# 网络

## 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。数据单位为报文。

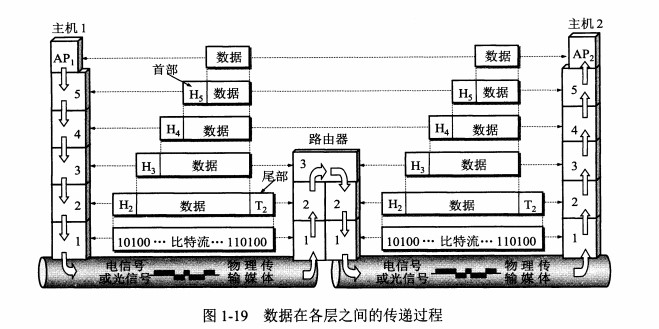
传输层：提供的是进程间的通用数据传输服务。由于应用层协议很多，定义通用的传输层协议就可以支持不断增多的应用层协议。传输层包括两种协议：传输控制协议 TCP，提供面向连接、可靠的数据传输服务，数据单位为**报文段**；用户数据报协议 UDP，提供无连接、尽最大努力的数据传输服务，数据单位为**用户数据报**。TCP 主要提供完整性服务，UDP 主要提供及时性服务。

网络层：为主机之间提供数据传输服务，而运输层协议是为主机中的进程提供服务。网络层把运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结点提供服务。数据链路层把网络层传来的分组封装成帧。

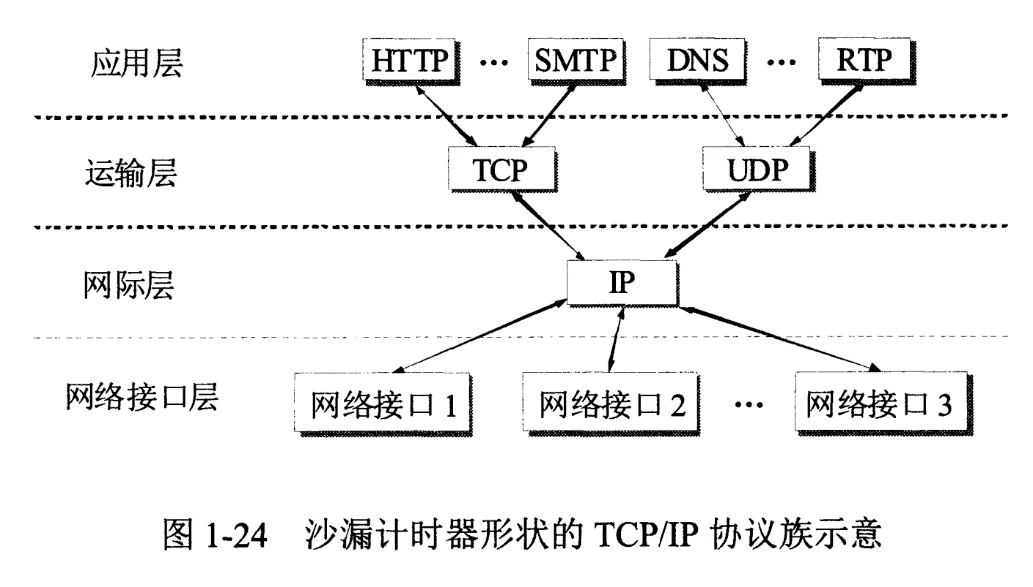
物理层：考虑的是怎样在传输媒体上传输数据比特流，而不是指具体的传输媒体。物理层的作用是尽可能屏蔽传输媒体和通信手段的差异，使数据链路层感觉不到这些差异。

## 数据在各层传递过程



路由器只有下面三层协议，路由器位于网络层，传输层涉及到TCP和UDP了。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，因此也就不需要运输层和应用层。

TCP/IP体系结构只有四层，底下的数据链路层和物理层合并为网络接口层。TCP/IP 协议族是一种沙漏形状，中间小两边大，IP 协议在其中占用举足轻重的地位。

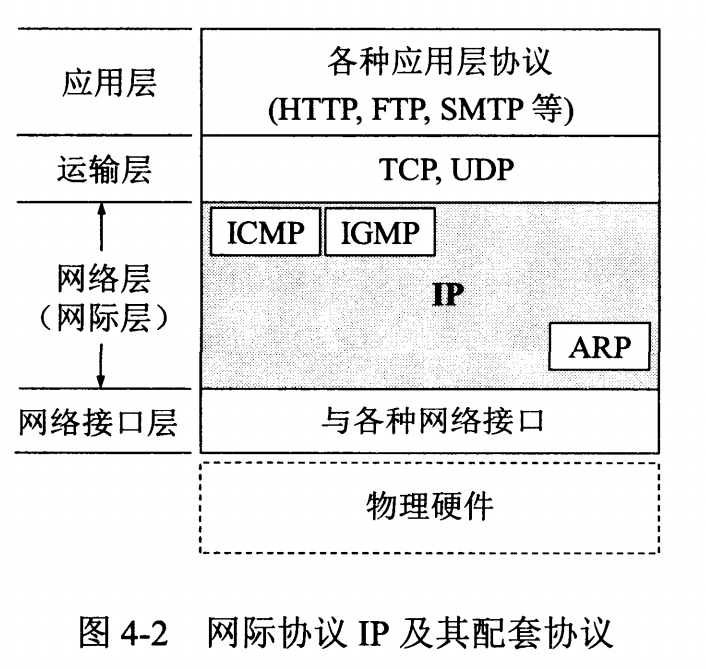


## 网络层

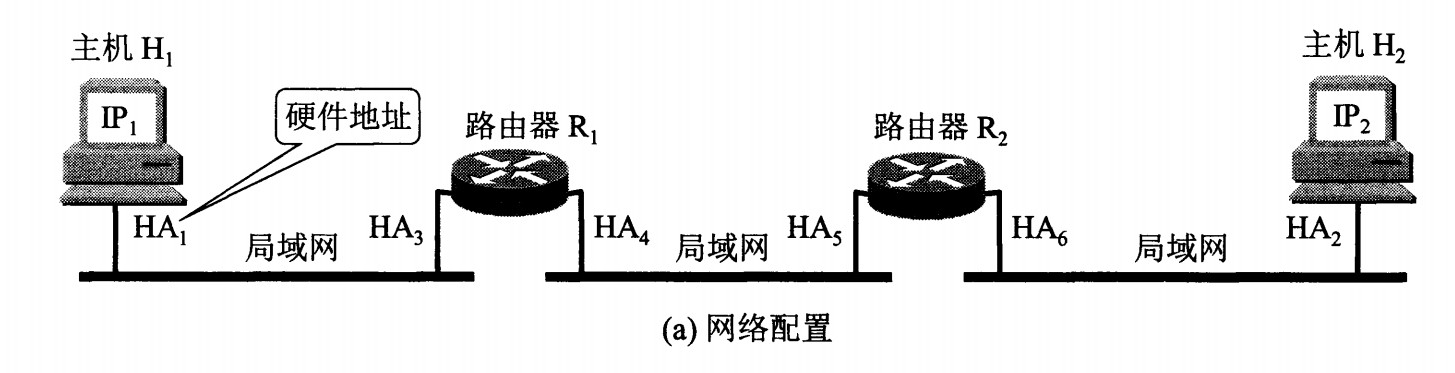
因为网络层是整个互联网的核心，因此应当让网络层尽可能简单。网络层向上只提供简单灵活的、无连接的、尽最大努力交互的数据报服务。使用 IP 协议，可以把异构的物理网络连接起来，使得在网络层看起来好像是一个统一的网络。

与IP协议配套使用的还有三个协议：

1. 地址解析协议ARP（Address Resolution Protocol）；
2. 网际控制报文协议 ICMP（Internet Control Message Protocol）；
3. 网际组管理协议 IGMP（Internet Group Management Protocol）。



网络层实现主机之间的通信，而链路层实现具体每段链路之间的通信。因此在通信过程中，IP 数据报的源地址和目的地址始终不变，而 MAC 地址随着链路的改变而改变。



### ARP Address Resolution Protocol

每个主机都有一个 ARP 高速缓存，里面有**本局域网上的各主机和路由器的 IP 地址到硬件地址的映射表**。

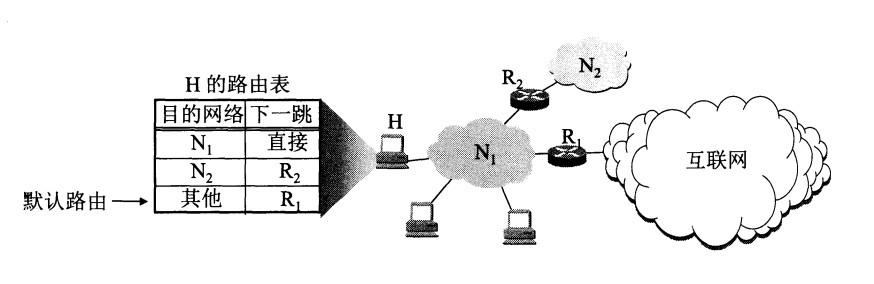
如果主机 A 知道主机 B 的 IP 地址，但是 ARP 高速缓存中没有该 IP 地址到 MAC 地址的映射，此时主机 A 通过广播的方式发送 ARP 请求分组，主机 B 收到该请求后会发送 ARP 响应分组给主机 A 告知其 MAC 地址，随后主机 A 向其高速缓存中写入主机 B 的 IP 地址到硬件地址的映射。

### 路由器

路由器主要功能：路由选择和分组转发。

#### 分组转发

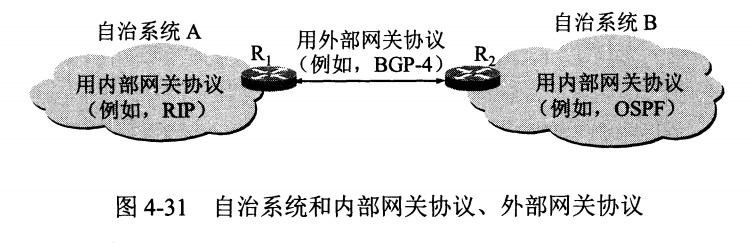
1. 从数据报的首部提取目的主机的 IP 地址 D，得到目的网络地址 N。
2. 若 N 就是与此路由器直接相连的某个网络地址，则进行直接交付；
3. 若路由表中有目的地址为 D 的特定主机路由，则把数据报传送给表中所指明的下一跳路由器；
4. 若路由表中有到达网络 N 的路由，则把数据报传送给路由表中所指明的下一跳路由器；
5. 若路由表中有一个默认路由，则把数据报传送给路由表中所指明的默认路由器；
6. 报告转发分组出错。



#### 路由选择

可以把路由选择协议划分为两大类：

1. 内部网关协议 IGP（Interior Gateway Protocol）：在 AS 内部使用，如 **RIP** 和 **OSPF**。
2. 外部网关协议 EGP（External Gateway Protocol）：在 AS 之间使用，如 **BGP**。



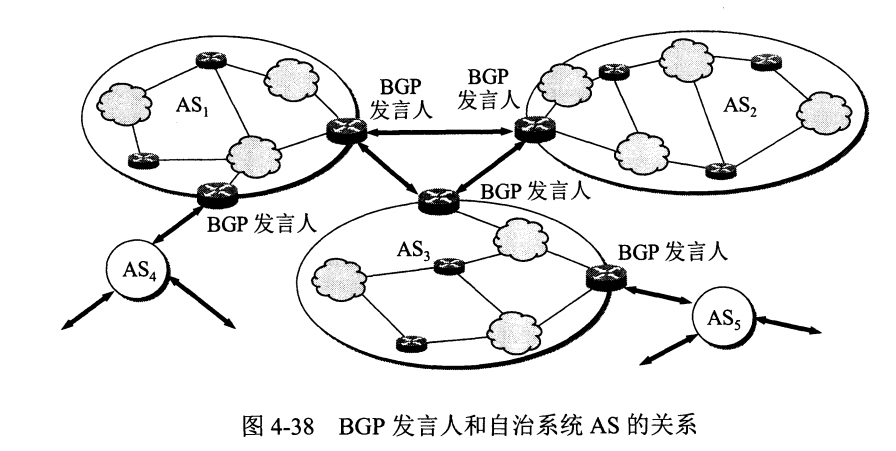
RIP：距离向量的路由选择协议，距离是指跳数，直接相连的路由器跳数为 1，跳数最多为 15，超过 15 表示不可达。

OSPF：是为了克服 RIP 的缺点而开发出来的，开放最短路径优先，最短路径优先表示使用了 Dijkstra 提出的最短路径算法 SPF。

BGP：AS 之间的路由选择很困难，主要是互联网规模很大。并且各个 AS 内部使用不同的路由选择协议，就无法准确定义路径的度量。并且 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。

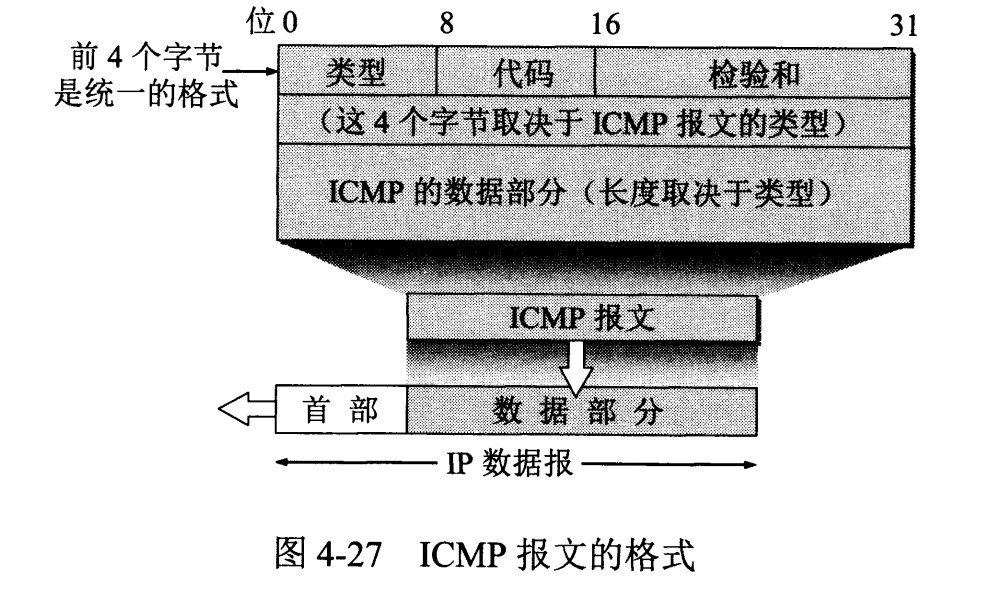
BGP 只能寻找一条比较好的路由，而不是最佳路由。它采用路径向量路由选择协议。

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。

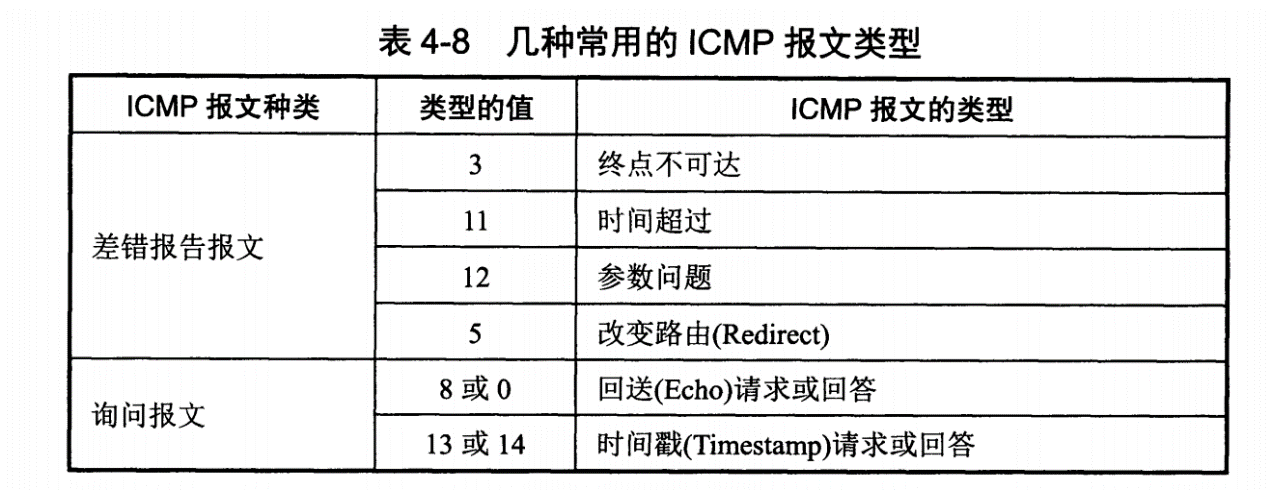


### ICMP Internet Control Message Protocol

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。



ICMP 报文分为**差错报告报文**和**询问报文**。



因为ICMP有这些类型，所以ICMP主要的应用：Ping和Traceroute。

Ping 发送的 IP 数据报封装的是无法交付的 UDP 用户数据报。

Traceroute用来跟踪一个分组从源点到终点的路径。

### 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 转换表把运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。

## 传输层

网络层只把分组发送到目的主机，但是真正通信的并不是主机而是主机中的进程。传输层提供了进程间的逻辑通信，传输层向高层用户屏蔽了下面网络层的核心细节，使应用程序看见的好像在两个运输层实体之间有一条端到端的逻辑通信信道。

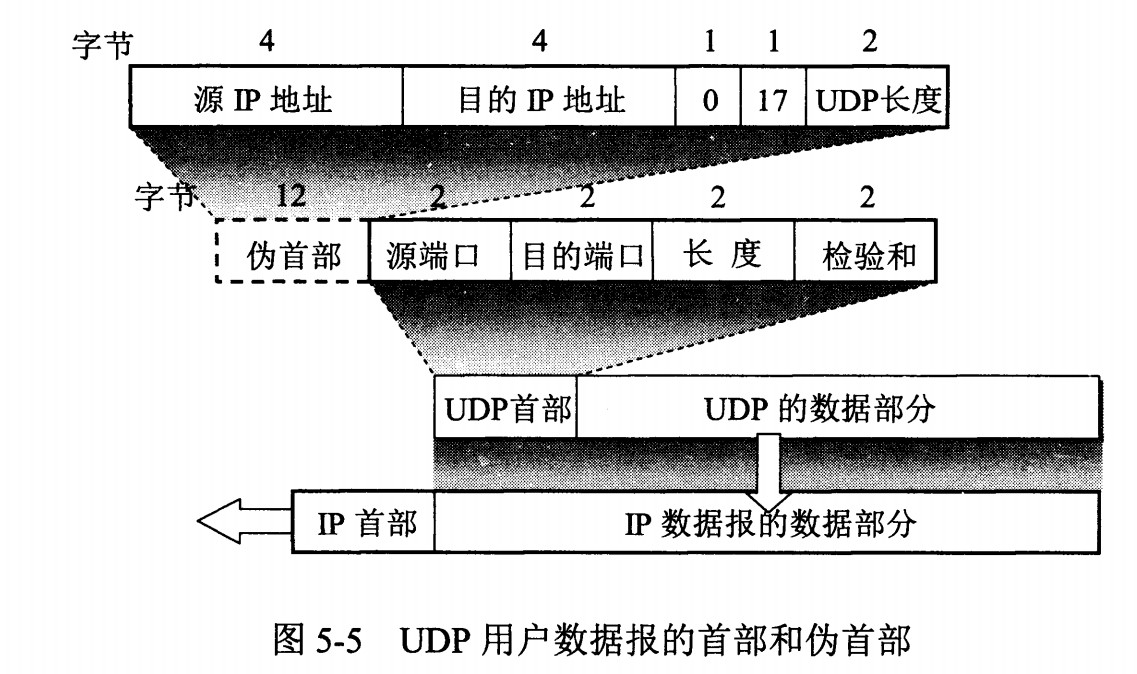
### UDP和TCP

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**）。

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**）。

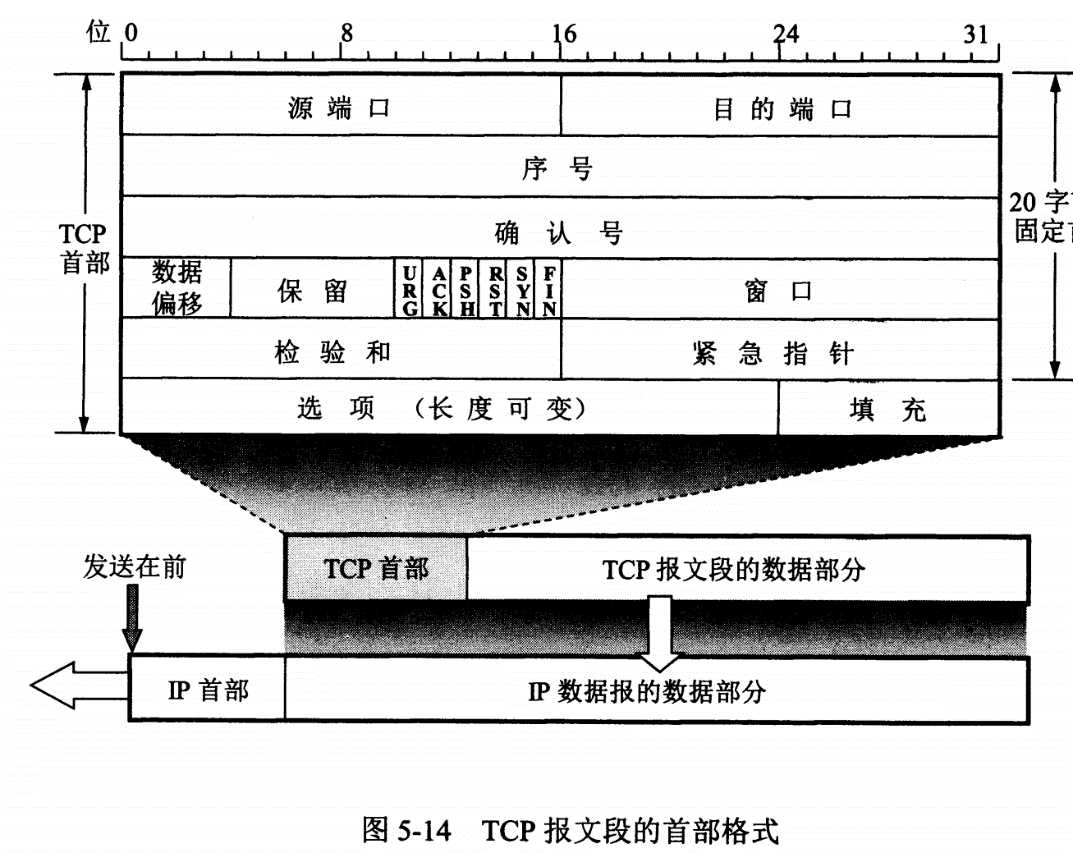
### UDP

UDP的首部字段只有8字节，其中包括源端口、目的端口、长度和校验和。首部之前还有12个字节的伪首部是为了计算校验和临时添加的。



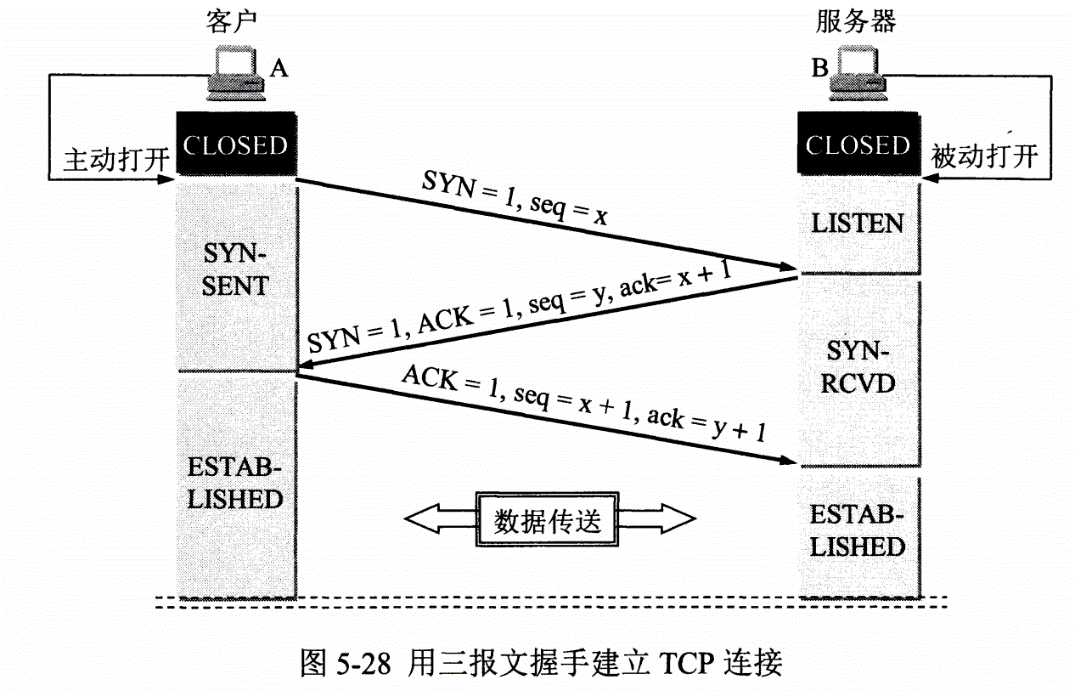
### TCP

TCP是面向连接的，所以TCP的首部比较复杂，比如有序号、确认号、一些标志位等。



* 序号 ：用于**对字节流进行编号**，例如序号为 301，表示第一个字节的编号为 301，如果携带的数据长度为 100 字节，那么下一个报文段的序号应为 401。
* 确认号 ：期望收到的下一个报文段的序号。例如 **B 正确收到** **A 发送来的一个报文段**，序号为 501，携带的数据长度为 200 字节，因此 B 期望下一个报文段的序号为 701，**B 发送给 A 的确认报文段中确认号就为 701**。
* 数据偏移 ：指的是数据部分距离报文段起始处的偏移量，实际上指的是首部的长度。
* 确认 ACK ：当 ACK=1 时确认号字段有效，否则无效。TCP 规定，在连接建立后所有传送的报文段都必须把 ACK 置 1。
* 同步 SYN ：在连接建立时用来同步序号。当 SYN=1，ACK=0 时表示这是一个连接请求报文段。若对方同意建立连接，则响应报文中 SYN=1，ACK=1。
* 终止 FIN ：用来释放一个连接，当 FIN=1 时，表示此报文段的发送方的数据已发送完毕，并要求释放运输连接。
* 窗口 ：窗口值作为接收方让发送方设置其发送窗口的依据。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。

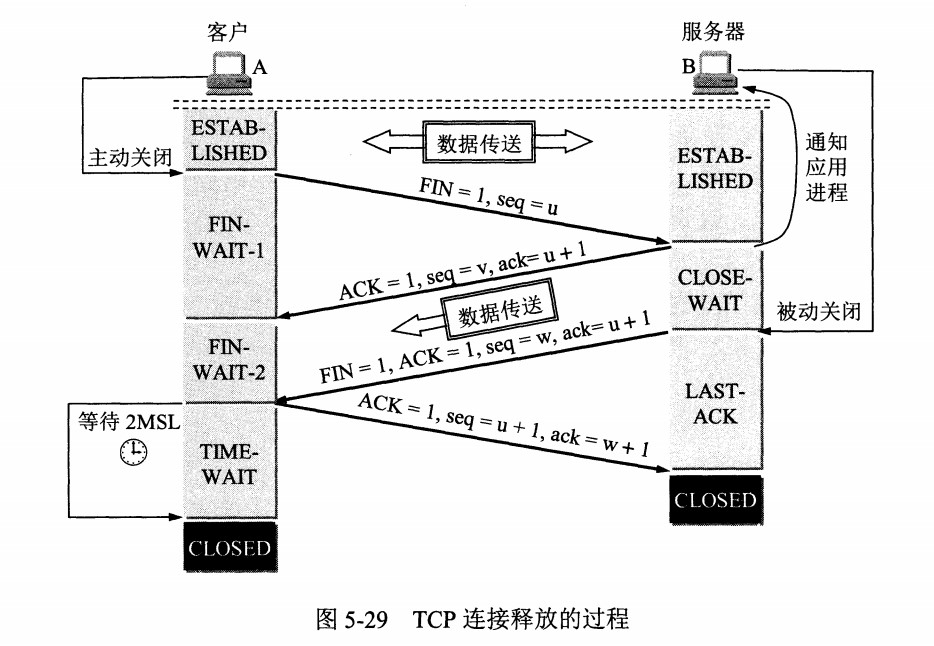
#### 三次握手



1. 服务器B处于listen状态，等待客户端A的连接；
2. A向B发起连接请求，SYN=1，ACK=0，选择一个数x；
3. B收到A发起的连接请求，如果同意建立连接，向A发送连接确认报文段，此时SYN=1，ACK=1，确认号=x+1，同时选择一个数y；
4. A收到B的连接确认报文段后，向B发送确认，确认号为y+1，序号是x+1；
5. B收到A的确认报文段后，连接建立。

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。**失效的连接请求是指，客户端发送的连接请求在网络中滞留，**客户端因为没及时收到服务器端发送的连接确认，因此就重新发送了连接请求。**滞留的连接请求并不是丢失，之后还是会到达服务器。如果不进行第三次握手，那么服务器会误认为客户端重新请求连接，然后打开了连接。但是并不是客户端真正打开这个连接，因此客户端不会给服务器发送数据，这个连接就白白浪费了。

#### 四次挥手



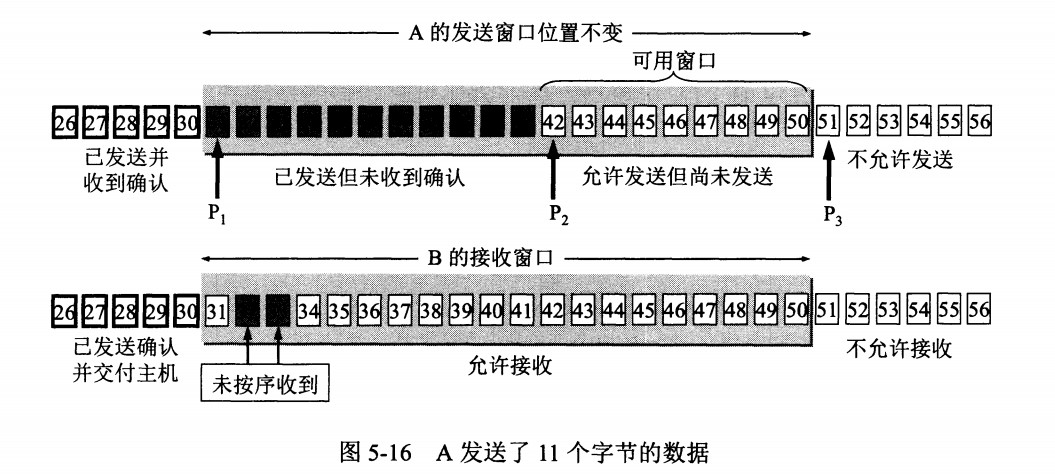
1. A发送连接释放报文段，FIN=1；
2. B收到之后发出确认，此时TCP属于**半关闭状态**，B能向A发送当时A不能像B发送；
3. 当B不再需要连接时，发送连接释放请求报文段给A，FIN=1；
4. A收到后先想B发出确认，进入TIME-WAIT状态，等待2ms后释放连接；
5. B收到A的确认后释放连接。

客户端发送了 FIN 连接释放报文之后，服务器收到了这个报文，就进入了 CLOSE-WAIT 状态。这个状态是为了**让服务器端发送还未传送完毕的数据**，传送完毕之后，服务器会发送 FIN 连接释放报文。

客户端接收到服务器端的 FIN 报文后进入TIME\_WAIT状态，此时并不是直接进入 CLOSED 状态，还需要等待一个时间计时器设置的时间 2MSL。这么做有两个理由：

1. 确保最后一个确认报文段能够到达。如果 B 没收到 A 发送来的确认报文段，那么就会重新发送连接释放请求报文段，A 等待一段时间就是为了处理这种情况的发生。
2. 等待一段时间是为了让本连接持续时间内所产生的所有报文段都从网络中消失，使得下一个新的连接不会出现旧的连接请求报文段。

### 滑动窗口



窗口是缓存的一部分，用来暂时存放字节流。发送方和接收方各有一个窗口，**接收方通过 TCP 报文段中的窗口字段告诉发送方自己的窗口大小**（说明是在三次握手的第二步，也就是服务器返回确认消息给客户端的时候告诉客户端滑动窗口的大小），发送方根据这个值和其它信息设置自己的窗口大小。

发送窗口左部的字节已经发送并且收到了确认，那么就将发送窗口向右滑动一定距离，**直到左部第一个字节不是已发送并且已确认的状态**；接收窗口的滑动类似，**接收窗口左部字节已经发送确认并交付主机**，就向右滑动接收窗口。

接收窗口只会对窗口内最后一个按序到达的字节进行确认，例如接收窗口已经收到的字节为 {31, 32, 34, 35}，其中 {31, 32} 按序到达，而 {34, 35} 就不是，因此只对字节 32 进行确认。发送方得到一个字节的确认之后，就知道这个字节之前的所有字节都已经被接收。

### TCP可靠传输

TCP 使用超时重传来实现可靠传输：如果一个已经发送的报文段在**超时时间**内没有收到确认，那么就重传这个报文段。

这个超时时间通过一定的方法计算得到，一个报文段从**发送再到接收到确认**所经过的时间称为往返时间 RTT，加权平均往返时间：

https://camo.githubusercontent.com/02e9a4a23b05bcf814d9e5d0c1b79a6bb5ae2aee/68747470733a2f2f6c617465782e636f6465636f67732e636f6d2f6769662e6c617465783f525454733d28312d61292a2852545473292b612a525454

超时时间 RTO 应该略大于 RTTs，TCP 使用的超时时间计算如下：

https://camo.githubusercontent.com/da3da0a6cad7ef2af7d7adcd2940277780c35325/68747470733a2f2f6c617465782e636f6465636f67732e636f6d2f6769662e6c617465783f52544f3d525454732b342a5254545f64

### TCP流量控制

流量控制是为了控制发送方发送速率，保证接收方来得及接收。

接收方发送的确认报文中的窗口字段可以用来控制发送方窗口大小，从而影响发送方的发送速率。将窗口字段设置为 0，则发送方不能发送数据。

### TCP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受，而拥塞控制是为了降低整个网络的拥塞程度。

TCP 主要通过四种算法来进行拥塞控制：慢开始、拥塞避免、快重传、快恢复。

* 慢开始和拥塞避免

发送的最初执行慢开始，令 cwnd=1，发送方只能发送 1 个报文段；当收到确认后，将cwnd 加倍，因此之后发送方能够发送的报文段数量为：2、4、8 ...

注意到慢开始每个轮次都将 cwnd 加倍，这样会让 cwnd 增长速度非常快，从而使得发送方发送的速度增长速度过快，网络拥塞的可能也就更高。设置一个慢开始门限 ssthresh，当 cwnd >= ssthresh 时，进入**拥塞避免**，每个轮次只将 cwnd 加 1。

如果出现了超时，则令 ssthresh = cwnd/2，然后重新执行慢开始。

* 快重传与快恢复

在接收方，要求每次接收到报文段都应该**发送对已收到有序报文段的确认**，例如已经接收到 M1 和 M2，此时收到 M4，应当发送对 M2 的确认。

在发送方，如果收到三个重复确认，那么可以确认下一个报文段丢失，例如收到三个 M2 ，则 M3 丢失。此时执行**快重传**，立即重传下一个报文段。

## 应用层