

网络把许多计算机连接在一起，而**互连网**则把许多网络通过**路由器**连接在一起。与网络相连的计算机常称为**主机**。

internet（互连网）是一个通用名词，它泛指由多个计算机网络互连而成的计算机网络。

Internet（互联网/因特网）是一个专用名词，它指当前全球最大的、开放的、由众多网络相互连接而成的特定互连网，它采用 **TCP/IP** 协议族作为通信的规则，且其前身是美国的 **ARPANET**。

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**。只要两台主机都运行了对等连接软件（P2P 软件），它们就可以进行平等的、对等连接通信。实际上，对等连接方式从本质上看仍然是使用客户-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。

路由器是实现**分组交换**的关键构件，其任务是转发收到的分组，这是网络核心部分最重要的功能。

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（message）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**（header）后，就构成了一个**分组**（packet）。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。

互联网的核心部分是由许多网络和把它们互连起来的路由器组成的，而主机处在互联网的边缘部分。**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。**

位于网络边缘的主机和位于网络核心部分的路由器都是计算机，但它们的作用却很不一样。**主机是为用户进行信息处理的**，并且可以和其他主机通过网络交换信息。**路由器则是用来转发分组的，即进行分组交换的**。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**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（即**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，好像在一个管道中传送。

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

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

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**。

在计算机网络中，**带宽**用来表示网络中某通道传送数据的能力，因此网络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**”。这种带宽的单位就是数据率的单位 bit/s，是“比特每秒”。

吞吐量 (throughput) 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** (transmission delay)：**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** (propagation delay)：**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（或高带宽链路）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。因此，**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。因此，**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被利用的（有数据通过）。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。信道利用率并非越高越好。信道利用率过

高会产生非常大的时延。

为进行网络中的数据交换而建立的规则、标准或约定称为**网络协议**。网络协议由以下三个要素组成：

- (1) **语法**，即数据与控制信息的结构或格式。
- (2) **语义**，即需要发出何种控制信息，完成何种动作以及做出何种响应。
- (3) **同步**，即事件实现顺序的详细说明。

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由不同的应用层协议**。在互联网中的应用层协议很多，如域名系统 DNS，支持万维网应用的 HTTP 新协议，支持电子邮件的 SMTP 协议，等等。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**（segment）。
- (2) 用户数据报协议 UDP——提供**无连接的、尽最大努力**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**，或简称为**数据报**。网络层的另一个任务就是要**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（Internet Protocol）和许多种路由选择协议，因此互联网的网络层也叫做网际层或 IP 层。

数据链路层：简称为链路层。两台主机之间的数据传输，总是在一段一段地

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是否有差错。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。这种方法会使数据链路层的协议复杂些。

物理层：在物理上所传数据的单位是比特。发送方发送 1（或 0）时，接收方应当收到 1（或 0）而不是 0（或 1）。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、同轴电缆、光缆、无线通道等，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，然后再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**。也就是说，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)，它可以与 PDU 不一样。例如，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)：通信的双端可以同时发送和接收信息。

频分复用的所有用户在同样的时间占用不同的带宽资源。

时分复用的所有用户是在不同的时间占用同样的频带宽度。

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，因此过程比较复杂。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—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—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。—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在进行通信时的主要步骤如下：

- （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（framing）就是在—段数据的前后分别添加首部和尾部，这样就构成了—个帧。—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—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—帧的最前面，表示帧的首部开始**。另—个**控制字符 EOT（End Of Transmission）表示帧的结束**。请注意，SOH 和 EOT 都是控制字符的名称。它们的十六进制编码分别是 01（二进制是 00000001）和 04（二进制是 00000100）。SOH（或 EOT）并不是 S，O，H（或 E，O，T）三个字符。

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—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—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。

当传送的帧时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”**（其十六进制编码是 1B，二进制是 00011011）。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。这种方法称为“字节填充”或“字符填充”。如果转义字符也出现在数据当中，那么解决办法仍然是在转义字符的前面插入一个转义字符。因此，当接收端收到连续的两个转义字符时，就删除其中前面的一个。

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（例如，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。

点对点协议 PPP（Point-to-Point）是目前使用得最广泛的数据链路层协议。

计算机与外界局域网的连接是通过通信**适配器**（adapter）进行的。适配器本来是在主机箱内插入的一块**网络接口板**（或者是在笔记本电脑中插入一块 PCMCIA 卡——个人计算机存储卡接口适配器）。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。现在的电子技术可以很容易做到：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。因此，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。例如，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）的要点归纳如下：

(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。**

集线器的一些特点如下:

(1) 使用集线器的以太网在逻辑上仍是一个总线网, 各站共享逻辑上的总线, 使用的还是 **CSMA/CD 协议** (更具体地说, 是各站中的适配器执行 CSMA/CD 协议)。网络中的各站必须竞争对传输媒体的控制, 并且**在同一时刻至多只允许一个站发送数据**。

(2) 一个集线器有许多接口, 每个接口通过 RJ-45 插头用两对双绞线与一台计算机上的适配器相连。因此, 一个集线器很像**一个多接口的转发器**。

(3) 集线器工作在物理层, 它的每个接口仅仅简单地转发比特——收到 1 就转发 1, 收到 0 就转发 0, 不进行碰撞检测。若两个接口同时有信号输入 (即发生碰撞), 那么所有的接口都将收不到正确的帧。

(4) 集线器采用了专门的芯片, 进行**自适应串音回波抵消**。这样就可**使接口转发出去的较强信号不致对该接口收到的较弱信号产生干扰** (这种干扰即近端串音)。每个比特在转发之前还要再进行再生整形并重新定时。

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**。因此,

(1) 假定连接在局域网上的一台计算机的适配器坏了而我们更换了一个新的适配器，那么这台计算机的局域网的“地址”也就改变了，虽然这台计算机的地理位置一点也没有变化，所接入的局域网也没有任何改变。

(2) 假定我们把位于南京的某局域网上的一台笔记本电脑携带到北京，并连接在北京的某局域网上。虽然这台电脑的地理位置变了，但只要电脑中的适配器不变，那么该电脑在北京的局域网中的“地址”仍然和它在南京的局域网中的“地址”一样。

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。

在物理层扩展以太网：

在过去使用粗缆或细缆以太网时，常使用工作在物理层的**转发器**来扩展以太网的地理覆盖范围。那时，两个网段可用一个转发器连接起来。

如果使用多个集线器，就可以连接成覆盖更大范围的多级星形结构的以太网。但这种多级结构的集线器以太网也带来了一些缺点：

(1) 当某个系的两个站在通信时所传送的数据会通过所有的集线器进行转发，使得其他系的内部在这时都不能通信（一发送数据就会碰撞）。

(2) 如果不同的系使用不同的以太网技术（如数据率不同），那么就不可能用集线器将它们互联起来。如果一个系使用 10Mbit/s 的适配器，而另外两个系使用 10/100Mbit/s 的适配器，那么用集线器连接起来后，大家都只能工作在 10Mbit/s 的速率。**集线器基本上是个多接口的转发器，它并不能把帧进行缓存。**

以太网交换机实质上就是一个**多接口的网桥**，通常都有十几个或更多的接口，和工作在物理层的转发器、集线器有很大的差别。以太网交换机的每个接口都直接与一个单台主机或另一个以太网交换机相连，并且一般都工作在全双工方式。以太网交换机还具有**并行性**，即能同时连通多对接口，使多对主机能同时通信（而网桥只能一次分析和转发一个帧）。相互通信的主机都是**独占传输媒体**，无碰撞地传输数据。

以太网交换机的接口还有存储器，能在输出端口繁忙时把到来的帧进行缓存。

以太网交换机是一种即插即用的设备，其内部的帧交换表（又称为地址表）是通过自学习算法自动地逐渐建立起来的。以太网交换机由于使用了专用的交换结构芯片，用硬件转发，其转发速率要比使用软件转发的网桥快很多。

对于传统的 10Mbit/s 的共享式以太网，若有 10 个用户，则每个用户占有的平均带宽只有 1Mbit/s。若使用以太网交换机来连接这些主机，虽然在每个接口到主机的带宽还是 10Mbit/s，但**只有一个用户在通信时时独占而不是和其他网络用户共享传输媒体的带宽**，因此对于拥有 10 个接口的交换机的总容量则为 100Mbit/s。这正是交换机的最大优点。

虽然许多以太网交换机对收到的帧采用**存储转发**方式进行转发，但也有一些交换机采用直播的交换方式。直通交换不必把整个数据帧先缓存后再进行处理，而是在接收数据帧的同时就立即按数据帧的目的 MAC 地址决定该帧的转发接口，因而提高了帧的转发速度。直通交换的一个缺点是它不检查差错就直接把帧转发出去，因此有可能也将一些无效的帧转发给其他的站。

虚拟局域网 VLAN (Virtual LAN) 是**由一些局域网网段构成的与物理位置无关的逻辑组**，而这些网段具有某些共同的需求。每一个 VLAN 的帧都有一个明确的标识符，指明发送这个帧的计算机属于哪一个 VLAN。

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叫做**转发器** (repeater)。
- (2) 数据链路层使用的中间设备叫做**网桥或桥接器** (bridge)。
- (3) 网络层使用的中间设备叫做**路由器** (router)。

(4) 在网络层以上使用的中间设备叫做**网关** (gateway)。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内是唯一的**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** (net-id), 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** (host-id), 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**。

这种两级的 IP 地址可以记为：

IP 地址 :: = {<网络号>, <主机号>}

A 类、B 类和 C 类地址都是**单播**地址 (一对一通信), 是最常用的。

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, IP 地址中全为 0 表示 “这个 (this)”。**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 (即 01111111) 保留作为**本地软件环回测试** (loopback test) 本主机的进程之间的通信之用。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** (例如, 一主机的 IP 地址为 5.6.7.8, 则该主机所在的网络地址就是 5.0.0.0), 而全 1 表示 “所有的 (all)”, 因此**全 1 的主机号字段表示该网络上的所有主机**。

IP 地址空间公有 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 (10) 已经固定了, 只剩下 14 位可以进行分配。因为网络号字段后面的 14 位无论怎样取值也不可能出现使

整个 2 字节的网络号字段成为全 0 或全 1，因此这里不存在网络总数减 2 的问题。但实际上 **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 (110)，还有 21 位可以进行分配。C 类网络地址 192.0.0.0 也是不指派的，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（第一级），而剩下的主机号（第二级）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（而不考虑主机号）**，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，因此，**用转发器或网桥连接起来的若干个局域网仍未一个网络，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 (ARP cache)，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样地址呢？我们可以通过下面的例子来说明。

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，就再 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。

当主机 A 向主机 B 发送数据报时，有可能以后不久主机 B 还要向主机 A 发送数据报，因而主机 B 也可能要向 A 发送 ARP 请求分组。为了减少网络上的通

信量，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。**以后主机 B 向 A 发送数据报时就方便了。

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**（例如从路由器 R1 转发到 R2，再转发到 R3，然后又转发到 R1），因而白白消耗网络资源。

“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。

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。

分组转发算法如下：

- (1) 从数据报的首部提取目的主机的 IP 地址 D，得出目的网络地址为 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**，不需要再经过其他的路由器，直接把数据报交付目的主机（这里包括把目的主机地址 D 转换为具体的硬件地址，把数据报封装为 MAC 帧，再发送此帧）；否则就是间接交付，执行 (3)。

(3) 若路由表中有目的地址为 D 的**特定主机路由**，则把数据报传送给路由表中所指明的下一跳路由器；否则，执行 (4)。

(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；否则，执行 (5)。

(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；否则，执行 (6)。

(6) 报告转发分组出错。

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。于是两级 IP 地址在本单位内部就变为三级 IP 地址：网络号、子网号和主机号：

IP 地址： $::= \{ \langle \text{网络号} \rangle, \langle \text{子网号} \rangle, \langle \text{主机号} \rangle \}$

(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？

我们知道，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。现在互联网的标准规定：**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。

在划分子网的情况下，路由器转发分组的算法如下：

- (1) 从收到的数据报的首部提取目的 IP 地址 D。
- (2) 先判断是否为直接交付。对路由器直接相连的网络逐个进行检查：用各网络的子网掩码和 D 逐位相“与”，看结果是否和相应的网络地址匹配。若匹配，则把分组进行直接交付（当然还需要把 D 转换成物理地址，把数据封装成帧发送出去），转发任务结束。否则就是间接交付，执行 (3)。
- (3) 若路由表中有目的地址为 D 的特定主机路由，则把数据报传送给路由表中所指明的下一跳路由器；否则，执行 (4)。
- (4) 对路由表中的每一行（目的网络地址，子网掩码，下一跳地址），用其中的子网掩码和 D 逐位相“与”，其结果为 N。若 N 与该行的目的网络地址匹配，则把数据报传送给该行指明的下一跳路由器；否则，执行 (5)。
- (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；否则，执行 (6)。
- (6) 报告转发分组出错。

无分类域间路由选择 CIDR (Classless Inter-Domain Routing) 最主要的特点有两个：

- (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，并且在新的 IPv6 使用之前容许互联网的规模继续增长。CIDR 把 32 位的 IP 地址划分为前后两个部分。前面部分是“网络前缀”，用来指明网络，后面部分则用来指明主机。因此 CIDR 使 IP 地址从三级编址（使用子网掩码）又回到了两级编址，但这已是无分类的两级编址。其记法是：

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。例如, 已知 IP 地址 128.14.35.7/20 是某 CIDR 地址块中的一个地址, 现在把它写成二进制表示, 其中的前 20 位是网络前缀, 而前缀后面的 12 位是主机号:

128.14.35.7/20 = 10000000 00001110 00100011 00000111

这个地址所在的地址块中的最小地址和最大地址可以很方便地得出:

最小地址: 128.14.32.0 10000000 00001110 00100000 00000000

最大地址: 128.14.47.255 10000000 00001110 00101111 11111111

为了方便地进行路由选择, CIDR 使用 32 位的**地址掩码** (address mask)。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。虽然 CIDR 不使用子网了, 但由于目前仍有一些网络还使用子网划分和子网掩码, 因此 CIDR 使用的地址掩码也可继续成为子网掩码。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** (route aggregation), 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** (Internet Control Message Protocol)。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，处于效率的考虑，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。

倘若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, 则只有两大类, 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**, 其特点是简单和开销较小, 但不能及时适应网络状态的变化。对于很简单的小网络, 完全可以采用静态路由选择, 用人工配置每一条路由。动态路由选择也叫做**自适应路由选择**, 其特点是能较好地适应网络状态的变化, 但实现起来较为复杂, 开销也比较大。因此, 动态路由选择适用于较复杂的大网络。

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。

为此, 可以把整个互联网划分为许多较小的**自治系统** (autonomous system), 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** (Interior Gateway Protocol)

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“**距离相量**”)。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。

对每一个相邻路由器发送过来的 RIP 报文，进行以下步骤：

(1) 对地址为 X 的相邻路由器发来的 RIP 报文，先修改此报文中的所有项目：把“下一跳”字段中的地址都改为 X，并把所有的“距离”字段的值加 1。每一个项目都有三个关键数据，即：到目的网络 N，距离是 d，下一跳路由器是 X。

(2) 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 (即在路由表中有目的网络 N，这时就再查看下一跳路由器地址)

若下一跳路由器地址是 X，则把收到的项目替换原路由表中的项目。

否则 (即这个项目是：到目的网络，但下一跳路由器不是 X)

若收到的项目中的 d 小于路由表中的距离，则进行更新。

否则什么也不做。

(3) 若 3 分钟还没有收到相邻路由器的更新路由表，则把此相邻路由器记为不可达的路由器，即把距离置为 16。

(4) 返回。

上面给出的距离向量算法的基础就是 Bellman-Ford 算法。这种算法的要点是这样的：

设 X 是结点 A 到 B 的最短路径上的一个结点。若把路径 $A \rightarrow B$ 拆成两段路径 $A \rightarrow X$ 和 $X \rightarrow B$ ，则每一段路径 $A \rightarrow X$ 和 $X \rightarrow B$ 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 协议存在一个问题：**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** (flooding)，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** (link-state database)，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 (这称为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，等等。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** (area)。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** (backbone area)。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** (area border router) 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（不能兜圈子），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（即目的网络），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**”。

请注意“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线，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。这就是说，**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。在讨论路由选择的原理时，往往不去区分转发表和路由表的区别，而可以笼统地都使用路由表这一名词。

三种常用的交换结构：(1) 通过**存储器**。(2) 通过**总线**。(3) 通过**互连网络**。

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。这样大的地址空间在可预见的将来是不会用完的。

(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，这时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** (即首部长度的必须是 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** (base header) 和后面的**有效载荷** (payload)。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 (extension header), 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。这可以有两种办法。(1) 租用电信公司的通信线路为本机构专用。这种方法虽然简单方便，但线路的租金太高，一般难于承受。(2) 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。按照 OSI (Open Systems Interconnection Reference Model) 的术语，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** (segment) 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	应用层协议	运输层协议
名字转换	DNS（域名系统）	UDP
文件传送	TFTP（简单文件传送协议）	UDP
路由选择协议	RIP（路由信息协议）	UDP
IP 地址配置	DHCP（动态主机配置协议）	UDP
网络管理	SNMP（简单网络管理协议）	UDP
远程文件服务器	NFS（网络文件系统）	UDP
IP 电话	专用协议	UDP
流式多媒体通信	专用协议	UDP
多播	IGMP（网际组管理协议）	UDP
电子邮件	SMTP（简单邮件传送协议）	TCP
远程终端接入	TELNET（远程终端协议）	TCP
万维网	HTTP（超文本传送协议）	TCP
文件传送	FTP（文件传送协议）	TCP

应用层所有的应用进程都可以通过运输层再传送到 IP 层（网络层），这就是**复用**。运输层从 IP 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，这就是**分用**。

协议端口号（protocol port number）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（当然，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。这对某些实时应用是很重要的。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**。也就是说，应用进程之间的通信好像在“打电话”：通话前要先拨号建立连接，通话结束后要挂机释放连接。

(2) 每一条 TCP 连接只能有**两个端点** (endpoint)，每一条 TCP 连接只能是点对点的（**一对一**）。

(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和识别收到的字节流，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楚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。

累积确认有优点也有缺点。优点是：容易实现，即使确认丢失也不必重传。但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 (ACKnowledgement)，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 (ReSeT)，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 (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，则在响应的**

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

终止 FIN (FINis)，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 (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**。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，这里都还没有考虑首部中的选项部分)，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需暂时保留，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，但这样做对网络资源的利用不利（因为发送方会重复传送较多的数据）。因此**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。但请注意两点。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。TCP 标准规定,确认推迟的时间不应超过 0.5 秒。若收到一连串具有最大长度的报文段,则必须每隔一个报文段就发送一个确认。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。

若收到的确认是对重传报文段的确认,但却被源主机当成是对原来的报文段的确认,则这样计算出的 RTTs 和超时重传时间 RTO (RetransmissionTime-Out) 就会偏大。若后面再发送的报文段又是经过重传后才收到确认报文段,则按此方法得出的超时重传时间 RTO 就越来越长。

同样,若收到的确认是对原来的报文段的确认,但被当成是对重传报文段的确认,则由此计算出的 RTTs 和 RTO 都会偏小。这就必然导致报文段过多地重传。这样就有可能使 RTO 越来越短。

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 (flow control) 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** (persistence timer)。只要 TCP 连接的一方收到对方的零窗口通知,就启动持续计数器。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** (仅携带 1 字节的数

据)，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**。算法如下：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 (silly window syndrome)。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地数据注入到网络中，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（例如，输入到网络的某些分组被某个结点丢弃了）。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。也就是说，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。更值得注意的是，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。

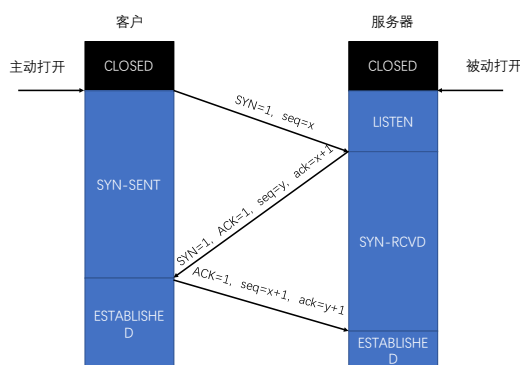
采用**快重传**算法可以**让发送方尽早知道发生了个别报文段的丢失**。快重传算法首先要求接收方不要等待自己发送数据时才进行捎带确认，而是要立即发送确认，**即使收到了失序的报文段也要立即发出对已收到的报文段的重复确认**。快重传算法规定，**发送方只要一连收到 3 个重复确认，就知道接收方确实没有收到报**

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= $\text{Min}[\text{rwnd}, \text{cwnd}]$ ，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。应当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。



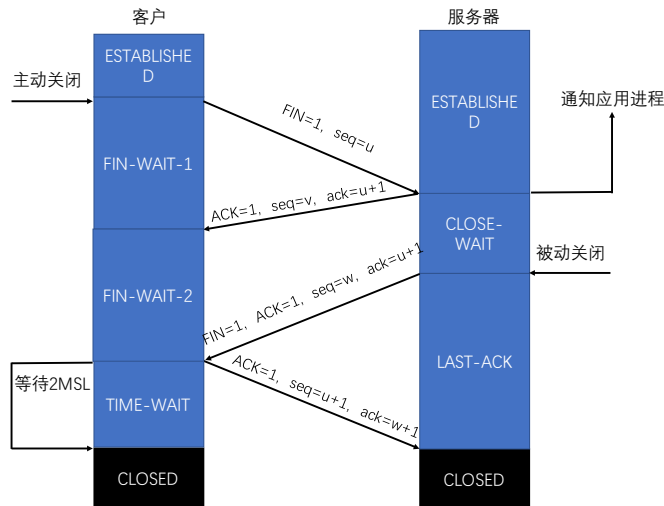
为什么客户端最后还要发送一次确认呢？这主要是为了防止已失效的连接请求报文段突然又传送到了 B，因而产生错误。

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”。

先假定出现一种异常情况，即**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？这两个理由。

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 还有一个**保活计时器** (keepalive timer)。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。总之，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。**

域名系统 DNS-----UDP

文件传送协议 FTP-----TCP

网络文件系统 NFS-----UDP

简单文本传送协议 TFTP-----UDP

远程终端协议 TELNET-----TCP

万维网 WWW (World Wide Web) 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（也就是所谓的“**链接到另一个站点**”，从而主动地按需获取丰富的信息。

万维网使用**统一资源定位符 URL** (Uniform Resource Locator) 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。

URL 的一般形式由以下四个部分组成：

<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，它是万维网上能够可靠地交换文件（包括文本、声音、图像等各种多媒体文件）的重要基础。

HTTP 协议本身是**无连接**的。这就是说，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。也就是说，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（假定现在服务器还没有把该页面更新），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（即经过了一个 RTT 时间后），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（与文档大小成正比）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，使同一个客户（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**。

HTTP 有两类报文：请求报文、响应报文。

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。可以看出，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。首部可以由好几行，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（操作）	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。

Cookie 只是一个小小的文本文件，不是计算机的可执行程序，因此不可能传播计算机病毒，也不可能用来获取用户计算机硬盘中的信息。

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。

这里增加了一个机制，叫做**通用网关接口 CGI** (Common Gateway Interface)。

CGI 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** (script)。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。它的工作原理是通过搜索软件（例如一种叫做“蜘蛛”或“网络机器人”的 Spider 程序）到互联网上的各网站收集信息，找到一个网站后可以从这个网站再链接到另一个网站，像蜘蛛爬行一样。然后按照一定的规则建立一个很大的在线索引数据库供用户查询。用户在查询时只要输入关键词，就从已经建立的索引数据库里进行查询（**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（例如很多年前的）。建立这种索引数据库的网站必须定期对已建立的数据库进行更新维护。现在全球最大的并且最受欢迎的全文搜索引擎就是谷歌 Google。

分类目录搜索引擎并不采集网站的任何信息，而是利用各网站向搜索引擎提交网络信息时填写的关键词和网站描述等信息，经过人工审核编辑后，如果认为符合网站登录的条件，则输入到分类目录的数据库中，供网上用户查询。

垂直搜索引擎针对某一特定领域、特定人群或某一特定需求提供搜索服务。垂直搜索也是提供关键字来进行搜索的，但被放到了一个行业知识的上下文中，

返回的结果更倾向于信息、消息、条目等。例如，对买房的人讲，它希望查找的是房子的具体供求信息（如面积、地点、价格等），而不是有关房子供求的一致性的论文或新闻、政策等。

元搜索引擎把用户提交的检索请求发送到多个独立的搜索引擎上去搜索，并把检索结果集中统一处理，以统一的格式提供给用户，因此是搜索引擎上的搜索引擎。

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。更具体些，就是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（User Agent）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。因此用户代理又称为**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。

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。**用户可以根据需要为自己的邮箱创建便于分类管理的层次式的邮箱文件夹，并且能够将存放的邮件从某一个文件夹中移动到另一个文件夹中。用户也可按某种条件对邮件进行查找。在用户未发出删除邮件的命令之前，IMAP 服务器邮箱中的邮件一直保存着。

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再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。

三种常用的内容传送编码：

- (1) **7 位 ASCII 码**。
- (2) **quoted-printable**，这种编码方法适用于所传送的数据中只有少量的非 ASCII 码，例如汉字。
- (3) **base64 编码**。

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(客户或服务器)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(存储器空间、CPU 空间、网络带宽等)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(小的整数)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(建立连接、收发数据、调整网络通信参数等)都必须使用这个套接字描述符。所以，几乎所有的网络系统调用都把这个套接字描述符

作为套接字的许多参数中的第一个参数。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。由此可见，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（绑定）来指明套接字的本地地址（本地端口号和本地 IP 地址）。在服务器端调用 bind 时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（收听）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（接受），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。这时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。

总之，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，客户必须指明远地端点（即远地服务器的 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新系统内核的缓存中。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**，一种会“传染”其他程序的程序，“传染”是通过修改其他程序来把自身或自己的变种复制进去而完成的。
- **计算机蠕虫**，一种通过网络的通信功能将自身从一个结点发送到另一个结点并自动启动运行的程序。
- **特洛伊木马**，一种程序，它执行的功能并非所声称的功能而是某种恶意功能。
- **逻辑炸弹**，一种当运行环境满足某种特定条件时执行其他特殊功能的程序。
- **后门入侵**，是指利用系统实现中的漏洞通过网络入侵系统。
- **流氓软件**，一种未经用户允许就在用户计算机上安装运行并损害用户利益的软件。

(3) 拒绝服务 DoS (Denial of Service)

攻击者向互联网上的某个服务器不停地发送大量分组，使该服务器无法提高

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)。有时也称这种攻击为**网络带宽攻击**或**连通性攻击**。

安全的计算机网络：(1) **保密性**；(2) **端点鉴别**；(3) **信息的完整性**；(4) **运行的安全性**。

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**。**数据加密标准 DES** (Data Encryption Standard) 属于对称密钥密码体制。**DES 的保密性仅取决于对密钥的保密，而算法是公开的**。

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 (public key, 即公钥) 是向公众公开的，而**解密密钥 SK (secret key, 即私钥或秘钥) 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。

鉴别是要验证通信的对方的确是自己所要通信的对象，而不是其他的冒充者，并且所传送的报文是完整的，没有被其他人篡改过。有时可再把鉴别细分为两种。一种是**报文鉴别**，即鉴别所收到的报文的确是报文的发送者所发送的，而不是其他人伪造的或篡改的。这就包含了**端点鉴别**和**报文完整性的鉴别**。另一种则是**实体鉴别**，即仅仅鉴别发送报文的实体。实体可以是一个人，也可以是一个进程。这就是端点鉴别。

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)

运输层的安全协议由 **SSL (安全套接字层)** 和 **TLS (运输层安全)**。SSL 最新的版本是 SSL 3.0, 它是保护万维网 HTTP 通信量所公认的事实上的标准。SSL 不仅被所有常用的浏览器和万维网服务器所支持, 而且也是 TLS 的基础。

PGP (Pretty Good Privacy) 是一个完整的电子邮件安全软件包, 包括加密、鉴别、电子签名和压缩等技术。PGP 并没有使用什么新的概念。它只是把现有的一些加密算法 (如 RSA 公钥加密算法或 MD5 报文摘要算法) 综合在一起而已。

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** (主要的), 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 (属于这类的有**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 (Intrusion 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。