互联网交换点 IXP（Internet eXchange Point）的主要作用就是允许两个网络直接相连并交换分组，而不需要再通过第三个网络来转发分组。

计算机之间通信：主机 A 的某个**进程**和主机 B 上的另一个**进程**进行通信。

对等连接（peer-to-peer，P2P）是指两台主机在通信时并**不区分哪一个是服务请求方哪一个是服务提供方**。对等连接方式从本质上看仍然是使用客户端-服务器方式，只是**对等连接中的每一台主机既是客户又同时是服务器**。

互联网按工作方式可分为**边缘部分**与**核心部分**。**主机在网络的边缘部分**，其作用是进行信息管理。**路由器在网络的核心部分**，其作用是按存储转发方式进行**分组交换**。

分组交换采用**存储转发**技术。通常我们把要发送的整块数据称为一个**报文**。在发送报文之前，先把较长的报文划分成为一个个更小的等长数据段。在每一个数据段前面，加上一些由必要的控制信息组成的**首部**后，就构成了一个**分组**。分组又称为“**包**”，而分组的首部也可称为“**包头**”。分组是在互联网中传送的数据单元。分组中的首部是非常重要的，正是由于分组的首部包含了诸如目的地址和源地址等重要控制信息，每一个分组才能在互联网中独立地选择传输路径，并被正确地交付到分组传输的终点。

在互联网核心部分的路由器之间一般都用高速链路相连接，而在网络边缘的主机接入到核心部分则通常以相对较低速率的链路相连接。

路由器收到一个分组，先暂时存储一下，检查其首部，查找转发表，按照首部中的目的地址，找到合适的接口转发出去，把分组交给下一个路由器。这样一步一步以存储转发的方式，把分组交付最终的目的主机。**各路由器之间必须经常交换彼此掌握的路由信息，以便创建和动态维护路由器中的转发表，使得转发表能够在整个网络拓扑发生变化时及时更新**。

路由器暂时存储的是一个个**短分组**，而不是整个长报文。短分组是暂时存在路由器的存储器（**内存**）中而不是存储在磁盘中的。这就保证了较高的交换速率。

分组交换**在传送数据之前不必先占用一条端到端的链路的通信资源。分组在哪条链路上传送才占用这段链路的通信资源**。分组到达一个路由器后，先暂时存

储下来，查找转发表，然后从一条合适的链路转发出去。分组在传输时就这样一段一段地断续占用通信资源，而且还**省去了建立连接和释放连接的开销**，因而数据的传输效率更高。

为了提高分组交换网的可靠性，互联网的核心部分常采用**网状拓扑**结构，使得当发生网络拥塞或少数结点、链路出现故障时，路由器可灵活地改变转发路由而不致引起通信的中断或全网的瘫痪。此外，通信网络的主干线路往往由一些**高速链路**构成，这样就可以较高的数据率迅速地传送计算机数据。

分组交换的优点

优点	所采用的手段
高效	在分组传输的过程中动态分配传输带宽，对通信链路是逐段占用
灵活	为每一个分组独立地选择最合适的转发路由
迅速	以分组作为传送单位，可以不先建立连接就能向其他主机发送分组
可靠	保证可靠性的网络协议；分布式多路由的分组交换网，使网络有很好的生存性

分组交换也带来一些新的问题。例如，分组在**各路由器存储转发时需要排队**，这就会造成一定的时延。此外，由于分组交换不像电路交换那样通过建立连接来保证通信时所需的各种资源，因而**无法确保通信时端到端所需的带宽**。分组交换带来的另一个问题是**各分组必须携带的控制信息也造成了一定的开销**。整个分组交换网还需要**专门的管理和控制机制**。

电路交换：整个报文的比特流连续地从源点直达终点，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（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。

若需要连续传送大量的数据，且其传送时间远大于连接建立时间，则电路交换的传输速率较快。报文交换和分组交换不需要预先分配传输带宽，在传送突发数据时可提高整个网络的信道利用率。**由于一个分组的长度往往远小于整个报文的长度，因此分组交换比报文交换的时延小，同时也具有更好的灵活性**。

带宽表示在单位时间内网络中的某信道所能通过的“**最高数据率**”，单位是“比特每秒”。**吞吐量**表示在单位时间内通过某个网络（或信道、接口）的实际的数据量。

总时延 = **发送时延** + **传播时延** + **处理时延** + **排队时延**

（1）**发送时延**：**主机或路由器发送数据帧所需要的时间**。

发送时延 = 数据帧长度（bit） / 发送速率（bit/s）

（2）**传播时延**：**电磁波在信道中传播一定的距离需要花费的时间**。

传播时延 = 信道长度（m） / 电磁波在信道上的传播速率（m/s）

发送时延发生在机器内部的发送器中，与**传输信道的长度没有任何关系**。传播时延发生在机器外部的传输信道媒体上，而**与信号的发送速率无关。信号传送的距离越远，传播时延就越大**。

（3）**处理时延**：**主机或路由器在收到分组时要花费一定的时间进行处理**，例如分析分组的首部、从分组中提取数据部分、进行差错检验或查找适当的路由等，这就产生了处理时延。

（4）**排队时延**：分组在经过网络传输时，要经过许多路由器。但分组在进入路由器后要先在输入队列中排队等待处理。在路由器确定了转发接口后，还要在传输队列中排队等待转发。这就产生了排队时延。**排队时延的长短往往取决于网络当时的通信量**。当网络的通信量很大时会发生队列溢出，使分组丢失，这相当于排队时延为无穷大。

“**在高速链路上，比特会传送得更快些**”，这个说法是不对的。**对于高速网络链路，我们提高的仅仅是数据的发送速率而不是比特在链路上的传播速率**。荷载信息的电磁波在通信线路上的传播速率取决于通信线路的介质材料，而与数据的发送速率无关。**提高数据的发送速率只是减小了数据的发送时延**。数据的发送速率的单位是每秒发送多少个比特，这是指在某个点或某个接口上的发送速率。而传播速率的单位是每秒传播多少公里，是指在某一段传输线路上比特的传播速率。**通常所说的“光纤信道的传输速率高”是指可以用很高的速率向光纤信道发送数据，而光纤信道的传播速率实际上还要比铜线的传播速率略低一些**。

传播时延带宽积 = 传播时延×带宽

假设某段链路的传播时延为 20ms，带宽为 10Mbit/s，算出时延带宽积为 200000bit。这就表明，若发送端连续发送数据，则在发送的第一个比特即将达到

终点时，发送端就已经发送了 20 万个比特，而这 20 万个比特都正在链路上向前移动。**链路的传播时延带宽积又称为以比特为单位的链路长度**。

利用率有**信道利用率**和**网络利用率**两种。**信道利用率**指出某信道有百分之几的时间是有数据通过的。完全空闲的信道利用率是零。**网络利用率**则是全网络的信道利用率的加权平均值。信道利用率过高会产生非常大的时延。

应用层：是体系结构中的最高层。其任务是**通过应用进程间的交互来完成特定网络应用**。应用层协议定义的是**应用进程间通信和交互的规则**。这里的进程就是指主机中正在运行的程序。**对于不同的网络应用需要有不同的应用层协议**。我们把应用层交互的数据单元称为**报文**。

运输层：任务是负责**向两台主机中进程之间的通信提供通用的数据传输服务**。应用进程利用该服务传送应用层报文。所谓“通用的”，是指并不针对某个特定网络应用，而是**多种应用可以使用同一个运输层服务**。由于一台主机可同时运行多个进程，因此运输层有复用和分用的功能。**复用就是多个应用层进程可同时使用下面运输层的服务，分用和复用相反，是运输层把收到的信息分别交付上面应用层中的相应进程**。

运输层主要使用以下两种协议：

（1）传输控制协议 TCP——提供**面向连接的**、**可靠的**数据传输服务，其数据传输的单位是**报文段**。

（2）用户数据报协议 UDP——提供**无连接的**、**尽最大努力交付**的数据传输服务（不保证数据传输的可靠性），其数据传输的单位是**用户数据报**。

网络层：负责**为分组交换网上的不同主机提供通信服务**。在发送数据时，**网络层把运输层产生的报文段或用户数据报封装成分组或包进行传送**。在 TCP/IP 体系中，由于网络层使用 IP 协议，因此分组也叫做 **IP 数据报**。网络层的另一个任务就是**选择合适的路由，使源主机运输层所传下来的分组，能够通过网络中的路由器找到目的主机**。

互联网是由大量的**异构网络**通过**路由器**相互连接起来的。互联网使用的网络层协议是**无连接的**网际协议 IP 和许多种路由选择协议。

数据链路层：简称为链路层。两台主机之间的数据传输，总是在一段一段的链路上传送的，这就需要使用专门的链路层的协议。在两个相邻结点之间传送数据时，**数据链路层将网络层交下来的 IP 数据报组装成帧，在两个相邻结点间的链路上传送帧**。每一帧包括**数据**和**必要的控制信息**（如**同步信息**、**地址信息**、**差**

错控制等）。

在接收数据时，控制信息使接收端能够知道一个帧从哪个比特开始和到哪个比特结束。这样，**数据链路层在收到一个帧后，就可从中提取出数据部分，上交给网络层**。

控制信息还使接收端能够**检测到所收到的帧中有无差错**。如**发现有差错，数据链路层就简单地丢弃这个出了差错的帧，以免继续在网络中传送下去白白浪费网络资源**。**如果需要改正数据在数据链路层传输时出现的差错，那么就要采用可靠传输协议来纠正出现的差错**。

物理层：在物理层所传数据的单位是**比特**。发送方发送 1 时，接收方应当收到 1 而不是 0。因此物理层要**考虑用多大的电压代表“1”或“0”，以及接收方如何识别出发送方所发送的比特**。物理层还要确定连接电缆的插头应当有多少根引脚以及各引脚应如何连接。解释比特代表的意思，不是物理层的任务。传递信息所利用的一些物理媒体，如双绞线等，并不在物理层协议之内而是在物理层协议的下面。因此也有人把物理层下面的物理媒体当作第 0 层。

协议数据单元 PDU（Protocol Data Unit）：对等层之间传送的数据单位。

当这一串的比特流离开主机 1 经网络的物理媒体传送到路由器时，就**从路由器的第 1 层依次上升到第 3 层。每一层都根据控制信息进行必要的操作，然后将控制信息剥去，将该层剩下的数据单元上交给更高的一层。当分组上升到了第 3 层时，就根据首部中的目的地址查找路由器中的转发表，找出转发分组的接口，然后往下传送到第 2 层，加上新的首部和尾部后，再到最下面的第 1 层，然后在物理媒体上把每一个比特发送出去**。

实体：任何可发送或接收信息的硬件或软件**进程**。**协议**：控制两个对等实体进行通信的**规则的集合**。

在协议的控制下，两个对等实体间的通信使得本层能够向上一层提供服务。要实现本层协议，还**需要使用下面一层所提供的服务**。

协议和服务在概念上是很不一样的。

首先，协议的实现保证了能向上一层提供服务。**使用本层服务的实体只能看见服务而无法看见下面的协议**。下面的协议对上面的实体是透明的。

其次，**协议是“水平的”**，即**协议是控制对等实体之间通信的规则**。但**服务是“垂直的”**，即**服务是由下层向上层通过层间接口提供的**。另外，并非在一个层

内完成的全部功能都称为服务。**只有那些能够被高一层实体“看得见”的功能才能称之为“服务”**。上层使用下层所提供的服务必须通过与下层交换一些命令，这些命令在 OSI 中称为**服务原语**。

在同一系统中相邻两层的实体进行交互的地方，通常称为**服务访问点 SAP**（Service Access Point）。服务访问点 SAP 是一个抽象的概念，它实际上就是一个**逻辑接口**，但这种层间接口和两个设备之间的硬件接口并不一样。OSI 把**层与层之间交换的数据的单位**称为**服务数据单元 SDU**（Service Data Unit），可以是多个 SDU 合成为一个 PDU，也可以是一个 SDU 划分为几个 PDU。

TCP/IP 协议可以**为各式各样的应用提供服务**，同时 TCP/IP 协议也允许 IP 协议**在各式各样的网络构成的互联网上运行**。

单向通信（单工通信）：只能有一个方向的通信而没有反方向的交互。
双向交替通信（半双工通信）：通信双方都可以发送消息，但不能同时发送。
双向同时通信（全双工通信）：通信的双方可以同时发送和接收信息。

数据链路层使用的信道主要有以下两种类型：

（1）**点对点信道**。这种信道使用一对一的点对点通信方式。

（2）**广播信道**。这种信道使用一对多的广播通信方式。广播信道上连接的主机很多，因此必须使用**专用的共享信道协议**来协调这些主机的数据发送。

数据链路层的三个基本问题：**封装成帧**、**透明传输**和**差错检测**。

链路就是从一个结点到相邻结点的一段物理线路（有线或无线），而中间没有任何其他的交换结点。

数据链路是另一个概念。当需要在一条线路上传送数据时，除了必须有一条物理线路外，还必须有一些必要的通信协议来控制这些数据的传输。若把实现这些协议的**硬件和软件**加到链路上，就构成了数据链路。现在最常用的方法是使用网络适配器（既有硬件，也包括软件）来实现这些协议。一般的适配器都包括了数据链路层和物理层这两层的功能。

点对点信道的数据链路层通信时的步骤：

（1）结点 A 的数据链路层把网络层交下来的 IP 数据报添加**首部**和**尾部**封装成帧。

（2）结点 A 把封装好的帧发送给结点 B 的数据链路层。

（3）若结点 B 的数据链路层收到的帧无差错，则从收到的帧中提取出 IP 数据报交给上面的网络；否则丢弃这个帧。

封装成帧就是在一段数据的前后分别添加**首部**和**尾部**，这样就构成了一个帧。一个帧的帧长等于**帧的数据部分长度**加上**帧首部**和**帧尾部**的长度。首部和尾部的一个重要作用就是进行**帧定界**。发送帧时，是从帧首部开始发送的。**为了提高帧的传输效率，应当使帧的数据部分长度尽可能地大于首部和尾部的长度**。但是，每一种链路层协议都规定了所能传送的帧的数据部分长度上限——**最大传送单元 MTU**（Maximum Transfer Unit）。

当数据是由可打印的 ASCII 码组成的文本文件时，帧定界可以使用特殊的帧定界符。**控制字符 SOH（Start Of Header）放在一帧的最前面，表示帧的首部开始**。另一个**控制字符 EOT（End Of Transmission）表示帧的结束**。

当数据在传输中出现差错时，帧定界符的作用更加明显。假定发送端在尚未发送完一个帧时突然出故障，中断了发送。但随后很快又恢复正常，于是重新从头开始发送刚才未发送完的帧。由于使用了帧定界符，接收端就知道前面收到的数据是个不完整的帧（只有首部开始符 SOH 而没有传输结束符 EOT），必须丢弃。而后面收到的数据有明确的帧定界符（SOH 和 EOT），因此这是一个完整的帧，应当收下。

由于帧的开始和结束的标记使用专门指明的控制字符，因此，所传输的数据中的任何 8 比特的组合一定不允许和用作帧定界的控制字符的比特编码一样，否则就会出现帧定界的错误。

当传送的帧是用文本文件组成的帧时（文本文件中的字符都是从键盘上输入的），其数据部分显然不会出现像 SOH 或 EOT 这样的帧定界控制字符。可见**不管从键盘上输入什么字符都可以放在这样的帧中传输过去**，因此这样的传输就是**透明传输**。

“在数据链路层透明传送数据”表示**无论什么样的比特组合的数据，都能够按照原样没有差错地通过这个数据链路层**。因此，**对所传送的数据来说，这些数据就“看不见”数据链路层有什么妨碍数据传输的东西**。或者说，**数据链路层对**

这些数据来说是透明的。

为了解决透明传输问题，就必须设法使数据中可能出现的控制字符“SOH”和“EOT”在接收端不被解释为控制字符。具体的方法是：发送端的数据链路层**在数据中出现控制字符“SOH”或“EOT”的前面插入一个转义字符“ESC”**。而在接收端的数据链路层在把数据送往网络层之前删除这个插入的转义字符。

比特在传输过程中可能会出现差错：1 可能会变成 0，而 0 也可能变成 1。这就叫做**比特差错**。在一段时间内，传输错误的比特占所传输比特总数的比率称为**误码率 BER**（Bit Error Rate）。

循环冗余检验 CRC（Cyclic Redundancy Check）是一种检错方法，而为了进行检测而添加的冗余码叫做**帧检验序列 FCS**（Frame Check Sequence）。在数据链路层，发送端帧检验序列 FCS 的生成和接收端的 CRC 检验都是用**硬件**完成的，处理很迅速，因此并不会延误数据的传输。

如果我们在传送数据时不以帧为单位来传送，那么就无法加入冗余码以进行差错检验。因此，**如果要在数据链路层进行差错检验，就必须把数据划分为帧，每一帧都加上冗余码，一帧接一帧地传送，然后在接收方逐帧进行差错检验。凡是接收端数据链路层接受的帧，我们都能以非常接近于 1 的概率认为这些帧在传输过程中没有产生差错**。

“**无比特差错**”和“**无传输差错**”并不是同样的概念。“无传输差错”还必须避免**帧丢失**、**帧重复**和**帧失序**。在数据链路层使用 CRC 检验，能够实现无比特差错的传输，但这还不是可靠传输。

对于通信质量良好的**有线**传输链路，数据链路层协议不使用确认和重传机制，即不要求数据链路层向上提供可靠传输的服务。如果在数据链路层传输数据时出现了差错并且需要进行改正，那么改正差错的任务就由上层协议（运输层的 TCP 协议）来完成。

对于通信质量较差的**无线**传输链路，数据链路层协议使用**确认**和**重传**机制，数据链路层向上提供可靠传输的服务。

计算机与外界局域网的连接是通过**通信适配器**进行的。适配器本来是在主机

箱内插入的一块**网络接口板**。这种接口板又称为**网络接口卡 NIC**（Network Interface Card）或简称为 “**网卡**”。在这种通信适配器上面装有**处理器**和**存储器**（包括 RAM 和 ROM）。适配器和局域网之间的通信是通过电缆或双绞线以**串行**传输方式进行的，而适配器和计算机之间的通信则是通过计算机主板上的 I/O 总线以**并行**传输方式进行的。因此，**适配器的一个重要功能就是要进行数据串行传输和并行传输的转换**。由于网络上的数据率和计算机总线上的数据率并不相同，因此在适配器中必须装有**对数据进行缓存的存储芯片**。在主板上插入适配器时，还必须把**管理该适配器的设备驱动程序**安装在计算机的操作系统中。这个驱动程序以后就会告诉适配器，应当从存储器的什么位置上把多长的数据块发送到局域网，或者应当在存储器的什么位置上把局域网传送过来的数据块存储下来。适配器还要能够实现以太网协议。

适配器所实现的功能包含了**数据链路层和物理层**这两个层次的功能。**适配器在接收和发送各种帧时，不使用计算机的 CPU。这时计算机中的 CPU 可以处理其他任务**。当适配器收到有差错的帧时，就把这个帧直接丢弃而不必通知计算机。当适配器收到正确的帧时，它就使用**中断**来通知该计算机，并交付协议栈中的网络层。当计算机要发送 IP 数据报时，就由协议栈把 IP 数据报向下交给适配器，组装成帧后发送到局域网。**计算机的硬件地址在适配器的 ROM 中，而计算机的软件地址——IP 地址，在计算机的存储器中**。

总线通信的特点是：当一台计算机发送数据时，总线上的所有计算机都能检测到这个数据。这种就是**广播通信**方式。为了在总线上实现一对一的通信，可以**使每一台计算机的适配器拥有一个与其他适配器都不同的地址**。在发送数据帧时，在帧的首部写明接收站的地址。仅当数据帧中的目的地址与适配器 ROM 中存放的硬件地址一致时，该适配器才能接收这个数据帧。适配器对不是发送给自己的数据帧就丢弃。这样，具有广播特性的总线上就实现了一对一的通信。

为了通信的简便，以太网采取了以下两种措施：

第一，采用较为灵活地**无连接**的工作方式，即不必先建立连接就可以直接发送数据。适配器对发送的数据帧不进行编号，也不要求对方发回确认。这样做可以使以太网工作起来非常简单，而局域网信道的质量很好，因通信质量不好产生差错的概率是很小的。以太网提供的服务是**尽最大努力的交付**，即**不可靠的交付**。当目的站收到有差错的数据帧时，就把帧丢弃，其他什么也不做。**对有差错的帧是否需要重传则由高层来决定**。如果高层使用 TCP 协议，那么 TCP 就会发现丢

失了一些数据。于是经过一定的时间后，TCP 就把这些数据重新传递给以太网进行重传。但以太网并不知道这是重传帧，而是当作新的数据帧来发送。

第二，以太网发送的数据都使用**曼彻斯特编码**的信号。

强化碰撞：当发送数据的站一旦发现发生了碰撞时，除了立即停止发送数据外，还要再继续发送 32 比特或 48 比特的**人为干扰信号**，以便让所有用户都知道现在已经发生了碰撞。

载波监听多点接入/碰撞检测 CSMA/CD（Carrier Sense Multiple Access With Collision Detection）的要点：

（1）**准备发送**：适配器从网络层获得一个分组，加上以太网的**首部**和**尾部**，组成以**太网帧**，放入适配器的缓存中。但在发送之前，必须先检测信道。

（2）**检测信道**：若检测到信道忙，则应不停地检测，一直等待信道转为空闲。若检测到信道空闲，并在 96 比特时间内信道保持空闲（保证了帧间最小间隔），就发送这个帧。

（3）在发送过程中仍不停地检测信道，即**网络适配器要边发送边监听**。这里只有两种可能性：

①**发送成功**：在争用期内一直未检测到碰撞。这个帧肯定能够发送成功。发送完毕后，其他什么也不做。然后回到（1）。

②**发送失败**：在争用期内检测到碰撞。这时立即停止发送数据，并按规定发送人为干扰信号。适配器接着就执行指数退避算法，等待 r 倍 512 比特时间后，返回到步骤（2），继续检测信道。但若重传达 16 次仍不能成功，则停止重传而向上报错。

以太网每发送完一帧，一定要把已发送的帧暂时保留一下。**如果在争用期内检测出发生了碰撞，那么还要在推迟一段时间后再把这个暂时保留的帧重传一次**。

在局域网中，**硬件地址**又称为**物理地址**或 **MAC 地址**（因为这种地址用在 **MAC 帧**中）。IEEE 802 标准为局域网规定了一种 48 位的全球地址，是指局域网上的每一台计算机中**固化在适配器的 ROM 中的地址**。

当路由器通过适配器连接到局域网时，适配器上的硬件地址就用来标志路由器的某个接口。**路由器如果同时连接到两个网络上，那么它就需要两个适配器和两个硬件地址**。

网络层向上只提供简单灵活地、**无连接**的、**尽最大努力交付**的数据报服务。

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

从一般的概念来讲，将网络互相连接起来要使用一些中间设备。根据中间设备所在的层次，可以有以下四种不同的中间设备：

（1）物理层使用的中间设备有**转发器**、**集线器**。

（2）数据链路层使用的中间设备有**网桥**、**桥接器**、**交换机**。

（3）网络层使用的中间设备叫做**路由器**。

（4）在网络层以上使用的中间设备叫做**网关**。用网关连接两个不兼容的系统需要在高层进行协议的转换。

整个互联网就是一个单一的、抽象的网络。IP 地址就是给互联网上的每一台主机(或路由器)的每一个接口分配一个在全世界范围是唯一的 **32 位**的标识符。

所谓“分类的 IP 地址”就是将 IP 地址划分为若干个固定类，每一类地址都由两个固定长度的字段组成，其中第一个字段是**网络号**，它标志主机(或路由器)所连接到的网络。**一个网络号在整个互联网范围内必须是唯一的**。第二个字段是**主机号**，它标志该主机（或路由器）。**一台主机号在它签名的网络号所指明的网络范围内必须是唯一的**。由此可见，**一个 IP 地址在整个互联网范围内是唯一的**。

A 类、B 类和 C 类地址都是**单播**地址（一对一通信）。

A 类、B 类和 C 类地址的网络号字段分别为 1 个、2 个和 3 个字节长，而在网络号字段的最前面有 1~3 位的类别位，其数值分别规定为 0，10 和 110。

A 类、B 类和 C 类地址的主机号字段分别为 3 个、2 个和 1 个字节长。

D 类地址（前 4 位是 1110）用于**多播**（一对多通信）。

E 类地址（前 4 位是 1111）保留为以后用。

A 类地址的网络号字段占 1 个字节，只有 7 位可供使用（该字段的第一位已固定为 0），但可指派的网络号是 126 个（即 $2^7 - 2$）。减 2 的原因是：第一，**网络号字段为全 0 的 IP 地址是个保留地址，意思是“本网络”**；第二，网络号为 127 保留作为**本地软件环回测试**。若主机发送一个目的地址为环回地址（例如

127.0.0.1）的 IP 数据报，则本主机中的协议软件就处理数据报中的数据，而不会把数据报发送到任何网络。**目的地址为环回地址的 IP 数据报永远不会出现在任何网络上，因为网络号为 127 的地址根本不是一个网络地址**。

A 类地址的主机号占 3 个字节，因此每一个 A 类网络中的最大主机数是 $2^{24}-2$。这里减 2 的原因是：**全 0 的主机号字段表示该 IP 地址是“本主机”所连接到的单个网络地址**，而**全 1 的主机号字段表示该网络上的所有主机**。

IP 地址空间共有 2^{32} 个地址。整个 A 类地址空间共有 2^{31} 个地址，占整个 IP 地址空间的 50%。

B 类地址的网络号字段有 2 个字节，但前面两位（1 0）已经固定了，只剩下 14 位可以进行分配。**B 类网络地址 128.0.0.0 是不指派的**，而可以指派的 B 类最小网络地址是 128.1.0.0。因此 B 类地址可指派的网络数为 $2^{14}-1$。B 类地址的每一个网络上的最大主机数是 $2^{16}-2$。这里需要减 2 是因为要扣除全 0 和全 1 的主机号。整个 B 类地址空间共约有 2^{30} 个地址，占整个 IP 地址空间的 25%。

C 类地址有 3 个字节的网络字段，最前面的 3 位是（1 1 0），还有 21 位可以进行分配。**C 类网络地址 192.0.0.0 是不指派的**，可以指派的 C 类最小的网络地址是 192.0.1.0，因此 C 类地址可指派的网络总数是 $2^{21}-1$。每一个 C 类地址的最大主机数是 $2^{8}-2$。整个 C 类地址空间共约有 2^{29} 个地址，占整个 IP 地址的 12.5%。

IP 地址具有以下一些重要特点：

（1）每一个 IP 地址都由**网络号**和**主机号**两部分组成。从这个意义上来说，IP 地址是一种分等级的地址结构。分两个等级的好处是：第一，**IP 地址管理机构在分配 IP 地址时只分配网络号，而剩下的主机号则由得到该网络号的单位自行分配**。这样就方便了 IP 地址的管理；第二，**路由器仅根据目的主机所连接的网络号来转发分组**，这样就可以使路由表中的项目数大幅度减少，从而减小了路由表所占的存储空间以及查找路由表的时间。

（2）实际上 IP 地址是标志一台主机（或路由器）和一条链路的接口。当一台主机同时连接到两个网络上时，该主机就必须同时具有两个相应的 IP 地址，其网络号必须是不同的。这种主机称为**多归属主机**（multihomed host）。由于一个路由器至少应当连接到两个网络，因此**一个路由器至少应当有两个不同的 IP 地址**。

（3）按照互联网的观点，**一个网络是指具有相同网络号 net-id 的主机的集合，用转发器或网桥连接起来的若干个局域网仍为一个网络，因为这些局域网都**

具有同样的网络号。具有不同的网络号的局域网必须使用路由器进行互连。

（4）在 IP 地址中，所有分配到网络号的网络都是**平等**的。

物理地址是数据链路层和物理层使用的地址，而 IP 地址是网络层和以上各层使用的地址，是一种逻辑地址。

在 IP 层抽象的互联网上只能看到 IP 数据报。路由器只根据目的站的 IP 地址的网络号进行路由选择。在局域网的链路层，只能看见 MAC 帧。

尽管互连在一起的网络的硬件地址体系各不相同，但 **IP 层抽象的互联网却屏蔽了下层这些很复杂的细节。只要我们在网络层上讨论问题，就能够使用统一、抽象的 IP 地址研究主机和主机或路由器之间的通信。**

ARP 协议的用途是为了从网络层使用的 IP 地址，解析出在数据链路层使用的硬件地址。

每一台主机都设有一个 **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，这些都是该主机目前知道的一些地址。那么主机怎样知道这些地址呢？

当主机 A 要向**本局域网**上的某台主机 B 发送 IP 数据报时，就先在其 ARP 高速缓存中查看有无主机 B 的 IP 地址。如有，就在 ARP 高速缓存中查出其对应的硬件地址，再把这个硬件地址写入 MAC 帧，然后通过局域网把该 MAC 帧发往此硬件地址。

也有可能查不到主机 B 的 IP 地址的项目。这可能是主机 B 才入网，也可能是主机 A 刚刚加电，其高速缓存还是空的。在这种情况下，主机 A 就自动运行 ARP，然后按以下步骤找出主机 B 的硬件地址。

（1）**ARP 进程**在**本局域网**上广播发送一个 ARP 请求分组。ARP 请求分组的主要内容是：“我的 IP 地址是 209.0.0.5，硬件地址是 00-00-C0-15-AD-18。我想知道 IP 地址为 209.0.0.6 的主机的硬件地址。”

（2）在本局域网上的所有主机上运行的 ARP 进程都收到此 ARP 请求分组。

（3）主机 B 的 IP 地址与 ARP 请求分组中要查询的 IP 地址一致，就收下这个 ARP 请求分组，并向主机 A 发送 ARP 响应分组，同时在这个 ARP 响应分组中写入自己的硬件地址。由于其余的所有主机的 IP 地址都与 ARP 请求分组中要查询的 IP 地址不一致，因此都不理睬这个 ARP 请求分组。ARP 响应分组的主要内容是：“我的 IP 地址是 209.0.0.6，我的硬件地址是 08-00-2B-00-EE-0A。” 请注

意：**虽然 ARP 请求分组是广播发送的，但 ARP 响应分组是普通的单播，即从一个源地址发送到一个目的地址**。

（4）主机 A 收到主机 B 的 ARP 响应分组后，就在其 ARP 高速缓存中写入主机 B 的 IP 地址到硬件地址的映射。

主机 A 在发送其 ARP 请求分组时，就把自己的 IP 地址到硬件地址的映射写入 ARP 请求分组。**当主机 B 收到 A 的 ARP 请求分组时，就把主机 A 的这一地址映射写入主机 B 自己的 ARP 高速缓存中**。

既然在网络链路上传送的帧最终是按照硬件地址找到目的主机的，那么为什么我们还要使用抽象的 IP 地址，而不直接使用硬件地址进行通信呢？

由于全世界存在着各式各样的网络，**它们使用不同的硬件地址**。要使这些异构网络能够互相通信就必须进行**非常复杂的硬件地址转换工作**，因此由用户或用户主机来完成这项工作几乎是不可能的事。但 IP 编址把这个复杂问题解决了。连接到互联网的主机只需要各自拥有一个唯一的 IP 地址，它们之间的通信就像连接在同一个网络上那样简单方便，因为上述的调用 ARP 的复杂过程都是由计算机软件自动进行的，对用户来说是看不见这种调用过程的。

生存时间：**防止无法交付的数据报无限制地在互联网中兜圈子**。“**跳数限制**”：**路由器在每次转发数据报之前就把 TTL 值减 1，若 TTL 值减小到零，就丢弃这个数据报，不再转发**。

在 IP 数据报的首部中没有地方可以用来指明“下一跳路由器的 IP 地址”。在 IP 数据报的首部写上的 IP 地址是源 IP 地址和目的 IP 地址，而没有中间经过的路由器的 IP 地址。既然 IP 数据报中没有下一跳路由器的 IP 地址，那么待转发的数据报又怎样能找到下一跳路由器呢？

当路由器收到一个待转发的数据报，**在从路由表得出下一跳路由器的 IP 地址后，不是把这个地址填入 IP 数据报，而是送交数据链路层的网络接口软件。网络接口软件负责把下一跳路由器的 IP 地址转换成硬件地址（必须使用 ARP），并将此硬件地址放在链路层的 MAC 帧的首部，然后根据这个硬件地址找到下一跳路由器**。由此可见，当发送一连串的数据报时，上述的这种查找路由表、用 ARP 得到硬件地址、把硬件地址写入 MAC 帧的首部等过程，将不断重复进行，造成了一定的开销。

划分子网的基本思路如下：

（1）一个拥有许多物理网络的单位，可将所属的物理网络划分为若干个“子网”。划分子网纯属一个单位内部的事情。**本单位以外的网络看不见这个网络是由多少个子网组成，因为这个单位对外仍然表现为一个网络**。

（2）划分子网的方法是从网络的主机号借用若干位作为子网号，当然主机号也就相应减少了同样的位数。

IP 地址：：={**<网络号>，<子网号>，<主机号>**}

（3）凡是从其他网络发送给本单位某台主机的 IP 数据报，仍然是根据 IP 数据报的目的网络号找到连接在本单位网络上的路由器。但**此路由器在收到 IP 数据报后，再按目的网络号和子网号找到目的子网，把 IP 数据报交付目的主机**。

假定有一个数据报（其目的地址是 145.13.3.10）已经到达了路由器 R1，那么这个路由器如何把它转发到子网 145.13.3.0 呢？

从 IP 数据报的首部无法看出源主机或目的主机所连接的网络是否进行了子网的划分。这是因为 32 位的 IP 地址本身以及数据报的首部都没有包含任何有关子网划分的信息。因此必须另外想办法，这就是使用**子网掩码**。

使用子网掩码的好处是：**不管网络有没有划分子网，只要把子网掩码和 IP 地址进行逐位的“与”运算，就能立即得出网络地址来**。

在不划分子网时，既然没有子网，为什么还要使用子网掩码？这就是为了更便于查找路由表。互联网的标准规定：**所有的网络都必须使用子网掩码，同时在路由器的路由表中也必须有子网掩码这一栏**。如果一个网络不划分子网，那么该网络的子网掩码就使用**默认子网掩码**。默认子网掩码中的 1 的位置和 IP 地址中的网络号字段 net-id 正好相对应。因此，若用默认子网掩码和某个不划分子网的 IP 地址逐位相“与”，就应当能够得出该 IP 地址的网络地址来。这样做可以**不用查找该地址的类别位就能知道这是哪一类的 IP 地址**。

同样的 IP 地址和不同的子网掩码可以得到相同的网络地址。

在使用子网划分后，路由表必须包含以下三项内容：**目的网络地址**、**子网掩码**和**下一跳地址**。

无分类域间路由选择 CIDR（Classless Inter-Domain Routing）的特点：

（1）**CIDR 消除了传统的 A 类、B 类和 C 类地址以及划分子网的概念，因而**

能更加有效地分配 IPv4 的地址空间。

IP 地址：：={**<网络前缀>，<主机号>**}

（2）**CIDR 把网络前缀都相同的连续的 IP 地址组成一个 “CIDR 地址块”**。我们只要知道 CIDR 地址块中的任何一个地址，就可以知道这个地址块的起始地址（即最小地址）和最大地址，以及地址块中的地址数。

为了更方便地进行路由选择，CIDR 使用 32 位的**地址掩码**。地址掩码由一串 1 和一串 0 组成，而 1 的个数就是网络前缀的长度。例如，/20 地址块的地址掩码是：11111111 11111111 11110000 00000000（20 个连续的 1）。**斜线记法中，斜线后面的数字就是地址掩码中 1 的个数**。

由于一个 CIDR 地址块中有很多地址，所以在路由表中就利用 CIDR 地址块来查找目的网络。这种地址的聚合常称为**路由聚合**，它使得路由表中的一个项目可以表示原来传统分类地址的很多个路由。路由聚合也称为**构成超网**。

在使用 CIDR 时，由于采用了网络前缀这种记法，IP 地址由网络前缀和主机号这两个部分组成，因此在路由表中的项目也要有相应的改变。这时，每个项目由“网络前缀”和“下一跳地址”组成。但是**在查找路由表时可能会得到不止一个匹配结果**。我们应当**从匹配结果中选择具有最长网络前缀的路由**。

二叉线索树只是提供了**一种可以快速在路由表中找到匹配的叶节点的机制**。但这是否和网络前缀匹配，还要和**子网掩码**进行一次**逻辑与**的运算。

为了更有效地转发 IP 数据报和提高交付成功的机会，在网际层使用了**网际控制报文协议 ICMP**。ICMP 允许主机或路由器报告差错情况和提供有关异常情况的报告。ICMP 是互联网的标准协议。但 **ICMP 不是高层协议**（看起来好像是高层协议，因为 **ICMP 报文是装在 IP 数据报中，作为其中的数据部分**），而**是 IP 层的协议**。ICMP 报文作为 IP 层数据报的数据，加上数据报的首部，组成 IP 数据报发送出去。

ICMP 报文的种类有两种，即 **ICMP 差错报告报文**和 **ICMP 询问报文**。

IP 数据报首部的校验和并不校验 IP 数据报的内容，因此不能保证经过传输的 ICMP 报文不产生差错。

ICMP 差错报告报文共有 4 种，即：

（1）**终点不可达**

当路由器或主机不能交付数据报时就向源点发送终点不可达报文。

（2）**时间超过**

当路由器收到生存时间为零的数据报时，除丢弃该数据报外，还要向源点发送时间超过报文。当终点在预先规定的时间内不能收到一个数据报的全部数据报片时，就把已收到的数据报片都丢弃，并向源点发送时间超过报文。

（3）**参数问题**

当路由器或目的主机收到的数据报的首部中有的字段不正确时，就丢弃该数据报，并向源点发送参数问题报文。

（4）**改变路由（重定向）**

路由器把改变路由报文发送给主机，让主机知道下次应将数据报发送给另外的路由器（可通过更好的路由）。

在互联网的主机中也要有一个路由表。当主机要发送数据报时，首先是查找主机自己的路由表，看应当从哪一个接口把数据报发送出去。在互联网中主机的数量远大于路由器的数量，出于效率的考虑，这些主机不和连接在网络上的路由器定期交换路由信息。在主机刚开始工作时，一般都在路由表中设置一个默认路由器的 IP 地址。不管数据要发送到哪个目的地址，都一律先把数据报传送给这个默认路由器，而这个默认路由器知道到每一个目的网络的最佳路由（通过和其他路由器交换路由信息）。如果默认路由器发现主机发往某个目的地址的数据报的最佳路由应当经过网络上的另一个路由器 R 时，就用**改变路由报文**把这情况告诉主机。于是，该主机就在其路由表中增加一个项目：到某某目的地址应经过路由器 R（而不是默认路由器）。

常用的 **ICMP 询问报文**有两种，即：

（1）**回送请求和回答**

ICMP 回送请求报文是由主机或路由器向一个特定的目的主机发出的询问。收到此报文的主机必须给源主机或路由器发送 ICMP 回送回答报文。这种询问报文用来**测试目的站是否可达以及了解其有关状态**。

（2）**时间戳请求和回答**

ICMP 时间戳请求报文是请某台主机或路由器回答当前的日期和时间。在 ICMP 时间戳回答报文中有一个 32 位的字段，其中写入的整数代表从 1900 年 1 月 1 日起到当前时刻一共有多少秒。时间戳请求与回答可用于**时钟同步**和**时间测量**。

ICMP 的一个重要应用就是**分组网间探测 PING**（Packet InterNet Groper），用来测试两台主机之间的连通性。**PING 使用了 ICMP 回送请求与回送回答报文**。**PING 是应用层直接使用网络层 ICMP 的一个例子**。它没有通过 TCP 或 UDP。

另一个非常有用的应用是 traceroute，它用来跟踪一个分组从源点到终点的路径。在 Windows 操作系统中这个命令是 tracert。

traceroute 从源主机向目的主机发送一连串的 IP 数据报，**数据报中封装的是无法交付的 UDP 用户数据报**。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，路由器 R1 先收下它，接着把 TTL 的值减 1。由于 TTL 等于零了，R1 就把 P1 丢弃了，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 先到达路由器 R1，R1 收下后把 TTL 减 1 再转发给路由器 R2。R2 收到 P2 时 TTL 为 1，但减 1 后 TTL 变为零了。R2 就丢弃 P2，并向源主机发送一个 ICMP 时间超过差错报告报文。这样一直持续下去。**当最后一个数据报刚刚到达目的主机时，数据报的 TTL 是 1。主机不转发数据报，也不把 TTL 值减 1。但因 IP 数据报中封装的是无法交付的运输层的 UDP 用户数据报，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**。

从路由算法能否随网络的通信量或拓扑自适应地进行调整变化来划分，有两大类，即**静态路由选择**策略与**动态路由选择**策略。静态路由选择也叫做**非自适应路由选择**，简单，开销小，但不能及时适应网络状态的变化。动态路由选择也叫做**自适应路由选择**，能适应网络状态的变化，但实现复杂，开销大。

互联网采用的路由选择协议主要是**自适应**的（即动态的）、**分布式**路由选择协议。由于以下两个原因，互联网采用分层次的路由选择协议：

（1）互联网的规模非常大。如果让所有的路由器知道所有的网络应怎样到达，则这种路由表将非常大，处理起来也太花时间。而所有这些路由器之间交换路由信息所需的带宽就会使互联网的通信链路饱和。

（2）许多单位不愿意外界了解自己单位网络的布局细节和本部门所采用的路由选择协议，但同时还希望连接到互联网上。

可以把整个互联网划分为许多较小的**自治系统**，一般都记为 AS。**自治系统 AS 是在单一技术管理下的一组路由器，而这些路由器使用一种自治系统内部的路由选择协议和共同的度量**。一个 AS 对其他 AS 表现出的是一个单一的和一致的路由选择策略。

在目前的互联网中，一个大的 ISP 就是一个自治系统。这样，互联网就把路由选择协议划分为两大类，即：

（1）**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，而这与在互联网中的其他自治系统选用什么路由选择协议无关。目前这类路由选择协议使用得最多，如 RIP 和 OSPF 协议。

（2）**外部网关协议 EGP**（External Gateway Protocol）

若源主机和目的主机处在不同的自治系统中（这两个自治系统可能使用不同的内部网关协议），当数据报传到一个自治系统的边界时，就需要使用一种协议将路由选择信息传递到另一个自治系统中。这样的协议就是外部网关协议 EGP。目前使用最多的外部网关协议是 BGP 的版本 4（BGP-4）。

RIP（Routing Information Protocol）叫做**路由信息协议**，是一种分布式的基于**距离向量**的路由选择协议。RIP 协议要求网路中的每一个路由器都要维护从它自己到其他每一个目的网络的距离记录（因此，这是**一组距离**，即“**距离向量**”）。RIP 协议将“距离”定义如下：

从一路由器到直接连接的网络的距离定义为 1。从一路由器到非直接连接的网络的距离定义为所经过的路由器数加 1。

RIP 协议的“**距离**”也称为“**跳数**”，因为每经过一个路由器，跳数就增加 1。RIP 认为好的路由就是它通过的路由器的数目少，即“距离短”。RIP 允许一条路径最多只能包含 15 个路由器。因此“距离”等于 16 时即相当于不可达。可见 **RIP 只适用于小型互联网**。

RIP 协议的特点是：（1）**仅和相邻路由器交换信息**。（2）**路由器交换的信息是当前本路由器所知道的全部信息，即自己现在的路由表**。（3）**按固定的时间间隔交换信息**。

RIP 协议的距离向量算法的基础是 Bellman-Ford 算法：

设 X 是结点 A 到 B 的最短路径上的一个结点。若把路径 A->B 拆成两段路径 A->X 和 X->B，则每一段路径 A->X 和 X->B 也都分别是结点 A 到 X 和结点 X 到 B 的最短路径。

RIP 协议使用运输层的**用户数据报 UDP** 进行传送。

RIP 协议的问题：**当网络出现故障时，要经过比较长的时间才能将此信息传送到所有的路由器**。

开放最短路径优先 OSPF（Open Shortest Path First）使用了 **Dijkstra 提出的最短路径算法 SPF**。OSPF 最主要的特征就是使用分布式的**链路状态协议**（link state protocol）。和 RIP 协议相比，OSPF 的三个要点和 RIP 的都不一样：

（1）**向本自治系统中所有路由器发送信息**。这里使用的方法是**洪泛法**，这就是**路由器通过所有输出端口向所有相邻的路由器发送信息**。而**每一个相邻路由器又将此信息发往其所有的相邻路由器（但不再发送给刚刚发来信息的那个路由器）**。这样，最终整个区域中所有的路由器都得到了这个信息的**一个副本**。我们应注意，RIP 协议是仅仅向自己相邻的几个路由器发送信息。

（2）发送的信息就是**与本路由器相邻的所有路由器的链路状态**，但这只是路由器所知道的**部分信息**。所谓“链路状态”就是说明本路由器都和哪些路由器相邻，以及该链路的“**度量**”。OSPF 将这个“度量”用来表示费用、距离、时延、带宽，等等。这些都由网络管理人员来决定，因此较为灵活。有时为了方便就称这个度量为“**代价**”。我们应注意，对于 RIP 协议，发送的信息是：“到所有网络的距离和下一跳路由器”。

（3）**只有当链路状态发生变化时，路由器才向所有路由器用洪泛法发送此信息**。而不像 RIP 那样，不管网络拓扑有无发生变化，路由器之间都要定期交换路由表的消息。

由于各路由器之间频繁地交换链路状态信息，因此所有的路由器最终都能建立一个**链路状态数据库**，这个数据库实际上就是**全网的拓扑结构图**。这个拓扑结构图在全网范围内是**一致**的（链路状态数据库的同步）。因此，每一个路由器都知道全网共有多少个路由器，以及哪些路由器是相连的，其代价是多少。每一个路由器使用链路状态数据库中的数据，构造出自己的路由表。我们注意到，RIP 协议的每一个路由器虽然知道到所有网络的距离以及下一跳路由器，但却**不知道全网的拓扑结构**（只有到了下一跳路由器，才能知道再下一跳应当怎样走）。

为了使 OSPF 能够用于规模很大的网络，OSPF 将一个自治系统再划分为若干个更小的范围，叫做**区域**。在一个区域内的路由器最好不超过 200 个。

划分区域的好处就是把利用洪泛法交换链路状态信息的范围局限于每一个区域而不是整个的自治系统，这就减少了整个网络上的通信量。在一个区域内部的路由器只知道本区域的完整网络拓扑，而不知道其他区域的网络拓扑的情况。为了使每一个区域能够和本区域以外的区域进行通信，OSPF 使用层次结构的区域划分。在上层的区域叫做**主干区域**。主干区域的作用是用来连通其他在下层的区域。从其他区域来的信息都由**区域边界路由器**进行概括。每一个区域至少应当有一个区域边界路由器，在主干区域内的路由器叫做**主干路由器**。一个主干路由

器可以同时是区域边界路由器。在主干区域内还要有一个路由器专门和本自治系统外的其他自治系统交换路由信息。这样的路由器叫做**自治系统边界路由器**。

OSPF 不用 UDP 而是直接用 **IP 数据报**传送。

边界网关协议 BGP 力求寻找一条能够到达目的网络且**比较好**的路由，而并非要寻找一条最佳路由。BGP 采用了**路径向量（path vector）路由选择协议**。在配置 BGP 时，每一个自治系统的管理员要选择至少一个路由器作为该自治系统的“**BGP 发言人**”。一个 BGP 发言人与其他的 BGP 发言人要交换路由信息，就要先建立 **TCP 连接**，然后在此连接上交换 BGP 报文以建立 **BGP 会话**。BGP 协议交换路由信息的结点数量级是**自治系统个数**的量级。

路由器是一种具有**多个输入端口**和**多个输出端口**的专用计算机，其任务是**转发分组**。从路由器某个输入端口收到的分组，按照分组要去的目的地，把该分组从路由器的某个合适的输出端口转发给下一跳路由器。下一跳路由器也按照这种方法处理分组，直到该分组到达终点为止。

整个路由器结构可划分为两大部分：**路由选择**部分和**分组转发**部分。

路由选择部分也叫做**控制部分**，其核心构件是**路由选择处理机**。路由选择处理机的任务是根据所定的路由选择协议构造出路由表，同时经常或定期地和相邻路由器交换路由信息而不断地更新和维护路由表。

交换结构又称为**交换组织**，它的作用就是根据**转发表**对分组进行处理，将某个输入端口进入的分组从一个合适的输出端口转发出去。交换结构本身就是一种网络，但这种网络完全包含在路由器之中，因此交换结构可看成是“**在路由器中的网络**”。

“转发”和“路由选择”是有区别的。在互联网中，“**转发**”就是**路由器根据转发表把收到的 IP 数据报从路由器合适的端口转发出去**。“转发”仅仅涉及到**一个路由器**，但“**路由选择**”则涉及到**很多路由器**，路由表则是许多路由器协同工作的结果。这些路由器按照复杂的路由算法，得出整个网络的拓扑变化情况，因而能够动态地改变所选择的路由，并由此构造出整个的路由表。路由表一般仅包含从目的网络到下一跳（用 IP 地址表示）的映射，而转发表是从路由表得出的。转发表必须包含完成转发功能所需的信息。**在转发表的每一行必须包含从要到达的目的网络到输出端口和某些 MAC 地址信息的映射**。将转发表和路由表用不同的数据结构实现会带来一些好处，这是因为**在转发分组时，转发表的结构应当使**

查找过程最优化，但路由表需要对网络拓扑变化的计算最优化。路由表总是用**软件**实现的，但转发表则甚至可用特殊的**硬件**来实现。

IPv6 所引进的主要变化如下：

（1）**更大的地址空间**。IPv6 把地址从 IPv4 的 32 位增大到 4 倍，即增大到 **128 位**，使地址空间增大了 2^{96} 倍。

（2）**扩展的地址层次结构**。IPv6 由于地址空间很大，因此可以划分为更多的层次。

（3）**灵活的首部格式**。IPv6 数据报的首部和 IPv4 的并不兼容。IPv6 定义了许多可选的扩展首部，不仅可提供比 IPv4 更多的功能，而且还可提高路由器的处理效率，这是因为路由器对扩展首部不进行处理（除逐跳扩展首部外）。

（4）**改进的选项**。IPv6 允许数据报包含有选项的控制信息，因而可以包含一些新的选项。但 **IPv6 的首部长度是固定的，其选项放在有效载荷中**。我们知道，**IPv4 所规定的选项是固定不变的，其选项放在首部的可变部分**。

（5）**允许协议继续扩充**。这一点很重要，因为技术总是在不断地发展（如网络硬件的更新）而新的应用也还会出现。但我们知道，IPv4 的功能是固定不变的。

（6）**支持即插即用（即自动配置）**。因此 IPv6 不需要动态主机设置协议 DHCP（Dynamic Host Configuration Protocol）。

（7）**支持资源的预分配**。IPv6 支持实时视像等要求保证一定的带宽和时延的应用。

（8）IPv6 首部改为 **8 字节对齐**（即首部长度必须是 8 字节的整数倍）。原来的 IPv4 首部是 4 字节对齐。

IPv6 数据报由两大部分组成，即**基本首部**和后面的**有效载荷**。有效载荷也称为**净负荷**。有效载荷允许有零个或多个扩展首部，再后面是**数据部分**。但请注意，**所有的扩展首部并不属于 IPv6 数据报的首部**。

IPv4 向 IPv6 的过渡策略：（1）**双协议栈**；（2）**隧道技术**。

双协议栈主机在和 IPv6 主机通信时采用 IPv6 地址，而和 IPv4 主机通信时则采用 IPv4 地址。但双协议栈主机怎样知道目的主机是采用哪一种地址呢？它是使用**域名系统 DNS** 来查询的。若 DNS 返回的是 IPv4 地址，双协议栈的源主机就使用 IPv4 地址。但当 DNS 返回的是 IPv6 地址，源主机就使用 IPv6 地址。

隧道技术的要点就是在 IPv6 数据报要进入 IPv4 网络时，把 IPv6 数据报封装称为 IPv4 数据报，使得整个的 IPv6 数据报变成 IPv4 数据报的数据部分。

在互联网上进行多播就叫做 **IP 多播**，IP 多播所传送的分组需要使用**多播 IP 地址**。

网际组管理协议 IGMP（Internet Group Management Protocol）是让连接在本地局域网上的多播路由器知道本局域网上是否有主机（严格讲，是主机上的某个进程）参加或退出了某个多播组。

反向路径广播 RPB（Reverse Path Broadcasting）的要点是：**每一个路由器在收到一个多播数据报时，先检查数据报是否是从源点经最短路径传送来的**。进行这种检查很容易，只要从本路由器寻找到源点的最短路径上（之所以叫做反向路径，因为在计算最短路径时是把源点当终点）的第一个路由器是否就是刚才把多播数据报送来的路由器。若是，就向所有其他反向转发刚才收到的多播数据段（但进入的方向除外），否则就丢弃而不转发。如果本路由器有好几个相邻路由器都处在到源点的最短路径上（也就是说，存在几条同样长度的最短路径），那么只能选择一条最短路径，选择的准则就是看这几条最短路径中的相邻路由器谁的 IP 地址最小。

专用地址只能用作本地地址而不能用作全球地址。**在互联网中的所有路由器，对目的地址是专用地址的数据报一律不进行转发**。有时一个很大的机构的许多部门分布的范围很广，这些部门经常要相互交换信息。这时可以利用公用的互联网作为本机构各专用网之间的通信载体，这样的专用网又称为**虚拟专用网 VPN**（Virtual Private Network）。

之所以称为“专用网”是因为这种网络是为本机构的主机用于机构内部的通信，而不是用于和网络外非本机构的主机通信。如果专用网不同网点之间的通信必须经过公用的互联网，但又有保密的要求，那么所有通过互联网传送的数据都必须加密。（**IP 隧道技术**）

网络地址转换 NAT（Network Address Translation）方法需要在专用网连接到互联网的路由器上安装 NAT 软件。装有 NAT 软件的路由器叫做 NAT 路由器，它至少有一个有效地外部全球 IP 地址。这样，所有使用本地地址的主机在和外界通信时，都要**在 NAT 路由器上将其本地地址转换成全球 IP 地址**，才能和互联网连接。

IP 数据报分为**首部**和**数据**两部分。首部的前一部分是**固定长度**，共 20 字节，是所有 IP 数据报必须具有的（源地址、目的地址、总长度等重要字段都在固定首部中）。一些长度可变的可选字段放在固定首部的后面。

从通信和信息处理的角度看，运输层向它上面的应用层提供通信服务，它属于**面向通信部分的最高层**，同时也是**用户功能中的最低层**。当网络的边缘部分中的两台主机使用网络的核心部分的功能进行端到端的通信时，**只有主机的协议栈才有运输层，而网络核心部分中的路由器在转发分组时都只用到下三层的功能**。

网络层为**主机**之间提供逻辑通信，而运输层为**应用进程**之间提供端到端的逻辑通信。两个对等运输实体在通信时传送的数据单位叫做**运输协议数据单元 TPDU**（Transport Protocol Data Unit）。但在 TCP/IP 体系中，则根据所使用的协议是 TCP 或 UDP，分别称之为 **TCP 报文段**或 **UDP 用户数据报**。

UDP 在传送数据之前**不需要先建立连接**。远地主机的运输层在收到 UDP 报文后，**不需要给出任何确认**。虽然 UDP **不提供可靠交付**，但在某些情况下 UDP 却是一种最有效的工作方式。

TCP 则提供**面向连接**的服务。在传送数据之前必须先建立连接，数据传送结束后要释放连接。TCP **不提供广播或多播服务**。由于 TCP 要提供可靠的、面向连接的运输服务，因此不可避免地增加了许多的开销，如**确认**、**流量控制**、**计时器**以及**连接管理**等。这不仅使协议数据单元的首部增大很多，还要占用许多的处理机资源。

应用层协议	运输层协议	端口
DNS（域名系统）	UDP	53
TFTP（简单文件传送协议）	UDP	
RIP（路由信息协议）	UDP	
DHCP（动态主机配置协议）	UDP	
SNMP（简单网络管理协议）	UDP	
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	

IGMP（网际组管理协议）	UDP	
SMTP（简单邮件传送协议）	TCP	25
TELNET（远程终端协议）	TCP	23
HTTP（超文本传送协议）	TCP	80
FTP（文件传送协议）	TCP	21

复用：应用层所有的应用进程都可以通过运输层再传送到网络层。**分用**：运输层从网络层收到发送给各应用进程的数据后，必须分别交付指明的各应用进程。

协议端口号是协议栈层间的抽象，是**软件端口**，是**应用层的各种协议进程与运输实体进行层间交互的一种地址**。

UDP 的主要特点是：

（1）UDP 是**无连接的**，即发送数据之前不需要建立连接（发送数据结束时也没有连接可释放），因此减少了开销和发送数据之前的时延。

（2）UDP 使用**尽最大努力交付**，即不保证可靠交付，因此主机不需要维持复杂的连接状态表。

（3）UDP 是**面向报文的**。发送方的 UDP 对应用程序交下来的报文，在添加首部后就向下交付 IP 层。UDP 对应用层交下来的报文，既不合并，也不拆分，而是保留这些报文的边界。这就是说，**应用层交给 UDP 多长的报文，UDP 就照样发送，即一次发送一个报文**。在接收方的 UDP，对 IP 层交上来的 UDP 用户数据报，在去除首部后就原封不动地交付上层的应用进程。也就是说，UDP 一次交付一个完整的报文。因此，应用程序必须选择合适大小的报文。若报文太长，UDP 把它交给 IP 层后，IP 层在传送时可能要进行分片，这会降低 IP 层的效率。反之，若报文太短，UDP 把它交给 IP 层后，会使 IP 数据报的首部的相对长度太长，这也降低了 IP 层的效率。

（4）UDP **没有拥塞控制**，因此网络出现的拥塞不会使源主机的发送速率降低。很多的实时应用（如 **IP 电话、实时视频会议**等）要求源主机以恒定的速率发送数据，并且允许在网络发生拥塞时丢失一些数据，但却不允许数据有太大的时延。UDP 正好适合这种要求。

（5）UDP 支持**一对一、一对多、多对一和多对多**的交互通信。

（6）UDP 的**首部开销小，只有 8 个字节**，比 TCP 的 20 个字节的首部要短。

虽然在 UDP 之间的通信要用到其端口号，但由于 UDP 的通信是无连接的，因此**不需要使用套接字（TCP 之间的通信必须要在两个套接字之间建立连接）**。

TCP 最主要的特点：

（1）TCP 是**面向连接**的运输层协议。这就是说，应用程序在使用 TCP 协议之前，必须**先建立 TCP 连接**。在传送数据完毕后，必须**释放已经建立的 TCP 连接**。

（2）每一条 TCP 连接只能是点对点的。

（3）TCP 提供**可靠交付**的服务。通过 TCP 连接传送的数据，**无差错、不丢失、不重复，并且按序到达**。

（4）TCP 提供**全双工**通信。TCP 允许通信双方的应用进程在任何时候都能发送数据。**TCP 连接的两端都设有发送缓存和接收缓存，用来临时存放双向通信的数据**。在发送时，应用程序在把数据传送给 TCP 的缓存后，就可以做自己的事，而 TCP 在合适的时候把数据发送出去。在接收时，TCP 把收到的数据放入缓存，上层的应用进程在合适的时候读取缓存中的数据。

（5）**面向字节流**。TCP 中的“**流**”（stream）指的是流入到进程或从进程流出的字节序列。“面向字节流”的含义是：**虽然应用程序和 TCP 的交付是一次一个数据块（大小不等），但 TCP 把应用程序交下来的数据仅仅看成是一连串无结构的字节流**。TCP 并不知道所传送的字节流的含义。**TCP 不保证接收方应用程序所收到的数据块和发送方应用程序所发出的数据块具有对应大小的关系**（例如，发送方应用程序交给发送方的 TCP 共 10 个数据块，但接收方的 TCP 可能只用了 4 个数据块就把接收到的字节流交付上层的应用程序）。但接收方应用程序收到的字节流必须和发送方应用程序发出的字节流完全一样。当然，接收方的应用程序必须有能力识别收到的字节流，把它还原成有意义的应用层数据。

TCP 和 UDP 在发送报文时所采用的方式完全不同。TCP 并不关心应用进程一次把多长的报文发送到 TCP 的缓存中，而是根据**对方给出的窗口值**和**当前网络拥塞的程度**来决定一个报文段应包含多少个字节（UDP 发送的报文长度是应用进程给出的）。如果应用进程传送到 TCP 缓存的数据块太长，TCP 就可以把它划分短一些再传送。如果应用进程一次只发来一个字节，TCP 也可以等待积累有足够多的字节后再构成报文段发送出去。

套接字 socket=（IP 地址：端口号）

每一条 TCP 唯一地被通信两端的两个端点（即**两个套接字**）所确定。

“**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。在收到确认后再发送下一个分组**。

A 只要**超过了一段时间仍然没有收到确认，就认为刚才发送的分组丢失了，因而重传前面发送过的分组**。这就叫做**超时重传**。要实现超时重传，就要在每发送完一个分组时设置一个**超时计时器**。如果在超时计时器到期之前收到了对方的确认，就撤销已设置的超时计时器。

这里应注意以下三点：

（1）第一，A 在发送完一个分组后，**必须暂时保留已发送的分组的副本**（在发生超时重传时使用）。只有在收到相应的确认后才能清除暂时保留的分组副本。

（2）第二，分组和确认分组都必须进行**编号**。这样才能明确是哪一个发送出去的分组收到了确认，而哪一个分组还没有收到确认。

（3）超时计时器设置的**重传时间应当比数据在分组传输的平均往返时间更长一些**。

连续 ARQ（Automatic Repeat reQuest）协议

滑动窗口。

接收方一般都是采用**累积确认**的方式。这就是说，接收方不必对收到的分组逐个发送确认，而是在收到几个分组后，**对按序到达的最后一个分组发送确认**，这就表示：到这个分组为止的所有分组都已经正确收到了。

累积确认的优点是：容易实现，即使确认丢失也不必重传。缺点是不能向发送方反映出接收方已经正确收到的所有分组的信息。

例如，如果发送方发送了前 5 个分组，而中间的第 3 个分组丢失了。这时接收方只能对前两个分组发出确认。发送方无法知道后面三个分组的下落，而只好把后面的三个分组都再重传一次。这就叫做 **Go-back-N**（回退 N），表示需要再退回来重传已发送过的 N 个分组。可见当通信线路质量不好时，连续 ARQ 协议会带来负面的影响。

确认号，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。**若确认号=N，则表明：到序号 N－1 为止的所有数据都已正确收到**。

确认 ACK，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，**在连接建立后所有传送的报文段都必须把 ACK 置 1**。

复位 RST，当 RST=1 时，**表明 TCP 连接中出现严重差错**（如由于主机崩溃或其他原因），**必须释放连接，然后再重新建立运输连接**。RST 置 1 还用来**拒绝一个非法的报文段或拒绝打开一个连接**。RST 也可称为重建位或重置位。

同步位 SYN，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，则在响应的报文段中使 SYN=1 和 ACK=1**。SYN 置 1 就表示这是一个**连接请求或连接接受报文**。

终止 FIN，用来释放一个连接。当 FIN=1 时，表明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。

窗口指的是发送本报文段的一方的接收窗口（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。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**。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着。

最大报文长度 MSS（Maximum Segment Size），每一个 TCP 报文段中的数据字段的最大长度。

为什么要规定一个最大报文段长度 MSS 呢？这并不是考虑接收方的接收缓存可能放不下 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），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**。设想在极端的情况下，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装配成原来的 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。

发送窗口表示：在没有收到 B 的确认的情况下，A 可以连续把窗口内的数据都发送出去。凡是已经发送过的数据，在未收到确认之前都必须暂时保留，以便在超时重传时使用。

发送窗口后沿的后面部分表示已发送且已收到了确认。这些数据显然不需要再保留了。而发送窗口前沿的前面部分表示不允许发送的，因为接收方都没有为这部分数据保留临时存放的缓存空间。

发送窗口的位置由窗口前沿和后沿的位置共同确定。发送窗口后沿的变化情况有两种可能，即**不动（没有收到新的确认）**和**前移（收到了新的确认）**。发送窗口后沿不可能向后移动，因为不能撤销掉已收到的确认。发送窗口前沿通常是**不断向前移动**，但也有可能不动。这对应于两种情况：一是**没有收到新的确认，对方通知的窗口大小也不变**；二是**收到了新的确认但对方通知的窗口缩小了**，使得发送窗口前沿正好不动。

发送窗口前沿也可能**向后收缩**。这发生在对方通知的窗口缩小了。但 **TCP 的标准强烈不赞成这样做**。

发送缓存用来暂时存放：

（1）发送应用程序传送给发送方 TCP **准备发送**的数据。

（2）TCP **已发送出但尚未收到确认**的数据。

发送窗口通常只是发送缓存的一部分。已被确认的数据应当从发送缓存中删除，因此**发送缓存和发送窗口的后沿是重合的。发送应用程序最后写入发送缓存的字节减去最后被确认的字节，就是还保留在发送缓存中的被写入的字节数**。发送应用程序必须控制写入缓存的速率，不能太快，否则发送缓存就会没有存放数据的空间。

接收缓存用来临时存放：

（1）**按序到达的、但尚未被接收应用程序读取**的数据。

（2）**未按序到达**的数据。

如果收到的分组被检测出差错，则要丢弃。如果接收应用程序来不及读取收到的数据，接收缓存最终就会被填满，使接收窗口减小到零。反之，如果接收应用程序能够及时从接收缓存中读取收到的数据，接收窗口就可以增大，但最大不能超过接收缓存的大小。

虽然 A 的发送窗口是根据 B 的接收窗口设置的，但**在同一时刻，A 的发送窗口并不总是和 B 的接收窗口一样大**。

对于不按序到达的数据应如何处理，TCP 标准并无明确规定。如果接收方把不按序到达的数据一律丢弃，那么接收窗口的管理将会比较简单，但这样做为发送方会重复传送较多的数据。**TCP 通常对不按序到达的数据是先临时存放在接收窗口中，等到字节流中所缺少的字节收到后，再按序交付上层的应用进程**。

TCP 要求接收方必须有**累积确认**的功能，这样可以减小传输开销。接收方可以在合适的时候发送确认，也可以在自己有数据要发送时把确认信息顺便**捎带**上。注意两点：一是**接收方不应过分推迟发送确认，否则会导致发送方不必要的重传，这反而浪费了网络的资源**。二是**捎带确认实际上并不经常发生，因为大多数应用程序很少同时在两个方向上发送数据**。

报文段的往返时间 RTT 的测验，实现起来相当复杂。

发送出一个报文段，设定的重传时间到了，还没有收到确认。于是重传报文段。经过了一段时间后，收到了确认报文段。现在的问题是：**如何判定此确认报文段是对先发送的报文段的确认，还是对后来重传的报文段的确认**？由于重传的报文段和原来的报文段完全一样，因此源主机在收到确认后，就无法做出正确的判断，而正确的判断对确定加权平均 RTTs 的值关系很大。

正确的方法是：**报文每重传一次，就把超时重传时间设为原来的 2 倍。当不再发生报文段的重传时，才根据原方法计算超时重传时间**。

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。

现在我们考虑一种情况，B 向 A 发送了零窗口的报文段后不久，B 的接收缓存又有了一些存储空间。于是 B 向 A 发送了 rwnd=400 的报文段。然而这个报文段在传送过程中丢失了。A 一直等待收到 B 发送的非零窗口的通知，而 B 也一直等待 A 发送的数据。如果没有其他措施，这种互相等待的死锁局面将一直延续下去。

为了解决这个问题，TCP 为每一个连接设有一个**持续计数器**。只要 TCP 连接的一方收到对方的零窗口通知，就启动持续计数器。若持续计数器设置的时间到期，就发送一个**零窗口探测报文段**，而对方就在确认这个探测报文段时给出了现在的窗口值。如果窗口仍然是零，那么收到这个报文段的一方就重新设置持续计时器。如果窗口不是零，那么死锁的僵局就可以打破了。

在 TCP 的实现中广泛使用 **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。当发送方收到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。这样做，就可以有效地提高网络的吞吐性能。

糊涂窗口综合征。设想一种情况：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 个字节（这样就使接收缓存空间仅腾出 1 个字节），然后向发送方发送确认，并把窗口设置为 1 个字节（但发送的数据报是 40 字节长）。接着，发送方又发来 1 个字节的数据（请注意，发送方发送的 IP 数据报是 41 字节长）。接收方发回确认，仍然将窗口设置为 1 个字节。这样进行下去，使网络的效率很低。

要解决这个问题，可以**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**接收缓存已有一半空闲的空间**。只要出现这两种情况之一，接收方就发出确认报文，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。

在计算机网络中的链路容量（即带宽）、交换结点中的缓存和处理机等，都是网络的资源。**在某段时间，若对网络中某一资源的需求超过了该资源所能提供的可用部分，网络的性能就要变坏**。这种情况就叫做**拥塞**（congestion）。拥塞常常趋于恶化。

拥塞控制就是防止过多的数据注入到网络中，这样可以使网络中的路由器或链路不致过载。拥塞控制所要做的都有一个前提，就是**网络能够承受现有的网络负荷**。拥塞控制是一个**全局性**的过程。

流量控制往往是指点对点通信量的控制，是个**端到端**的问题。

具有理想拥塞控制的网络，在吞吐量饱和之前，网络吞吐量应等于提供的负载，故吞吐量曲线是 45°的斜线。但当提供的负载超过某一限度时，由于网络资源受限，吞吐量不再增长而保持为水平，即吞吐量达到饱和。这就表明提供的负

载中有一部分损失掉了。虽然如此，在这种理想的拥塞控制作用下，网络的吞吐量仍然维持在其所能达到的最大值。

但是，实际网络的情况就很不相同了。随着提供的负载的增大，网络吞吐量的增长速率逐渐减小。在网络吞吐量还未达到饱和时，就已经有一部分的输入分组被丢弃了。当网络的吞吐量明显地小于理想的吞吐量时，网络就进入了**轻度拥塞**的状态。当提供的负载达到某一数值时，网络的吞吐量反而随提供的负载的增大而下降，这时**网络就进入了拥塞状态**。当提供的负载继续增大到某一数值时，网络的吞吐量就下降到零，网络已无法工作，这即是所谓的**死锁**。

判断网络拥塞的依据就是出现了超时。

慢开始：当主机开始发送数据时，由于并不清楚网络的负荷情况，所以如果立即把大量数据字节注入到网络，那么就有可能引起网络发生拥塞。经验证，较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。**每经过一个传输轮次，拥塞窗口 cwnd 就加倍**。

慢开始门限 **ssthresh**：

（1）当 cwnd<ssthresh 时，使用慢开始算法。

（2）**当 cwnd>ssthresh 时，停止使用慢开始算法而改用拥塞避免算法**。

（3）当 cwnd=ssthresh 时，既可使用慢开始算法，也可使用拥塞避免算法。

拥塞避免算法的思路是让拥塞窗口 cwnd 缓慢地增大，即每经过一个往返时间 RTT 就把发送方的拥塞窗口 cwnd 加 1，而不是像慢开始阶段那样加倍增长。

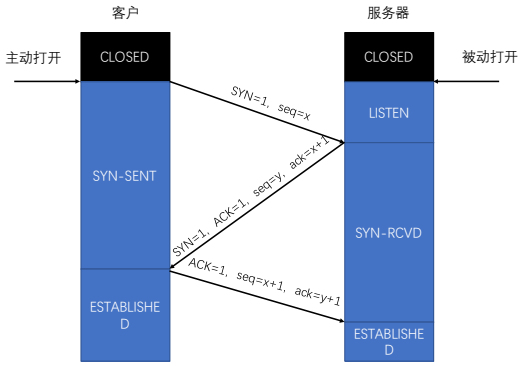
采用**快重传**算法可以**让发送方尽早知道发生了个别报文段的丢失**。快重传算法首先要求接收方**不要等待自己发送数据时才进行捎带确认，而是要立即发送确认，即使收到了失序的报文段也要立即发出对已收到的报文段的重复确认**。快重传算法规定，**发送方只要一连收到 3 个重复确认，就知道接收方确实没有收到报文段 M，因而应当立即进行重传**。

快恢复算法。当发送方收到 3 个重复确认时，不执行慢开始算法，而是执行拥塞避免算法。

发送窗口的上限值=**Min[rwnd，cwnd]**，rwnd 为**接收方窗口**，cwnd 为**拥塞**

窗口。

主动队列管理 AQM（Active Queue Management）。所谓“主动”就是不要等到路由器的队列长度已经达到最大值时才不得不丢弃后面到达的分组。这样就太被动了。**在队列长度达到某个值得警惕的数值时（即当网络拥塞有了某些拥塞征兆时），就主动丢弃到达的分组**。这样就提醒了发送方放慢发送的速率，因而有可能使网络拥塞的程度减轻，甚至不出现网络拥塞。

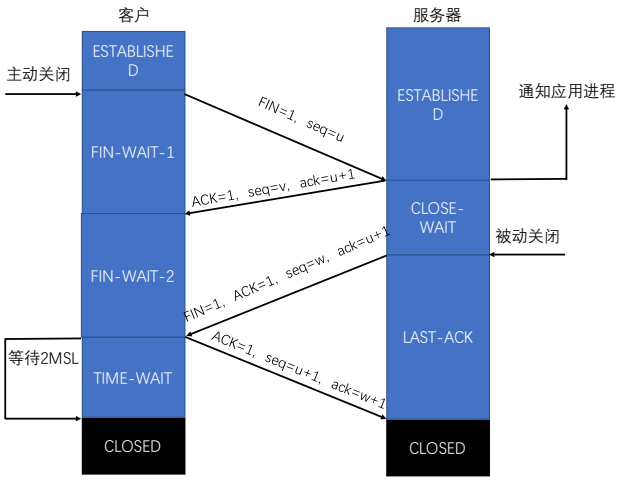


为什么客户端最后还要发送一次确认呢？这主要是为了**防止已失效的连接请求报文段突然又传送到了 B**，因而产生错误。

所谓“**已失效的连接请求报文段**”是这样产生的。考虑一种正常情况，客户端发出连接请求，但因连接请求报文丢失而未收到确认。于是客户端再重传一次连接请求。后来收到了确认，建立了连接。数据传输完毕后，就释放了连接。客户端共发送了两个连接请求报文段，其中第一个丢失了，第二个到达了服务器，没有“已失效的连接请求报文段”。

现假定出现一种异常情况：**客户端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服务器**。本来这是一个早已失效的报文段。但服务器收到此失效的连接请求报文段后，就误认为是客户端又发出一次新的连接请求。于是就向客户端发出确认报文段，同意建立连接。假定不采用报文握手，那么只要服务器发出确认，新的连接就建立了。

由于现在客户端并没有发出建立连接的请求，因此不会理睬服务器的确认，也不会向服务器发送数据。但服务器却以为新的运输连接已经建立了，并一直等待客户端发来数据。服务器的许多资源就这样白白浪费了。



为什么客户端在 TIME-WAIT 状态必须等待 2MSL 的时间呢？

第一，**为了保证客户端发送的最后一个 ACK 报文段能够到达服务器**。这个 ACK 报文段有可能丢失，因而使处在 LAST-ACK 状态的服务器收不到对已发送的 FIN+ACK 报文的确认。服务器会超时重传这个 FIN+ACK 报文段，而客户端就能在 2MSL 时间内收到这个重传的 FIN+ACK 报文段。接着客户端重传一次确认，重新启动 2MSL 计时器。最后，客户端和服务器都正常进入到 CLOSED 状态。如果客户端在 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，那么就无法收到服务器重传的 FIN+ACK 报文段，因而也不会再发送一次确认报文。这样，服务器就无法按照正常步骤进入 CLOSED 状态。

第二，**防止“已失效的连接请求报文段”出现在本连接中**。客户端在发送完最后一个 ACK 报文段后，再经过 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。

服务器只要收到了客户端发出的确认，就进入 CLOSED 状态。同样，服务器在撤销相应的传输控制块 TCB 后，就结束了这次 TCP 连接。我们注意到，服务器结束 TCP 连接的时间要比客户端早一些。

除时间等待计时器外，TCP 还设有一个**保活计时器**。设想有这样的情况：客户已主动与服务器建立了 TCP 连接。但后来客户端的主机突然出故障。显然，服务器以后就不能再收到客户发来的数据。因此，应当**有措施使服务器不要再白白等待下去。这就是使用保活计时器**。服务器每收到一次客户的数据，就重新设置保活计时器，时间的设置通常是两小时。若两小时没有收到客户的数据，服务器就发送一个探测报文段，以后则每隔 75 秒钟发送一次。若一连发送 10 个探测报文段后仍无客户的响应，服务器就认为客户端出了故障，接着就关闭这个连接。

运输层用一个 16 位端口号来标志一个端口。端口号具有本地意义，它只是为了标志本计算机应用层中的各个进程在和运输层交互时的层间接口。在互联网的不同计算机中，相同的端口号是没有关联的。

一个域名服务器所负责管辖的范围叫做**区**。一个区中的所有节点必须是能够连通的。每一个区设置相应的权限域名服务器，用来保存该区中的所有主机的域名到 IP 地址的映射。DNS 服务器的管辖范围不是以“域”为单位，而是以“区”为单位。区是 DNS 服务器实际管辖的范围。**区可能等于或小于域，但一定不能大于域**。

万维网并非某种特殊的计算机网络。**万维网是一个大规模的、联机式的信息储藏所**，英文简称为 Web。万维网用**链表**的方法能非常方便地从互联网上的一个站点访问另一个站点。

万维网使用**统一资源定位符 URL** 来标志万维网上的各种文档，并使每一个文档在整个互联网的范围内具有唯一的标识符 URL：

<协议>://<主机>:<端口>/<路径>

HTTP 的默认端口号是 80。HTTP 是**面向事务**的应用层协议。

HTTP 协议本身是**无连接**的。**虽然 HTTP 使用了 TCP 连接，但通信的双方在交换 HTTP 报文之前不需要先建立 HTTP 连接**。

HTTP 协议是**无状态**的。同一个客户第二次访问同一个服务器上的页面时，服务器的响应与第一次被访问时的相同，因为服务器并不记得曾经访问过的这个客户，也不记得为该客户曾经服务过多少次。**HTTP 的无状态特性简化了服务器的设计，使服务器更容易支持大量并发的 HTTP 请求**。

用户在点击鼠标链接某个万维网文档时，HTTP 协议首先要和服务器建立 TCP 连接。这需要使用三报文握手。当**建立 TCP 连接的三报文握手的前两部分完成后，万维网客户就把 HTTP 请求报文，作为建立 TCP 连接的三报文握手中的第三个报文的数据，发送给万维网服务器**。服务器收到 HTTP 请求报文后，就把所请求的文档作为响应报文返回给客户。请求一个万维网文档所需的时间是该文档的传输时间加上两倍往返时间 RTT（一个 RTT 用于连接 TCP 连接，另一个 RTT 用于请求和接收万维网文档。**TCP 建立连接的三报文握手的第三个报文段中的数据，就是客户对万维网文档的请求报文**）。

非持续连接会使万维网服务器的负担很重。好在浏览器都能够打开 5~10 个并行的 TCP 连接，而每一个 TCP 连接处理客户的一个请求。使用并行的 TCP 连接可以缩短响应时间。

持续连接就是万维网服务器在发送响应后仍然在一段时间内保持这条连接，使同一个浏览器和该服务器可以继续在这条连接上传送后续的 HTTP 请求报文和响应报文。

HTTP/1.1 协议的持续连接有两种工作方式，即**非流水线方式**和**流水线方式**。非流水线方式的特点，**是客户在收到前一个响应后才能发出下一个请求**。流水线方式的特点，是**客户在收到 HTTP 的响应报文之前就能够接着发送新的请求报文**。

由于 HTTP 是**面向文本**的，因此在报文中的每一个字段都是一些 ASCII 码串，因而各个字段的长度都是不确定的。HTTP 请求报文和响应报文都是由三个部分组成的。这两种报文的区别就是开始行不同。

（1）**开始行**，用于区分是请求报文还是响应报文。在请求报文中的开始行叫做**请求行**，而在响应报文中的开始行叫做**状态行**。在开始行的三个字段之间都以空格分隔开，最后的“CR”和“LF”分别代表“回车”和“换行”。

（2）**首部行**，用来说明浏览器、服务器或报文主体的一些信息。首部可以有好几行，但也可以不使用。在每一个首部行中都有首部字段名和它的值，每一行在结束的地方都要有“回车”和“换行”。**整个首部行结束时，还有一空行将首部行和后面的实体主体分开**。

（3）**实体主体**，在请求报文中一般都不用这个字段，而在响应报文中也可能没有这个字段。

请求报文的第一行“请求行”只有三个内容，即**方法**，**请求资源的 URL**，以及 **HTTP 的版本**。

方法	意义
OPTION	请求一些选项的信息
GET	**请求读取由 URL 所标志的信息**
HEAD	请求读取由 URL 所标志的信息的首部

POST	**给服务器添加信息（例如，注释）**
PUT	**在指明的 URL 下存储一个文档**
DELETE	**删除指明的 URL 所标志的资源**
TRACE	用来进行环回测试的请求报文
CONNECT	用于代理服务器

响应报文的状态行包括三项内容，即 **HTTP 的版本**，**状态码**，以及**解释状态码的简单短语**。

1xx 表示通知信息，如请求收到了或正在进行处理。

2xx 表示成功，如接受或知道了。

3xx 表示重定向，如要完成请求还必须采取进一步的行动。

4xx 表示客户的差错，如请求中有错误的语法或不能完成。

5xx 表示服务器的差错，如服务器失效无法完成请求。

Cookie 是这样工作的。当用户 A 浏览某个使用 Cookie 的网站时，该网站的服务器就为 A 产生一个唯一的识别码，并以此作为索引在服务器的后端数据库产生一个项目。接着再给 A 的 HTTP 响应报文中添加一个叫做 Set-cookie 的首部行。这里的“首部字段名”就是“Set-cookie”，而后面的“值”就是赋予该用户的 “识别码”。例如，这个首部行是这样的：

Set-cookie：31d4d96e407aad42

当 A 收到这个响应时，其浏览器就在它管理的特定 Cookie 文件中添加一行，其中包括**这个服务器的主机名**和 **Set-cookie 后面给出的识别码**。当 A **继续浏览这个网站**时，每发送一个 HTTP 请求报文，其浏览器就会从其 Cookie 文件中取出这个网站的识别码，并放到 HTTP 请求报文的 Cookie 首部行中：

Cookie：31d4d96e407aad42

于是，这个网站就能够跟踪用户 A 在该网站的活动。

动态文档是指文档的内容是在浏览器访问万维网服务器时才由应用程序动态创建的。当浏览器请求到达时，万维网服务器要运行另一个应用程序，并把控制转移到此应用程序。接着，该应用程序对浏览器发来的数据进行处理，并输出 HTTP 格式的文档，万维网服务器把应用程序的输出作为对浏览器的响应。由于对浏览器每次请求的响应都是临时生成的，因此用户通过动态文档所看到的内容

是不断变化的。

动态文档和静态文档之间的主要**差别体现在服务器一端**。这主要是文档内容的生成方法不同。而从浏览器的角度看，这两种文档并没有区别。

通用网关接口 CGI（Common Gateway Interface）是一种标准，它定义了动态文档应如何创建，输入数据应如何提供给应用程序，以及输出结果应如何使用。

CGI 程序的正式名称是 **CGI 脚本**。按照计算机科学的一般概念，“脚本”指的是一个程序，它**被另一个程序（解释程序）而不是计算机的处理机来解释或执行**。一个脚本运行起来比一般的编译程序要慢，因为它的每一条指令被另一个程序来处理（这就要一些附加的指令），而不是直接被指令处理器来处理。脚本不一定是一个独立的程序，它可以是一个动态装入的库，甚至是服务器的一个子程序。

有两种技术可用于浏览器屏幕显示的连续更新。一种技术称为**服务器推送**，这种技术是将所有的工作都交给服务器。服务器不断地运行与动态文档相关联的应用程序，定期更新信息，并发送更新过的文档。另一种提供屏幕连续更新的技术是**活动文档**。这种技术是把所有的工作都转移给浏览器端。每当浏览器请求一个活动文档时，服务器就**返回一段活动文档程序副本，使该程序副本在浏览器端运行**。这时，活动文档程序可与用户直接交互，并可连续地改变屏幕的显示。只要用户运行活动文档程序，活动文档的内容就可以连续地改变。由于活动文档技术不需要服务器的连续更新传送，对网络带宽的要求也不会太高。

全文搜索引擎是一种纯技术型的检索工具。它建立了一个很大的在线索引数据库供用户查询，**并不是实时地在互联网上检索到的信息**。因此很可能有些查到的信息已经是过时的。必须定期对已建立的数据库进行更新维护。

谷歌采用的 **PageRank** 技术把整个互联网当作了一个整体对待，检查整个网络链接的结构，并确定哪些网页重要性最高。如果有很多网站上的链接都指向页面 A，那么页面 A 就比较重要。**PageRank 对链接的数目进行加权统计**。

一个电子邮件系统由**用户代理**、**邮件服务器**、**邮件发送协议**（SMTP）和**邮件读取协议**（POP3）组成。

用户代理 UA 是用户与电子邮件系统的接口，在大多数情况下它就是运行在用户电脑中的一个程序。用户代理又称为**电子邮件客户端软件**。

邮件服务器必须能够同时充当客户和服务器。

SMTP、POP3（或 IMAP）都是使用 TCP 连接来传送邮件的，使用 TCP 的目的是为了可靠地传送邮件。

邮件不会在互联网中的某个中间邮件服务器落地。

在使用**网际报文存取协议 IMAP**（Internet Message Access Protocol）时，在用户的计算机上运行 IMAP 客户程序，然后与接收方的邮件服务器上的 IMAP 服务器程序建立 TCP 连接。用户在自己的计算机上就可以操纵邮件服务器的邮箱，就像在本地操纵一样，因此 IMAP 是一个**联机协议**。当用户计算机上的 IMAP 客户程序打开 IMAP 服务器的邮箱时，用户就可看到邮件的首部。**若用户需要打开某个邮件，则该邮件才传到用户的计算机上**。

发送人的用户代理向发送方邮件服务器发送邮件，以及发送方邮件服务器向接收方邮件服务器发送邮件，都是使用 SMTP 协议。而 **POP3 或 IMAP 则是用户代理从接收方邮件服务器上读取邮件所使用的协议**。

使用万维网电子邮件不需要在计算机中再安装用户代理软件。用户在浏览器中浏览各种信息时需要使用 HTTP 协议。因此，**在浏览器和互联网上的邮件服务器之间传送邮件时，仍然使用 HTTP 协议**。但是**在各邮件服务器之间传送邮件时，则仍然使用 SMTP 协议**。

通用互联网邮件扩充 MIME（Multipurpose Internet Mail Extensions）并没有改动或取代 SMTP。MIME 的意图是**继续使用原来的邮件格式，但增加了邮件主体的结构，并定义了传送非 ASCII 码的编码规则**。也就是说，MIME 邮件可在现有的电子邮件程序和协议下传送。

动态主机配置协议 DHCP（Dynamic Host Configuration Protocol）提供了一种**即插即用连网机制**，允许一台计算机加入新的网络和获取 IP 地址而不用手工参与。Windows 系统中的“自动获得 IP 地址”和“自动获得 DNS 服务器地址”，就表示使用 DHCP 协议。

当应用进程需要使用网络进行通信时，必须首先发出 socket 系统调用，请求操作系统为其**创建一个“套接字”**。这个调用的实际效果是请求操作系统把网络通信所需要的一些系统资源分配给该应用进行。操作系统为这些资源的总和用一个叫做**套接字描述符**的号码来表示，然后把这个套接字描述符返回给应用进程。此后，应用进程所进行的网络操作都必须使用这个套接字描述符。在处理系统调用的时候，通过套接字描述符，操作系统就可以识别出应该使用哪些资源来完成应用进程所请求的服务。通信完毕后，应用进程通过一个关闭套接字的 close 系统调用通知操作系统回收与该套接字描述符相关的所有资源。**套接字是应用进程为了获得网络通信服务而与操作系统进行交互时使用的一种机制**。

（1）**连接建立**阶段

当套接字被创建后，它的端口号和 IP 地址都是空的，因此**应用进程要调用 bind 来指明套接字的本地端口号和本地 IP 地址。在服务器端调用 bind 就是把熟知端口号和本地 IP 地址填写到已创建的套接字中**。这就叫做把本地地址**绑定**到套接字。在客户端也可以不调用 bind，这时由操作系统内核自动分配一个动态端口号（通信结束后由系统收回）。

服务器在调用 bind 后，还必须**调用 listen 把套接字设置为被动方式，以便随时接受客户的服务请求。UDP 服务器由于只提供无连接服务，不使用 listen 系统调用**。

服务器紧接着就调用 accept，以便把远地客户进程送来的连接请求提取出来。系统调用 accept 的一个变量就是要指明是从哪一个套接字发起的连接。

调用 accept 要完成的动作很多。这是因为一个服务器必须能够同时处理多个连接。这样的服务器常称为**并发方式工作的服务器**。**主服务器进程 M 一调用 accept，就为每一个新的连接请求创建一个新的套接字，并把这个新创建的套接字的标识符返回给发起连接的客户方**。与此同时，**主服务器进程还要创建一个从属服务器进程来处理新建立的连接**。这样，从属服务器进程用这个新创建的套接字和客户进程建立连接，而**主服务器进程用原来的套接字重新调用 accept，继续接受下一个连接请求**。在已建立的连接上，从属服务器进程就使用这个新创建的套接字传送和接收数据。数据通信结束后，从属服务器进程就关闭这个新创建的套接字，同时这个从属服务器也被撤销。

在任一时刻，服务器中总是有**一个主服务器进程**和**零个或多个从属服务器进程**。**主服务器进程用原来的套接字接受连接请求，而从属服务器进程用新创建的套接字和相应的客户建立连接并可进行双向传送数据**。

当使用 TCP 协议的客户已经调用 socket 创建了套接字后，客户进程就调用

connect，以便和远地服务器建立连接（这就是**主动打开**，相当于客户发出的连接请求）。在 connect 系统调用中，客户必须指明远地服务器的 IP 地址和端口号。

（2）**数据传送**阶段

客户和服务器都在 TCP 连接上使用 send 系统调用传送数据，使用 recv 系统调用接收数据。通常客户使用 send 发送请求，而服务器使用 send 发送回答。服务器使用 recv 接收客户用 send 调用发送的请求。客户在发完请求后用 recv 接收回答。

调用 send 需要三个变量：数据要发往的套接字的描述符、要发送的数据的地址以及数据的长度。通常 send 调用把数据复制到操作系统内核的缓存中。若系统的缓存已满，send 就暂时阻塞，直到缓存有空间存放新的数据。

调用 recv 也需要三个变量：要使用的套接字的描述符、缓存的地址以及缓存空间的长度。

（3）**连接释放**阶段

一旦客户或服务器结束使用套接字，就把套接字撤销。这时就调用 close 释放连接和撤销套接字。

计算机网络的通信面临两大类威胁，即**被动攻击**和**主动攻击**。

被动攻击是指攻击者从网络上窃听他人的通信内容。通常把这类攻击称为截获。在被动攻击中，攻击者只是观察和分析某一个协议数据单元 PDU 而不干扰信息流。

主动攻击有如下几种最常见的方式。

（1）**篡改**攻击者故意篡改网络上传送的报文。

（2）恶意程序**计算机病毒**、**计算机蠕虫**、 **特洛伊木马**、**逻辑炸弹**、**后门入侵**、**流氓软件**。

（3）**拒绝服务 DoS**

攻击者向互联网上的某个服务器不停地发送大量分组，使该服务器无法提供正常服务。若从互联网上的成百上千个网站集中攻击一个网站，则称为**分布式拒绝服务 DDoS**（Distributed Denial of Service）。

所谓**对称密钥密码体制**，即**加密密钥与解密密钥是使用相同的密码体制**。

公钥密码体制使用不同的加密密钥和解密密钥。公钥密码体制的产生主要有两个方面的原因，一是由于**对称密钥密码体制的密钥分配问题**，二是由于**对数字**

签名的需求。在公钥密码体制中，加密密钥 PK（公钥）是向公众公开的，而**解密密钥 SK（私钥）则是需要保密的**。加密算法 E 和解密算法 D 也都是公开的。

先后对 X 进行 D 运算和 E 运算或进行 E 运算和 D 运算，结果都是一样的。

数字签名必须保证能够实现以下三点功能：

（1）**报文鉴别**：接收者能够核实发送者对报文的签名。

（2）**报文的完整性**：接收者确信所收到的数据和发送者发送的完全一样而没有被篡改过。

（3）**不可否认**：发送者事后不能抵赖对报文的签名。

A 用自己的私钥加密，之后发送，B 用公钥解密。——缺失了保密性。

A 用自己的私钥加密，再用 B 的公钥加密，之后发送，B 用自己的私钥解密，再用 A 的公钥解密。——实现了数字签名的保密性。

对称密钥的分配：**密钥分配中心 KDC**（Key Distribution Center）。

非对称密钥的分配：**认证中心 CA**（Certification Authority）

运输层的安全协议有 **SSL（安全套接字层）**和 **TLS（运输层安全）**。

防火墙是一种特殊编程的**路由器**，安装在一个网点和网络的其余部分之间，目的是实施访问控制策略。防火墙里面的网络称为“**可信的网络**”，而把防火墙外面的网络称为“**不可信的网络**”。防火墙的功能有两个：一个是**阻止**，另一个是**允许**。防火墙技术分为：**网络级防火墙**，用来防止整个网络出现外来非法的入侵（**分组过滤**和**授权服务器**）；**应用级防火墙**，用来进行访问控制（用**应用网关**或**代理服务器**来区分各种应用）。

入侵检测系统 IDS（Instrusion Detection System）是在入侵已经开始，但还没有造成危害或在造成更大危害前，及时检测到入侵，以便尽快阻止入侵，把危害降低到最小。

响应状态码	含义
200 OK	请求已正常处理。
204 No Content	请求处理成功，但没有任何资源可以返回给客户端，一般在只需要从客户端往

	服务器发送信息，而对客户端不需要发送新信息内容的情况下使用。
206 Partial Content	是对资源某一部分的请求，该状态码表示客户端进行了范围请求，而服务器成功执行了这部分 GET 请求。响应报文中包含由 Content-Range 指定范围的实体内容。
301 Moved Permanently	资源的 URI 已更新。永久性重定向，请求的资源已经被分配了新的 URI，以后应使用资源现在所指的 URI。
302 Found	资源的 URI 已临时定位到其他位置了。临时性重定向。和 301 相似，但 302 代表的资源不是永久性移动，只是临时性性质的。已移动的资源对应的 URI 将来还有可能发生改变。
303 See Other	资源的 URI 已更新。该状态码表示用于请求对应的资源存在着另一个 URL，应使用 GET 方法定向获取请求的资源。303 状态码和 302 状态码有着相同的功能，但 303 状态码明确表示客户端应当采用 GET 方法获取资源。
304 Not Modified	资源已找到，但未符合条件请求。该状态码表示客户端发送附带条件的请求时（采用 GET 方法的请求报文中包含 If-Match，If-Modified-Since，If-None-Match，If-Range，If-Unmodified-Since 中任一首部）服务端允许请求访问资源，但因发生请求未满足条件的情况后，直接返回 304。
307 Temporary Redirect	临时重定向。与 302 有相同的含义。
400 Bad Request	服务器端无法理解客户端发送的请求，请求报文中可能存在语法错误，如 HTTP1.1 不带 Host 首部。
401 Unauthorized	该状态码表示发送的请求需要有通过 HTTP 认证的认证信息。
403 Forbidden	不允许访问那个资源。该状态码表明对请求资源的访问被服务器拒绝了。（权限，未授权 IP 等）
404 Not Found	服务器上没有请求的资源。路径错误等。
500 Internal Server Error	貌似内部资源出故障了。该状态码表明服务器端在执行请求时发生了错误。也有可能是 Web 应用存在 BUG 或某些临时故障。
503 Service Unavailable	抱歉，我现在正在忙着。该状态码表明服务器暂时处于超负载或正在停机维护，现在无法处理请求。

HTTP1.1 和 HTTP1.0 的区别：

（1）**缓存处理**。在 HTTP1.0 中主要使用首部的 **If-Modified-Since**，**Expire** 来作为缓存判断的标准，HTTP1.1 则引入了**更多的缓存控制策略**例如 If-Unmodified-Since，If-Match，If-None-Match 等更多可供选择的缓存头来控制缓存策略。

（2）**带宽优化及网络连接的使用**。HTTP1.0 中，存在一些浪费带宽的现象，例如客户端只是需要某个对象的一部分，而服务器却将整个对象送过来了，并且不支持断点续传功能，HTTP1.1 则在请求头引入了 range 头域，它允许只请求资源的某个部分，即返回码是 206（Partial Content），这样就方便了开发者自由地选择以便于充分利用带宽和连接。

（3）**错误通知的管理**。在 HTTP1.1 中新增了 24 个错误状态码，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地删除。

（4）**Host 头处理**。在 HTTP1.0 中认为每台服务器都锁定一个唯一的 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）。

（5）**长连接**。HTTP1.1 支持长连接和请求的流水线处理，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，在 HTTP1.1 中默认开启 Connection：keep-alive，一定程度上弥补了 HTTP1.1 每次请求都要创建连接的缺点。