据接收方检验和结果判断，如果结果为0，表示此数据报有效，不为0就丢弃该数据报。

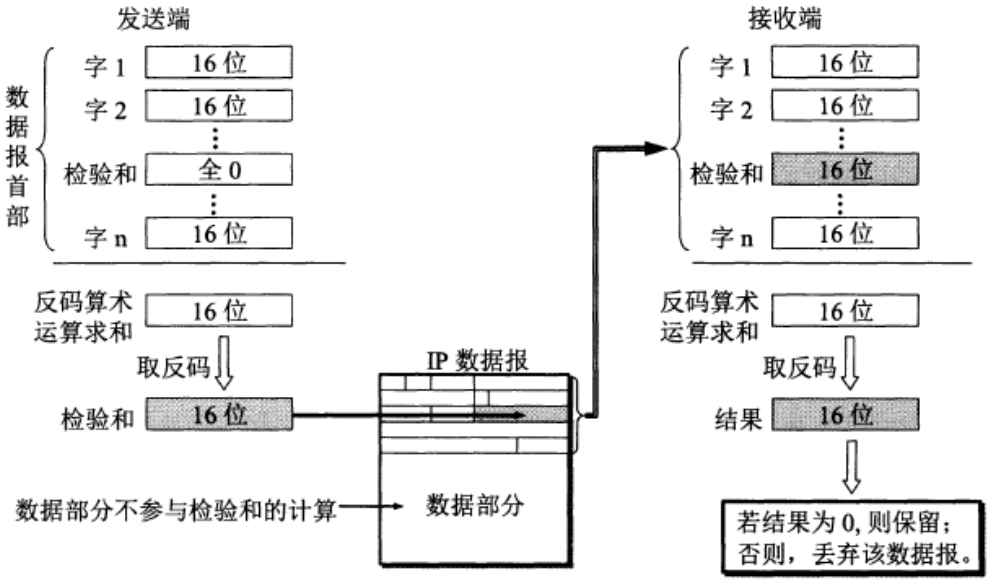


图 4-15　IP 数据报首部检验和的计算过程

32位源端IP地址和32位目的IP地址：用来指定发送端和接收端的。

1.2IP分片

若IP协议在传输数据包时，IP报文长度大于转发接口的MTU，则将数据报文分为若干分片进行传输，分片报文到达接收方时，由接收方完成重组。

对于不同的传输层协议，在IP层上，需不需要进行分片是不同的：

1.2.1对于TCP来说

它是尽量避免分片的。因为当在IP层进行了分片后，如果其中的某片数据丢失，则需对整个数据报进行重传。因为IP层本身没有超时重传机制，当来自TCP报文段的某一片丢失后，TCP在超时后重发整个TCP报文段，该报文段对应于一份IP数据报，没有办法只重传数据报中的一个数据报片。而且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，因此基于这种原因，TCP是经常要避免分片的。

那么TCP层是如何避免IP层的分片呢?
首先，TCP在建立连接时会进行3次握手，而在这3次握手中，客户端和服务端通常会协商一个值，那就是**MSS（最长报文大小）**，用来表示本段所能接收的最大长度的报文段。MSS=MTU-TCP首部大小-IP首部大小，MTU值通过查询链路层得知。

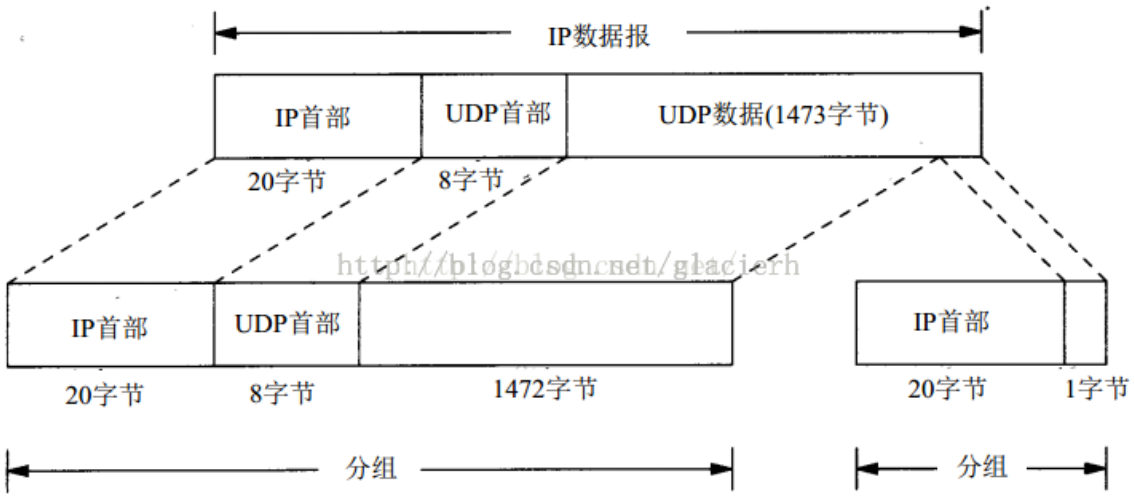
当两端确认好MSS后进行通信，TCP层往IP层传输数据时，如果TCP层缓冲区的大小大于MSS，那么TCP层都会将其中的数据分组进行传输，这样就避免了在IP层进行分片。

1.2.2对于UDP而言

由于UDP是不需要保证可靠性的，没有超时和重传机制，这使得UDP很容易导致IP分片。

当IP数据报超过帧的MTU(最大传输单元)时，它将会被分片传输。分片能发生在发送端或者中转路由器，且在传输过程中可能被多次分片。在最后的目标机器上这些分片才会被内核的的IP模块重新组装。

在分片的数据中，传输层的首部只会出现在第一个分片中，如下图所示。因为传输层的数据格式对IP层是透明的，传输层的首部只有在传输层才会有它的作用，IP层不知道也不需要保证在每个分片中都有传输层首部。所以，在网络上传输的数据包是有可能没有传输层首部的。



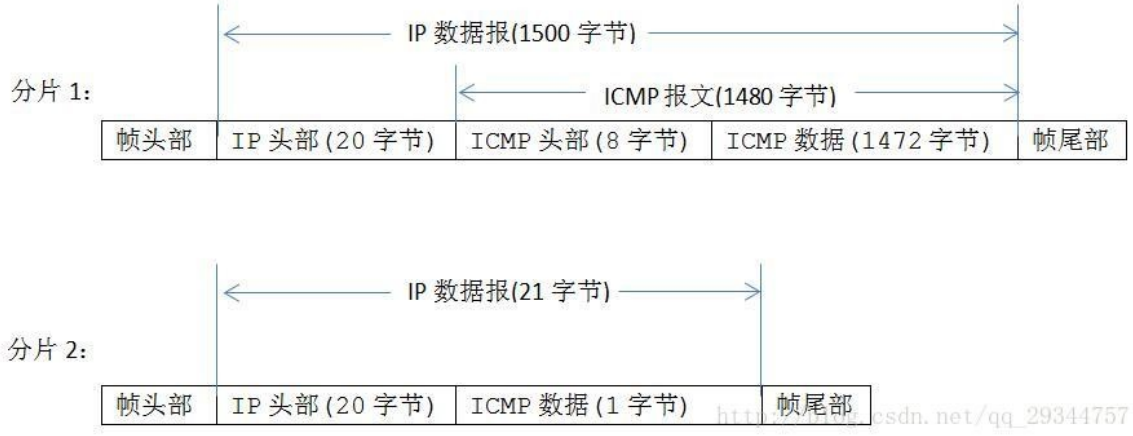
在IPv4的头部信息中有3个字段专门为IP分片服务的:标识（16位）、标志（3位）、分片偏移（13位）。

以太网帧的MTU是1500字节，因此它的数据部分最大为1480字节(IP头部占用20字节)。为观察IP分片的数据报，这里采用ICMP协议发送一个长度为1501字节的IP数据报，其中IP头部占用20字节，ICMP报文占据1481字节。1481字节的ICMP数据报中含8字节ICMP头部，其他1473字节为数据部分。长度为1501的IP数据报被拆分为2个IP分片：

- 第1个IP分片：1480字节ICMP数据报文(含8字节的ICMP头部信息) + 20 字节IP头部信息 = 1500字节的IP数据报，设置了MF位
- 第2个IP分片：1字节的ICMP数据报文(不含8字节的ICMP头部信息) + 20 字节的IP头部信息 = 21字节的IP数据包，没有设置MF位。

用户要发送的以太网帧：

分片后：



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，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分片，传输过程中，哪怕丢失一片数据也要重传整个数据报。IP本身没有超时重传的机制，由更高层的TCP来完成，一些UDP应用程序也具有超时和重传。

在分片时，除最后一片外，其他每一片中的数据部（除IP首部外的其余部分）必须是8字节的整数倍。只有第一个分片携带四层信息，IP数据报（从传输层给网络层，由网络层发下来的数据叫数据报）是指IP层端到端的传输单元（分片之前和重组之后），分组（在网络中传输的叫分组）是指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。

当路由器收到一份需要分片的数据报，而在IP首部又设置了不分片（DF）的标志比特，路由器就会返回一个ICMP不可达信息。

MTU发现机制，在IPsec VPN中，要防止IP分片，因为有分片的话会严重影响效率，所以两台路由器都会做一个 path MTU发现技术，发送一个数据报为1500字节的IP数据报，并且DF置1。如果中间路由器MTU比1500小，就会对源端发送一个ICMP不可达信息，在这个ICMP不可达信息中，携带此路由器的最大MTU值，这时源端主机收到后修改数据报大小继续发送，如果中间路由器还有跟小的，又会向源发送一个ICMP不可达信息，源端主机再次修改，如此重复，只到到达目的地，就知道单向路径MTU的最小值了。

1.2.3分片带来的影响

1、分片带来的性能消耗

分片和重组会消耗发送方、接收方一定的CPU等资源，如果存在大量的分片报文的话，可能会造成较为严重的资源消耗；

分片对接收方内存资源的消耗较多，因为接收方要为接收到的每个分片报文分配内存空间，以便于最后一个分片报文到达后完成重组。

2、分片丢包导致的重传问题

如果某个分片报文在网络传输过程中丢失，那么接收方将无法完成重组，如果应用进程要求重传的话，发送方必须重传所有分片报文而不是仅重传被丢弃的那个分片报文，这种效率低下的重传行为会给端系统和网络资源带来额外的消耗。

3、分片攻击

黑客构造的分片报文，但是不向接收方发送最后一个分片报文，导致接收方要为所有的分片报文分配内存空间，可由于最后一个分片报文永远不会达到，接收方的内存得不到及时的释放（接收方会启动一个分片重组的定时器，在一定时间内如果无法完成重组，将向发送方发送ICMP重组超时差错报文，，只要这种攻击的分片报文发送的足够多、足够快，很容易占满接收方内存，让接收方无内存资源处理正常的业务，从而达到DOS的攻击效果。

4、安全隐患

由于分片只有第一个分片报文具有四层信息而其他分片没有，这给路由器、防火墙等中间设备在做访问控制策略匹配的时候带来了麻烦。

如果路由器、防火墙等中间设备不对分片报文进行安全策略的匹配检测而直接放行IP分片报文，则有可能给接收方带来安全隐患和威胁，因为黑客可以利用这个特性，绕过路由器、防火墙的安全策略检查对接收方实施攻击；

如果路由器、防火墙等中间设备对这些分片报文进行重组后在匹配其安全策略，那么又会对这些中间设备的资源带来极大的消耗，特别是在遇到分片攻击的时候，这些中间设备会在第一时间内消耗完其所有内存资源，从而导致全网中断的严重后果。

1.2.4重组

为了重新组合这些数据报分片，接收主机在第一个分片到达时分配一个存储缓冲区。这个主机还将启动一个计时器。当数据报的后续分片到达时，数据被复制到缓冲区存储器中片偏移量字段指出的位置。当所有分片都到达时，完整的未分片的原始数据包就被恢复了。处理如同未分片数据报一样继续进行。

如果计时器超时并且分片保持尚未认可状态，则数据报被丢弃。这个计时器的初始值称为IP数据报的生存期值。它是依赖于实现的。一些实现允许对它进行配置。在某些IP主机上可以使用netstat命令列出分片的细节。如TCP/IP for OS/2中的netstat-i命令。

重组的步骤：
在接收方，一个由发送方发出的原始IP数据报，其所有分片将被重新组合，然后才能提交到上层协议。每一个将被重组的IP数据报都用一个ipq结构实例来表示，因此先来看看ipq这个非常重要的结构。

为了能高效地组装分片，用于保存分片的数据结构必须能做到以下几点：

1、快速定位属于某一个数据报的一组分组
2、在属于某一个数据报的一组分片中快速插入新的分片
3、有效地判断一个数据报的所有分片是否已经全部接收
4、具有组装超时机制，如果在重组完成之前定时器溢出，则删除该数据报的所有内容

2、网际控制报文协议ICMP

ICMP 是为了更有效地转发 IP 数据报和提高交付成功的机会。它封装在 IP 数据报中，但是不属于高层协议。

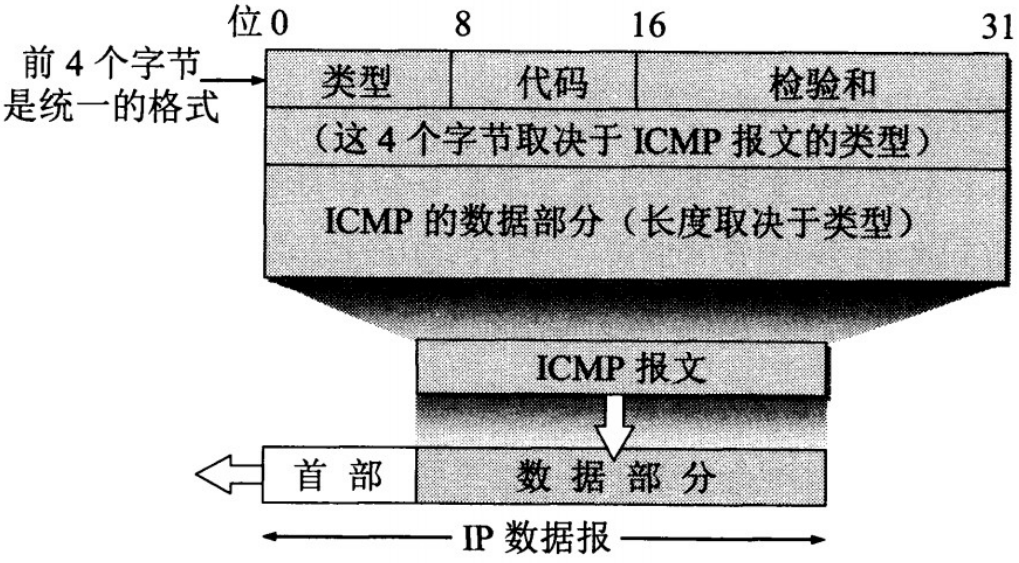


图 4-27　ICMP 报文的格式

类型：占一字节，标识ICMP报文的类型，目前已定义了14种，从类型值来看ICMP报文可以分为两大类。第一类是取值为1~127的差错报文，第2类是取值128以上的信息报文。
代码：占一字节，标识对应ICMP报文的代码。它与类型字段一起共同标识了ICMP报文的详细类型。
校验和：这是对包括ICMP报文数据部分在内的整个ICMP数据报的校验和，以检验报文在传输过程中是否出现了差错。其计算方法与在我们介绍IP报头中的校验和计算方法是一样的。

ICMP报文分为两种类型（1）ICMP差错报告报文，即通知出错原因的错误消息（如traceroute） （2）ICMP询问报文，即用于诊断的查询消息（如ping）
差错报告报文主要用来向 IP 数据报源主机返回一个差错报告信息， 这个错误报告信息产生的原因是路由器或主机不能对当前数据报进行正常的处理，例如无法将数据报递交给有效的协议上层，又例如数据报因为生存时间 TTL 为 0 而被删除等。
查询报文用于一台主机向另一台主机查询特定的信息，通常查询报文都是成对出现的，即源主机发起一个查询报文，在目的主机收到该报文后，会按照查询报文约定的格式为源主机返回一个应答报文。

表 11-1　ICMP 报文分类

ICMP 报文种类	具体类型	报文功能
差错报告报文	3	目的站不可达
	4	源站抑制
	5	重定向（改变路由）
	11	数据报超时
	12	数据报参数错误
查询报文	8 或 0	回送请求或回答
	10 或 9	路由器询问和通告
	13 或 14	时间戳请求或回答
	15 或 16	信息请求或回答
	17 或 18	地址掩码请求或回答

往返时间rtt（round trip time）是应答时间-发送时间
生存时间ttl（time to live）是IP首部中的一个字段，指的是报文能够在网络中存活的时间，一般ttl的值是255，不同系统的ttl值不一样。每经过一个路由器时，ttl值就减1，当减到0时，这个数据包就被路由器抛弃了。

2.1Ping

- 能验证网络的连通性

- 会统计响应时间和TTL(IP包中的Time To Live，生存周期)

Ping 的原理是通过向目的主机发送 ICMP Echo 请求报文，目的主机收到之后会发送 Echo 回答报文。Ping 会根据时间和成功响应的次数估算出数据包往返时间以及丢包率。

2.2Traceroute

打印出可执行程序主机，一直到目标主机之前经历多少路由器。

Traceroute 发送的 IP 数据报封装的是无法交付的 UDP 用户数据报，并由目的主机发送终点不可达差错报告报文。

- 源主机向目的主机发送一连串的 IP 数据报。第一个数据报 P1 的生存时间 TTL 设置为 1，当 P1 到达路径上的第一个路由器 R1 时，R1 收下它并把 TTL 减 1，此时 TTL 等于 0，R1 就把 P1 丢弃，并向源主机发送一个 ICMP 时间超过差错报告报文；
- 源主机接着发送第二个数据报 P2，并把 TTL 设置为 2。P2 先到达 R1，R1 收下后把 TTL 减 1 再转发给 R2，R2 收下后也把 TTL 减 1，由于此时 TTL 等于 0，R2 就丢弃 P2，并向源主机发送一个 ICMP 时间超过差错报文。
- 不断执行这样的步骤，直到最后一个数据报刚刚到达目的主机，主机不转发数据报，也不把 TTL 值减 1。但是因为数据报封装的是无法交付的 UDP，因此目的主机要向源主机发送 ICMP 终点不可达差错报告报文。
- 之后源主机知道了到达目的主机所经过的路由器 IP 地址以及到达每个路由器的往返时间。

怎么知道UDP到没到达目的主机呢？这就涉及一个技巧的问题，TCP和UDP协议有一个端口号定义，而普通的网络程序只监控少数的几个号码较 小的端口，比如说80,比如说23,等等。而traceroute发送的是端口号>30000(真变态)的UDP报，所以到达目的主机的时候，目的 主机只能发送一个端口不可达的ICMP数据报给主机。主机接到这个报告以后就知道，主机到了。

3、网络地址转换 NAT

专用网内部的主机使用本地 IP 地址又想和互联网上的主机通信时，可以使用 NAT 来将本地 IP 转换为全球 IP。

在以前，NAT 将本地 IP 和全球 IP 一一对应，这种方式下拥有 n 个全球 IP 地址的专用网内最多只可以同时有 n 台主机接入互联网。为了更有效地利用全球 IP 地

址，现在常用的 NAT 转换表把传输层的端口号也用上了，使得多个专用网内部的主机共用一个全球 IP 地址。使用端口号的 NAT 也叫做网络地址与端口转换 NAPT。

表 4-12　NAPT 地址转换表举例

方向	字段	旧的 IP 地址和端口号	新的 IP 地址和端口号
出	源 IP 地址:TCP 源端口	192.168.0.3:30000	172.38.1.5:40001
出	源 IP 地址:TCP 源端口	192.168.0.4:30000	172.38.1.5:40002
入	目的 IP 地址:TCP 目的端口	172.38.1.5:40001	192.168.0.3:30000
入	目的 IP 地址:TCP 目的端口	172.38.1.5:40002	192.168.0.4:30000

4、路由器分组转发流程

从数据报的首部提取目的主机的 IP 地址 D，得到目的网络地址 N。

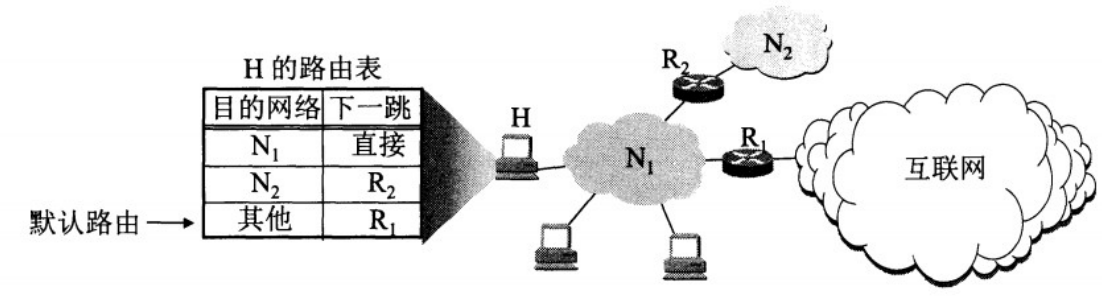
若 N 就是与此路由器直接相连的某个网络地址，则进行直接交付；

若路由表中有目的地址为 D 的特定主机路由，则把数据报传送给表中所指明的下一跳路由器；

若路由表中有到达网络 N 的路由，则把数据报传送给路由表中所指明的下一跳路由器；

若路由表中有一个默认路由，则把数据报传送给路由表中所指明的默认路由器；

报告转发分组出错。



5、路由选择协议

路由选择协议都是自适应的，能随着网络通信量和拓扑结构的变化而自适应地进行调整。

互联网可以划分为许多较小的自治系统 AS，一个 AS 可以使用一种和别的 AS 不同的路由选择协议。

可以把路由选择协议划分为两大类：

- 自治系统内部的路由选择：RIP 和 OSPF
- 自治系统间的路由选择：BGP

5.1内部网关协议 RIP

RIP 是一种基于距离向量的路由选择协议。距离是指跳数，直接相连的路由器跳数为 1。跳数最多为 15，超过 15 表示不可达。

RIP 按固定的时间间隔仅和相邻路由器交换自己的路由表，经过若干次交换之后，所有路由器最终会知道到达本自治系统中任何一个网络的最短距离和下一跳路由器地址。

距离向量算法：

- 对地址为 X 的相邻路由器发来的 RIP 报文，先修改报文中的所有项目，把下一跳字段中的地址改为 X，并把所有的距离字段加 1；
- 对修改后的 RIP 报文中的每一个项目，进行以下步骤：
- 若原来的路由表中没有目的网络 N，则把该项目添加到路由表中；
- 否则：若下一跳路由器地址是 X，则把收到的项目替换原来路由表中的项目；否则：若收到的项目中的距离 d 小于路由表中的距离，则进行更新（例如原始路由表项为 Net2, 5, P，新表项为 Net2, 4, X，则更新）；否则什么也不做。
- 若 3 分钟还没有收到相邻路由器的更新路由表，则把该相邻路由器标为不可达，即把距离置为 16。

RIP 协议实现简单，开销小。但是 RIP 能使用的最大距离为 15，限制了网络的规模。并且当网络出现故障时，要经过比较长的时间才能将此消息传送到所有路由器。

缺点：

1). RIP的15跳限制，超过15跳的路由被认为不可达

2). RIP不能支持可变长子网掩码(VLSM)，导致IP地址分配的低效率

3). 周期性广播整个路由表，在低速链路及广域网云中应用将产生很大问题

4). 收敛速度慢于OSPF，在大型网络中收敛时间需要几分钟

5). RIP没有网络延迟和链路开销的概念，路由选路基于跳数。拥有较少跳数的路

由总是被选为最佳路由即使较长的路径有低的延迟和开销

6). RIP没有区域的概念，不能在任意比特位进行路由汇

5.2内部网关协议 OSPF

开放最短路径优先 OSPF，基于链路状态及最短路径树算法，是为了克服 RIP 的缺点而开发出来的。

OSPF 具有以下特点：

- 向本自治系统中的所有路由器发送信息，这种方法是洪泛法。
- 发送的信息就是与相邻路由器的链路状态，链路状态包括与哪些路由器相连以及链路的度量，度量用费用、距离、时延、带宽等来表示。
- 只有当链路状态发生变化时，路由器才会发送信息。

5.3外部网关协议 BGP

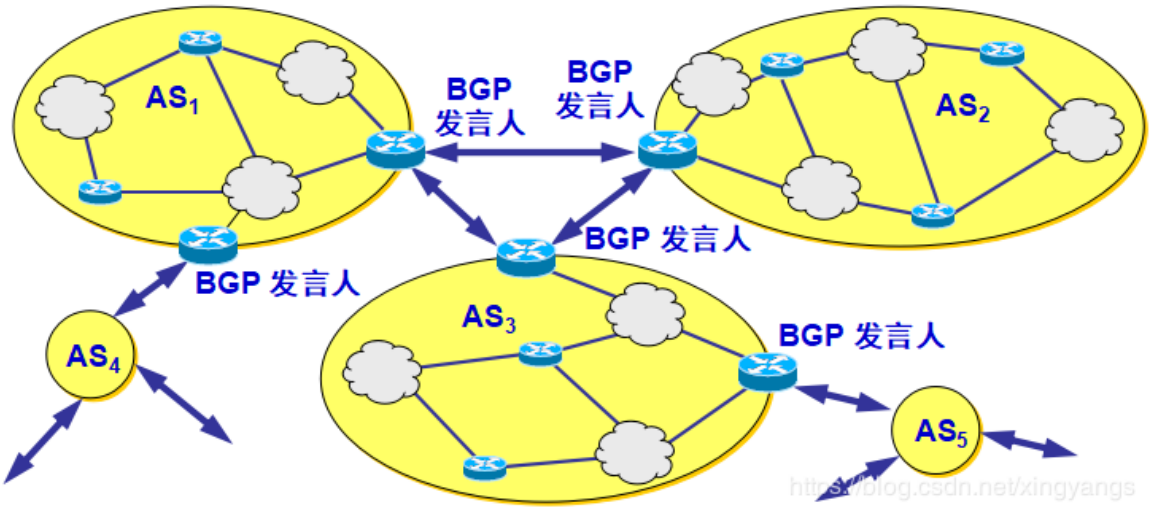
BGP（Border Gateway Protocol，边界网关协议）

AS（自治系统）之间的路由选择很困难，主要是由于：

- 互联网规模很大；
- 各个 AS 内部使用不同的路由选择协议，无法准确定义路径的度量；
- AS 之间的路由选择必须考虑有关的策略，比如有些 AS 不愿意让其它 AS 经过。

BGP 只能寻找一条比较好的路由，而不是最佳路由。

每个 AS 都必须配置 BGP 发言人，通过在两个相邻 BGP 发言人之间建立 TCP 连接来交换路由信息。



四、传输层

1、UDP和TCP的特点

用户数据报协议 UDP（User Datagram Protocol）是无连接的，尽最大可能交付，没有拥塞控制，面向报文（对于应用程序传下来的报文不合并也不拆分，只是添加 UDP 首部），支持一对一、一对多、多对一和多对多的交互通信。

传输控制协议 TCP（Transmission Control Protocol）是面向连接的，提供可靠交付，有流量控制，拥塞控制，提供全双工通信，面向字节流（把应用层传下来的报文看成字节流，把字节流组织成大小不等的数据块），每一条 TCP 连接只能是点对点的（一对一）。

2、UDP

- UDP无连接，时间上不存在建立连接需要的时延。空间上，TCP需要在端系统中维护连接状态，需要一定的开销。此连接装入包括接收和发送缓存，拥塞控制参数和序号与确认号的参数。UDP不维护连接状态，也不跟踪这些参数，开销小。空间和时间上都具有优势。
- 分组首部开销小，TCP首部20字节，UDP首部8字节。
- UDP没有拥塞控制，应用层能够更好的控制要发送的数据和发送时间，网络中的拥塞控制也不会影响主机的发送速率。某些实时应用要求以稳定的速度发送，能容忍一些数据的丢失，但是不能允许有较大的时延（比如实时视频，直播等）
- UDP提供尽最大努力的交付，不保证可靠交付。所有维护传输可靠性的工作需要用户在应用层来完成。没有TCP的确认机制、重传机制。如果因为网络原因没有传送到对端，UDP也不会给应用层返回错误信息
- UDP是面向报文的，对应用层交下来的报文，添加首部后直接向下交付为IP层，既不合并，也不拆分，保留这些报文的边界。对IP层交上来UDP用户数据报，在去除首部后就原封不动地交付给上层应用进程，报文不可分割，是UDP数据报处理的最小单位。
  正是因为这样，UDP显得不够灵活，不能控制读写数据的次数和数量。比如我们要发送100个字节的报文，我们调用一次sendto函数就会发送100字节，对端也需要用recvfrom函数一次性接收100字节，不能使用循环每次获取10个字节，获取十次这样的做法。
- UDP支持一对一、一对多、多对一和多对多的交互通信。

2.1首部格式

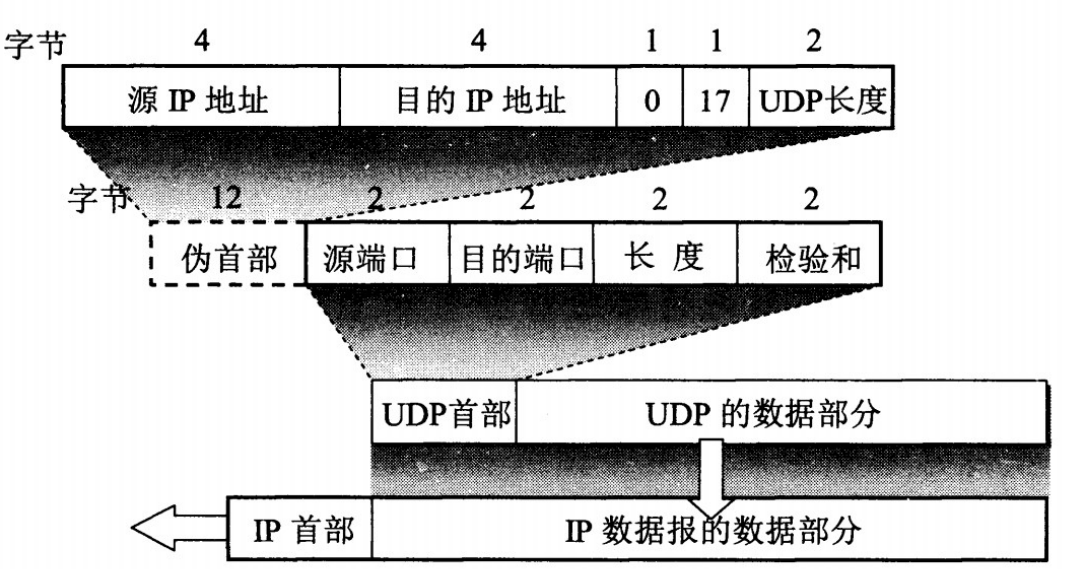


图 5-5　UDP 用户数据报的首部和伪首部

UDP首部有8个字节，由4个字段构成，每个字段都是两个字节，

源端口： 源端口号，需要对方回信时选用，不需要时全部置0.

目的端口：目的端口号，在终点交付报文的时候需要用到。

长度：UDP的数据报的长度（包括首部和数据）其最小值为8（只有首部）

校验和：检测UDP数据报在传输中是否有错，有错则丢弃。

该字段是可选的，当源主机不想计算校验和，则直接令该字段全为0.

当传输层从IP层收到UDP数据报时，就根据首部中的目的端口，把UDP数据报通过相应的端口，上交给应用进程。

如果接收方UDP发现收到的报文中的目的端口号不正确（不存在对应端口号的应用进程0,），就丢弃该报文，并由ICMP发送“端口不可达”差错报文给对方。

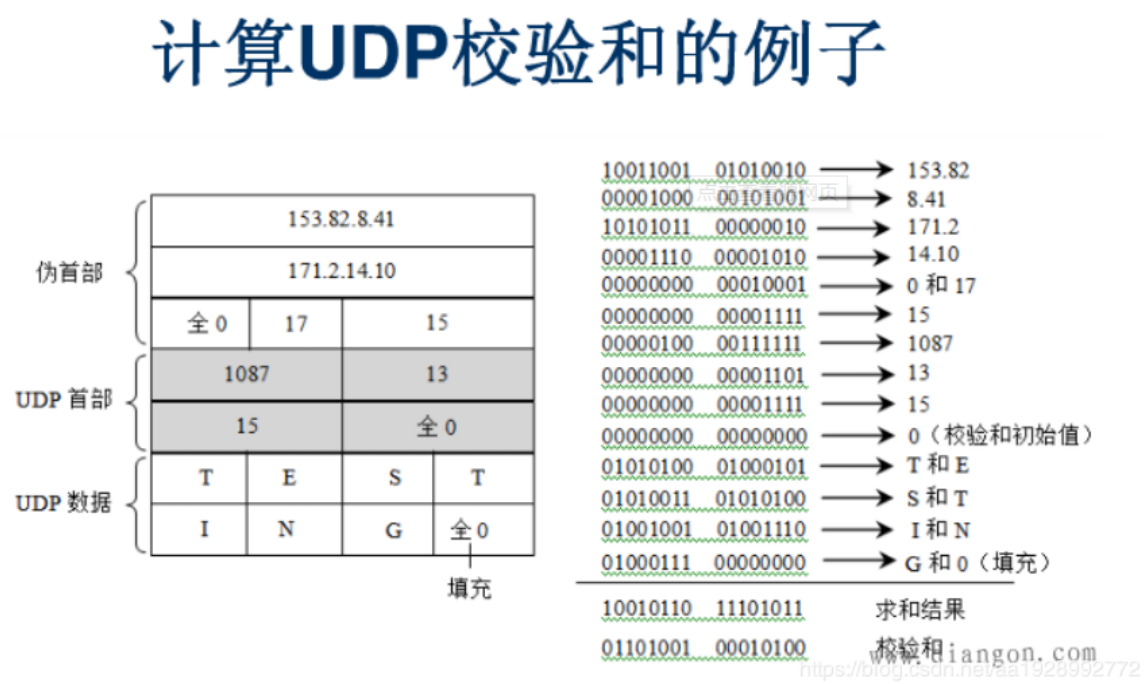
2.2UDP校验

在计算校验和的时候，需要在UDP数据报之前增加12字节的伪首部，伪首部并不是UDP真正的首部。只是在计算校验和，临时添加在UDP数据报的前面，得到一个临时的UDP数据报。校验和就是按照这个临时的UDP数据报计算的。伪首部既不向下传送也不向上递交，而仅仅是为了计算校验和。这样的校验和，既检查了UDP数据报，又对IP数据报的源IP地址和目的IP地址进行了检验。

UDP伪首部结构和TCP伪首部相同。只不过UDP伪首部中协议字段值为17，而TCP为6.

UDP校验和的计算方法和IP数据报首部校验和的计算方法相似，都使用二进制反码运算求和再取反，但不同的是：IP数据报的校验和之检验IP数据报和首部，但UDP的校验和是把首部和数据部分一起校验。

发送方，首先是把全零放入校验和字段并且添加伪首部，然后把UDP数据报看成是由许多16位的子串连接起来，若UDP数据报的数据部分不是偶数个字节，则要在数据部分末尾增加一个全零字节（此字节不发送），接下来就按照二进制反码计算出这些16位字的和。将此和的二进制反码写入校验和字段。在接收方，把收到得UDP数据报加上伪首部（如果不为偶数个字节，还需要补上全零字节）后，按二进制反码计算出这些16位字的和。当无差错时其结果全为1。否则就表明有差错出现，接收方应该丢弃这个UDP数据报。



1. 校验时，若UDP数据报部分的长度不是偶数个字节，则需要填入一个全0字节，但是次字节和伪首部一样，是不发送的。
2. 如果UDP校验和校验出UDP数据报是错误的，可以丢弃，也可以交付上层，但是要附上错误报告，告诉上层这是错误的数据报。
3. 通过伪首部，不仅可以检查源端口号，目的端口号和UDP用户数据报的数据部分，还可以检查IP数据报的源IP地址和目的地址。

2.3UDP可靠性传输实现

从应用层角度考虑：

- 提供超时重传，能避免数据报丢失。

- 提供确认序列号，可以对数据报进行确认和排序。

本端：首先在UDP数据报定义一个首部，首部包含确认序列号和时间戳，时间戳是用来计算RTT(数据报传输的往返时间)，从何计算出合适的RTO(重传的超时时间)。然后以等-停的方式发送数据报，即收到对端的确认之后才发送下一个的数据报。当时间超时，本端重传数据报，同时RTO扩大为原来的两倍，重新开始计时。
对端：接受到一个数据报之后取下该数据报首部的时间戳和确认序列号，并添加本端的确认数据报首部之后发送给对端。根据此序列号对已收到的数据报进行排序并丢弃重复的数据报。

发送方的处理：
1） 包发送确认后，由于还没有收到确认，先缓存
2） 收到确认包后，从缓存中删除发送的包
3） 接收方将丢失的包通知过来，或者超过一定的时候，若还没有收到确认的包，进行重传（注意，这个由接收线程触发）

接收方的处理：
1） 接收到包的数据，先将数据放到缓存中，a. 若有丢包现象，通知发送方，同时记录丢失的包 b.若是重传的包，从丢失的列表中删除
2） 发送确认包
3） 丢失的包，超时会让发送方再次发送

一些情况分析：
情况1：发送包a，接收方确认a，发送方收到确认：正常
情况2：发送包a，接收方确认a，发送方没有收到确认：发送方会重发此包，接收方收到此包忽略
情况3：发送包a，接收方没有收到a：发送方重发此包
情况4：发送包a，一直收不到确认，超过一定次数或时间后，结束
情况4：发送包a失败，结束

2.4UDP为什么会乱序

tcp 协议头有 seq 和 ack_seq 用来保证 tcp包的时序, 如果出现缺少其中一块, 会进行重传, 而 udp没有seq 和 ack_seq 来保证, 所以会乱序

3、TCP

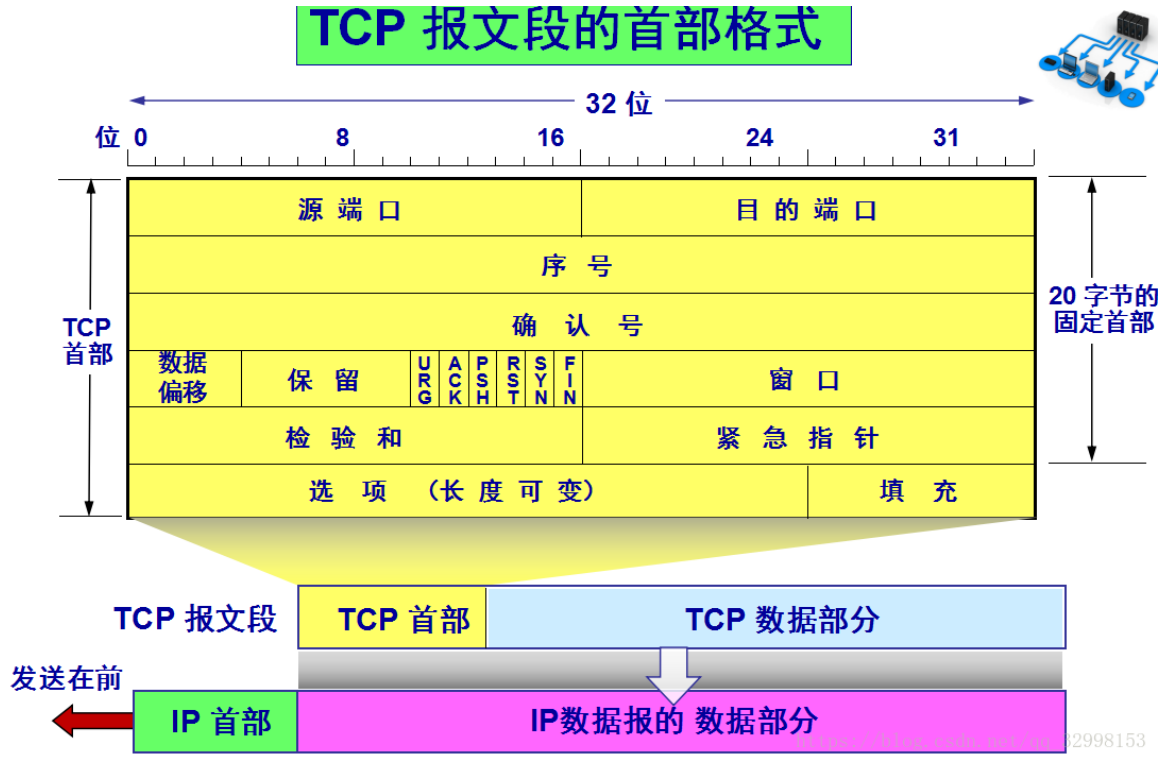
TCP是一个**面向连接的**(connection-oriented)、**可靠的**(reliable)、**字节流式**(byte stream)传输协议。

面向连接：在应用TCP协议进行通信之前双方通常需要通过**三次握手**来建立TCP连接，连接建立后才能进行正常的数据传输，因此广播和多播不会承载在TCP协议上。(谷歌提交了一个RFC文档，建议在TCP三次握手的过程允许SYN数据包中带数据，即 TFO(TCP Fast Open)，目前ubuntu14.04已经支持该TFO功能)。但是同时面向连接的特性给TCP带来了复杂的连接管理以及用于检测连接状态的存活检测机制。

可靠性：由于TCP处于多跳通信的IP层之上，而IP层并不提供可靠的传输，因此在TCP层看来就有四种常见传输错误问题，分别是比特错误(packet bit errors)、包乱序(packet reordering)、包重复(packet duplication)、丢包(packet erasure或称为packet drops)，TCP要提供可靠的传输，就需要有额外的机制处理这几种错误。因此个人理解可靠性体现在三个方面，首先TCP通过**超时重传和快速重传**两个常见手段来保证数据包的正确传输，也就是说接收端在没有收到数据包或者收到错误的数据包的时候会触发发送端的数据包重传(处理比特错误和丢包)。其次TCP接收端会缓存接收到的乱序到达数据，重排序后在向应用层提供有序的数据(处理包乱序)。最后TCP发送端会维持一个发送"窗口"动态的调整发送速率以适用接收端缓存限制和网络拥塞情况，避免了网络拥塞或者接收端缓存满而大量丢包的问题(降低丢包率)。因此**可靠性需要TCP协议具有超时与重传管理、窗口管理、流量控制、拥塞控制**等功能。另外TFO下TCP有可能向应用层提供重复的数据，也就是不可靠传输，但是只会发生在连接建立阶段，我们后续会进行介绍。

字节流式：应用层发送的数据会在TCP的发送端缓存起来，统一分片(例如一个应用层的数据包分成两个TCP包)或者打包(例如两个或者多个应用层的数据包打包成一个TCP数据包)发送，到接收端的时候接收端也是直接按照字节流将数据传递给应用层。作为对比，同样是传输层的协议，UDP并不会对应用层的数据包进行打包和分片的操作，一般一个应用层的数据包就对应一个UDP包。这个也是伴随TCP窗口管理、拥塞控制等。

3.1TCP首部格式



源端口和目的端口：各占2个字节，分别写入源端口和目的端口。

序号seq：占4字节。序号范围是【0，2^32 - 1】，共2^32（即4294967296）个序号。序号增加到2^32-1后，下一个序号就又回到0。也就是说，序号使用mod 2^32运算。TCP是面向字节流的。在一个TCP连接中传送的字节流中的每一个字节都按顺序编号。整个要传送的字节流的起始序号必须在连接建立时设置。首部中的序号字段值则是指的是本报文段所发送的数据的第一个字节的序号。例如，一报文段的序号是301，而接待的数据共有100字节。这就表明：本报文段的数据的第一个字节的序号是301，最后一个字节的序号是400。显然，下一个报文段（如果还有的话）的数据序号应当从401开始，即下一个报文段的序号字段值应为401。这个字段的序号也叫“报文段序号”。

确认号ack：占4字节，是期望收到对方下一个报文段的第一个数据字节的序号。例如，B正确收到了A发送过来的一个报文段，其序号字段值是501，而数据长度是200字节（序号501~700），这表明B正确收到了A发送的到序号700为止的数据。因此，B期望收到A的下一个数据序号是701，于是B在发送给A的确认报文段中把确认号置为701。注意，现在确认号不是501，也不是700，而是701。
总之：若确认号为= N，则表明：到序号N-1为止的所有数据都已正确收到。

数据偏移：占4位，它指出TCP报文段的数据起始处距离TCP报文段的起始处有多远。这个字段实际上是指出TCP报文段的首部长度。由于首部中还有长度不确定的选项字段，因此数据偏移字段是必要的，但应注意，“数据偏移”的单位是32位

字（即以4字节的字为计算单位）。由于4位二进制数能表示的最大十进制数字是15，因此数据偏移的最大值是60字节，这也是TCP首部的最大字节（即选项长度不能超过40字节）。

保留：占6位，保留为今后使用，但目前应置为0 。

下面有6个控制位，用来说明本报文段的性质

紧急URG（URGent）：当URG=1时，表明紧急指针字段有效。它告诉系统此报文段中有紧急数据，应尽快发送（相当于高优先级的数据），而不要按原来的排队顺序来传送。例如，已经发送了很长的一个程序要在远地的主机上运行。但后来发现了一些问题，需要取消该程序的运行，因此用户从键盘发出中断命令。如果不使用紧急数据，那么这两个字符将存储在接收TCP的缓存末尾。只有在所有的数据被处理完毕后这两个字符才被交付接收方的应用进程。这样做就浪费了很多时间。

当URG置为1时，发送应用进程就告诉发送方的TCP有紧急数据要传送。于是发送方TCP就把紧急数据插入到本报文段数据的最前面，而在紧急数据后面的数据仍然是普通数据。这时要与首部中紧急指针（Urgent Pointer）字段配合使用。

应用场景：强制关闭时可使用

确认ACK（ACKnowledgment）：仅当ACK = 1时确认号字段才有效，当ACK = 0时确认号无效。TCP规定，在连接建立后所有的传送的报文段都必须把ACK置为1。

推送 PSH（PuSH）：当两个应用进程进行交互式的通信时，有时在一端的应用进程希望在键入一个命令后立即就能收到对方的响应。在这种情况下，TCP就可以使用推送（push）操作。这时，发送方TCP把PSH置为1，并立即创建一个报文段发送出去。接收方TCP收到PSH=1的报文段，就尽快地（即“推送”向前）交付接收应用进程。而不用再等到整个缓存都填满了后再向上交付。

复位RST（ReSeT）：当RST=1时，表名TCP连接中出现了严重错误（如由于主机崩溃或其他原因），必须释放连接，然后再重新建立传输连接。RST置为1还用来拒绝一个非法的报文段或拒绝打开一个连接。

应用场景：端口未打开、请求超时、提前关闭

同步SYN（SYNchronization）：在连接建立时用来同步序号。当SYN=1而ACK=0时，表明这是一个连接请求报文段。对方若同意建立连接，则应在响应的

报文段中使SYN=1和ACK=1，因此SYN置为1就表示这是一个连接请求或连接接受报文。

终止FIN（FINis，意思是“完”“终”）：用来释放一个连接。当FIN=1时，表明此报文段的发送发的数据已发送完毕，并要求释放运输连接。

窗口：占2字节。窗口值是【0，2^16-1】之间的整数。窗口指的是发送本报文段的一方的接受窗口（而不是自己的发送窗口）。窗口值告诉对方：从本报文段首部中的确认号算起，接收方目前允许对方发送的数据量（以字节为单位）。之所以要有这个限制，是因为接收方的数据缓存空间是有限的。总之，窗口值作为接收方让发送方设置其发送窗口的依据。
例如，发送了一个报文段，其确认号是701，窗口字段是1000.这就是告诉对方：“从701算起，我（即发送方报文段的一方）的接收缓存空间还可接受1000个字节数据（字节序号是701~1700），你在给我发数据时，必须考虑到这一点。”
总之：窗口字段明确指出了现在允许对方发送的数据量。窗口值经常在动态变化。

检验和：占2字节。检验和字段检验的范围包括首部和数据这两部分。和UDP用户数据报一样，在计算检验和时，要在TCP报文段的前面加上12字节的伪首部。伪首部的格式和UDP用户数据报的伪首部一样。但应把伪首部第4个字段中的17改为6（TCP的协议号是6）；把第5字段中的UDP中的长度改为TCP长度。接收方收到此报文段后，仍要加上这个伪首部来计算检验和。若使用TPv6,则相应的伪首部也要改变。

紧急指针：占2字节。紧急指针仅在URG=1时才有意义，它指出本报文段中的紧急数据的字节数（紧急数据结束后就是普通数据）。因此，在紧急指针指出了紧急数据的末尾在报文段中的位置。当所有紧急数据都处理完时，TCP就告诉应用程序恢复到正常操作。值得注意的是，即使窗口为0时也可以发送紧急数据。

选项：长度可变，最长可达4字节。当没有使用“选项”时，TCP的首部长度是20字节。
TCP最初只规定了一种选项，即**最大报文段长度MSS**（Maximum Segment Szie）。注意MSS这个名词含义。MSS是每一个TCP报文段中的数据字段的最大长度。数据字段加上TCP首部才等于整个的TCP报文段。所以MSS并不是整个TCP报文段的最大长度，而是“TCP报文段长度减去TCP首部长度”

为什么要规定一个最大报文长度MSS呢？这并不是考虑接受方的接收缓存可能存放不下TCP报文段中的数据。实际上，MSS与接收窗口值没有关系。我们知道，TCP报文段的数据部分，至少要加上40字节的首部（TCP首部20字节和IP首部20字节，这里还没有考虑首部中的可选部分）才能组装成一个IP数据报。若选择较小的MSS长度，网络的利用率就降低。设想在极端情况下，当TCP报文段只含有1字节的数据时，在IP层传输的数据报的开销至少有40字节（包括TCP报文段的首部和IP数据报的首部）。这样，对网络的利用率就不会超过1/41。到了数据链路层还要加上一些开销。但反过来，若TCP报文段非常长，那么在IP层传输时就有可能要分解成多个短数据报片。在终点要把收到的各个短数据报片组成成原来的TCP报文段，当传输出错时还要进行重传，这些也都会使开销增大。

因此，MSS应尽可能大些，只要在IP层传输时不需要分片就行。由于IP数据报所经历的路径是动态变化的，因此在这条路径上确定的不需要的分片的MSS，如果改走另一条路径就可能需要进行分片。因此最佳的MSS是很难确定的。在连接过程中，双方都把自己能够支持的MSS写入这一字段，以后就按照这个数值传输数据，两个传送方向可以有不同的MSS值。若主机未填写这一项，则MSS的默认值是536字节长。因此，所有在互联网上的主机都应该接受的报文段长度是536+20（固定首部长度）=556字节。

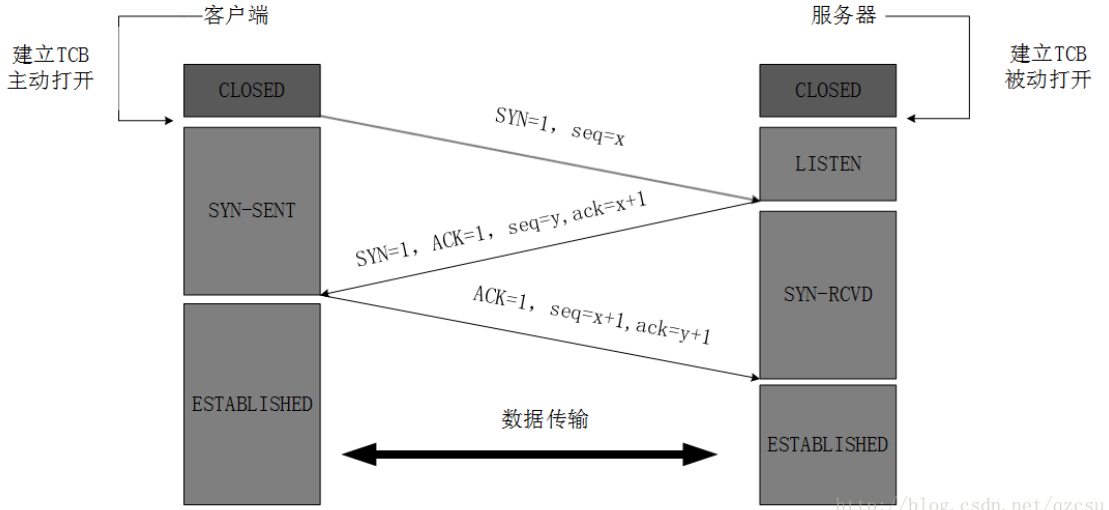
窗口扩大选项是为了扩大窗口。我们知道，TCP首部中窗口字段长度是16位，因此最大的窗口大小为64K字节。虽然这对早期的网络是足够用的，但对于包含卫星信道的网络，传播时延和宽带都很大，要获得高吞吐量需要更大的窗口大小。
窗口扩大选项占3字节，其中有一个字节表示移位值S。新的窗口值等于TCP首部中的窗口位数从16增大到（16+S）。移位值允许使用的最大值是14，相当于窗口最大值增大到2^（16+14）-1=2^30-1。
窗口扩大选项可以在双方初始建立TCP连接时进行协商。如果连接的某一端实现了窗口扩大，当它不再需要扩大其窗口时，可发送S=0选项，使窗口大小回到16。

时间戳选项占10字节，其中最主要的字段是时间戳字段（4字节）和时间戳回送回答字段（4字节）。时间戳选项有以下两个概念：

- 用来计算往返时间RTT。发送方在发送报文段时把当前时钟的时间值放入时间戳字段，接收方在确认该报文段时把时间戳字段复制到时间戳回送回答字段。因此，发送方在收到确认报文后，可以准确地计算出RTT来。
- 用于处理TCP序号超过2^32的情况，这又称为防止序号绕回PAWS。我们知道，TCP报文段的序号只有32位，而每增加2^32个序号就会重复使用原来用过的序号。当使用高速网络时，在一次TCP连接的数据传送中序号很可能被重

复使用。例如，当使用1.5Mbit/s的速度发送报文段时，序号重复要6小时以上。但若用2.5Gbit/s的速率发送报文段，则不到14秒钟序号就会重复。为了使接收方能够把新的报文段和迟到很久的报文段区分开，则可以在报文段中加上这种时间戳。

3.2TCP三次握手



假设 A 为客户端，B 为服务器端。

- 首先 B 处于 LISTEN（监听）状态，等待客户的连接请求。
- A 向 B 发送连接请求报文，SYN=1，ACK=0，选择一个初始的序号 x。
- B 收到连接请求报文，如果同意建立连接，则向 A 发送连接确认报文，SYN=1，ACK=1，确认号为 x+1，同时也选择一个初始的序号 y。
- A 收到 B 的连接确认报文后，还要向 B 发出确认，确认号为 y+1，序号为 x+1。
- B 收到 A 的确认后，连接建立。

注：

TCP规定，SYN报文段（SYN=1的报文段）不能携带数据，但需要消耗掉一个序号。

TCP规定，ACK报文段可以携带数据，但是如果不携带数据则不消耗序号。

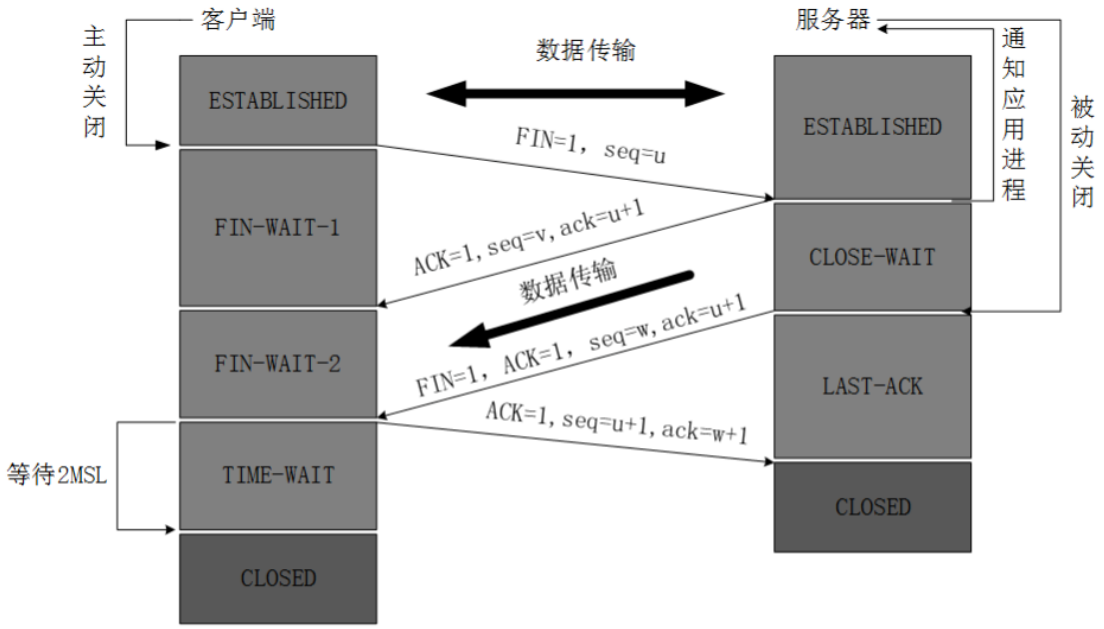
TCP规定，FIN报文段即使不携带数据，也要消耗一个序号。

3.2.1三次握手的原因（为什么TCP客户端最后还要发送一次确认呢）

第三次握手是为了防止失效的连接请求到达服务器，让服务器错误打开连接。

客户端发送的连接请求如果在网络中滞留，那么就会隔很长一段时间才能收到服务器端发回的连接确认。客户端等待一个超时重传时间之后，就会重新请求连接。但是这个滞留的连接请求最后还是会到达服务器，如果不进行三次握手，那么服务器就会打开两个连接。如果有第三次握手，客户端会忽略服务器之后发送的对滞留连接请求的连接确认，不进行第三次握手，因此就不会再次打开连接。

3.3TCP四次挥手



- A 发送连接释放报文，FIN=1，seq = u，此时A进入FIN-WAIT-1状态。
- B 收到之后发出确认，ACK=1，ack=u+1，并且带上自己的序列号seq=v，此时B就进入了CLOSE-WAIT状态。TCP服务器通知高层的应用进程，客户端向服务器的方向就释放了，这时 TCP 属于**半关闭状态**，B 能向 A 发送数据但是 A 不能向 B 发送数据。
- A收到B的确认请求之后，A进入了FIN-WAIT-2状态，并等待B发送释放连接报文。
- 当 B 不再需要连接时，发送连接释放报文，FIN=1，ack=u+1，由于在半连接状态，B可能又发送了一些数据，假定此时序列号为seq=w。此时B就进入了LAST-ACK状态，等待A的确认。
- A 收到后发出确认，ACK=1，ack=w+1，seq=u+1，此时A进入 TIME-WAIT 状态，等待 2 MSL（最大报文存活时间）后释放连接。
- B 收到 A 的确认后释放连接。可以看到，服务器结束TCP连接的时间要比客户端早一些。

3.3.1四次挥手的原因

建立连接的时候， 服务器在LISTEN状态下，收到建立连接请求的SYN报文后，把ACK和SYN放在一个报文里发送给客户端。
而关闭连接时，服务器收到对方的FIN报文时，仅仅表示对方不再发送数据了但是还能接收数据，而自己也未必全部数据都发送给对方了，所以己方可以立即关闭，也可以发送一些数据给对方后，再发送FIN报文给对方来表示同意现在关闭连接，因此，己方ACK和FIN一般都会分开发送，从而导致多了一次。

3.3.2TIME_WAIT

MSL（Maximum Segment Lifetime），**最长报文段寿命**，TCP允许不同的实现可以设置不同的MSL值。

- 第一，保证客户端发送的最后一个ACK报文能够到达服务器，因为这个ACK报文可能丢失，站在服务器的角度看来，我已经发送了FIN+ACK报文请求断开了，客户端还没有给我回应，应该是我发送的请求断开报文它没有收到，于是服务器又会重新发送一次，而客户端就能在这个2MSL时间段内收到这个重传的报文，接着给出回应报文，并且会重启2MSL计时器。
- 第二，防止类似与“三次握手”中提到了的“已经失效的连接请求报文段”出现在本连接中。客户端发送完最后一个确认报文后，在这个2MSL时间中，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。这样新的连接中不会出现旧连接的请求报文。

3.3.3如果已经建立了连接，但是客户端突然出现故障了怎么办?

TCP还设有一个保活计时器，显然，客户端如果出现故障，服务器不能一直等下去，白白浪费资源。服务器每收到一次客户端的请求后都会重新复位这个计时器，时间通常是设置为2小时，若两小时还没有收到客户端的任何数据，服务器就会发送一个探测报文段，以后每隔75秒发送一次。若一连发送10个探测报文仍然没反应，服务器就认为客户端出了故障，接着就关闭连接。

3.3.4UDP和TCP校验的时候为什么要加入伪首部?

伪首部并非TCP&UDP数据报中实际的有效成分。伪首部是一个虚拟的数据结构，其中的信息是从数据报所在IP分组头的分组头中提取的，既不向下传送也不向上递交，而仅仅是为计算校验和。这样的校验和，既校验了TCP&UDP用户数据的源端口号和目的端口号以及TCP&UDP用户数据报的数据部分，又检验了IP数据报的源IP地址和目的地址。伪报头保证TCP&UDP数据单元到达正确的目的地址。因此，伪报头中包含IP地址并且作为计算校验和需要考虑的一部分。最终目的端根据伪报

头和数据单元计算校验和以验证通信数据在传输过程中没有改变而且到达了正确的目的地址。

为什么伪首部要有目的IP地址

学习过通信系统原理后，我们知道数据传输过程中会产生误码，0可能变为1，1可能变为0，并且每种校验码都有一定的查错能力，超过这个范围，就无法察觉错误了，而早期的通信环境大概比较糟糕，因此，在传输过程中出现误码，可能使IP报文的目的地址出现错误，接收主机的UDP计算校验和时，目的IP地址来自IP层，由于目的IP地址出现错误，导致发送主机计算检验和时使用的目的IP地址与接收主机计算检验和时使用的目的IP地址不同，UDP发现错误，丢弃报文

为什么伪首部要有源IP地址

为了让接收主机确认源IP地址没有出现错误。

假设我们想要开发一款基于UDP的程序，A发送UDP报文给B，B要发送回应报文给A，假设传输过程出现误码，源IP地址出现错误，则A计算检验和时使用的源IP地址与B计算校验和时使用的源IP地址不同，B就可以发现错误，从而丢弃报文，定时重传等可靠性由应用程序自己保证

为什么伪首部要有协议字段

为了确认操作系统交付给UDP的报文是UDP报文

为什么伪首部要有UDP的长度

可能是为了防止UDP头部的长度字段出现错误，导致解析UDP报文时出现差错

3.3.5TCP同时打开和同时关闭

3.3.5.1同时打开

两个应用程序同时彼此执行主动打开的情况是可能的，尽管发生的可能性极小。每一方必须发送一个SYN，且这些SYN必须传递给对方。这需要每一方使用一个对方熟知的端口最为本地端口。

当出现同时打开的情况时，状态迁移图就与标准的连接状态迁移图不一样了。两端几乎同时发送SYN并进入SYN_SENT状态。当每一端收到SYN时，状态变为SYN_RCVD，同时它们都再发SYN并对收到的SYN进行确认。当双方都接收到SYN及相应的ACK时，状态都变为了ESTABLISHED。

一个同时打开的连接需要交换需要交换4个报文段，比正常的三次握手多一个。没有任何一端称为客户或服务器，因为每一端既是客户又是服务器。

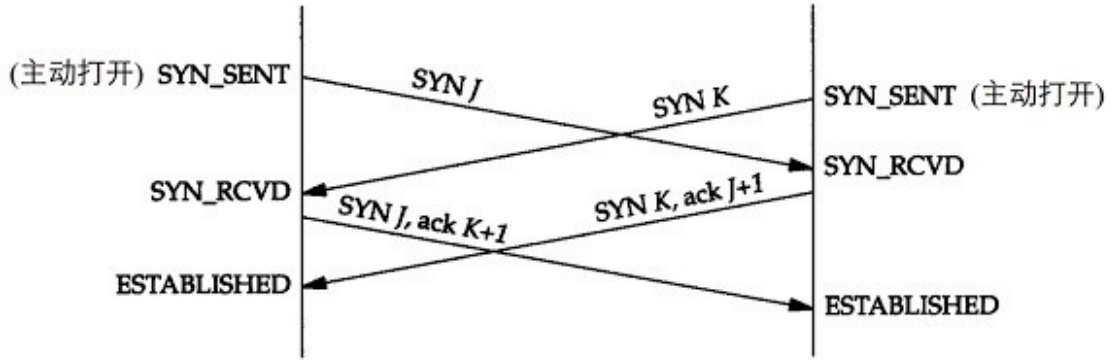


图18-17 同时打开期间报文段的交换

3.3.5.2同时关闭

在标准的情况下通过一方发送FIN来关闭连接，但是双方都执行主动关闭也是有可能的，TCP协议也允许这样的同时关闭。当应用层发出关闭命令时，两方均从ESTABLISHED变为FIN_WAITE_1。这将导致双发各发送一个FIN，两个FIN经过网络传输后分别达到另一端。收到FIN后，状态由FIN_WAIT_1变签到CLOSING，并将发送最后的ACK。当收到最后的ACK时，状态变化为TIME_WAIT。

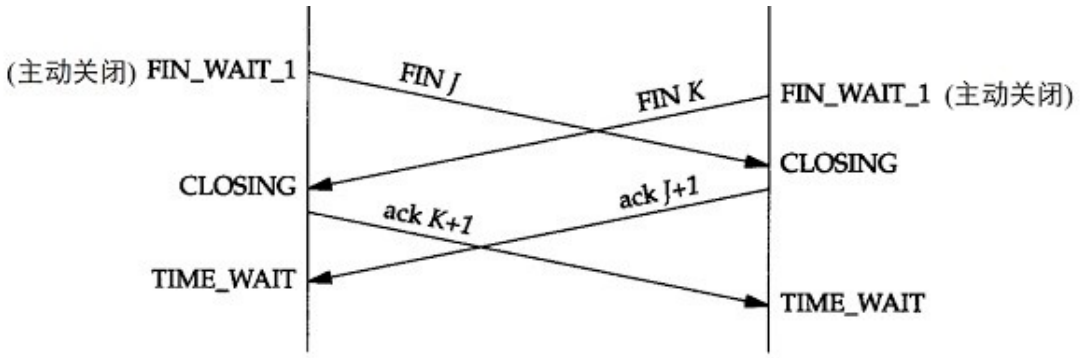


图18-19 同时关闭期间的报文段交换

3.4TCP可靠性传输

1. 确认应答机制：每个ACK都有对应的确认序列号，意思是告诉发送者已经收到了数据，下一个数据应该从哪里开始发送。
2. 超时重传机制：两种情况
   1）如果主机A发送给主机B的报文，主机B在规定的时间内没有及时收到主机A发送的报文，我们可以认为是ACK丢了，这时就需要触发超时重传机制。
   2）如果主机A未收到B发来的确认应答，也可能是因为ACK丢了。因此主机B会收到很多重复的数据，那么，TCP协议需要能够识别出那些包是重复的包，

并且把重复的包丢弃，这时候我们可以用前面提到的序列号，很容易做到去重的效果
3. 流量控制
4. 拥塞控制
5. 校验和
6. 连接管理机制（三次握手，四次挥手）

3.5TCP流量控制：滑动窗口原理

滑动窗口协议**允许发送方在停止并等待确认前发送多个数据分组。**由于发送方不必每发一个分组（IP数据包或TCP报文段）就停下来等待确认，因此该协议可以加速数据的传输，增大了吞吐量。

停止等待协议是最简单但也是最基础的数据链路层协议。与滑动窗口协议不同的是停止等待协议就是每发送完一个分组就停止发送，等待对方的确认，在收到确认后再发送下一个分组。当发送窗口和接收窗口的大小都等于**1**时，就是停止等待协议。

TCP的可靠数据传输服务确保进程从接收缓冲区中读出的数据流是正确地，有序的，不重复的，即读出的字节流与连接的另一方系统发送的字节流是完全一样的，但是实际上接收方应用进程不一定时刻都在读数据，那么如果应用进程读取数据相当慢，而发送方发送的数据太多、太快，就很容易使接收方的接收缓冲区溢出。所以**TCP采用大小可变的滑动窗口给应用程序提供流量控制服务**，用以消除接收缓冲区溢出的可能。实际上TCP报文段头部的16位窗口字段写入的数值就是接收方当前给对方设置的发送窗口的上限，发送窗口在连接建立时由双方商定，但是在通信过程中，接收方根据自己的接收缓冲区资源大小随时动态的调整对方发送窗口的上限（滑动窗口即可以滑动滴）。

流量控制（用滑动窗口实现）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。

TCP连接发送方发送一个数据报，对端会进行确认在发送下一个数据包。

举个例子：TCP通信是全双工的，这里为了方便理解，就以一个方向为例，假设A为发送方，B为接收方。A会有一个发送窗口，B有一个接收窗口。在连接建立的时候B告诉A其接收窗口是400字节，因此发送方的发送窗口不能超过接收方给的接收窗口大小。（假设一个报文段100字节，报文段序号初始值为1.图中大写ACK

表示头部中的确认位ACK，小写ack表示确认字段的值）

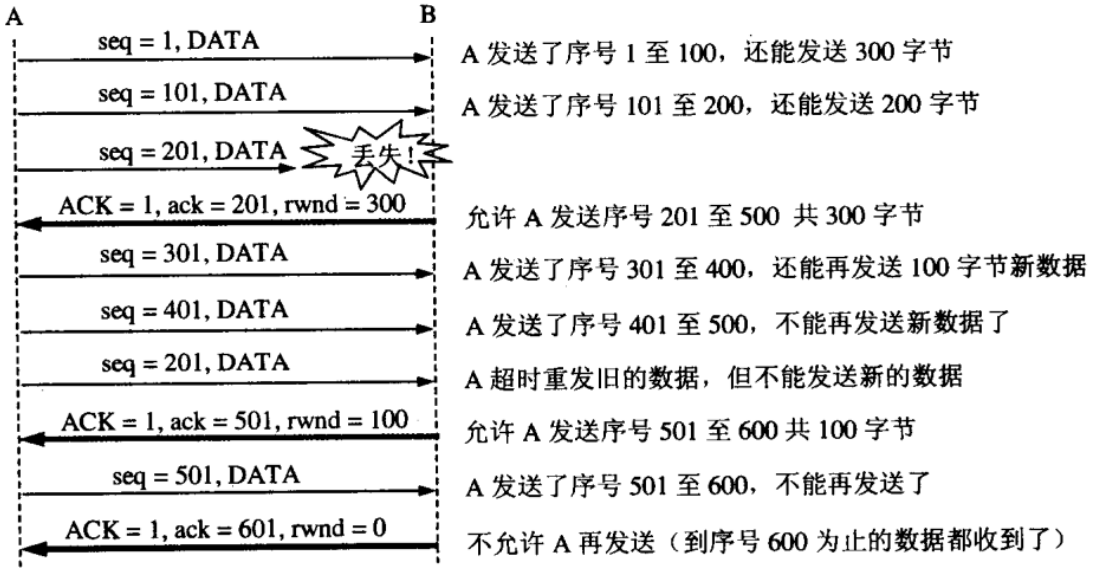


图 5-22　利用可变窗口进行流量控制举例

为了便于大家看，我们采用另一种方式描述发送窗口，假设A现在收到B的确认报文段，窗口是20字节，确认号是31（表示B期望收到的下一个序号是31，而序号30为止的数据已经收到了），根据此条件，A构造出自己的发送窗口，如下图：

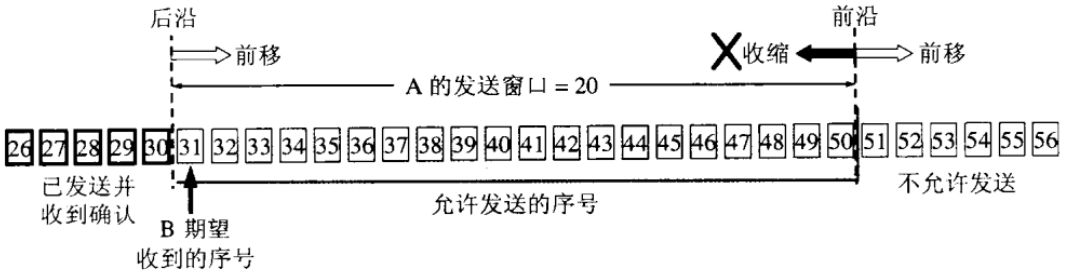


图 5-15　根据 B 给出的窗口值，A 构造出自己的发送窗口

发送窗口表示在没有收到B确认的情况下，A也可以连续把发送窗口的数据发送出去。但是已经发送过的数据在未收到确认之前，它还需要暂时保留，以便于超时重传时使用。发送窗口越大，它就可以在收到对方确认之前发送更多的数据，因而获得更高的传输效率。（但是接收方要来得及处理之前的数据）

发送窗口的位置由窗口前沿和后沿的位置共同确定。
它后沿变化有两种，（1）不动（没有收到新的确定）（2）前移（收到新的确认）；

前沿是不断向前移动的，但也可能不动
（1）不动（没有收到新确认，对方窗口也不变）

（2）不动（收到新的确认，对方的应用层没有读取数据，接收窗口缩小了，使前沿正好不变，后沿前移窗口就变小了）

假定A发送了序号为31～41的数据。这时，发送窗口的位置并未改变，发送窗口内靠后的11个字节（黑色部分）表示 已发送但未收到确认。

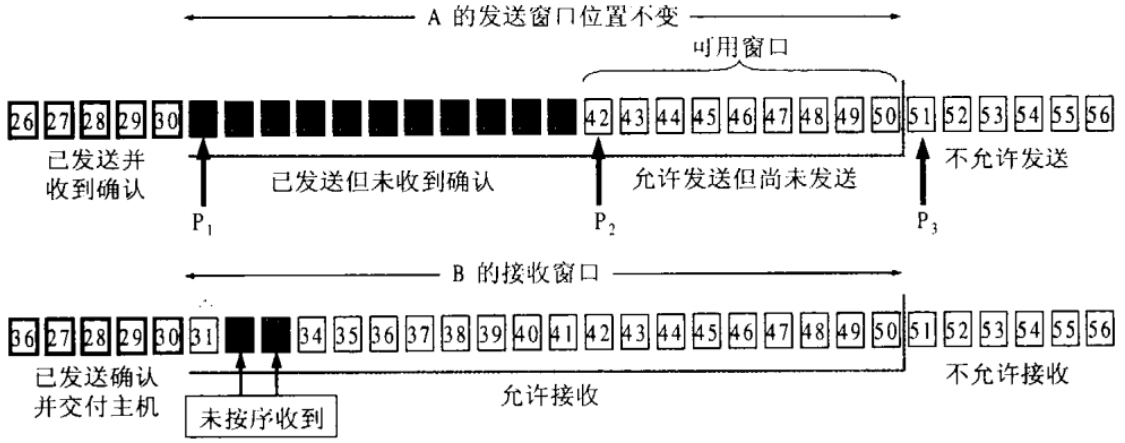


图 5-16　A 发送了 11 个字节的数据

即小于P1的是已发送并收到确认的序号，而大于P3的是不允许发送的部分。

P3-P1=A的发送窗口

P2-P1=已发送但是未接收到确认的字节数

P3-P2=允许发送但是未发送的字节数（可用窗口或有效窗口）

再看B的接收窗口大小是20，在接收窗口外面到30号位置的数据是接收并确认的，因此可以丢弃。在接收窗口中，B收到了32和33的数据（白色的是没有接收到的），但它们不是按序到达的，因为并没有收到31号数据。B只能对按序达收到的数据中的最高序号给出确认，因此B发送的确认报文字段的确认号依然是31号（确认号就是告诉发送方期望下一次收到数据包的序号）。

现在假定B收到了序号为31的数据，并把31～33的数据交付主机，然后B删除这些数据。接着把窗口向前移动3个序号，如下图，同时给a发送确认，其中的窗口值仍为20，但确认号变为34。表明B已经收到序号33为止的数据。

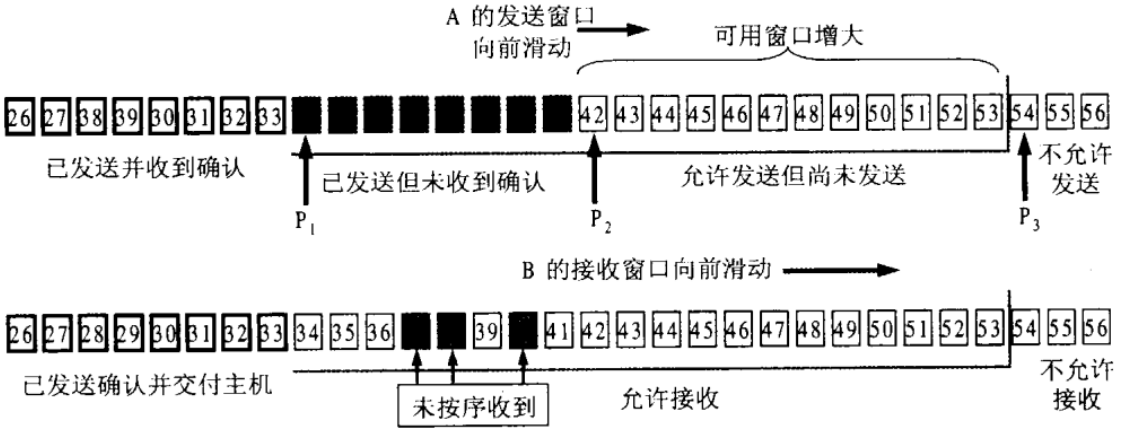


图 5-17　A 收到新的确认号，发送窗口向前滑动

当A发送完窗口内的所有数据后，未收到任何确认时，窗口内就没有可发送的数据。经过一段时间后（超时计时器），若仍未收到确认，A将会重新发送。

此外发送方和接收方都有自己的缓冲区

发送缓冲区存放：

1、发送发TCP准备发送的数据

2、TCP已发送但尚未收到确认的数据（为超时重传准备）

接收缓冲区存放：

1、按序到达但未被应用程序读取的数据

2、未按序到达的数据

实际上发送窗口大小不仅仅是由接收端决定的，其还受网络的拥塞程度决定。发送窗口=MIN{接收窗口，拥塞窗口}

3.6TCP拥塞控制

如果网络出现拥塞，分组将会丢失，此时发送方会继续重传，从而导致网络拥塞程度更高。因此当出现拥塞时，应当控制发送方的速率。这一点和流量控制很像，但是出发点不同。流量控制是为了让接收方能来得及接收（端到端的问题），而拥塞控制是为了降低整个网络的拥塞程度。

TCP 主要通过四个算法来进行拥塞控制：慢开始、拥塞避免、快重传、快恢复。

拥塞的标志

1. 重传计时器超时
2. 接收到三个重复确认

3.6.1慢开始

算法原理：当主机开始发送数据时，如果立即将大量数据字节注入到网络，那么就有可能因为不清楚当前网络的负荷情况而引起网络阻塞。所以，最好的方法是先探测一下，即由小到大逐渐增大发送窗口，也就是说，由小到大逐渐增大拥塞窗口数值。通常在刚刚发送报文段时，先把**拥塞窗口cwnd**设置为一个最大报文段MSS的数值（虽然 TCP 的窗口基于字节，但是这里设窗口的大小单位为报文段）。而在每收到一个新的报文段的确认后，把拥塞窗口增加至多一个MSS的数值。用这样的方法逐步增大发送方的拥塞窗口cwnd，可以使分组注入到网络的速率更加合理。（慢开始当中的“慢”并不是指cwnd的增长速率慢，而是在TCP开始发送报文段时先设置cwnd = 1,使得发送方在开始时只发送一一个报文段）
当rwnd足够大的时候，为了防止拥塞窗口cwnd的增长引起网络阻塞，还需要另外一个变量------**慢开始门限ssthresh**.

- 当cwnd < ssthresh时，使用上述慢启动算法；
- 当cwnd >ssthresh时，停止使用慢启动算法，改用拥塞避免算法；

慢开始的局限性

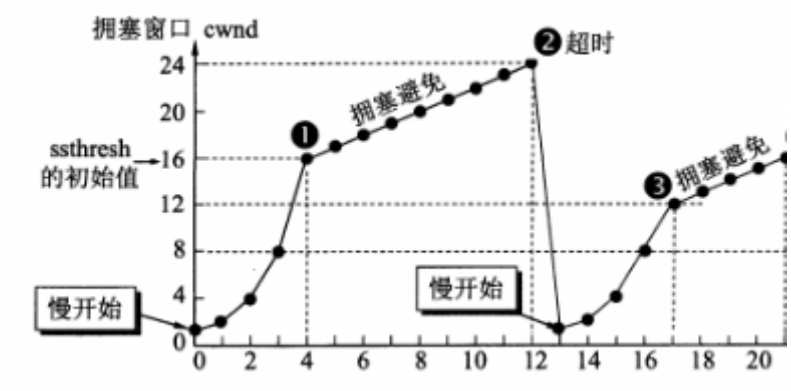
- 需要获得网络内部流量分布的信息，浪费可用的网络容量，额外开销；
- 估算合理的ssthresh值并不容易，可能耗时较长；

3.6.2拥塞避免（按线性规律增长）

拥塞避免并非完全能够避免拥塞，是说在拥塞避免阶段将拥塞窗口控制为按线性规律增长，使网络比较不容易出现拥塞。

思路：让拥塞窗口cwnd缓慢地增大，即每经过一个往返时间RTT就把发送方的拥塞控制窗口加一。

无论是在慢开始阶段还是在拥塞避免阶段，只要发送方判断网络出现拥塞（其根据就是没有收到确认，虽然没有收到确认可能是其他原因的分组丢失，但是因为无法判定，所以都当做拥塞来处理），就把慢开始门限设置为出现拥塞时的发送窗口大小的一半（ssthresh=cwnd/2）。然后把拥塞窗口设置为1，执行慢开始算法。



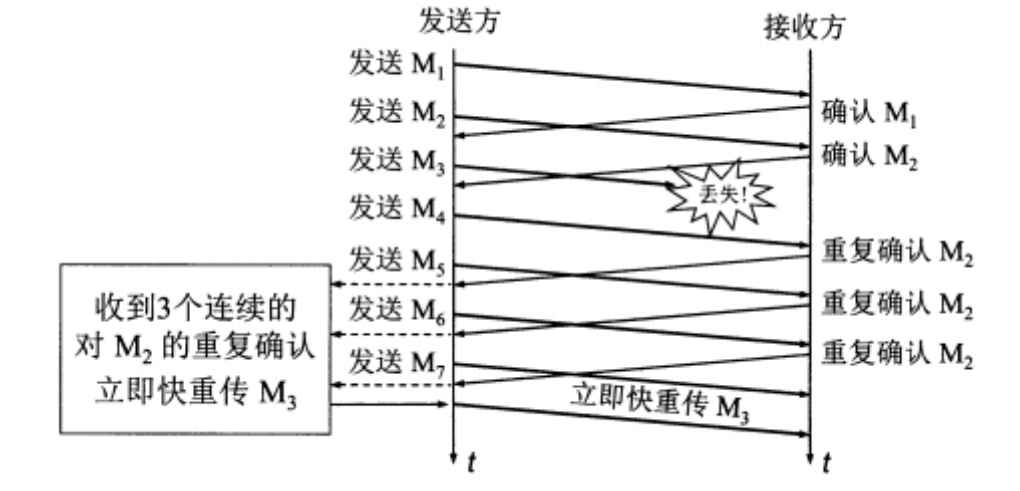
**加法增大与乘法减小 **

乘法减小：无论是慢开始阶段还是拥塞避免，只要出现了网络拥塞（超时），就把慢开始门限值ssthresh减半

加法增大：执行拥塞避免算法后，拥塞窗口线性缓慢增大，防止网络过早出现拥塞

3.6.3快重传

快重传要求接收方在收到一个失序的报文段后就立即发出重复确认（为的是使发送方及早知道有报文段没有到达对方）而不要等到自己发送数据时捎带确认。快重传算法规定，发送方只要一连收到三个重复确认就应当立即重传对方尚未收到的报文段，而不必继续等待设置的重传计时器时间到期。



由于不需要等待设置的重传计时器到期，能尽早重传未被确认的报文段，能提高整个网络的吞吐量。

3.6.4快恢复（与快重传配合）

1. 采用快恢复算法时，慢开始只在TCP连接建立时和网络出现超时时才使用。

2. 当发送方连续收到三个重复确认时，就执行“乘法减小”算法，把ssthresh门限减半。但是接下去并不执行慢开始算法。
3. 考虑到如果网络出现拥塞的话就不会收到好几个重复的确认，所以发送方现在认为网络可能没有出现拥塞。所以此时不执行慢开始算法，而是将cwnd设置为ssthresh的大小，然后执行拥塞避免算法。

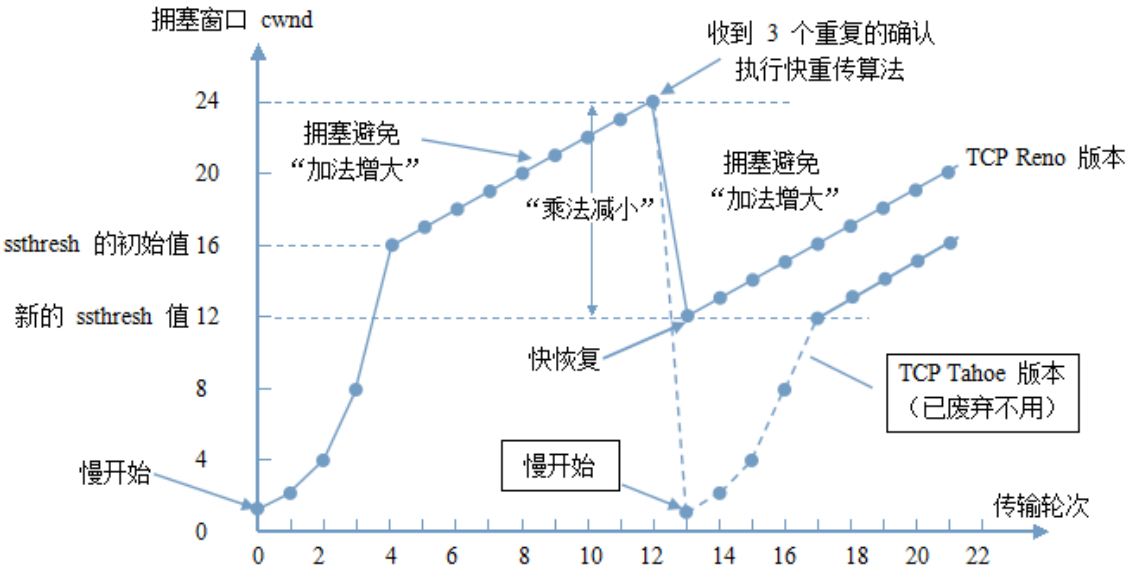


图 从连续收到三个重复的确认转入拥塞避免

3.7TCP四种计时器

3.7.1重传计时器

在滑动窗口协议中，接受窗口会在连续收到的包序列中的最后一个包向接收端发送一个ACK，当网络拥堵的时候，发送端的数据包和接收端的ACK包都有可能丢失。TCP为了保证数据可靠传输，就规定在重传的“时间片”到了以后，如果还没有收到对方的ACK，就重发此包，以避免陷入无限等待中。

当TCP发送报文段时，就创建该特定报文的重传计时器。可能发生两种情况：

1. 若在计时器截止时间到之前收到了对此特定报文段的确认，则撤销此计时器。
2. 若在收到了对此特定报文段的确认之前计时器截止时间到，则重传此报文段，并将计时器复位。

3.7.2持久计时器

先来考虑一下情景：发送端向接收端发送数据包知道接受窗口填满了，然后接受窗口告诉发送方接受窗口填满了停止发送数据。此时的状态称为“零窗口”状态，发送端和接收端窗口大小均为0.直到接受TCP发送确认并宣布一个非零的窗口大小。但这个确认会丢失。我们知道TCP中，对确认是不需要发送确认的。若确认

丢失了，接受TCP并不知道，而是会认为他已经完成了任务，并等待着发送TCP接着会发送更多的报文段。但发送TCP由于没有收到确认，就等待对方发送确认来通知窗口大小。双方的TCP都在永远的等待着对方。

要打开这种死锁，TCP为每一个链接使用一个持久计时器。当发送TCP收到窗口大小为0的确认时，就坚持启动计时器。当坚持计时器期限到时，发送TCP就发送一个特殊的报文段，叫做**探测报文**。这个报文段只有一个字节的数据。他有一个序号，但他的序号永远不需要确认；甚至在计算机对其他部分的数据的确认时该序号也被忽略。探测报文段提醒接受TCP：确认已丢失，必须重传。

坚持计时器的值设置为重传时间的数值。但是，若没有收到从接收端来的响应，则需发送另一个探测报文段，并将坚持计时器的值加倍和复位。发送端继续发送探测报文段，将坚持计时器设定的值加倍和复位，直到这个值增大到门限值（通常是60秒）为止。在这以后，发送端每个60秒就发送一个探测报文，直到窗口重新打开。

3.7.3保活计时器

保活计时器使用在某些实现中，用来防止在两个TCP之间的连接出现长时间的空闲。假定客户打开了到服务器的连接，传送了一些数据，然后就保持静默了。也许这个客户出故障了。在这种情况下，这个连接将永远的处理打开状态。

要解决这种问题，在大多数的实现中都是使服务器设置保活计时器。每当服务器收到客户的信息，就将计时器复位。通常设置为两小时。若服务器过了两小时还没有收到客户的信息，他就发送探测报文段。若发送了10个探测报文段（每一个间隔75秒）还没有响应，就假定客户除了故障，因而就终止了该连接。

这种连接的断开当然不会使用四次握手，而是直接硬性的中断和客户端的TCP连接。

3.7.4时间等待计时器

时间等待计时器是在四次握手的时候使用的。四次握手的简单过程是这样的：假设客户端准备中断连接，首先向服务器端发送一个FIN的请求关闭包（FIN=final），然后由established过渡到FIN-WAIT1状态。服务器收到FIN包以后会发送一个ACK，然后自己有established进入CLOSE-WAIT.此时通信进入半双工状态，即留给服务器一个机会将剩余数据传递给客户端，传递完后服务器发送

一个FIN+ACK的包，表示我已经发送完数据可以断开连接了，就这便进入LAST_ACK阶段。客户端收到以后，发送一个ACK表示收到并同意请求，接着由FIN-WAIT2进入TIME-WAIT阶段。服务器收到ACK，结束连接。此时（即客户端发送完ACK包之后），客户端还要等待2MSL（MSL=maxinum segment lifetime最长报文生存时间，2MSL就是两倍的MSL）才能真正的关闭连接。

3.8time_wait如何避免

首先服务器可以设置SO_REUSEADDR套接字选项来通知内核，如果端口忙，但TCP连接位于TIME_WAIT状态时可以重用端口。在一个非常有用的场景就是，如果你的服务器程序停止后想立即重启，而新的套接字依旧希望使用同一端口，此时SO_REUSEADDR选项就可以避免TIME_WAIT状态。

3.9三次握手的隐患

SYN- Flood攻击是当前网络上最为常见的DDoS攻击，也是最为经典的拒绝服务攻击，它就是利用了TCP协议实现上的一个缺陷，通过向网络服务所在端口发送大量 的伪造源地址的攻击报文，就可能造成目标服务器中的半开连接队列被占满，从而阻止其他合法用户进行访问。

原理：攻击者首先伪造地址对服务器发起SYN请求，服务器回应(SYN+ACK)包，而真实的IP会认为，我没有发送请求，不作回应。服务 器没有收到回应，这样的话，服务器不知 道(SYN+ACK)是否发送成功，默认情况下会重试5次（tcp_syn_retries）。这样的话，对于服务器的内存，带宽都有很大的消耗。攻击者如果处于公网，可以伪造IP的话，对于服务器就很难根据IP来判断攻击者，给防护带来很大的困难。

SYN Flood 防护措施

1. 无效连接监视释放
   这种方法不停的监视系统中半开连接和不活动连接，当达到一定阈值时拆除这些连接，释放系统资源。这种绝对公平的方法往往也会将正常的连接的请求也会被释放掉，” 伤敌一千，自损八百 “

2. 延缓TCB分配方法
   SYN Flood关键是利用了，SYN数据报文一到，系统立即分配TCB资源，从而占用了系统资源，因此有俩种技术来解决这一问题

Syn Cache技术

这种技术在收到SYN时不急着去分配TCB，而是先回应一个ACK报文，并在一个专用的HASH表中（Cache）中保存这种半开连接，直到收到正确的ACK报文再去分配TCB

Syn Cookie技术

Syn Cookie技术则完全不使用任何存储资源，它使用一种特殊的算法生成Sequence Number，这种算法考虑到了对方的IP、端口、己方IP、端口的固定信息，以及对方无法知道而己方比较固定的一些信息，如MSS、时间等，在收到对方 的ACK报文后，重新计算一遍，看其是否与对方回应报文中的（Sequence Number-1）相同，从而决定是否分配TCB资源。

3.10既然有了拥塞控制,为什么还会有拥塞比如游戏卡顿

路由器只是把溢出的流量丢弃处理，其他的什么都不做。
源主机发现丢包了，就会将流量降速差不多一半，这就是流量拥塞控制。
但是，是先发生“路由器流量溢出（丢包）”，然后才触发了“源主机的流量拥塞控制”。
源主机的流量拥塞控制（Congestion Control）机制，永远都无法避免路由节点的网络拥塞，从而造成的流量溢出（丢包）！

3.11close_wait作用，如果close_wait不关闭有什么问题?

在被动关闭连接情况下，在已经接收到FIN，但是还没有发送自己的FIN的时刻，连接处于CLOSE_WAIT状态。通常来讲，CLOSE_WAIT状态的持续时间应该很短，正如SYN_RCVD状态。

如果一直保持在CLOSE_WAIT状态，那么只有一种情况就是在对方关闭连接之后服务器程序自己没有进一步发出ack信号。换句话说，就是在对方连接关闭之后，程序里没有检测到，或者程序压根就忘记了这个时候需要关闭连接，于是这个资源就一直被程序占着。

出现大量close_wait的现象，主要原因是某种情况下对方关闭了socket链接，但是我方忙与读或者写，没有关闭连接。代码需要判断socket，一旦读到0，断开连接，read返回负，检查一下errno，如果不是AGAIN，就断开连接。

一直存在导致端口被占用

3.12TCP粘包、拆包

粘包问题主要还是因为接收方不知道消息之间的界限，不知道一次性提取多少字节的数据所造成的。

发送方引起的粘包是由TCP协议本身造成的，TCP为提高传输效率，发送方往往要收集到足够多的数据后才发送一个TCP段。若连续几次需要send的数据都很少，通常TCP会根据negal优化算法把这些数据合成一个TCP段后一次发送出去，这样接收方就收到了粘包数据。

两种粘包情况：

1. 发送端需要等缓冲区满才发送出去，造成粘包（发送数据时间间隔很短，数据了很小，会合到一起，产生粘包）
2. 接收方不及时接收缓冲区的包，造成多个包接收（客户端发送了一段数据，服务端只收了一小部分，服务端下次再收的时候还是从缓冲区拿上次遗留的数据，产生粘包）

两种拆包情况：

1. 要发送的数据大于TCP发送缓冲区剩余空间大小，将会发生拆包。
2. 待发送数据大于MSS（最大报文长度），TCP在传输前将进行拆包。

粘包、拆包解决办法

通过以上分析，我们清楚了粘包或拆包发生的原因，那么如何解决这个问题呢？解决问题的关键在于如何给每个数据包添加边界信息，常用的方法有如下几个：

1、发送端给每个数据包添加包首部，首部中应该至少包含数据包的长度，这样接收端在接收到数据后，通过读取包首部的长度字段，便知道每一个数据包的实际长度了。

2、发送端将每个数据包封装为固定长度（不够的可以通过补0填充），这样接收端每次从接收缓冲区中读取固定长度的数据就自然而然的把每个数据包拆分开来。

3、可以在数据包之间设置边界，如添加特殊符号，这样，接收端通过这个边界就可以将不同的数据包拆分开。

3.13UDP会不会粘包

TCP为了保证可靠传输并减少额外的开销（每次发包都要验证），采用了基于流的传输，基于流的传输不认为消息是一条一条的，是无保护消息边界的（保护消息边界：指传输协议把数据当做一条独立的消息在网上传输，接收端一次只能接受一条独立的消息）。

UDP则是面向消息传输的，是有保护消息边界的，接收方一次只接受一条独立的信息，所以不存在粘包问题。

举个例子：有三个数据包，大小分别为2k、4k、6k，如果采用UDP发送的话，不管接受方的接收缓存有多大，我们必须要进行至少三次以上的发送才能把数据包发送完，但是使用TCP协议发送的话，我们只需要接受方的接收缓存有12k的大小，就可以一次把这3个数据包全部发送完毕。

五、应用层

1、域名系统

DNS 是一个分布式数据库，提供了主机名和 IP 地址之间相互转换的服务。这里的分布式数据库是指，每个站点只保留它自己的那部分数据。

域名具有层次结构，从上到下依次为：根域名、顶级域名、二级域名。

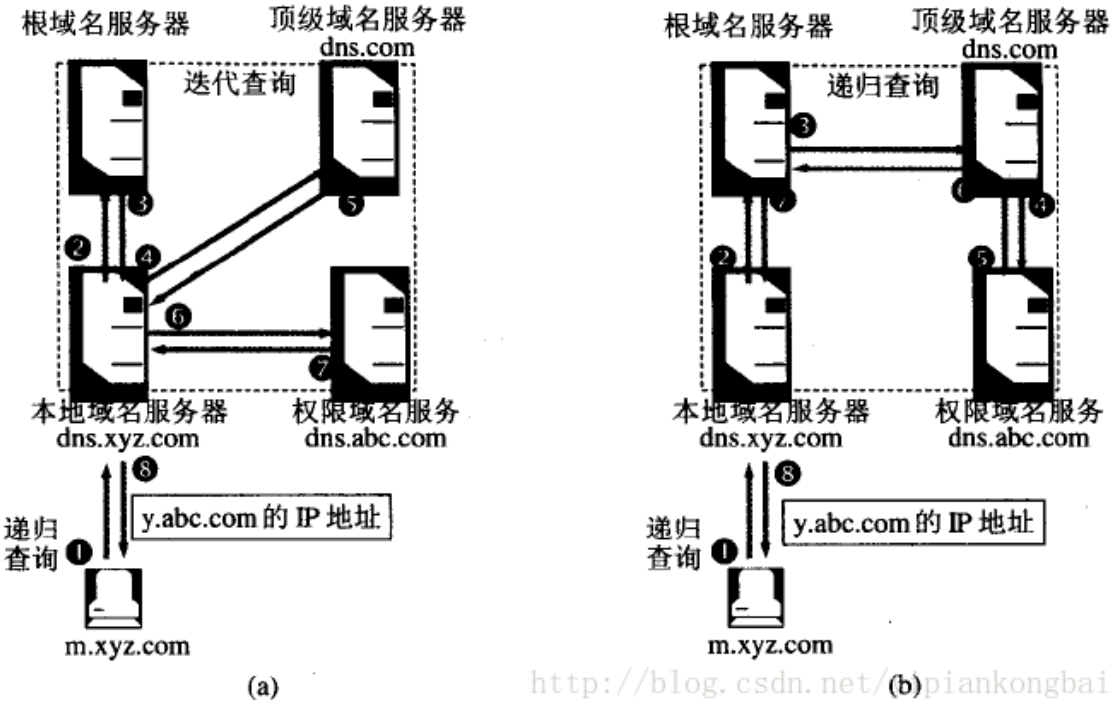
DNS 可以使用 UDP 或者 TCP 进行传输，使用的端口号都为 53。大多数情况下 DNS 使用 UDP 进行传输，这就要求域名解析器和域名服务器都必须自己处理超时和重传从而保证可靠性。在两种情况下会使用 TCP 进行传输：

- 如果返回的响应超过的 512 字节（UDP 最大只支持 512 字节的数据）。
- 区域传送（区域传送是主域名服务器向辅助域名服务器传送变化的那部分数据）。

1.1递归查询和迭代查询

递归查询：本机向本地域名服务器发出一次查询请求，就静待最终的结果。如果本地域名服务器无法解析，自己会以DNS客户机的身份向其它域名服务器查询，直到得到最终的IP地址告诉本机
迭代查询：本地域名服务器向根域名服务器查询，根域名服务器告诉它下一步到

哪里去查询，然后它再去查，每次它都是以客户机的身份去各个服务器查询。



1.2域名解析过程

域名解析总体可分为一下过程：

(1) 输入域名后,先查找自己主机对应的域名服务器，域名服务器先查找自己的数据库中的数据.

(2) 如果没有，就向上级域名服务器进行查找，依次类推

(3) 最多回溯到根域名服务器, 肯定能找到这个域名的IP地址

(4) 域名服务器自身也会进行一些缓存，把曾经访问过的域名和对应的IP地址缓存起来, 可以加速查找过程

具体可描述如下：

1. 主机先向本地域名服务器进行递归查询
2. 本地域名服务器采用迭代查询，向一个根域名服务器进行查询
3. 根域名服务器告诉本地域名服务器，下一次应该查询的顶级域名服务器的IP地址
4. 本地域名服务器向顶级域名服务器进行查询
5. 顶级域名服务器告诉本地域名服务器，下一步查询权限服务器的IP地址
6. 本地域名服务器向权限服务器进行查询
7. 权限服务器告诉本地域名服务器所查询的主机的IP地址
8. 本地域名服务器最后把查询结果告诉主机

1.3DNS缓存

为了增加访问效率，计算机有域名缓存机制，当访问过某个网站并得到其IP后，会将其域名和IP缓存下来，下一次访问的时候，就不需要再请求域名服务器获取IP，直接使用缓存中的IP，提高了响应的速度。当然缓存是有有效时间的，当过了有效时间后，再次请求网站，还是需要先请求域名解析。

但是域名缓存机制也可能会带来麻烦。例如IP已变化了，仍然使用缓存中的IP来访问，将会访问失败。再如 同一个域名在内网和外网访问时所对应的IP是不同的，如在外网访问时通过外网IP映射到内网的IP。同一台电脑在外网环境下访问了此域名，再换到内网来访问此域名，在DNS缓存的作用下，也会去访问外网的IP，导致访问失败。根据情况，可以手动清除DNS缓存或者禁止DNS缓存机制。

2、文件传送协议（FTP）

FTP 使用 TCP 进行连接，它需要两个连接来传送一个文件：

- 控制连接：服务器打开端口号 21 等待客户端的连接，客户端主动建立连接后，使用这个连接将客户端的命令传送给服务器，并传回服务器的应答。
- 数据连接：用来传送一个文件数据。

为什么要用这两个连接?
FTP协议约定，一个连接用于传输命令，一个连接用于传输数据。

根据数据连接是否是服务器端主动建立，FTP 有主动和被动两种模式：
主动模式(PORT)
服务器端主动建立数据连接，其中服务器端的端口号为 20，客户端的端口号随机，但是必须大于 1024，因为 0~1023 是熟知端口号。

被动模式(PASV)
客户端主动建立数据连接，其中客户端的端口号由客户端自己指定，服务器端的端口号随机。

主动模式要求客户端开放端口号给服务器端，需要去配置客户端的防火墙。被动模式只需要服务器端开放端口号即可，无需客户端配置防火墙。但是被动模式会导致服务器端的安全性减弱，因为开放了过多的端口号。

2.1FTP响应码

客户端发送 FTP 命令后，服务器返回响应码。
响应码用三位数字编码表示：
第一个数字给出了命令状态的一般性指示，比如响应成功、失败或不完整。
第二个数字是响应类型的分类，如 2 代表跟连接有关的响应，3 代表用户认证。
第三个数字提供了更加详细的信息。

第一个数字的含义如下：
1 表示服务器正确接收信息，还未处理。
2 表示服务器已经正确处理信息。
3 表示服务器正确接收信息，正在处理。
4 表示信息暂时错误。
5 表示信息永久错误。

第二个数字的含义如下：
0 表示语法。
1 表示系统状态和信息。
2 表示连接状态。
3 表示与用户认证有关的信息。
4 表示未定义。
5 表示与文件系统有关的信息。

2.2FTP断点续传

断点续传功能其实就是在发送的过程中，记录下发送的进度，当出现包括网络中断等发送出错的情况下，断开连接。等下次网络好的情况下，继续发送剩余文件的过程。

对于ftp的断点续传上传的功能实现，FTP协议中提供了一条APPE的控制命令用来追加文件，我们所实现的断点续传的命令就是围绕着这个命令进行的。
1、 在正常上传的过程中，记录下已经发送的文件的长度
2、 当网络发送异常时，记录当前发送文件长度，并关闭当前ftp连接，结束ftp的发送过程
3、 当网络正常后，重新开始建立ftp的连接，此时由上传文件改为APPE命令，并在数据连接上发送剩余的数据到ftp服务器。

2.3匿名FTP

匿名FTP是这样一种机制，用户可通过它连接到远程FTP服务器上，进行文件的上传或下载，而不需要成为其注册用户。系统管理员建立一个特殊的用户帐号(匿名帐号)，一般名为anonymous或助，互联网上的任何人在任何地方都可使用该用户的帐号。
通过FTP程序连接匿名FTP服务器的方式同连接普通FTP服务器的方式类似，只是在要求提供用户帐号时必须输人anonymous或ftp，而该用户帐号的口令可以是任意的字符。
当远程服务器提供匿名FTP服务时，会预先指定某些目录及文件向公众开放，允许匿名用户的存取，而系统中的其余目录则处于隐匿状态。作为一种安全措施，大多数匿名FTP服务器都只允许用户下载文件，而不允许用户上传文件。

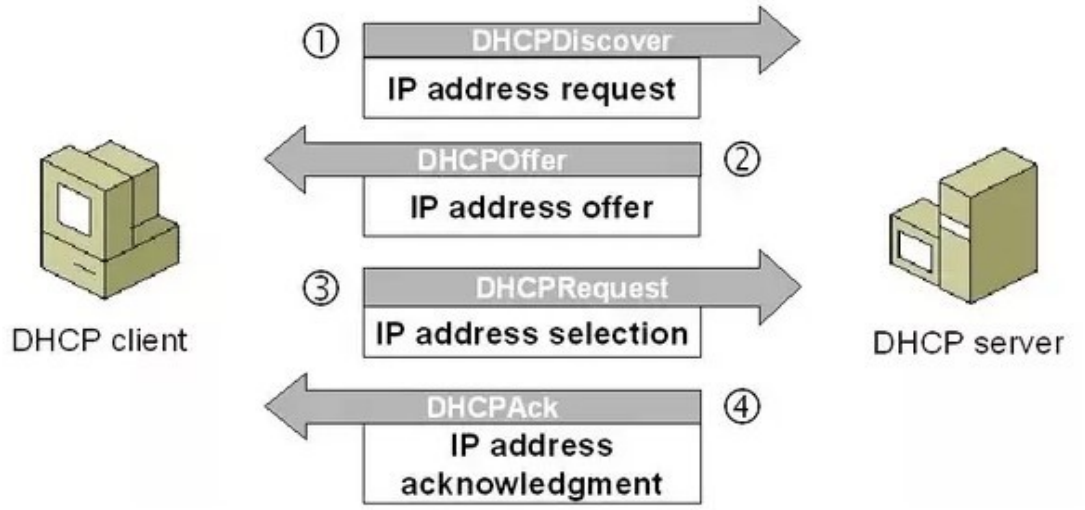
3、动态主机配置协议

DHCP (Dynamic Host Configuration Protocol) 提供了即插即用的连网方式，用户不再需要手动配置 IP 地址等信息。

DHCP 配置的内容不仅是 IP 地址，还包括子网掩码、网关 IP 地址。

DHCP 工作过程如下：

1. 客户端发送 Discover 报文，该报文的目的地址为 255.255.255.255:67，源地址为 0.0.0.0:68，被放入 UDP 中，该报文被广播到同一个子网的所有主机上。如果客户端和 DHCP 服务器不在同一个子网，就需要使用中继代理。
2. DHCP 服务器收到 Discover 报文之后，发送 Offer 报文给客户端，该报文包含了客户端所需要的信息。因为客户端可能收到多个 DHCP 服务器提供的信息，因此客户端需要进行选择。
3. 如果客户端选择了某个 DHCP 服务器提供的信息，那么就发送 Request 报文给该 DHCP 服务器。
4. DHCP 服务器发送 Ack 报文，表示客户端此时可以使用提供给它的信息。



4、远程登录协议

TELNET 用于登录到远程主机上，并且远程主机上的输出也会返回。

TELNET 可以适应许多计算机和操作系统的差异，例如不同操作系统系统的换行符定义。

5、电子邮件协议

一个电子邮件系统由三部分组成：用户代理、邮件服务器以及邮件协议。

邮件协议包含发送协议和读取协议，发送协议常用 SMTP，读取协议常用 POP3 和 IMAP。

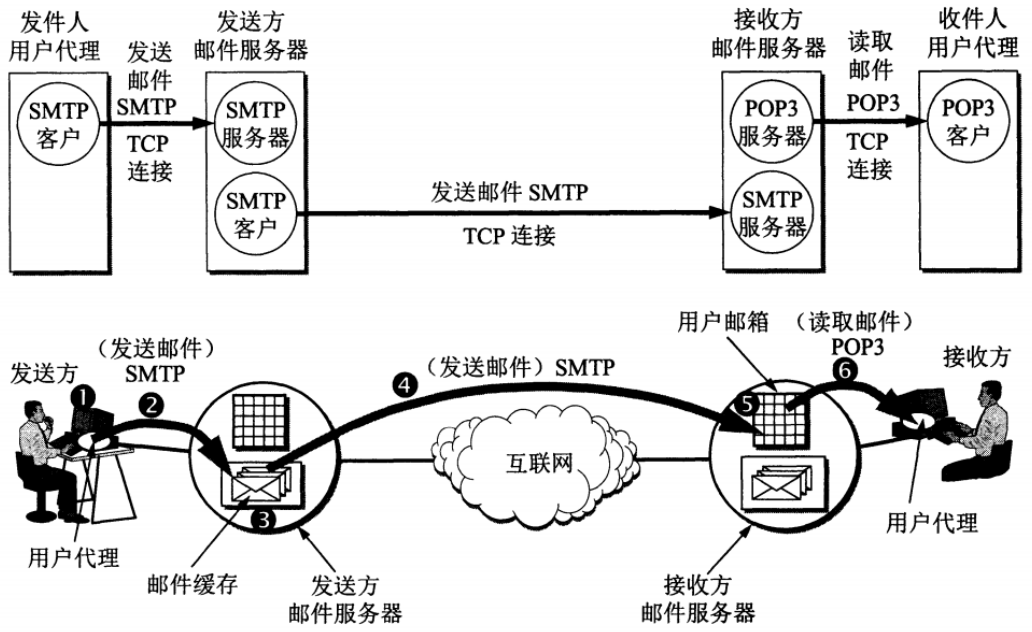
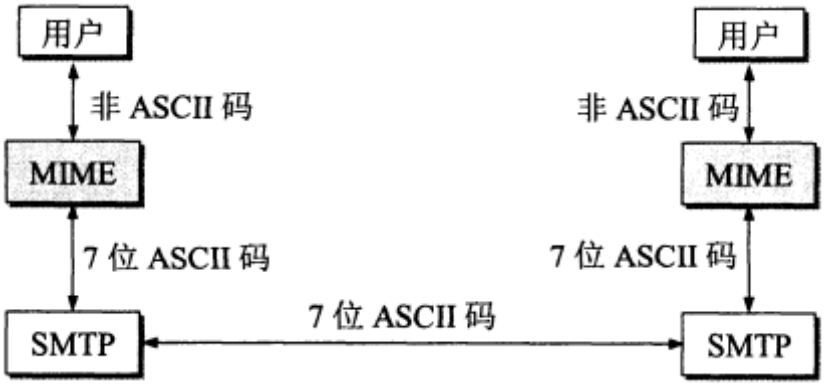


图 6-17　电子邮件的最主要的组成构件

5.1SMTP

SMTP 只能发送 ASCII 码，而互联网邮件扩充 MIME 可以发送二进制文件。MIME 并没有改动或者取代 SMTP，而是增加邮件主体的结构，定义了非 ASCII 码的编码规则。



5.2POP3

POP3 的特点是只要用户从服务器上读取了邮件，就把该邮件删除。但最新版本的 POP3 可以不删除邮件。

5.3IMAP

IMAP 协议中客户端和服务器上的邮件保持同步，如果不手动删除邮件，那么服务器上的邮件也不会被删除。IMAP 这种做法可以让用户随时随地去访问服务器上的

邮件。

6、HTTP

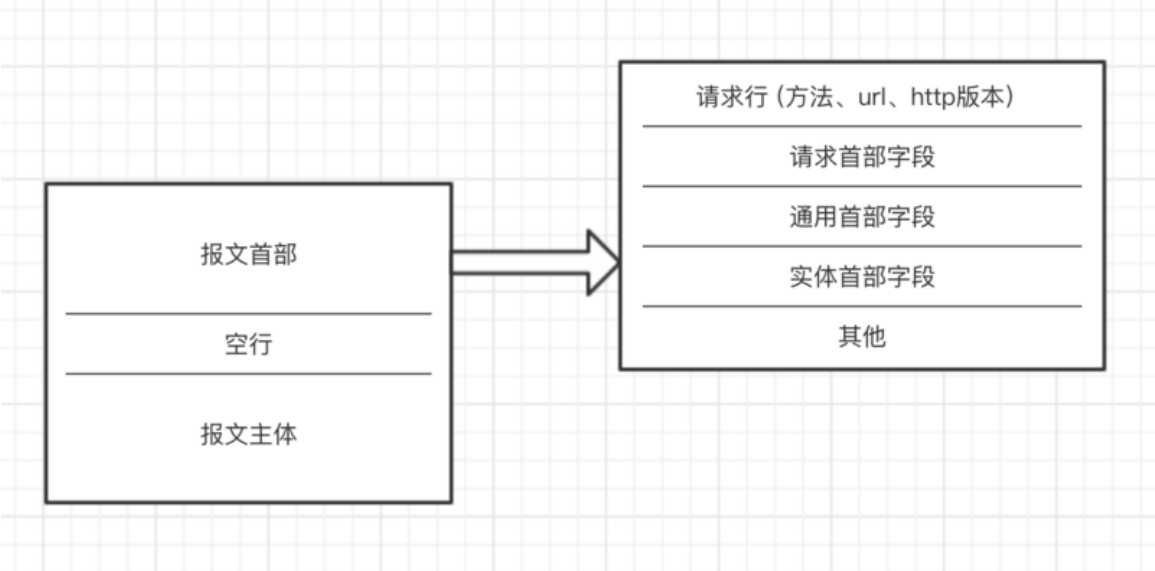
HTTP是基于TCP/IP协议的应用层协议，用于客户端和服务器之间的通信，默认80端口。

6.1HTTP首部

HTTP首部分为请求报文和响应报文。

6.1.1请求报文

HTTP 请求报文由请求行、请求头部、空行 和 请求包体 4 个部分组成



请求行：请求行由方法字段、URL 字段 和HTTP 协议版本字段 3 个部分组成，他们之间使用空格隔开。常用的 HTTP 请求方法有 GET、POST、HEAD、PUT、DELETE、OPTIONS、TRACE、CONNECT;

GET和POST

GET：当客户端要从服务器中读取某个资源时，使用GET 方法。GET 方法要求服务器将URL 定位的资源放在响应报文的数据部分，回送给客户端，即向服务器请求某个资源。使用GET 方法时，请求参数和对应的值附加在 URL 后面，利用一个问号(“?”)代表URL 的结尾与请求参数的开始，传递参数长度受限制。例如，/index.jsp?id=100&op=bind。

POST：当客户端给服务器提供信息较多时可以使用POST 方法，POST 方法向服务器提交数据，比如完成表单数据的提交，将数据提交给服务器处理。GET 一般用于获取/查询资源信息，POST 会附带用户数据，一般用于更新资源信息。POST

方法将请求参数封装在HTTP 请求数据中，以名称/值的形式出现，可以传输大量数据;

区别：

1、GET参数通过URL传递，POST放在Request body中。

2、GET请求会被浏览器主动cache，而POST不会，除非手动设置。

3、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。

4、Get 请求中有非 ASCII 字符，会在请求之前进行转码，POST不用，因为POST在Request body中，通过 MIME，也就可以传输非 ASCII 字符。

5、 一般我们在浏览器输入一个网址访问网站都是GET请求

6、HTTP的底层是TCP/IP。HTTP只是个行为准则，而TCP才是GET和POST怎么实现的基本。GET/POST都是TCP链接。GET和POST能做的事情是一样一样的。但是请求的数据量太大对浏览器和服务器都是很大负担。所以业界有了不成文规定，（大多数）浏览器通常都会限制url长度在2K个字节，而（大多数）服务器最多处理64K大小的url。

7、GET产生一个TCP数据包；POST产生两个TCP数据包。对于GET方式的请求，浏览器会把http header和data一并发送出去，服务器响应200（返回数据）；而对于POST，浏览器先发送header，服务器响应100 continue，浏览器再发送data，服务器响应200 ok（返回数据）。

8、在网络环境好的情况下，发一次包的时间和发两次包的时间差别基本可以无视。而在网络环境差的情况下，两次包的TCP在验证数据包完整性上，有非常大的优点。但并不是所有浏览器都会在POST中发送两次包，Firefox就只发送一次。

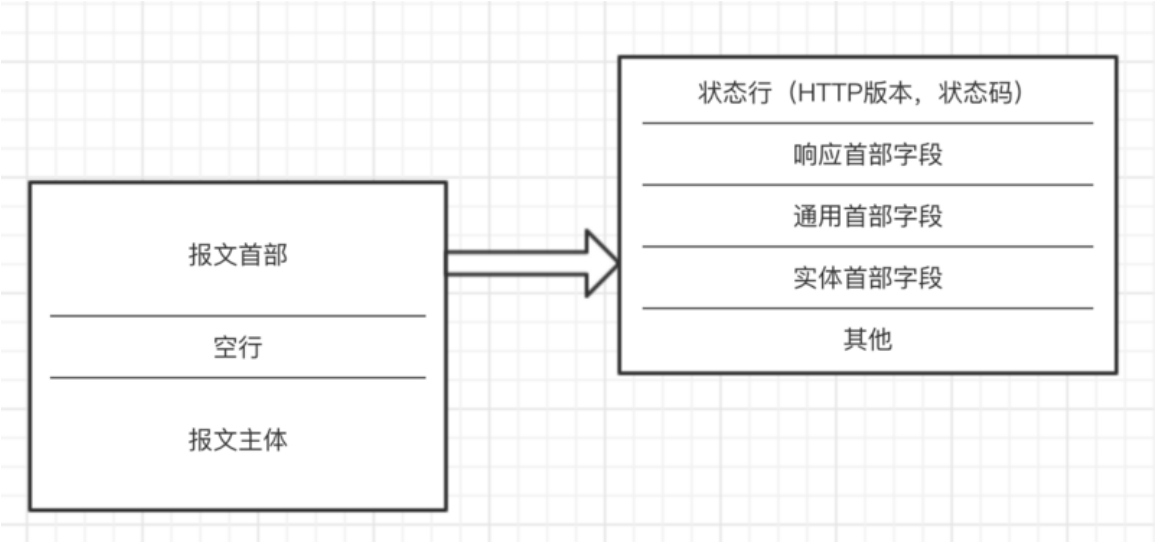
POST和PUT

POST：后一个请求不会把第一个请求覆盖掉。POST不是幂等（一个方法重复执行多次，若效果一样就是幂等的）的（所以Post用来增资源，比如新建用户）

PUT：如果两个请求相同，后一个请求会把第一个请求覆盖掉。PUT是幂等的。（所以PUT用来改资源，比如修改密码）

6.1.2响应报文

HTTP 响应报文由状态行、响应头部、空行 和 响应包体 4 个部分组成



6.1.3通用首部字段

1、Cache-Control：操作缓存的工作机制

参数：

```
public：明确表明其他用户也可以利用缓存
private：缓存只给特定的用户
no-cache：客户端发送这个指令，表示客户端不接收缓存过的响应，必须到服务器取；服务器返回这个指令，指缓存服务器不能对资源进行缓存。其实是不缓存过期资源，要向服务器进行有效期确认后再处理资源。
no-store：指不进行缓存
max-age：缓存的有效时间（相对时间）
```

2、Connection：

```
Connection：keep-Alive （持久连接）
Connection：不再转发的首部字段名
```

3、Date：表明创建http报文的日期和时间

4、Pragma：兼容http1.0，与Cache-Control：no-cache含义一样。但只用在客户端发送的请求中，告诉所有的中间服务器不返回缓存。形式唯一：Pragma：no-cache

5、Trailer：会事先说明在报文主体后记录了哪些首部字段，该首部字段可以应用在http1.1版本分块传输编码中。

6、Transfer-Encoding：chunked （分块传输编码），

规定传输报文主体时采用的编码方式，http1.1的传输编码方式只对分块传输编码

有效

7、Upgrade：升级一个成其他的协议，需要额外指定Connection：Upgrade。服务器可用101状态码作为相应返回。

8、Via：追踪客户端和服务器之间的请求和响应报文的传输路径。可以避免请求回环发生，所以在经过代理时必须要附加这个字段。

6.1.4请求首部字段

1、Accept：通知服务器，用户代理能够处理的媒体类型及媒体类型的相对优先级q表示优先级的权重值，默认为q = 1.0，范围是0~1（可精确到小数点后3位，1为最大值）
当服务器提供多种内容时，会先返回权重值最高的媒体类型

2、Accept-Charset：支持的字符集及字符集的相对优先顺序，跟Accept一样，用q来表示相对优先级。这个字段应用于内容协商机制的服务器驱动协商。

3、Accept-Encoding：支持的内容编码及内容编码的优先级顺序，q表示相对优先级。
内容编码：gzip、compress、deflate、identity（不执行压缩或者不会变化的默认编码格式）。
可以使用*作为通配符，指定任意的编码格式。

4、Accept-Language：能够处理的自然语言集，以及相对优先级。

6.1.5响应首部字段

1、Location：Location响应报头域用于重定向接受者到一个新的位置。例如：客户端所请求的页面已不存在原先的位置，为了让客户端重定向到这个页面新的位置，服务器端可以发回Location响应报头后使用重定向语句，让客户端去访问新的域名所对应的服务器上的资源;

2、Server：Server 响应报头域包含了服务器用来处理请求的软件信息及其版本。它和 User-Agent 请求报头域是相对应的，前者发送服务器端软件的信息，后者发送客户端软件(浏览器)和操作系统的信息。

3、Vary：指示不可缓存的请求头列表;

6.1.6状态码

1xx：指示信息--表示请求已接收，继续处理
2xx：成功--表示请求已被成功接收、理解、接受
3xx：重定向--要完成请求必须进行更进一步的操作
4xx：客户端错误--请求有语法错误或请求无法实现
5xx：服务器端错误--服务器未能实现合法的请求

100 Continue 表示继续发送剩余的，也就是，你这些材料我先收下，但是你要继续给我剩下的材料
101 Switching Protocols 表示换一种协议，也就是，老王路上被狗追了，心情不好，对工作人员不礼貌，然后工作人员就会说，你这什么态度，要换一种态度或者好一些。

2** 表示已经成功被服务器接收 ，就是差不多这趟进城算是办成了，具体来说
200 OK 表示成功，世界本来就该这样，老王办完事可以回去了

3** 表示重定向，就是老王进城找错地方了，具体来说
300 Multiple Choices 表示多种选择，虽然你找错地方了，我可以给你几个地址，你去那边就行了。
301 Moved Permanently 表示永久重定向 ，你找的这个地方搬家了，我给你他的地址，你去找。
302 Move temporarily 表示临时重定向，要找的地方临时搬家了，你要么去那边，要么过阵子再来
304 Not Modified 表示没有改变，情况跟上次一样，没必要跑这一趟
307Temporary Redirect 表示临时重定向，我这里虽然没有，但是我可以从别的地方拿来给你

4** 请求错误，表示老王自身出了点问题，具体来说
400Bad Request 表示请求错误，老王提的问题和要求有点问题
401 Unauthorized 表示无权限，这次来办事，本来需要介绍信的，你没有只能办完介绍信再来了
403Forbidden 表示拒绝，不用问为什么了，反正就是不给你办
404 Not Found 表示没找到，老王找错地方了，压根不该到这来
405Method Not Allowed 表示方法错误，应该先打电话预约的，却直接来了
410 Gone 表示资源以后不能再用，不要再来了，老王要找的这个人已经死了

5** 服务器错误，就是老王去的这家单位出问题了，具体来讲
500 Internal Server Error 表示服务器内部错误 就是单位着火了，没空给你办
501 Not Implemented 表示不能解决，就是这个问题确实归我们管，但是我们现在没想出办法
504 Gateway Timeout 表示请求超时，就是排队的人太多了，直到下班还没排到，只能下次再来了

6.2HTTP1.0、HTTP1.1和HTTP2.0的区别

1.0和1.1的区别

1、长连接（HTTP persistent connection ，也有翻译为持久连接），指数据传输完成了保持TCP连接不断开（不发RST包、不四次握手），等待在同域名下继续用这个通道传输数据；相反的就是短连接。

HTTP1.1支持长连接（PersistentConnection）和请求的流水线（Pipelining）处理，并且默认使用长连接，如果加入"Connection: close "，才关闭。

HTTP 1.0默认使用短连接，规定浏览器与服务器只保持短暂的连接，浏览器的每次请求都需要与服务器建立一个TCP连接，服务器完成请求处理后立即断开TCP连接，服务器不跟踪 每个客户也不记录过去的请求。要建立长连接，可以在请求消息中包含Connection: Keep-Alive头域，如果服务器愿意维持这条连接，在响应消息中也会包含一个Connection: Keep-Alive的头域。

2、流水线

请求的流水线（Pipelining）处理，在一个TCP连接上可以传送多个HTTP请求和响应，减少了建立和关闭连接的消耗和延迟。例如：一个包含有许多图像的网页文件的多个请求和应答可以在一个连接中传输，但每个单独的网页文件的请求和应答仍然需要使用各自的连接。 HTTP 1.1还允许客户端不用等待上一次请求结果返回，就可以发出下一次请求，但服务器端必须按照接收到客户端请求的先后顺序依次回送响应结果，以保证客户端能够区分出每次请求的响应内容。

3、HOST字段

HTTP1.1在Request消息头里头多了一个Host域，而且是必传的，HTTP1.0则没有这个域。

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。

4、100(Continue) Status

HTTP/1.1加入了一个新的状态码100（Continue）。客户端事先发送一个只带头域的请求，如果服务器因为权限拒绝了请求，就回送响应码401（Unauthorized）；如果服务器接收此请求就回送响应码100，客户端就可以继续发送带实体的完整请求了。100 (Continue) 状态代码的使用，允许客户端在发request消息body之前先用request header试探一下server，看server要不要接收request body，再决定要不要发request body。

5、错误通知的管理

在HTTP1.1中新增了24个错误状态响应码，如409（Conflict）表示请求的资源与资源的当前状态发生冲突；410（Gone）表示服务器上的某个资源被永久性的删除。

HTTP1.1和HTTP2.0

新的二进制格式（Binary Format），HTTP1.x的解析是基于文本。基于文本协议的格式解析存在天然缺陷，文本的表现形式有多样性，要做到健壮性考虑的场景必然很多，二进制则不同，只认0和1的组合。基于这种考虑HTTP2.0的协议解析决定采用二进制格式，实现方便且健壮。

多路复用（MultiPlexing），即连接共享，即每一个request都是是用作连接共享机制的。一个request对应一个id，这样一个连接上可以有多个request，每个连接的request可以随机的混杂在一起，接收方可以根据request的 id将request再归属到各自不同的服务端请求里面。

header压缩，HTTP1.x的header带有大量信息，而且每次都要重复发送，HTTP2.0使用encoder来减少需要传输的header大小，通讯双方各自cache一份header fields表，既避免了重复header的传输，又减小了需要传输的大小。

服务端推送（server push），同SPDY（是Google开发的基于TCP的应用层协议，用以最小化网络延迟，提升网络速度，优化用户的网络使用体验）一样，

HTTP2.0也具有server push功能。

6.3输入www.baidu.com后发生了什么

事件顺序:

1. 浏览器获取输入的域名www.baidu.com
2. 浏览器向DNS请求解析www.baidu.com的IP地址
   3.域名系统DNS解析出百度服务器的IP地址
3. 浏览器与该服务器建立TCP连接(默认端口号80)
4. 浏览器发出HTTP请求，请求百度首页
5. 服务器通过HTTP响应把首页文件发送给浏览器
6. TCP连接释放
7. 浏览器将首页文件进行解析，并将Web页显示给用户。

DHCP 配置主机信息

- 假设主机最开始没有 IP 地址以及其它信息，那么就需要先使用 DHCP 来获取。
- 主机生成一个 DHCP 请求报文，并将这个报文放入具有目的端口 67 和源端口 68 的 UDP 报文段中。
- 该报文段则被放入在一个具有广播 IP 目的地址(255.255.255.255) 和源 IP 地址（0.0.0.0）的 IP 数据报中。
- 该数据报则被放置在 MAC 帧中，该帧具有目的地址 FF:FF:FF:FF:FF:FF，将广播到与交换机连接的所有设备。
- 连接在交换机的 DHCP 服务器收到广播帧之后，不断地向上分解得到 IP 数据报、UDP 报文段、DHCP 请求报文，之后生成 DHCP ACK 报文，该报文包含以下信息：IP 地址、DNS 服务器的 IP 地址、默认网关路由器的 IP 地址和子网掩码。该报文被放入 UDP 报文段中，UDP 报文段有被放入 IP 数据报中，最后放入 MAC 帧中。
- 该帧的目的地址是请求主机的 MAC 地址，因为交换机具有自学习能力，之前主机发送了广播帧之后就记录了 MAC 地址到其转发接口的交换表项，因此现在交换机就可以直接知道应该向哪个接口发送该帧。
- 主机收到该帧后，不断分解得到 DHCP 报文。之后就配置它的 IP 地址、子网掩码和 DNS 服务器的 IP 地址，并在其 IP 转发表中安装默认网关。

ARP 解析 MAC 地址

- 主机通过浏览器生成一个 TCP 套接字，套接字向 HTTP 服务器发送 HTTP 请求。为了生成该套接字，主机需要知道网站的域名对应的 IP 地址。
- 主机生成一个 DNS 查询报文，该报文具有 53 号端口，因为 DNS 服务器的端口号是 53。
- 该 DNS 查询报文被放入目的地址为 DNS 服务器 IP 地址的 IP 数据报中。

- 该 IP 数据报被放入一个以太网帧中，该帧将发送到网关路由器。
- DHCP 过程只知道网关路由器的 IP 地址，为了获取网关路由器的 MAC 地址，需要使用 ARP 协议。
- 主机生成一个包含目的地址为网关路由器 IP 地址的 ARP 查询报文，将该 ARP 查询报文放入一个具有广播目的地址（FF:FF:FF:FF:FF:FF）的以太网帧中，并向交换机发送该以太网帧，交换机将该帧转发给所有的连接设备，包括网关路由器。
- 网关路由器接收到该帧后，不断向上分解得到 ARP 报文，发现其中的 IP 地址与其接口的 IP 地址匹配，因此就发送一个 ARP 回答报文，包含了它的 MAC 地址，发回给主机。

DNS 解析域名

- 知道了网关路由器的 MAC 地址之后，就可以继续 DNS 的解析过程了。
- 网关路由器接收到包含 DNS 查询报文的以太网帧后，抽取出 IP 数据报，并根据转发表决定该 IP 数据报应该转发的路由器。
- 因为路由器具有内部网关协议（RIP、OSPF）和外部网关协议（BGP）这两种路由选择协议，因此路由表中已经配置了网关路由器到达 DNS 服务器的路由表项。
- 到达 DNS 服务器之后，DNS 服务器抽取出 DNS 查询报文，并在 DNS 数据库中查找待解析的域名。
- 找到 DNS 记录之后，发送 DNS 回答报文，将该回答报文放入 UDP 报文段中，然后放入 IP 数据报中，通过路由器反向转发回网关路由器，并经过以太网交换机到达主机。

HTTP 请求页面

- 有了 HTTP 服务器的 IP 地址之后，主机就能够生成 TCP 套接字，该套接字将用于向 Web 服务器发送 HTTP GET 报文。
- 在生成 TCP 套接字之前，必须先与 HTTP 服务器进行三次握手来建立连接。生成一个具有目的端口 80 的 TCP SYN 报文段，并向 HTTP 服务器发送该报文段。
- HTTP 服务器收到该报文段之后，生成 TCP SYN ACK 报文段，发回给主机。
- 连接建立之后，浏览器生成 HTTP GET 报文，并交付给 HTTP 服务器。
- HTTP 服务器从 TCP 套接字读取 HTTP GET 报文，生成一个 HTTP 响应报文，将 Web 页面内容放入报文主体中，发回给主机。
- 浏览器收到 HTTP 响应报文后，抽取出 Web 页面内容，之后进行渲染，显示 Web 页面。

涉及到的协议:

(1) 应用层：HTTP(WWW访问协议)，DNS(域名解析服务)

DNS解析域名为目的IP，通过IP找到服务器路径，客户端向服务器发起HTTP会

话，然后通过运输层TCP协议封装数据包，在TCP协议基础上进行传输。
(2) 传输层：TCP(为HTTP提供可靠的数据传输)，UDP(DNS使用UDP传输)，HTTP会话会被分成报文段，添加源、目的端口；TCP协议进行主要工作。
(3)网络层：IP(IP数据数据包传输和路由选择)，ICMP(提供网络传输过程中的差错检测)，ARP(将本机的默认网关IP地址映射成物理MAC地址)为数据包选择路由，IP协议进行主要工作，相邻结点的可靠传输，ARP协议将IP地址转成MAC地址。

简单理解: 域名解析 --> 发起TCP的3次握手 --> 建立TCP连接后发起http请求 --> 服务器响应http请求，浏览器得到html代码 --> 浏览器解析html代码，并请求html代码中的资源（如js、css、图片等） --> 浏览器对页面进行渲染呈现给用户。

6.4微信扫码登陆原理

网页端+服务器

用户打开网站的登录页面的时候，向浏览器的服务器发送获取登录二维码的请求。服务器收到请求后，随机生成一个uuid，将这个id作为key值存入redis服务器，同时设置一个过期时间，再过期后，用户登录二维码需要进行刷新重新获取。同时，将这个key值和本公司的验证字符串合在一起，通过二维码生成接口，生成一个二维码的图片（二维码生成，网上有很多现成的接口和源码，这里不再介绍。）然后，将二维码图片和uuid一起返回给用户浏览器。

浏览器拿到二维码和uuid后，会每隔一秒向浏览器发送一次，登录是否成功的请求。请求中携带有uuid作为当前页面的标识符。这里有的同学就会奇怪了，服务器只存了个uuid在redis中作为key值，怎么会有用户的id信息呢?

这里确实会有用户的id信息，这个id信息是由手机服务器存入redis中的。

手机端+服务器

话说，浏览器拿到二维码后，将二维码展示到网页上，并给用户一个提示：请掏出您的手机，打开扫一扫进行登录。用户拿出手机扫描二维码，就可以得到一个验证信息和一个uuid。由于手机端已经进行过了登录，在访问手机端的服务器的时候，参数中都回携带一个用户的token，手机端服务器可以从中解析到用户的userId（这里从token中取值而不是手机端直接传userid是为了安全，直接传userid可能会被截获和修改，token是加密的，被修改的风险会小很多）。手机端将解析到的数据和用户token一起作为参数，向服务器发送验证登录请求（这里的

服务器是手机服务器，手机端的服务器跟网页端服务器不是同一台服务器）。服务器收到请求后，首先对比参数中的验证信息，确定是否为用户登录请求接口。如果是，返回一个确认信息给手机端。

手机端收到返回后，将登录确认框显示给用户（防止用户误操作，同时使登录更加人性化）。用户确认是进行的登录操作后，手机再次发送请求。服务器拿到uuId和userId后，将用户的userid作为value值存入redis中以uuid作为key的键值对中。

登录成功

然后，浏览器再次发送请求的时候，浏览器端的服务器就可以得到一个用户Id，并调用登录的方法，声成一个浏览器端的token，再浏览器再次发送请求的时候，将用户信息返回给浏览器，登录成功。这里存储用户id而不是直接存储用户信息是因为，手机端的用户信息，不一定是和浏览器端的用户信息完全一致。

7、HTTPS

WEB服务存在http和https两种通信方式，http默认采用80作为通讯端口，对于传输采用不加密的方式，https默认采用443，对于传输的数据进行加密传输。

7.1HTTPS原理

对称加密和非对称加密

对称加密加密与解密用的是同样的密钥，是最快速、最简单的一种加密方式。但对称加密有以下几个问题：

1、任意一端加密的密钥一旦泄露，则整个加密就失去了安全性。

2、加密的密钥自身的传输面临安全问题。

非对称加密使用了一对密钥，公钥（public key）和私钥（private key）。私钥只能由一方安全保管，不能外泄，而公钥则可以发给任何请求它的人。相比对称加密，他由于使用了一对密钥，其中私钥是从不外泄的因此有以下优势：

1、因为私钥不需要传输，泄露的可能性极小。

2、公钥本身可以公开传输，因此不存在泄露和安全问题。

但也存在一些不足，主要是加密效率上，相比对称加密需要更大的计算成本和时间成本。

对称加密和非对称加密配合使用，为现在网络安全传输提供了很好的解决方案。
1、每次建立连接时，生成对称加密的密钥。
2、使用非对称加密对密钥进行加密后传输，客户端和服务器进行密钥同步。
3、客户端和服务器端使用对称加密传输实际数据。
由于对称加密的密钥每次连接都不一样，是的泄露的风险大大降低，同时又保障了数据传输的安全性。SSL便是建立在对称加密和非对称加密配合使用的基础上的。

HTTPS在传输的过程中会涉及到三个密钥：

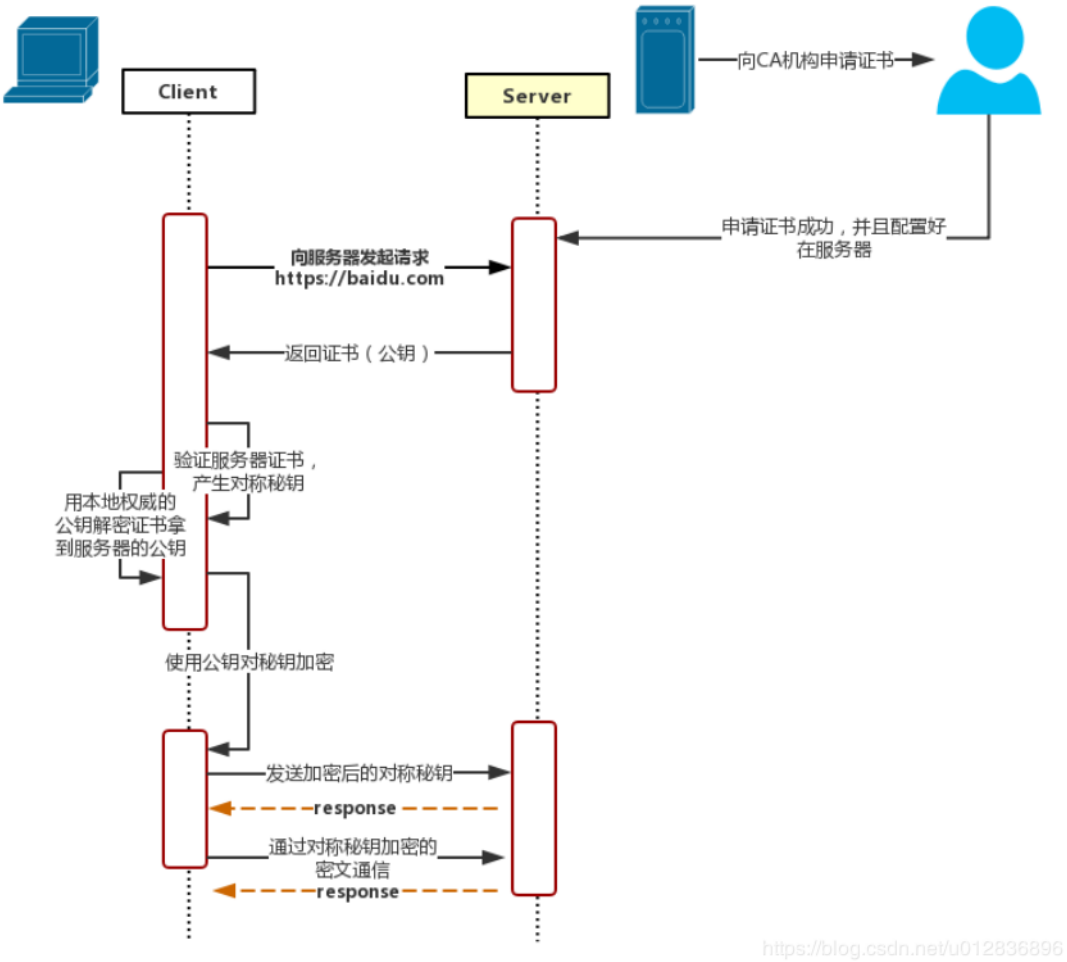
服务器端的公钥和私钥，用来进行非对称加密
客户端生成的随机密钥，用来进行对称加密

一个HTTPS请求实际上包含了两次HTTP传输，可以细分为8步：

1. 客户端向服务器发起HTTPS请求，连接到服务器的443端口
2. 服务器端有一个密钥对，即公钥和私钥，是用来进行非对称加密使用的，服务器端保存着私钥，不能将其泄露，公钥可以发送给任何人。
3. 服务器将自己的公钥发送给客户端。
4. 客户端收到服务器端的公钥之后，会对公钥进行检查，验证其合法性，如果发现发现公钥有问题，那么HTTPS传输就无法继续。严格的说，这里应该是验证服务器发送的数字证书的合法性，关于客户端如何验证数字证书的合法性。如果公钥合格，那么客户端会生成一个随机值，这个随机值就是用于进行对称加密的密钥，我们将该密钥称之为client key，即客户端密钥，这样在概念上和服务器端的密钥容易进行区分。然后用服务器的公钥对客户端密钥进行非对称加密，这样客户端密钥就变成密文了，至此，HTTPS中的第一次HTTP请求结束。
5. 客户端会发起HTTPS中的第二个HTTP请求，将加密之后的客户端密钥发送给服务器。
6. 服务器接收到客户端发来的密文之后，会用自己的私钥对其进行非对称解密，解密之后的明文就是客户端密钥，然后用客户端密钥对数据进行对称加密，这样数据就变成了密文。

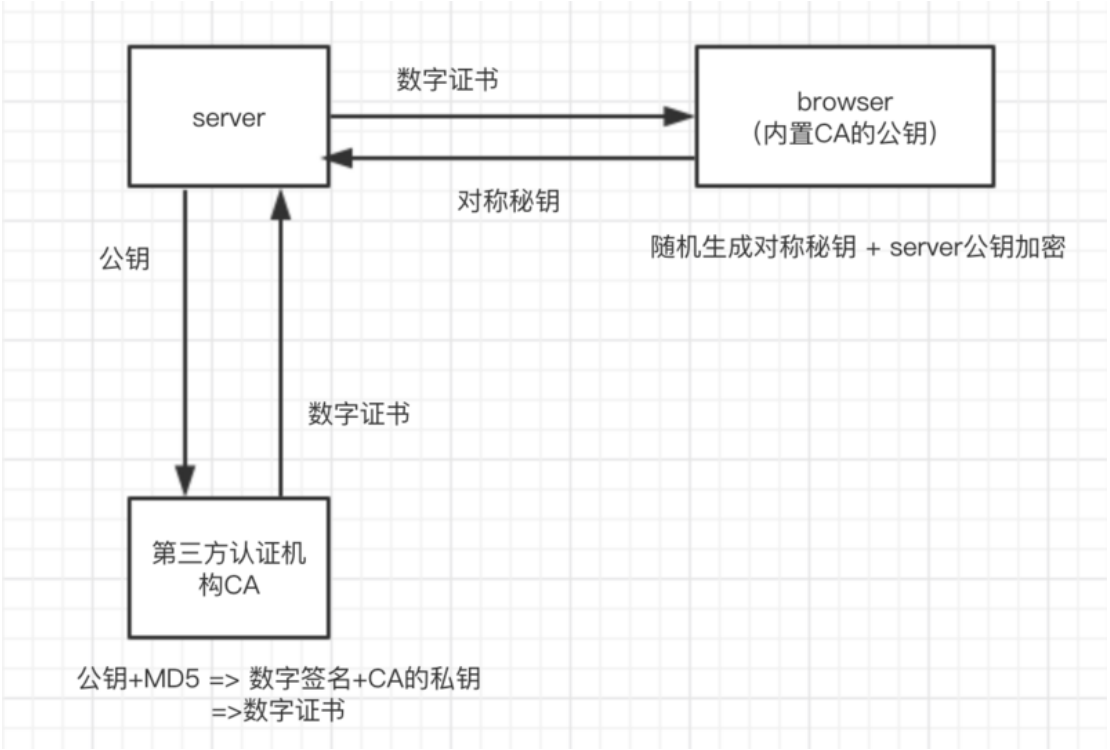
7. 然后服务器将加密后的密文发送给客户端。

8. 客户端收到服务器发送来的密文，用客户端密钥对其进行对称解密，得到服务器发送的数据。这样HTTPS中的第二个HTTP请求结束，整个HTTPS传输完成。



加密过程

- server生成一个公钥和私钥，把公钥发送给第三方认证机构（CA）；
- CA把公钥进行MD5加密，生成数字签名；再把数字签名用CA的私钥进行加密，生成数字证书。CA会把这个数字证书返回给server；
- server拿到数字证书之后，就把它传送给浏览器；
- 浏览器会对数字证书进行验证，首先，浏览器本身会内置CA的公钥，会用这个公钥对数字证书解密，验证是否是受信任的CA生成的数字证书；
- 验证成功后，浏览器会随机生成对称秘钥，用server的公钥加密这个对称秘钥，再把加密的对称秘钥传送给server；
- server收到对称秘钥，会用自己的私钥进行解密，之后，它们之间的通信就用这个对称秘钥进行加密，来维持通信。



7.2怎么保证服务器给客户端下发的公钥是真正的公钥，而不是中间人伪造的公钥呢?

使用权威机构的公钥解密数字证书，得到证书内容（服务器的公钥）以及证书的数字签名，然后根据证书上描述的计算证书签名的方法计算一下当前证书的签名，与收到的签名作对比，如果一样，表示证书一定是服务器下发的，没有被中间人篡改过。因为中间人虽然有权威机构的公钥，能够解析证书内容并篡改，但是篡改完成之后中间人需要将证书重新加密，但是中间人没有权威机构的私钥，无法加密，强行加密只会导致客户端无法解密，如果中间人强行乱修改证书，就会导致证书内容和证书签名不匹配。所以证书签名就能判断证书是否被篡改。

7.3证书如何安全传输，被掉包了怎么办?

中间人同样可以向权威机构申请一份证书，然后在服务器给客户端下发证书的时候劫持原证书，将自己的假证书下发给客户端，客户端收到之后依然能够使用权威机构的公钥解密证书，并且证书签名也没问题。但是这个时候客户端还需要检查证书中的域名和当前访问的域名是否一致。如果不一致，会发出警告!

8、HTTP缓存机制

HTTP的缓存分为强缓存和协商缓存（对比缓存）。

8.1强制缓存

在缓存数据未失效的情况下，可以直接使用缓存数据；在没有缓存数据的时候，浏览器向服务器请求数据时，服务器会将数据和缓存规则一并返回，缓存规则信息包含在响应header中。

- Expires：缓存过期时间（HTTP1.0）
  缺点：生成的是绝对时间，但是客户端时间可以随意修改，会导致误差。
- Cache-Control ： HTTP1.1，优先级高于Expires
  可设置参数：

```
private: 客户端可以缓存
public: 客户端和代理服务器都可缓存
max-age=xxx: 缓存的内容将在 xxx 秒后失效
no-cache: 需要使用协商缓存来验证缓存数据（后面介绍）
no-store: 所有内容都不会缓存，强制缓存，对比缓存都不会触发
```

Expires和Cache-Control决定了浏览器是否要发送请求到服务器，ETag和Last-Modified决定了服务器是要返回304+空内容还是新的资源文件。

8.2协商缓存

浏览器第一次请求数据时，服务器会将缓存标识与数据一起返回给客户端，客户端将二者备份至缓存数据库中。再次请求数据时，客户端将备份的缓存标识发送给服务器，服务器根据缓存标识进行判断，判断成功后，返回304状态码，通知客户端比较成功，可以使用缓存数据。

8.3缓存判断顺序

- 先判断Cache-Control，在Cache-Control的max-age之内，直接返回200 from cache；
- 没有Cache-Control再判断Expires，再Expires之内，直接返回200 from cache；
- Cache-Control=no-cache或者不符合Expires，浏览器向服务器发送请求；
- 服务器同时判断ETag和Last-Modified，都一致，返回304，有任何一个不一致，返回200。

8.4cookie和session

8.4.1cookie

解决http的无状态问题，是客户端保存用户信息的一种机制，用来记录用户的一些信息，来实现session的跟踪。

1、cookie属性

name、value :以key/value的形式存在

comment：说明该cookie的用处

domain：可以访问该cookie的域名

Expires/maxAge：cookie失效时间。负数：临时cookie，关闭浏览器就失效；0：表示删除cookie，默认为-1

path：可以访问此cookie的页面路径

size：cookie的大小

secure：是否以https协议传输

version：该cookie使用的版本号，0遵循Netscape规范，大多数用这种，1遵循W3C规范

HttpOnly：此属性为true，则只有在http请求头中会带有此cookie的信息，而不能通过document.cookie来访问此cookie，能防止XSS攻击。

2、cookie原理

客户端请求服务器时，如果服务器需要记录该用户状态，就使用response向客户端浏览器颁发一个Cookie。而客户端浏览器会把Cookie保存起来。当浏览器再请求服务器时，浏览器把请求的网址连同该Cookie一同提交给服务器。服务器通过检查该Cookie来获取用户状态。

3、cookie同源和跨域

cookie的同源是域名相同，忽略协议和端口，不可跨域。

8.4.2session

session是在服务端保存的一个数据结构，用来跟踪用户的状态，这个数据可以保存在集群、数据库、文件中。

session的运行依赖session id，而session id是存在cookie中的。

1、session原理

当客户端请求创建一个session的时候，服务器会先检查这个客户端的请求里是否已包含了一个session标识——sessionId。如果已包含这个sessionId，则说明以前已经为此客户端创建过session，服务器就按照sessionId把这个session检索出来使用（如果检索不到，可能会新建一个。如果客户端请求不包含sessionId，则为此客户端创建一个session并且生成一个与此session相关联的sessionId。

2、如果禁用了cookie

使用URL重写技术来进行会话跟踪。在 url 中传递 session id,即每次HTTP交互，URL后面都会被附加上一个诸如 sid=xxxxx 这样的参数，服务端据此来识别用户。

8.4.3区别

- cookie 数据存放在客户的浏览器上，session数据放在服务器上；
- 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;
- 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;
- 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。session可以达到5M甚至更多
- Cookie 只能存储 ASCII 码字符串，而 Session 则可以存储任何类型的数据，因此在考虑数据复杂性时首选 Session

9、HTTP断点续传

有时候可能下载的文件过大，一次性传输遇到网络问题就很可能传输失败而需要全部重新下载。另外，断点续传还能够提供并发的下载提高下载速率。

浏览器

如果浏览器发起的请求中没有带range字段，而服务器没有返回全部数据，而是返回部分数据和一个206的状态码，则浏览器不会发起对剩余字段的请求，此时下载失败。

如果浏览器发起的请求带range字段，服务器返回指定range范围的数据，则下载成功

迅雷等下载器

如果迅雷发起的请求中没有带range字段，而服务器没有返回全部数据，而是返回

部分数据和一个206的状态码，则迅雷会发起对剩余字段的请求，此时下载成功。

如果迅雷发起的请求带range字段，服务器返回指定range范围的数据，则下载成功。（服务器返回的range字段最好起始和结束点都一致，如果返回的range范围过小迅雷可能会提示下载失效）

结论

服务器的设计方面，应该让range的主动权应该掌握在客户端这边，如果请求中没有带range字段，则应该全部数据返回。如果带range字段，则应该返回确切范围内的数据。

如果下载的数据过大，服务器担心承受不住，可以断开TCP连接，则客户端那边会根据已经下载的内容再去发起对未下载内容的请求（带range字段）

这种设计模式下用迅雷下载，会先发起一个不带range的请求，拿到整个文件大小后再分段多线程发起对部分数据的请求。用谷歌浏览器下载，则直接是一个超级大的http包返回（状态码200）