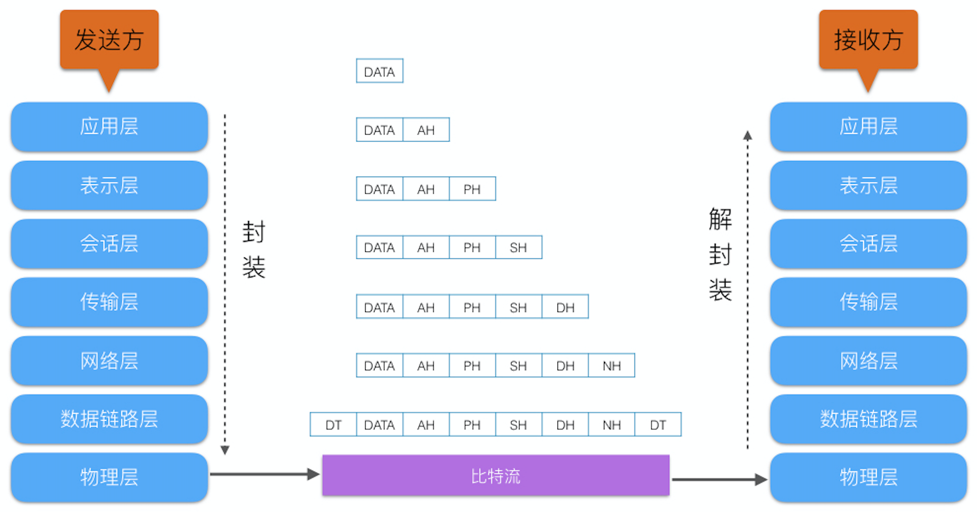
**OSI七层模型**

　　OSI(Open System Interconnection，开放系统互联)七层网络模型成为开放式系统互联参考模型，是一个把网络通信在逻辑上的定义，也可以理解成为定义了通用的网络通信规范。而我们的数据在网络中传输的过程，实际上就是如下图的封装和解封装的过程，发送方通过各种封装处理，把数据转换成比特流的形式，比特流在信号传输的硬件媒介中传输，接收方再把比特流进行解封装处理。

**1.物理层**

**规定了如何为网络通信实现最底层的物理连接，以及物理设备的机械、电气、功能和过程特性。**如：如何使用电缆和接头的类型、用来传送信号的电压等。需要注意的是，网络通信过程中所需的物理媒介(网线、线缆等)，其实并不属于物理层，因为物理层实际上是一种规定，规定这些物理媒介设备在连接网络时的各种规格、参数以及工作方式。但是同时，双绞线、线缆等物理媒介又是物理层的实现。

**2.数据链路层**

**规定了如何进行物理地址寻址、如何在物理线路上进行数据(帧frame)的可靠传递以及流量控制。**

　　数据链路层协议有SLIP协议、CSLIP协议、PPP协议等。交换机，对帧解码并根据帧中包含的信息把数据发送到正确的接收方，所以交换机是工作在数据链路层的。

**3.网络层**

**规定了通过哪些网络节点、什么样的网络路径来将数据(数据包)从发送方发送到接收方。**在网络层中，确定了从节点A发数据到节点B的网络路径，经过哪些节点。网络层既可以建立LAN通信系统，更主要的是可以在WAN网络系统中建立通信，这是因为它有自己的路由地址结构，**通过路由协议(又称可路由协议)进行网络通信的路由工作。**

**4.传输层**

**负责总体的数据传输和数据控制，提供端到端的交换数据的机制。**传输层对数据(段)进行分割和重组，并且进行流量控制和根据接收方的接收数据能力确定适当的传输速率。例如以太网无法处理大于1500字节的数据包，传输层将数据分割成数据片段，并对小数据片段进行序列编号。接收方的传输层将根据序列编号对数据进行重组。

**传输层协议有TCP协议、UDP协议等。**

**5.会话层**

**在网络中的两个节点之间建立、维持和终止通信。**

**6.表示层**

**在应用程序和网络之间对数据进行格式化，使之能够被另一方理解。**即发送方的表示层将应用程序数据的抽象语法转换成网络适用于OSI网络传输的传送语法，接收方则相反。除此之外，表示层还可对数据进行加密与解密。

**7.应用层**

**最顶层的OSI层，为应用程序提供网络服务。**如为电子邮件、文件传输功能提供协议支持。

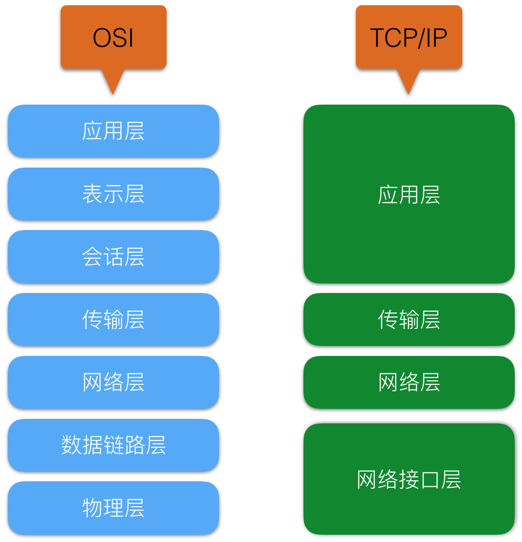
　　应用层协议有HTTP协议、FTP协议、SMTP协议等。

## 什么是TCP/IP协议？

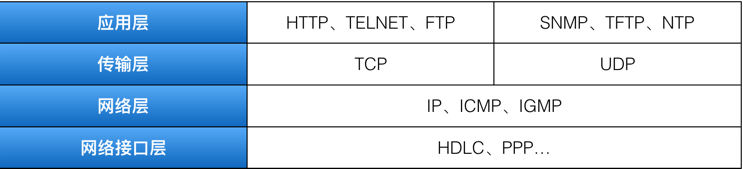
　　TCP/IP协议定义了今天的电子设备如何连入internet网络，以及数据如何在他们之间传输的标准。在今天的基于TCP/IP的互联网诞生之前，能够使用接口通信处理实现互联互通的电脑并不多，而且大部分电脑之间信息的交换并不兼容。后来好几个牛逼哄哄的歪果仁开始捣鼓一些协议，能够让电脑之间进行通信。终于在1974年12月，Bob Kahn和Vinton G.Cerf带领的团队首先制定出了通过详细定义的TCP/IP协议标准。当时作了一个试验，将信息包通过点对点的卫星网络，再通过陆地电缆，再通过卫星网络，再由地面传输，贯串欧洲和美国，经过各种电脑系统，全程9.4万公里竟然没有丢失一个数据位，远距离的可靠数据传输证明了TCP/IP协议的成功。1983年1月1日，运行较长时期曾被人们习惯了的NCP被停止使用，TCP/IP协议作为因特网上所有主机间的共同协议，从此以后被作为一种必须遵守的规则被肯定和应用。（让我们为Bob Kahn和Vinton G.Cerf两位老爷爷撒花鼓掌吧👏👏👏）

　　从“TCP/IP”名字上来看，貌似这只是tcp协议和ip协议，但是实际上，这是**很多协议(有人说有几百个协议，感觉待考证)的集合**。由于知识广度和时间限制，将不对TCP/IP协议进行全面剖析，主要对其中比较重要的几个点进行阐述。

　　从概念上来讲，TCP/IP协议族则把7层网络模型合并成4层，其对应关系如下：

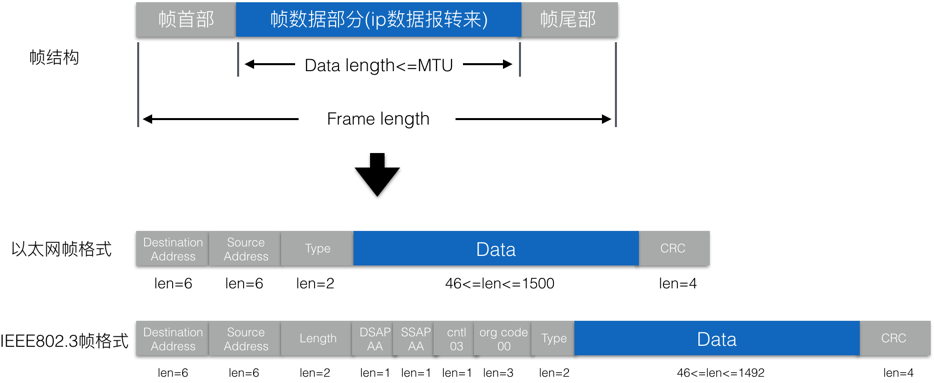


各层协议举例如下：



## TCP/IP——网络接口层

**主要作用一：数据封装/解封装成帧(frame)。**为了保证可靠传输，网络层传过来的数据在这里被加工成了可被物理层传输的结构包——帧。帧中除了包括需要**传输的数据**外，还包括发送方和接收方的**物理地址**以及**检错和控制信息**。其中的物理地址确定了帧将发送到何处，检错和控制信息则是用来保证数据的无差错到达。数据帧结构如下(Address均为mac地址)：



**主要作用二：控制帧的传输。**控制帧的传输主要体现在反馈重发、计时器、帧序号方面。接收方通过对帧的差错编码(奇偶校验码或 CRC 码)的检查，来判断帧在传输过程中是否出错，并向发送方进行反馈，如果传输发生差错，则需要重发纠正。作为发送方，如果在发送帧后，会同时启动定时器，如果帧发送后在一定时间内没有收到反馈，为了避免传输停滞不前，则在计时器Timeou后认为帧传输出错，自动重发。为了避免多次收到同一帧并将其递交给网络层的情况发生，则需要对每个发送的帧进行编号，接收方以此来判断该帧是否重复接受了。

**主要作用三：流量控制。**由于收发双方各自使用的设备工作速率和缓冲存储空间的差异，可能出现发送方的发送能力大于接收方接收能力的现象，此时若不对发送方的发送速率做适当的限制，前面来不及接收的帧将被后面不断发送来的帧“淹没”，从而造成帧的丢失而出错。由此可见，流量控制实际上是对发送方数据流量的控制，使其发送速率不超过接收方的速率。所以需要一些规则使得发送方知道在什么情况下可以接着发送下一帧，而在什么情况下必须暂停发送，以等待收到某种反馈信息后再继续发送。

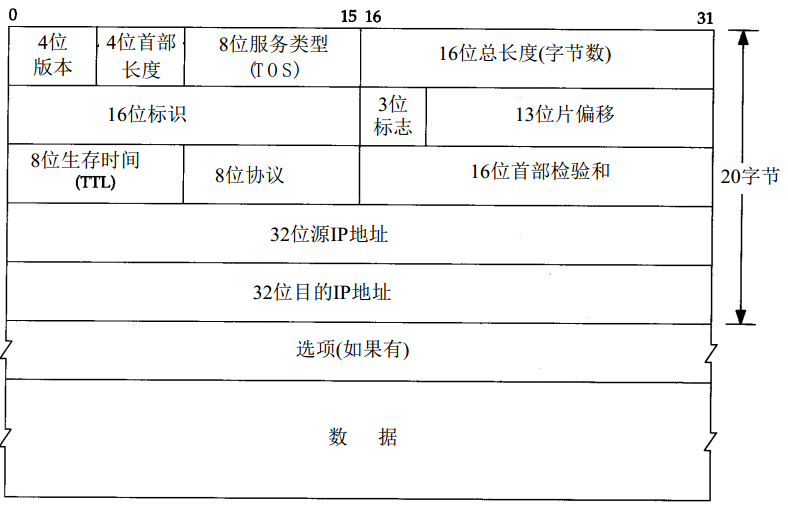
## TCP/IP——网络层

说到网络层不得不提的就**是IP协议，它是TCP/IP协议族中最为核心的协议**。所有的TCP、UDP、ICMP、IGMP协议数据都以**IP数据报格式传输**。IP协议提供的是不可靠的、无连接的数据报传输服务。

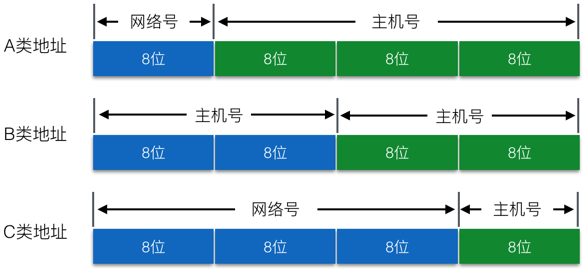
**不可靠是指IP协议不会保证数据报能否成功到达目的地，仅提供传输服务，传输出错，则会丢弃出错的数据报。**

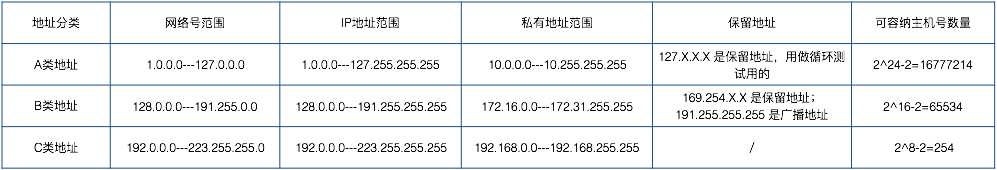
**无连接是指IP协议对数据报的处理是独立的，这也意味着接收方不一定会按照发送顺序接收数据报。**

**IP数据报格式如下：**



**IP地址分类：**





**子网划分：**

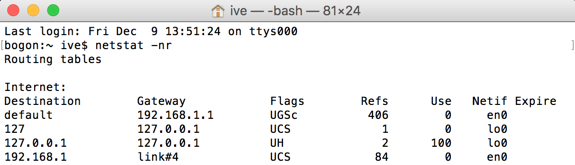
　　一个有500台主机的网络，如果使用C类地址，则无法满足主机数量需求，使用B类地址则会造成IP地址的浪费。由此，需要在ABC类网络的基础上进行子网划分：即占用主机号的前几位表示子网号。子网掩码的概念由此被引入。



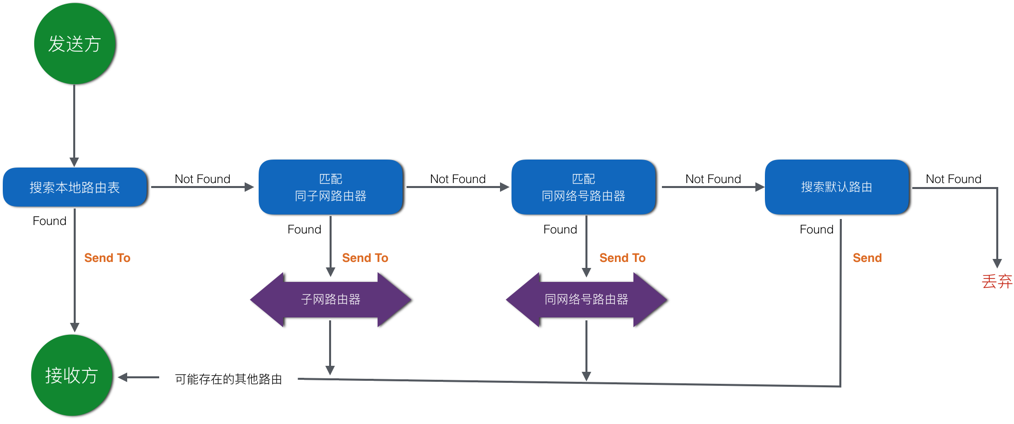
子网掩码中的 1 标识了 IP 地址中相应的网络号，0 标识了主机号。将 IP 地址和子网掩码进行逻辑与运算 ，结果就能得到网络号和子网号。

**IP路由：**

　　如果发送方和接收方在直接点对点连接或者在一个共享网络上，那么IP数据报则可以直接送达。但多数情况下，两台主机互联，需要通过多台路由器，需要路由转发送达消息。这个路由的过程是怎样的呢？这里有个重要的角色——路由表。路由表是一张存储在内存中的记录路由信息的表，定义着到达特定网络终端的路径，在某些情况下，还有一些与这些路径相关的度量。路由表中保存的信息，则用来指导消息从发送方到达接受方的网络路由路径。Linux下查看路由表的命令是route -n，mac os查看路由表的命令是netstat -nr。



　　有了路由表，网络路径的确认过程则如下所示：



　接收到数据报的路由器按照它自己的路由表进行继续转发，直到到达目的地址。而除了在路由寻址过程中失败会导致丢弃数据报之外，IP数据报的TTL(生命周期)被减为0，则该IP数据报也会被丢弃。

　　网络层还有几个比较重要的协议，ARP(Address Resolution Protocol地址解析协议)、ICMP(Internet Control Message Protocol报文控制协议)、IGMP(Internet Group Management Protocol组管理协议)，这里不做具体说明啦。

## TCP/IP——传输层

　　无论参考OSI还是TCP/IP的网络模型，我们从传输层向更底层看，各层的协议都是在直接或间接的服务于主机与主机之间的通信，而传输层则是在进程与进程通信层面上的。传输层有两个重要的协议——TCP协议和UDP协议。不同的应用进程则使用不同的传输层协议。



　端口的作用则正是体现在传输层的。用来区分网络消息由主机上的那一个进程处理。端口号有 0～65535 的编号，其中0～1023为系统端口号。



**UDP协议：**

　　UDP(User Datagram Protocol)即用户数据报协议，其传输机制决定了它的最大优点——快，同时也决定了它最大的缺点——不可靠、不稳定。

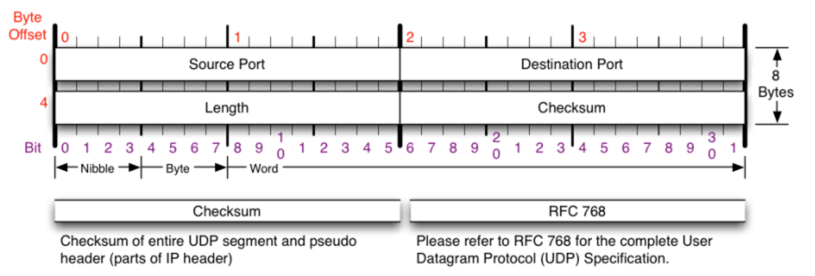
　　UDP是无连接的，发送数据之前不需要建立连接(TCP需要)。减少了开销和延时。

　　UDP是面向报文的，对IP数据报只做简单封装(8字节UDP报头)。减少报头开销。

　　UDP没有阻塞机制，宁愿阻塞时丢弃数据不传，也不阻塞造成延时。

　　UDP支持一对一、一对多、多对一、多对多通信。

　　UDP报文结构：



**TCP协议：**

　　TCP(Transmission Control Protocol)传输控制协议，相对于UDP，TCP是面向连接的、提供可靠的数据传输服务。同时也是较UDP开销较大的、传输速度较慢的。

　　TCP提供可靠的、面向连接的数据传输服务。使用TCP通信之前，需要进行“三次握手”建立连接，通信结束后还要使用“四次挥手”断开连接。

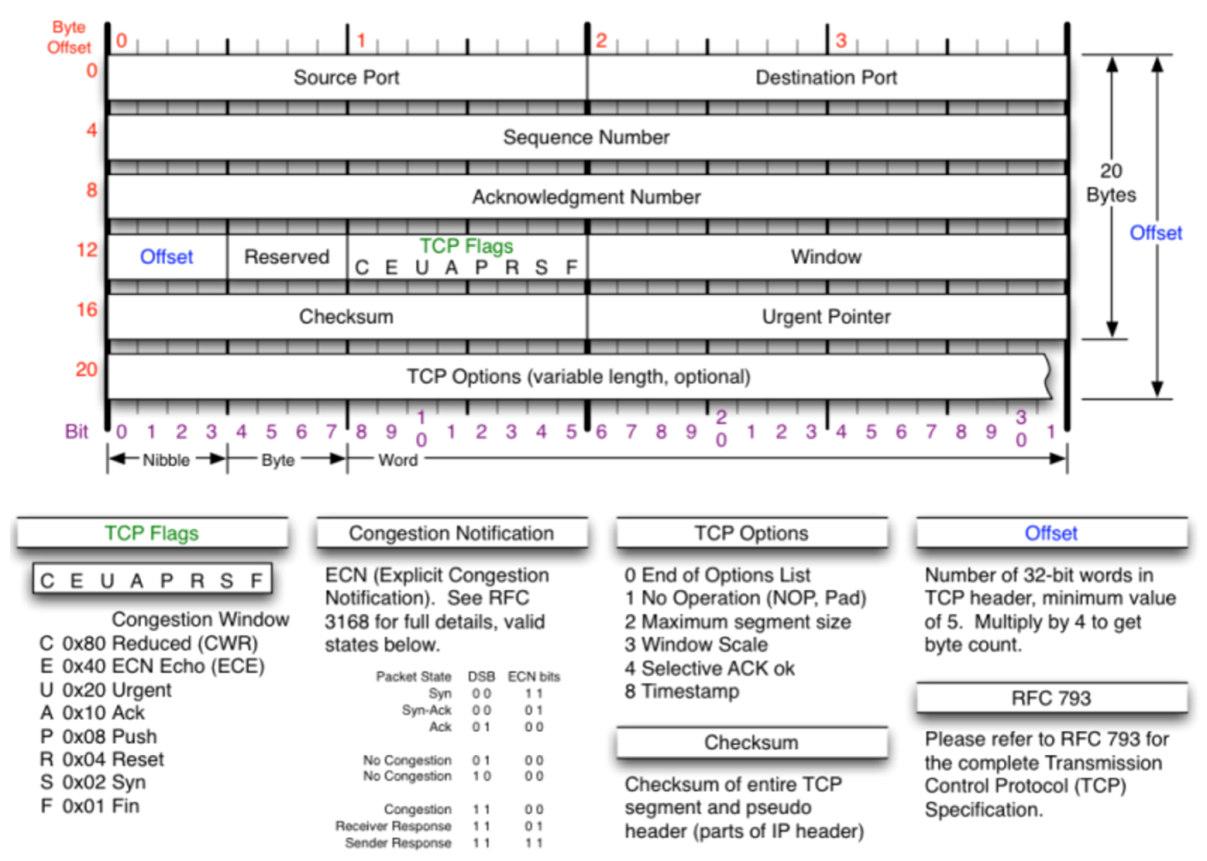
　　TCP是点对点的连接。一条TCP连接只能连接两个端点。

　　TCP 提供可靠传输，无差错、不丢失、不重复、按顺序。

　　TCP 提供全双工通信，允许通信双方任何时候都能发送数据，发送方设有发送缓存，接收方设有接收缓存。

　　TCP 面向字节流 。TCP 并不知道所传输的数据的含义，仅把数据看作一连串的字节序列，它也不保证接收方收到的数据块和发送方发出的数据块具有大小对应关系。

　　TCP报文结构：



　TCP是面向字节流的，通过 TCP 传送的字节流中的每个字节都按顺序编号，而报头中的Sequence Number字段值则指的是本报文段数据的第一个字节的序号。Acknowledgment Number是期望收到对方下个报文段的第一个数据字节的序号。

　　Offset：占4位，指 TCP 报文段的报头长度，包括固定的20字节和TCP Options字段。

　　Reserved：占6位，保留为今后使用，目前为0。

　　TCP flags的C、E、U、A、P、R、S、F字段用来说明该报文的性质。意义如下：

　　　　C(CWR)和E(ECE)用来支持ECN(显示阻塞通告)。

　　　　U(URGENT)：当 URG=1时，它告诉系统此报文中有紧急数据，应优先传送(比如紧急关闭)，这要与紧急指针字段配合使用。

　　　　A(ACK)：仅当 ACK=1时确认号字段才有效。建立 TCP 连接后，所有报文段都必须把 ACK 字段置为 1。

　　　　P(PUSH)：若TCP连接的一端希望另一端立即响应，PSH字段便可以“催促”对方，不再等到缓存区填满才发送。

　　　　R(RESET)：若 TCP 连接出现严重差错，RST 置为 1，断开 TCP 连接，再重新建立连接。

　　　　S(SYN)：用于建立和释放连接，当SYN=1时，表示建立连接。

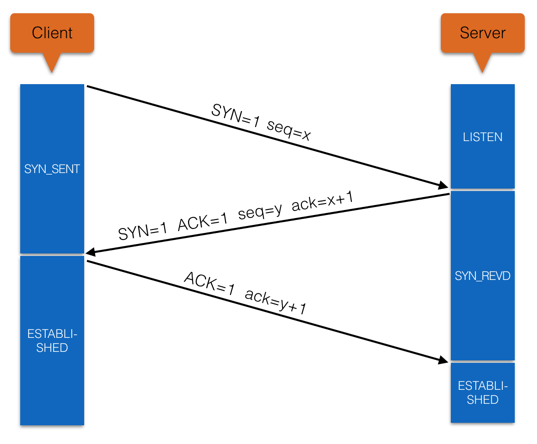
　　　　F(FIN)：用于释放连接，当 FIN=1，表明发送方已经发送完毕，要求释放TCP 连接。

　　Window：占2个字节。窗口值是指发送者自己的接收窗口大小，因为接收缓存的空间有限。

　　CheckSum：占2个字节。和UDP报文一样，有一个检验和，用于检查报文是否在传输过程中出差错。

　　Urgent Pointer：占2字节。当URG=1时才有效，指出本报文段紧急数据的字节数。

TCP建立连接的三次握手：



(1)Client首先向Server发送连接请求报文段，同步自己的seq(x)，Client进入SYN\_SENT状态。

　　(2)Server收到Client的连接请求报文段，返回给Client自己的seq(y)以及ack(x+1)，Server进入SYN\_REVD状态。

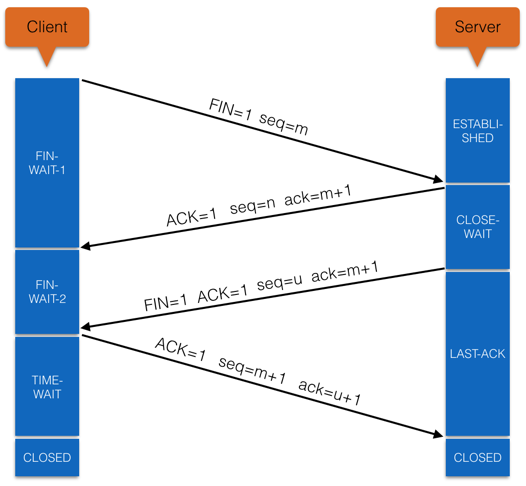
　　(3)Client收到Server的返回确认，再次向服务器发送确认报文段ack(y+1)，这个报文段已经可以携带数据了。Client进入ESTABLISHED状态。

　　(4)Server再次收到Client的确认信息后，进入ESTABLISHED状态。

　　TCP连接至此建立起来了。为什么要做三次握手呢？握手的过程实际上是在通知对方自己的初始化序号(Initial Sequence Number)，简称ISN，也就是上图中的x和y。x和y会被当作之后传输数据的一个依据，以保证TCP报文在传输过程中不会混乱。

　　我们回到TCP Header结构来看，Sequence Number和Acknowledgment Number都是占32位，所以seq和ack的取值范围是0 ～ 2^32-1。seq和ack每增加到2^32-1，则重新从0开始。值得一提的是，seq的初始值(ISN)并不是每次都从0开始的。我们设想一下，如果是从0开始，那么当TCP三次握手建立连接完成后，Client发送了30个报文，然后Client断线了。于是Client重连，再次用0作为初始的seq，这样就会出现两个报文具有相同的seq，就出现了混乱。事实上TCP的做法是每隔4微秒就对ISN做一次加1操作，当ISN到达2^32-1后再次从0开始的时候，已经过去了几个小时，之前的seq=0的报文已经不存在于这次连接中了，这样就避免了上面的问题。

　　TCP断开连接的四次挥手：



(1)Client向Server发送断开连接请求的报文段，seq=m(m为Client最后一次向Server发送报文段的最后一个字节序号加1)，Client进入FIN-WAIT-1状态。

　　(2)Server收到断开报文段后，向Client发送确认报文段，seq=n(n为Server最后一次向Client发送报文段的最后一个字节序号加1)，ack=m+1，Server进入CLOSE-WAIT状态。此时这个TCP连接处于半开半闭状态，Server发送数据的话，Client仍然可以接收到。

　　(3)Server向Client发送断开确认报文段，seq=u(u为半开半闭状态下Server最后一次向Client发送报文段的最后一个字节序号加1)，ack=m+1，Server进入LAST-ACK状态。

　　(4)Client收到Server的断开确认报文段后，向Server发送确认断开报文，seq=m+1，ack=u+1，Client进入TIME-WAIT状态。

　　(5)Server收到Client的确认断开报文，进入CLOSED状态，断开了TCP连接。

　　(6)Client在TIME-WAIT状态等待一段时间(时间为2\*MSL((Maximum Segment Life))，确认Client向Server发送的最后一次断开确认到达(如果没有到达，Server会重发步骤(3)中的断开确认报文段给Client，告诉Client你的最后一次确认断开没有收到)。如果Client在TIME-WAIT过程中没有再次收到Server的报文段，就进入CLOSES状态。TCP连接至此断开。

　　TCP连接可靠性的体现：

　　(1)TCP报文段的长度可变，根据收发双方的缓存状态、网络状态而调整。

　　(2)当TCP收到发自TCP连接另一端的数据，它将发送一个确认。

　　(3)当TCP发出一个段后，它启动一个定时器，等待目的端确认收到这个报文段，如果不能及时收到一个确认，将重发这个报文段。

　　(4)TCP将保持它首部和数据的检验和。如果通过检验和发现报文段有差错，这个报文段将被丢弃，等待超时重传。

　　(5)TCP将数据按字节排序，报文段中有序号，以确保顺序的正确性。

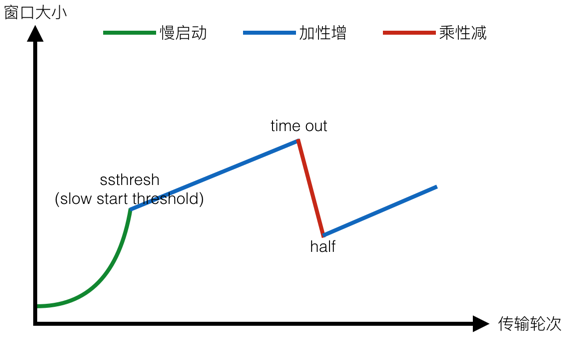
　　(6)TCP还能提供流量控制。TCP连接的每一方都有收发缓存。TCP的接收端只允许另一端发送接收端缓冲区所能接纳的数据。这将防止较快主机致使较慢主机的缓冲区溢出。

　　需要注意的是，TCP报文传输采用接受后返回确认的方式来保证报文传输的可靠性，并不是意味着发送方在发送一个报文段后就进入等待确认状态，让后面的报文段等着。也不是接收方在接收到一个报文后，对每一个报文都进行回复确认。

　　真实的情况是，对于发送方，在发送一个报文段后，复制一份该报文段的副本，然后继续进行下一个报文段的发送，如果没有得到发送方的回复确认，就对该报文段进行超时重发。对于接收方来说，则采用“积累确认”的方式进行回复。接收者收到多个连续的报文段后，只回复确认最后一个报文段，表示在这之前的数据都已收到。以此达到提升传输效率的目的。

　　TCP的流量控制和阻塞控制：

　　由于接收方缓存的限制，发送窗口不能大于接收方接收窗口。在报文段首部有一个字段就叫做窗口(rwnd)，这便是用于告诉对方自己的接收窗口，可见窗口的大小是可以变化的。



　总结起来如上图，TCP的流量和阻塞控制采用“慢启动”、“加性增”、“乘性减”的策略。

　　慢启动：初始的窗口值很小，但是按指数规律渐渐增长，直到达到慢开始门限(ssthresh)。

　　加性增：窗口值达到慢开始门限后，每发送一个报文段，窗口值增加一个单位量。

　　乘性减：无论什么阶段，只要出现超时，则把窗口值减小一半。

## TCP/IP——应用层

 　　关于应用层协议，不同的应用或者具体来说同一种应用不同的需求，都会使用不同的应用层协议，作为互联网软件开发工程师，我们可能对应用层协议最为熟悉，这篇文章中将不再对应用层协议进行总结。

# 最后说几句：ip是面向无连接的 为何tcp确能面向连接呢

主机是端到端联系的，你可以这样去理解：IP跟对端的IP打招呼，TCP跟对端的TCP打招呼。  
TCP向下封装在IP包里面，通过IP包传输到对端，IP跟对端的IP打上招呼了，就将TCP的报文解封装，并交接上一层的TCP，这样TCP也跟对端的TCP打上招呼了。  
IP打招呼的方式呢：是面向无连接的。  
TCP打招呼的方式呢：是面向连接的，三次握手验证：你跟对方说HI，对方跟你说HI，然后你跟对方建立起连接，这就叫面向连接的服务。