# 应用层

进程通过套接字的软件接口向网络发送报文和从网络接收报文。套接字是同一台主机应用层与传输层之间的接口。

## 1. Web和HTTP

Web的应用层协议是超文本传输协议HTTP。HTTP由客户程序和服务器程序实现，客户程序和服务器程序运行在不同的端系统中，通过交换HTTP报文进行会话。HTTP定义了这些报文的结构以及客户和服务器进行报文交换的方式。HTTP定义了Web客户向Web服务器请求Web页面的方式，以及服务器向客户传送Web页面的方式。

**HTTP使用TCP传输协议，端口号80**。因为HTTP服务器并不保存关于客户的任何信息，所以我们说HTTP是一个无状态协议。

Web缓存器也叫代理服务器，它是能够代表初始Web服务器来满足HTTP请求的网络实体。Web缓存器有自己的磁盘存储空间，并在存储空间中保存最近请求过的对象的副本。

### HTTP与HTTPS

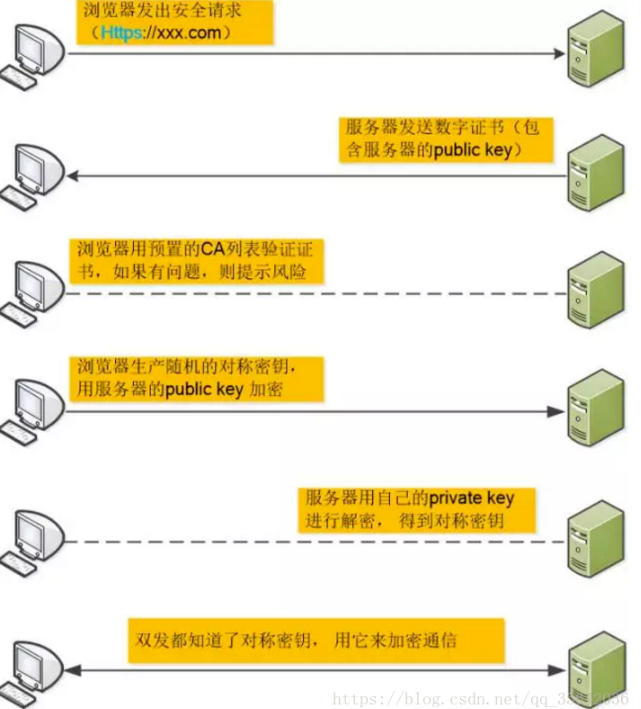
超文本传输协议HTTP协议被用于在Web浏览器和网站服务器之间传递信息，HTTP协议以明文方式发送内容，不提供任何方式的数据加密，如果攻击者截取了Web浏览器和网站服务器之间的传输报文，就可以直接读懂其中的信息，因此，HTTP协议不适合传输一些敏感信息，比如：信用卡号、密码等支付信息。

为了解决HTTP协议的这一缺陷，需要使用另一种协议：安全套接字层超文本传输协议HTTPS。为了数据传输的安全，HTTPS在HTTP的基础上加入了SSL/TLS协议，SSL/TLS依靠证书来验证服务器的身份，并为浏览器和服务器之间的通信加密。

HTTPS协议是由SSL/TLS+HTTP协议构建的可进行加密传输、身份认证的网络协议，要比http协议安全。

HTTPS协议的主要作用可以分为两种：一种是建立一个信息安全通道，来保证数据传输的安全；另一种就是确认网站的真实性。

### 客户端在使用HTTPS方式与Web服务器通信时的步骤



1. 客户使用https的URL访问Web服务器，要求与Web服务器建立SSL连接。
2. Web服务器收到客户端请求后，会将网站的证书信息（证书中包含公钥）传送一份给客户端。
3. 客户端的浏览器与Web服务器开始协商SSL/TLS连接的安全等级，也就是信息加密的等级。
4. 客户端的浏览器根据双方同意的安全等级，建立会话密钥，然后利用网站的公钥将会话密钥加密，并传送给网站。
5. Web服务器利用自己的私钥解密出会话密钥。
6. Web服务器利用会话密钥加密与客户端之间的通信。

### 面试题

##### HTTP协议和HTTPS协议区别？

1. HTTP协议是以明文的方式在网络中传输数据，而HTTPS协议传输的数据则是经过TLS加密后的，HTTPS具有更高的安全性
2. HTTPS在TCP三次握手阶段之后，还需要进行SSL 的handshake，协商加密使用的对称加密密钥
3. HTTPS协议需要服务端申请证书，浏览器端安装对应的根证书
4. HTTP协议端口是80，HTTPS协议端口是443

## 2. 文件传输协议FTP

FTP协议是Internet文件传送的基础，它由一系列规格说明文档组成，目标是提高文件的共享性，提供非直接使用远程计算机，使存储介质对用户透明和可靠高效地传送数据。简单的说，FTP就是完成两台计算机之间文件的拷贝。

当FTP的服务器端从该连接上收到一个文件传输的命令后(无论是向还是来自远程主机)，就发起一个到客户端的TCP数据连接。FTP在该数据连接上准确地传送一个文件，然后关闭该连接。在同一个会话期间，如果用户还需要传输另一个文件，FTP则打开另一个数据连接。**因而对FTP传输而言，控制连接贯穿了整个用户会话期间，但是对会话中的每一次文件传输都需要建立一个新的数据连接(即数据连接是非持续的)**。

**FTP运行在TCP之上**，使用了两条TCP连接：控制连接和数据连接。

* 控制连接：端口21，在两个主机之间传输控制信息，如用户标识、口令、命令（改变远程目录、存放命令、获取命令等）。称FTP的控制信息是带外传递的。
* 数据连接：端口20，用于实际发送一个文件。

### 2.2. FTP用户授权

要连上FTP服务器（即“登陆”），必须要有该FTP服务器授权的帐号，也就是说只有在有了一个用户标识和一个口令后才能登陆FTP服务器，享受FTP服务器提供的服务。

### 2.3. 匿名FTP

互连网中有很大一部分FTP服务器被称为“匿名”FTP 服务器。这类服务器的目的是向公众提供文件拷贝服务，不要求用户事先在该服务器进行登记注册，也不用取得FTP服务器的授权。匿名文件传输能够使用户与远程主机建立连接并以匿名身份从远程主机上拷贝文件，而不必是该远程主机的注册用户。用户使用特殊的用户名“anonymous”登陆FTP服务，就可访问远程主机上公开的文件。匿名FTP一直是Internet上获取信息资源的最主要方式。

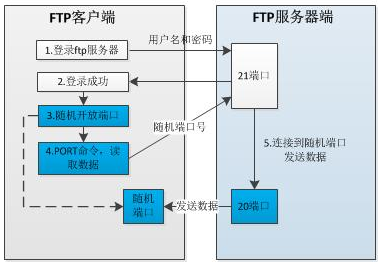
### 2.4. FTP的工作方式

FTP支持两种模式，一种方式叫做Standard(即PORT方式，主动方式)，一种是Passive (即PASV，被动方式)。 Standard模式FTP的客户端发送PORT命令到FTP服务器。Passive模式FTP的客户端发送PASV命令到FTP服务器。

#### 1. PORT模式

主动模式是指服务器主动连接客户端的数据端口。

PORT中文称为主动模式，工作的原理： FTP客户端连接到FTP服务器的21端口，发送用户名和密码登录，登录成功后要list列表或者读取数据时，客户端随机开放一个端口（1024以上），发送 PORT命令到FTP服务器，告诉服务器客户端采用主动模式并开放端口；FTP服务器收到PORT主动模式命令和端口号后，通过服务器的20端口和客户端开放的端口连接，发送数据，原理如下图：



**优点：**

服务端配置简单，利于服务器安全管理，服务器只需要开放21端口。

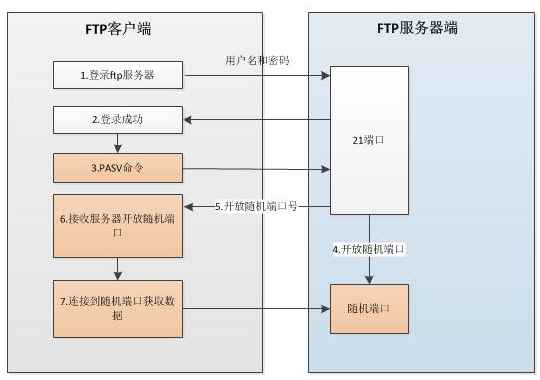
**缺点：**

如果客户端开启了防火墙，或客户端处于内网（NAT网关之后），那么服务器对客户端端口发起的连接可能会失败。

#### 2. PASV模式

被动模式的FTP是指**服务器被动地等待客户端连接自己的**20号**数据端口。**

PASV是Passive的缩写，称为被动模式，工作原理：FTP客户端连接到FTP服务器的21端口，发送用户名和密码登录，登录成功后要list列表或者读取数据时，发送PASV命令到FTP服务器，服务器在本地随机开放一个端口（1024以上），然后把开放的端口告诉客户端，客户端再连接到服务器开放的端口进行数据传输，原理如下图：



由于被动且自动建立连接，容易受到攻击，所以安全性差。

**缺点：**

服务器配置管理稍显复杂，不利于安全，服务器需要开放随机高位端口以便客户端可以连接，因此大多数FTP服务软件都可以手动配置被动端口的范围。

**优点：**

对客户端网络环境没有要求。

## 3. 因特网中的电子邮件

因特网电子邮件系统有3个主要组成部分：**用户代理**、**邮件服务器**和**简单邮件传输协议**（SMTP）。

**SMTP运行在TCP之上，端口25**。

## 4. DNS

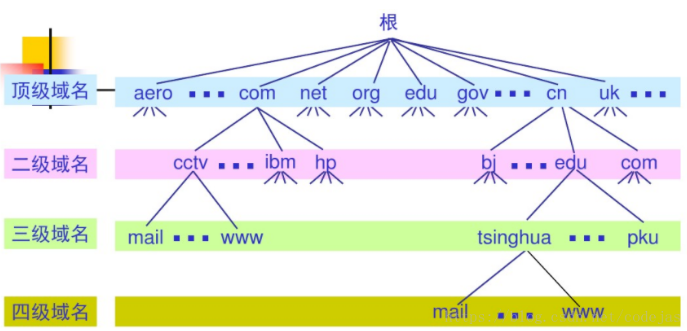
域名系统DNS：进行主机名到IP地址转换的目录服务。

**DNS协议运行在UDP之上，使用53号端口**。DNS通常是由其他应用层协议所使用的，包括HTTP、SMTP和FTP，将用户提供的主机名解析为IP地址。

域名系统DNS被设计成一个层次树状结构的联机分布式数据库系统，并且采取的是客户服务器的方式。DNS使大多数名字都在本地进行解析，只有少量的解析需要在互联网通信，因此效率很高。采取分布式的一个好处是，即使单个计算机出了故障，也不会妨碍DNS 系统的正常运行。

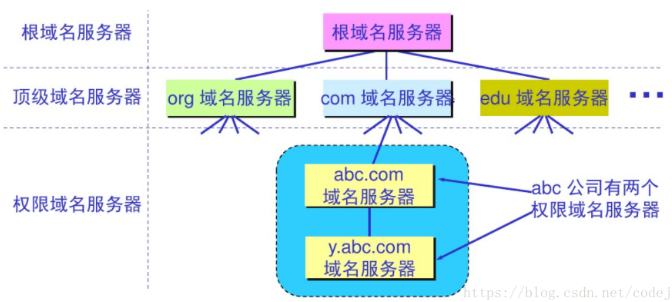
域名到IP地址的解析是通过许多分布在互联网上的域名服务器完成的。解析的主要过程如下：当一个主机中的进程需要把域名解析为IP地址时，该进程就会调用解析程序，并成为DNS的一个客户，把待解析的域名放在DNS的请求报中，以**UDP用户数据报**方式发送给本地域名服务器。本地域名服务器在查找域名后，把对应的IP地址放在回答报文中返回。获得IP地址后的主机即可进行通信。

### 4.1. 域名结构



通过域名树的方式来查看域名的结构，可以看出来，它实际上是一个倒过来的树，最上面的是根，没有对应的名字。因为根没有名字，所以根下面的一级节点就是顶级域名，往下同理。我们以www.baidu.com/为例，www是三级域名，baidu是二级域名，com是顶级域名，各级域名之间通过 . 相连。每个互联网上的主机域名都对应一个IP地址，并且这个域名在互联网中是唯一的。

### 4.2. 域名服务器

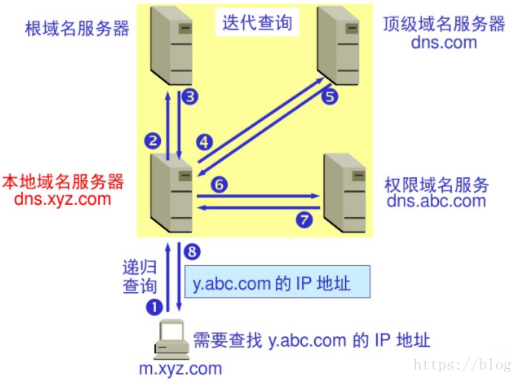


本地域名服务器并不属于上面图示的服务器层次结构，但是它在域名服务系统却发挥着至关重要的作用。当一台主机发出DNS查询请求时，这个查询请求报文就会发送给本地域名服务器。每一个互联网提供者，或者一个大学，甚至小到一个学院，都可以拥有一台本地域名服务器，这种域名服务器也被称为默认域名服务器。我们本地网络服务连接的域名服务器指的就是本地域名服务器。

### 4.3. 域名解析过程

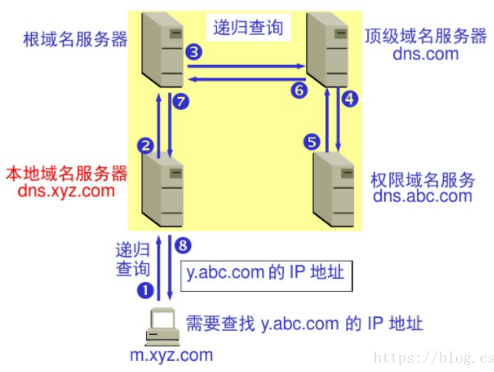
域名解析查询的方式有两种：迭代查询与递归查询。

#### 1. 迭代查询



**本地域名服务器向根域名服务器的查询方式通常采取迭代查询**。迭代查询有以下的特点：当根域名服务器收到本地域名服务器发出的迭代查询请求报时，要么给出所要查询的IP地址，要么告诉本地域名服务器：“我这里没有你要的查询结果，你需要向哪一台域名服务器进行查询”。然后本地域名服务器进行后续的查询(不替代本地域名服务器)。

#### 2. 递归查询



**主机向本地域名服务器的查询一般都采用递归查询(recursive query)。**所谓的递归查询就是：如果主机所询问的本地域名服务器不知道被查出来的域名的IP地址，那么本地域名服务器就以DNS客户的身份，向其他根域名服务器继续发出查询请求报文(替代该主机继续查询)，而不是主机自己进行下一步的查询。因此，递归查询返回的结果要么是所查询的IP地址，要么报错，表示无法查到所需要的IP。

### 4.4. DNS 缓存

为了提高 DNS 的查询效率，减轻根域名服务器的负荷和 DNS 数据报的查询数量，在域名服务器中广泛地使用了高速缓存。高速缓存用来存放最近查询过的域名以及从何处获得域名映射信息的记录。

假设我们要查询www.baidu.com/域名对应的IP地址，如果本地域名服务器上有该域名对应的IP地址，那么可以直接从本地域名服务器上获得对应的IP地址，而不需要到根域名服务器上进行查询。当本地域名服务器查询不到IP地址时，本地域名服务器也可以不向根域名服务器发送请求报文，而是直接向顶级域名服务器发送查询请求报文。

不仅在本地域名服务器中有高速缓存，在主机中也有。很多主机在启动的时候从本地域名服务器下载名字和地址的全部数据库，维护存放自己使用的域名的高速缓存，只有在缓存中找不到名字时才使用域名服务器。

### 4.5. DNS 记录

共同实现DNS分布式数据库的所有DNS服务器存储了资源记录RR。RR提供了主机名到IP地址的映射。每个DNS回答报文包含了一条或多条资源记录。

资源记录是一个包含了下列字段的4元组：

(Name , Value , Type , TTL)

1. TTL是该记录的生存时间，它决定了资源记录应当从缓存中删除的时间。
2. Name和Value的值取决于Type：
   1. 如果Type=A，则Name是主机名，Value是该主机名对应的IP地址。因此，一条类型为A的资源记录提供了标准的主机名到IP地址的映射。
   2. 如果Type = NS，则Name是个域，Value是个知道如何获得该域中主机IP地址的权威DNS服务器的主机名。这个记录用于沿着查询链来路由DNS查询。
   3. 如果Type = CNAME，则Value是别名为Name的主机对应的规范主机名。该记录能够向查询的主机提供一个主机名对应的规范主机名。
   4. 如果Type = MX，则Value是个别名为Name的邮件服务器的规范主机名。

# 二．传输层

## 1. 因特网运输层概述

运输层位于网络层之上。网络层提供了主机之间的逻辑通信，而运输层为运行在不同主机上的进程之间提供了逻辑通信。(网络层IP，IP只能标识主机；传输层是TCP/UDP，可以标识主机上某个服务，即某个进程)。

### 运输层在应用程序中的作用过程

* 在**发送端**，运输层将从发送应用程序进程接收到的报文转换成运输层分组，运输层分组称为报文段。
* 在**发送端系统中**，运输层将这些报文段传递给网络层，网路层将其封装成网络层分组(即数据报)并向目的地发送。
* 在**接收端**，网络层从数据报中提取运输层报文段，并将该报文段向上交给运输层。运输层则处理接收到的报文段，使该报文段中的数据为接收应用进程使用。

### 因特网运输层概述

**IP被称为不可靠服务：**因特网网络层协议IP，即网际协议。IP为主机之间提供了逻辑通信。IP的服务模型是尽力而为交付服务，但它并不做任何确保。特别是，它不确保报文段的交付，不保证报文段的按序交付，不保证报文段数据的完整性。所以，IP被称为不可靠服务。

UDP和TCP最基本的责任是，**将两个端系统间IP的交付服务扩展为运行在端系统上的两个进程之间的交付服务**。将主机间交付扩展到进程间交付被称为运输层的多路复用与多路分解。

## 2. 多路复用与多路分解

每个运输层报文段中具有几个字段。在接收端，运输层检查这些字段，标识出接收套接字，进而将报文段定向到该套接字。将运输层报文段中的数据交付到正确的套接字的工作称为多路分解。

在源主机从不同套接字中收集数据块，并为每个数据块封装上首部信息(这将在以后用于分解)从而生成报文段，然后将报文段传递到网络层，所有这些工作称为多路复用。

运输层多路复用要求:①套接字有唯一标识符；②每个报文段有特殊字段来指示该报文段所要交付到的套接字（比如源端口号字段和目的端口号字段）。

在主机上的每个套接字能够分配一个端口号，当报文段到达主机时，运输层检查报文段中的目的端口号，并将其定向到相应的套接字。然后报文段中的数据通过套接字进入其所连接的进程。

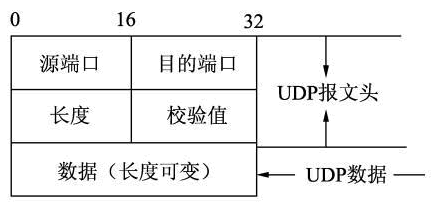
## 3. UDP

### 3.1. UDP特点

1. 无需连接建立
2. 无连接状态。TCP 需要在端系统中维护连接状态。此连接状态包括接收和发送缓存、拥塞控制参数以及序号与确认号的参数。要实现 TCP的可靠数据传输服务并提供拥塞控制，这些状态信息是必要的。在另一方面，UDP不维护连接状态，也不跟踪这些参数。
3. 分组首部开销小。每个TCP报文段都有20字节的首部开销，而UDP仅有8字节的开销。
4. UDP不保证可靠交付，主机不需要维护复杂的状态参数。
5. UDP支持一对一、一对多、多对一和多对多的交互通信。

### 3.2. UDP报文段结构

UDP首部只有4个字段，每个字段由两个字节组成



* **源端口**：16位，包含发送数据报的应用程序所使用的UDP端口。接收端的应用程序利用这个字段的值作为发送响应的目的地址。这个字段是可选的，所以发送端的应用程序不一定会把自己的端口号写入该字段中。如果不写入端口号，则把这个字段设置为0。这样，接收端的应用程序就不能发送响应了。
* **目的端口**：16位，接收端计算机上UDP软件使用的端口。
* **长度**：16位，表示UDP数据报长度，包含UDP报文头和UDP数据长度。因为UDP报文头长度是8个字节，所以这个值最小为8。
* **校验值**：16位，检验数据在传输过程中是否被损坏。

### 3.3. UDP 检验和

UDP检验和提供了**差错检测功能**。检验和用于确定当UDP报文段从源到达目的地移动时，其中的比特是否发生了改变，**但它对差错恢复无能为力（或者丢弃受损的报文段，或者交付给应用程序并给出警告）**。

## 4.可靠数据传输

TCP是在不可靠的(IP)端到端网络层之上实现的可靠数据传输协议。更一般的情况是，两个可靠通信端点的下层可能是由一条物理链路(如在链路级数据传输协议的场合下)组成或是由一个全球互联网络(如在运输级协议的场合下)组成。

### 4.1. 经具有比特差错信道的可靠数据传输

控制报文使得接收方可以让发送方知道哪些内容被正确接收（“OK”），哪些内容接收有误并因此需要重复（“请重复”）。在计算机网络环境中，基于这样重传机制的可靠数据传输协议称为**自动重传请求( Automatic Repeat Request , ARQ)协议**。

基本上，ARQ协议中还需要另外三种协议功能来处理存在比特差错的情况：

1. 差错检测。需要一种机制以使接收方检测到何时出现了比特差错（比如UDP的校验和字段）；
2. 接收方反馈。发送方要了解接收方情况(此时为分组是再被正确接收)的唯一途径就是让接收方提供明确的反馈信息给发送方。如在口述报文情况下回答的"肯定确认" (ACK)和 否定确认" (NAK)。
3. 重传。接收方收到有差错的分组时，发送方将重传该分组文

**ARQ停等(stop-and-wait)协议**：发送方将不会发送一块新数据除非发送方确信接收方已正确接收当前分组。

问题：如果ACK或NAK分组受损该如何处理？协议应该怎样纠正ACK或NAK分组中的差错？如果一个ACK或NAK分组受损，发送方无法知道接收方是否正确接收了上一块发送的数据。

解决：(几乎所有现有的数据传输协议中，包括TCP，都采用了这种方法)是**在数据分组中添加一新字段，让发送方对其数据分组编号，即将发送数据分组的序号放在该字段**。于是，接收方只需要检查序号即可确定收到的分组是否是一次重传。对于停等协议这种简单情况，1比特序号就足够了，因为它可让接收方知道发送方是否正在重传前一个发送分组(接收到的分组序号与最近收到的分组序号相同)，或是一个新分组(序号变化了，用模2运算"前向"移动) 。

### 4.2. 经具有比特差错的丢包信道的可靠数据传输

**怎样检测丢包以及发生丢包后该做些什么？**

我们让发送方负责检测和恢复丢包工作。假定发送方传输一个数据分组，该分组或者接收方对该分组的ACK发生了丢失。在这两种情况下，发送方都收不到应当到来的接收方的响应。如果发送方愿意等待足够长的时间以便确定分组已丢失，则它只需重传该数据分组即可。

发送方需要等待至少需要等待发送方与接收方之间的一个往返时延(可能会包括在中间路由器的缓冲时延)加上接收方处理一个分组所需的时间才能确定己丢包。因此实践中采取的方法是发送方明智地选择一个时间值，以判定可能发生了丢包(尽管不能确保)。如果在这个时间内没有收到ACK，则重传该分组。如果一个分组经历了一个特别大的时延，发送方可能会重传该分组，即使该数据分组及其ACK都没有丢失。这就在发送方到接收方的信道中引入了**冗余数据分组**的可能性。

为了实现基于时间的重传机制，需要一个倒计数定时器，在一个给定的时间量过期后，可中断发送方。因此，发送方需要能做到：①每次发送一个分组(包括第一次分组和重传分组)时，便启动一个定时器。②响应定时器中断(采取适当的动作)。③终止定时器。

### 4.3. 流水线可靠数据传输协议

由于前述协议是停等协议，其性能在高速网络中并不能让人满意。停等协议有着非常低的发送方利用率。（**网络协议限制底层网络硬件所提供的能力**）。

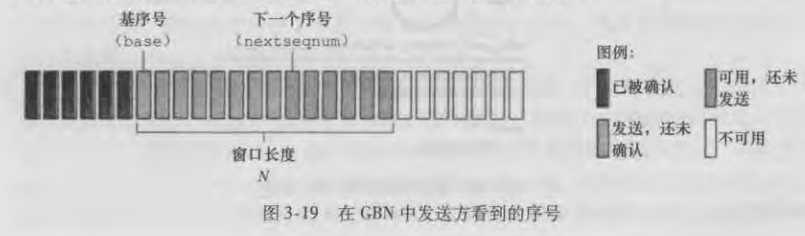
解决这种特殊的性能问题的一个简单方法是：**不使用停等方式运行，允许发送方发送多个分组而无需等待确认**。如发送方可以在等待确认之前发送3个报文，其利用率也基本提高3倍。因为许多从发送方向接收方输送的分组可以被看成是填充到一条流水线，故这种技术被称为流水线。

流水线技术对可靠数据传输协议可带来如下影响：

1. 必须增加序号范围，因为每个输送中的分组(不计算重传的)必须有一个唯一的序号，而且也许有多个在输送中未确认的报文 。
2. 协议的发送方和接收方两端也许必须缓存多个分组。发送方最低限度应当能缓冲那些已发送但没有确认的分组。接收方或许也需要缓存那些已正确接收的分组。
3. 所需序号范围和对缓冲的要求取决于数据传输协议如何处理丢失、损坏及延时过大的分组。解决流水线的差错恢复有两种基本方法是：回退N步(Go-Back-N，GBN)和选择重传(Selective Repeat, SR)。

### 4.4. 回退N步

在回退N步(GBN)协议中，允许发送方发送多个分组(当有多个分组可用时)而不需等待确认，但在流水线中未确认的分组数不能超过某个最大允许数N。



上图显示了发送方看到的GBN协议的序号范围。如果我们将基序号(base)定义为最早的未确认分组的序号，将下一个序号(nextseqnum)定义为最小的未使用序号(即下一个待发分组的序号)，则可将序号范围分割成4段：

1. [0, base-l]段内的序号对应于已经发送并被确认的分组。
2. [base, nextseqnum - 1]段内对应已经发送但未被确认的分组。
3. [nextseqnum, base+N-1]段内的序号能用于那些要被立即发送的分组，如果有数据来自上层的话。
4. [base + N, ∞)的序号是不能使用的，直到当前流水线中未被确认的分组(特别是序号为 base的分组)已得到确认为止。

N常被称为窗口长度(window size) , GBN协议也常被称为**滑动窗口协议**。

GBN发送方必须响应三种类型的事件：

1. 上层的调用。当上层调用rdt\_send时，发送方首先检查发送窗口是否已满，即是否有N个已发送但未被确认的分组。如果窗口未满，则产生一个分组并将其发送，并相应地更新变量。如果窗口已满，发送方只需将数据返回给上层，隐式地指示上层该窗口已满。然后上层可能会过一会儿再试。在实际实现中，发送方更可能缓存(并不立刻发送)这些数据，或者使用同步机制(如一个信号量或标志)允许上层在仅当窗口不满时才调用 rdt\_send()。
2. 收到一个ACK。在GBN协议中，对序号为n的分组的确认采取累积确认(cumulative acknowledgment)的方式，表明接收方已正确接收到序号为n的以前且包括n在内的所有分组。
3. 超时事件。就像在停等协议中那样，定时器将用于恢复数据或确认分组的丢失。如果出现超时，发送方重传所有已发送但还未被确认过的分组。如果收到一个ACK，但仍有已发送但未被确认的分组，则定时器被重新启动。如果没有已发送但未被确认的分组，该定时器被终止。

#### 接收方的动作

如果一个序号为n的分组被正确接收到，并且按序(即上次交付给上层的数据是序号为n-1的分组)，则接收方为分组n发送一个ACK，并将该分组中的数据部分交付到上层。在所有其他情况下，接收方丢弃该分组，并**为最近按序接收的分组重新发送ACK**。

#### GBN协议的不足

1. 当窗口长度和带宽时延积都很大时，在流水线中会有很多分组，影响性能；
2. 单个分组的差错就能够引起GBN重传大量分组，许多分组根本没有必要重传。随着信道差错率的增加，流水线可能会被这些不必要重传的分组所充斥。

### 4.5. 选择重传

选择重传(SR)协议通过**让发送方仅重传那些它怀疑在接收方出错(即丢失或受损)的分组而避免了不必要的重传**。这种个别的、按需的重传要求接收方逐个地确认正确接收的分组。

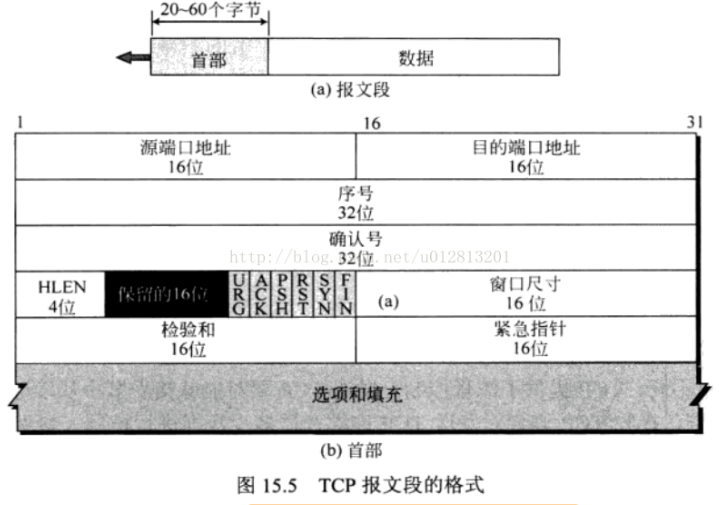
**窗口长度比须小于或等于序号空间大小的一半！**

## 5. 面向连接的运输: TCP

TCP被称为是面向连接，TCP连接提供的是全双工服务。TCP是一对一的连接。TCP的连接状态完全保留在两个端系统中。

TCP维护了一段发送缓存，当应用程序通过套接字填入待发送数据时，TCP将其填入发送缓存。TCP可从缓存中取出并放入报文段中的数据数量受限于**最大报文段长度 (Maximum Segment Size, MSS)**。MSS通常根据最初确定的由本地发送主机发送的最大链路层帧长度(即所谓的**最大传输单元 (Maximum Transmission Unit, MTU)**)来设置。设置该MSS要保证一个TCP报文段(当封装在一个IP数据报中)加上TCP/IP首部长度(通常40字节)将适合单个链路层帧。以太网和PPP链路层协议都具有1500字节的MTU，因此MSS的典型值为1460字节。

### 5.1. TCP 报文段结构



1. 源地址端口：16位，发送该报文段的主机中应用程序的端口号。
2. 目的端口地址：16位，发送该报文段的接收端主机中应用程序的端口号。
3. 序号：32位，指派给本报文段第一个数据字节的编号，TCP传输保证连接性，发送的每个字节都要编上号。序号就是告诉终点，报文段中的第一个字节是序列中的哪个。建立连接时，发收双发使用各自的随机数产生器产生一个初始序号(ISN)，通常，两个方向的ISN是不同的。
4. 确认号：32位，定义了接收方期望从对方接受的字节编号。如果报文段的接收方成功的接受了对方发过来的编号x的字节，那么返回x+1作为确认号，确认号可以和数据捎带一起发送。
5. HLEN（首部长度）：4位，指出TCP首部一共有多少个4字节，所以范围是5~15。
6. 保留：
7. 控制：定义了6中不同的控制位或者标志位：
   1. URG：紧急数据标志位
   2. ACK：确认标志位
   3. PSH：请求推送位，接收端应尽快把数据传送给应用层
   4. RST：连接复位，通常，如果TCP收到的一个分段明显不属于该主机的任何一个连接，则向远程发送一个复位包
   5. SYN：建立连接，让连接双方同步序列号
   6. FIN：释放连接
8. 窗口大小：定义TCP的窗口大小，以字节为单位。窗口的最大长度是65535字节（16位）。窗口大小的值被称为接受窗口，由接收方决定。
9. 检验和：TCP的使用检验和是强制性的，UDP使用检验和是可选的。在计算检验和是报文段要附加相同的伪首部，对于TCP来说伪首部的协议字段的值是6。
10. 紧急指针：只有当紧急标志置位时URG，该16位的字段才有效。紧急指针定义了一个数值，把这个数值加到序号上就得到报文段数据部分中最后一个紧急字节的编号。
11. 选项：40字节可选信息。

**例1：**假设主机A已收到了来自主机B的编号为0-535的所有字节，同时假设它打算发送一个报文段给主机B。主机A等待主机B的数据流中字节536及之后的所有字节。所以主机就会在它发往主机B的报文段的确认号字段中填上536。

**例2：**假设主机A己收到一个来自主机B的包含字节0-535的报文段，以及另一个包含字节900 -1000的报文段。由于某种原因，主机A还没有收到字节536 -899的报文段。在这个例子中，主机A为了重新构建主机B的数据流，仍在等待字节536 (和其后的字节)。因此，A到B的下一个报文段将在确认号字段中包含536。因为TCP只确认该流中至第一个丢失字节为止的字节，所以TCP被称为提供累积确认(cumulative acknowledgment)。

### 5.2. 往返时间的估计与超时

TCP采用超时/重传机制来处理报文段的丢失问题。

#### TCP是如何估计发送方与接收方之间往返时间的

RTT：Round-Trip Time.

报文段的样本RTT(SampleRTT)就是从某报文段被发出(即交给IP)到对该报文段的确认被收到之间的时间量。大多数TCP的实现仅在某个时刻做一次SampleRTT测量，而不是为每个发送的报文段测量一个SampleRTT。这就是说，在任意时刻，仅为1个已发送的但目前尚未被确认的报文段估SampleRTT，从而产生一个接近每个RTT的新SampleRTT值。另外，TCP决不为已被重传的报文段计算SampleRTT；它仅为传输一次的报文段测量SampleRTT。

TCP维持一个SampleRTT均值即**EstimatedRTT**。一旦获得一个新SampleRTT时，TCP就会根据下列公式来更新EstimatedRTT：

EstimatedRTT = (1 -α)\*EstimatedRTT +α\*SampleRTT

在[RFC 6298]中给出的α参考值是α=0.125。

[RFC 6298]定义了RTT偏差**DevRTT**，用于估算SampleRTT一般会偏离EstimatedRTT的程度:

DevRTT = (1 -β) \* DevRTT +β\*| SampleRTT – EstimatedRTT |

β的推荐值为0.25。

#### 设置和管理重传超时间隔

要求将超时间隔设为EstimatedRTT加上一定余量。当SampleRTT值波动较大时，这个余量应该大些;当波动较小时，这个余量应该小些。

TimeoutInterval = EstimatedRTT + 4 \* DevRTT

推荐的初始TimeoutInterval值为1秒。同样，当出现超时后，TimeoutInterval值将加倍，以免即将被确认的后继报文段过早出现超时。不管怎样，一旦报文段收到ACK并更新EstimatedRTT后，TimeoutInterval 就又使用上述公式计算了。

### 5.3. 可靠数据传输

TCP在IP不可靠的尽力而为服务之上创建了一种可靠数据传输服务。TCP的可靠数据传输服务确保一个进程从其接收缓存中读出的数据流是**无损坏、无间隔、非冗余和按序的数据流**，即该字节流与连接的另 一方端系统发送出的字节流是完全相同。

#### 1.超时间隔加倍

每当超时事件发生时，如前所述，TCP重传序号最小的还未被确认的报文段。只是每次TCP重传时都会将下一次的超时间隔设为先前值的两倍。

然而，每当定时器在另两个事件(即收到上层应用的数据和收到ACK)中的任意一个启动时，Timeoutlnterval由最近的EstimaledRTT值与DevRTT值推算得到。

#### 2.快速重传

超时触发重传存在的问题之一是超时周期可能相对较长。当一个报文段丢失时，这种长超时周期迫使发送方延迟重传丢失的分组，因而增加了端到端时延。

当TCP接收方收到一个具有这样序号的报文段时，即其序号大于下一个所期望的、按序的报文段，它检测到了数据流中的一个间隔，这就是说有报文段丢失。这个间隔可能是由于在网络中报文段丢失或重新排序造成的。因为TCP不使用否定确认，所以接收方不能向发送方发送一个显式的否定确认。**相反，它只是对已经接收到的最后一个按序字节数据进行重复确认(即产生一个冗余ACK) 即可**。

如果TCP发送方接收到对相同数据的3个冗余ACK，它把这当作一种指示，说明跟在这个已被确认过3次的报文段之后的报文段已经丢失。一旦收到3个冗余ACK，TCP就执行**快速重传**(fast retransmil) [RFC 5681 ]，即在该报文段的定时器过期之前重传丢失的报文段。

### 5.4. 流量控制

一条TCP连接每一侧主机都为该连接设置了接收缓存。如果某应用程序读取数据时相对缓慢，而发送方发送得太多、太快，发送的数据就会很容易地使连接的接收缓存溢出。

TCP 为它的应用程序提供了**流量控制服务**，以消除发送方使接收方缓存溢出的可能性。流量控制因此是一个速度匹配服务，即发送方的发送速率与接收方应用程序的读取速率相匹配。前面提到过，TCP发送方也可能因为IP网络的拥塞而被遏制；这种形式的发送方的控制被称为**拥塞控制**(congestion control)。

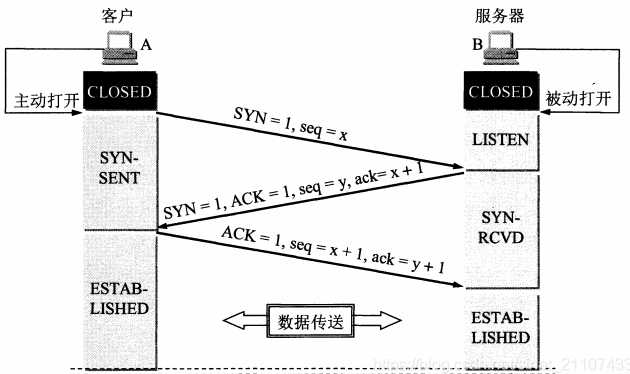
TCP 通过让发送方维护一个称为**接收窗口**(receive window) 的变量来提供流量控制。**接收窗口用于给发送方一个指示—该接收方还有多少可用的缓存空间**。因为TCP是全双工通信，在连接两端的发送方都各自维护一个接收窗口。

### 5.5. TCP连接管理

TCP连接建立过程中需要解决以下三个问题：

1. 要使每一方能够通知对方的存在；
2. 要允许双方协商一些参数（如最大窗口值、是否使用窗口扩大选项和时间戳选项等）；
3. 能够对运输实体资源（如缓存大小、连接表中的项目等）进行分配。

#### 1. 三次握手



TCP连接建立的过程如上，主机A是客户应用进程，B运行服务器应用程序。初始状态，两端的TCP进程都属于CLOSED（关闭）状态。B的TCP服务器先创建传输控制块TCB，准备接受客户进程的请求，然后就处于LISTEN（监听）状态，等待客户请求。

1. 第一次握手：A主动打开连接，创建传输控制模块TCB，然后向B发出请求报文段。TCP报文段首部的同步位SYN=1，同时选择一个初始序号seq=x。TCP规定，SYN报文段不能携带数据，但要消耗掉一个序号。此时，TCP客户进程处于同步已发送（SYN-SENT）状态。
2. 第二次握手：B收到请求报文段后，如果同意连接，则向A发送确认。确认报文段中，SYN=1，ACK=1，确认号是ack=x+1，同时也为自己选择一个初始序号seq=y。这个报文段也不能携带数据，同样要消耗掉一个序号。此时TCP服务进程处于同步收到（SYN-RCVD）状态。
3. 第三次握手：TCP客户程序收到B的确认后，再次向B发送确认。确认报文段ACK=1，确认号ack=y+1，而自己的序号seq=x+1。TCP规定，ACK报文段可以携带数据，但如果不携带数据则不消耗序号。此时，TCP连接已经建立，A进入已建立连接（ESTABLISHED）状态。B收到A的确认后，也进入已建立连接（ESTABLISHED）状态。

**面试题1：为何A（客户端应用进程）还要再发一次确认呢？（第三次握手）**

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。

考虑如下一种场景：A向B发送连接请求，但是由于某种原因（在某个网络节点长时间滞留）导致在A和B之前的一次TCP连接已经释放后的某个时间，该连接请求才到达B。如果没有第三次握手，B在收到这个迟到的连接请求后，会误认为A又请求建立连接，就又会建立连接。假定有第三次握手，即便之前的某个连接请求到达B，只要B没有收到A的第三次握手确认，TCP连接就不会建立。

**面试题2：请问tcp握手为什么两次不可以？为什么不用四次？**

两次不可以：tcp是全双工通信，两次握手只能确定单向数据链路是可以通信的，并不能保证反向的通信正常。

不用四次：

本来握手应该和挥手一样都是需要确认两个方向都能联通的，本来模型应该是：

1.客户端发送syn0给服务器

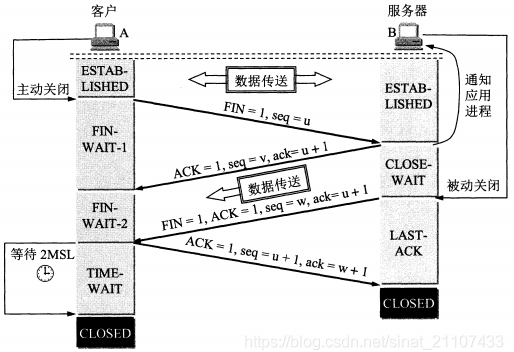
2.服务器收到syn0，回复ack(syn0+1)

3.服务器发送syn1

4.客户端收到syn1，回复ack(syn1+1)

因为tcp是全双工的，上边的四部确认了数据在两个方向上都是可以正确到达的，但是2，3步没有没有上下的联系，可以将其合并，加快握手效率，所有就变成了3步握手。

#### 2. 四次挥手



数据传输结束后，双方都可以释放连接。现在A和B都处于ESTABLISHED状态。

1. 第一次挥手：A的应用进程首先向其TCP发送连接释放报文段，并停止发送数据，主动关闭连接。A把连接释放报文段首部的终止控制位FIN=1，序号seq=u（序号等于前面已经传送过的数据的最后一个字节的序号+1）。此时A进入终止等待1（FIN-WAIT-1）状态。TCP规定，FIN报文段不能携带数据，也消耗一个序号。
2. 第二次挥手：B收到连接释放报文段后发出确认，确认号ack=u+1，序号seq=v（序号等于前面已经传送过的数据的最后一个字节的序号+1）。此时B进入关闭等待（CLOSE-WAIT）状态。TCP服务进程此时应通知高层应用进程，A到B这个方向的连接就释放了，此时TCP连接处于半关闭状态。即A已经没有数据要发送了，但如果B要发送数据，A仍要接收。即，B到A方向的连接并未关闭。A收到B的确认后，进入终止等待2（FIN-WAIT-2）状态，等待B发出连接释放报文段。
3. 第三次挥手：如果B已经没有数据发送给A，其应用进程将通知连接释放。B发出的连接释放报文段FIN=1，序号seq=w（半关闭状态B可能又发送了一些数据给A，如果没有，则序号seq=v+1），B必须重复发送已经发送过的确认号ack=u+1（因为B收到的A的报文段的最后一个序号始终是u）。此时B进入最后确认（LAST-ACK）状态，等待A的确认。
4. 第四次挥手：A收到B的连接释放报文段后，必须发出确认。ACK=1，确认号ack=w+1，自己的序号seq=u+1。然后自己进入时间等待（TIME-WAIT）状态。经过时间等待计时器（TIME-WAIT timer）设置的时间2MSL后，A进入CLOSED状态。

MSL指的是最长报文段寿命（Maximum Segment Life），RFC793建议为2分钟。2MSL即4分钟。

疑问：为什么A在TIME-WAIT状态必须等待2分钟呢？

1. 一方面是为了保证A发送的最后一个ACK报文段能够到达B。这个ACK报文段有可能丢失，因此处于LAST-ACK状态的B收不到对自己之前已经发送的ACK和FIN报文段的确认。B会超时重传这个ACK+FIN报文段，A就会再次发送一次ACK确认报文段，再启动计时2MSL。如果A发送完最后一个确认ACK后立即关闭TCP连接，如果ACK报文丢失，A收不到B重传的ACK+FIN，也就不会重传ACK确认报文段，这样B应用进程的TCP连接就无法关闭。
2. 另一方面，是防止上一节三次握手过程中的无效的由于网络原因迟到的连接请求报文出现在本连接过程中。MSL是任何IP数据报能够在因特网中存活的最长时间。A发送完最后一个ACK确认报文段后，等待2MSL时间，就会使本连接持续时间内所产生的所有报文段都从网络中消失。（一个方向上的分组最多存活MSL被丢弃，另一个方向上的ACK也最多存活MSL被丢弃）。

### 5.6. TCP拥塞控制

**拥塞**：网络中对某一资源的需求超过了该资源所能提供的可用部分，网络的性能将变坏。

**拥塞控制**：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。

#### 1. 拥塞的代价

1. 当分组的到达速率接近链路容量时，分组经历巨大的排队时延。
2. 发送方必须执行重传以补偿因为缓存溢出而丢弃(丢失)的分组。
3. 发送方在遇到大延时所进行的不必要重传会引起路由器利用其链路带宽来转发不必要的分组副本。
4. 当一个分组沿一条路径被丢弃时，每个上游路由器用于转发该分组到丢弃该分组而使用的传输容量最终被浪费掉了。

#### 2. 拥塞控制方法

慢开始、拥塞避免、快重传、快恢复。

#### 慢开始和拥塞避免

**拥塞窗口与控制：**

发送方维持一个叫做拥塞窗口cwnd的状态变量。Cwnd的大小取决于网络的拥塞程度而动态变化。发送方控制拥塞窗口的原则是：只要网络没有出现拥塞，拥塞窗口就增大一些，以便把更多的分组发送出去；但只要网络出现拥塞，拥塞窗口就减小一些，以减少注入到网络中的分组数。

发送方如何知道网络发生了拥塞：只要发送方没有按时受到应当到达的确认报文，就可以猜想网络可能出现了拥塞。

**慢开始和拥塞避免：**

当主机开始发送数据时，由小到大逐渐增大发送窗口，即由小到大逐渐增大拥塞窗口数值。通常在刚开始发报文段时，先把拥塞窗口cwnd设置为MSS的数值。而在每接收到对新的报文段的确认后，把拥塞窗口增加至多一个MSS的数值。

**Example：**

发送方此时可以连续发送两个数据报文段，接收方收到该数据报文段后，给发送方一次发回2个确认报文段；发送方收到这两个确认报文后，将拥塞窗口的值加2变为4，发送方此时可连续发送4个报文段；接收方收到4个报文段后，给发送方依次回复4个确认报文，发送方收到确认报文后，将拥塞窗口加4，置为8；发送方此时可以连续发送8个数据报文段，接收方收到该8个数据报文段后，给发送方一次发回8个确认报文段，发送方收到这8个确认报文后，将拥塞窗口的值加8变为16……

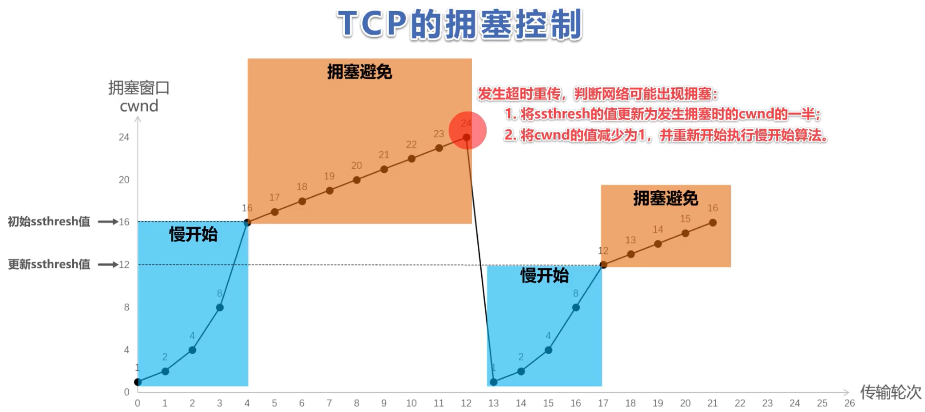
为防止拥塞窗口cwnd增长过大引起网络阻塞，需要设置一个慢开始门限ssthresh状态变量。Ssthresh用法如下：

当cwnd < ssthresh时，使用上述慢开始算法；

当cwnd > ssthresh时，停止使用慢开始算法而改用拥塞避免算法；

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

**拥塞避免算法思路：**让cwnd缓慢增大，即每经过一个往返时间RTT，就把发送方的拥塞窗口cwnd加1，而不是加倍，此时拥塞窗口cwnd按线性规律缓慢增长。



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

注意：拥塞避免并非能够完全避免，而是在拥塞避免阶段将拥塞窗口控制为按线性规律增长，是网络不那么容易出现拥塞。

#### 快重传和快恢复

快重传算法首先要求接收方每收到一个失序的报文段后就立即发出重复确认（为的是让发送方及早知道有报文段没有到达对方）而不要等自己发送数据时才捎带确认。发送方只要一连收到三个重复确认就应当立即重传对方尚未收到的报文段，而不必继续等待为其设置的重传计时器到期。

与快重传配合使用的还有快恢复算法：

1. 当发送方连续收到3个重复确认时，就执行乘法减小算法，把慢开始门限ssthresh减半。这是为了预防网络拥塞发生。请注意，接下去不执行慢开始算法。
2. 由于发送方现在认为网络很可能没有发生拥塞（如果发生了拥塞，就不会一连有好几个报文段连续到达接收方，也就不会导致接收方连续发送重复确认），因此与慢开始不同之处是现在不执行慢开始算法（即拥塞窗口现在不设置为1），而是把cwnd值设置为慢开始门限ssthresh减半后的数值，然后执行拥塞避免算法，使拥塞窗口缓慢增大。

### 面试题

##### 面试题1：TCP和UDP区别

1. **连接**：TCP是面向连接的传输层协议，即传输数据之前必须先建立好连接。UDP无连接。
2. **服务对象**：TCP是点对点的两点间服务，即一条TCP连接只能有两个端点；UDP支持一对一，一对多，多对一，多对多的交互通信。
3. **可靠性**：TCP是可靠交付：无差错，不丢失，不重复，按序到达。UDP是尽最大努力交付，不保证可靠交付。
4. **拥塞控制，流量控制**：TCP有拥塞控制和流量控制保证数据传输的安全性。UDP没有拥塞控制，网络拥塞不会影响源主机的发送效率。
5. **报文长度**：TCP是动态报文长度，即TCP报文长度是根据接收方的窗口大小和当前网络拥塞情况决定的。UDP面向报文，不合并，不拆分，保留上面传下来报文的边界。
6. **首部开销**：TCP首部开销大，首部20个字节。UDP首部开销小，8字节。（源端口，目的端口，数据长度，校验和）。

##### TCP套接口connect函数返回值？

如果是TCP套接口，函数connect激发TCP的三次握手过程，且仅在连接建立成功或出错时才返回，返回的错误可能有以下几种情况：

1. 如果TCP客户没有收到对SYN分节的响应，则返回ETIMEDOUT。重传
2. 如果对客户的SYN的响应是RST，则表明该服务器主机在指定的端口上没有进程在等待与之连接。客户一收到RST，马上就返回错误ECONNREFUSED。

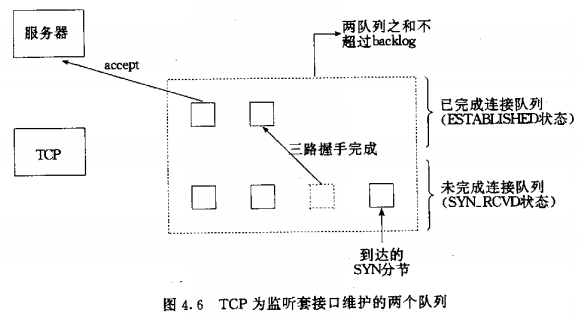
RST含义为复位，是TCP在某些错误情况下所发的一种TCP分节。有3个条件可以产生RST：SYN到达某端口但此端口上没有正在监听的服务器、TCP想取消一个已有连接、TCP接收了一个根本不存在的连接上的分节。

1. 如果某客户发出的SYN在中间的路由器上引发了一个目的地不可达ICMP错误，客户上的内核保存此消息，并间隔6秒、24秒在发送一个SYN，若总共等待了75秒后仍未收到响应，则把保存的消息（即ICMP错误）作为EHOSTUNREACH或ENETUNREACH返回给进程。

##### listen函数

对于给定的监听套接口，内核要维护两个队列：

1. 未完成连接队列，为每个这样的SYN分节开设一个条目：已由客户发出并到达服务器，服务器正在等待完成TCP三路握手过程，这些套接口都处于SYN\_RCVD状态；
2. 已完成连接队列，为每个已完成TCP三路握手过程的客户开设一个条目。这些套接口都处于ESTABLISHED状态。



注意，当一个客户SYN分节到达时，若两个队列都是满的，TCP就忽略此分节，且不发送RST。因为这种情况是暂时的，客户TCP将重发SYN，期望不久后就能在队列中找到空闲条目。如果服务器发送了RST，客户的connect函数将立即返回一个错误，强制应用进程处理这种情况，而不是让TCP正常的重传机制来处理。

##### TCP客户端/服务器连接终止的各种情况

**1、当处于正常连接中的服务器主机崩溃时，一直不重启**

当服务器主机崩溃时，已有的网络上已经发送不出任何东西，假如客户端服务器的场景是：

客户端write消息给服务器，等待服务器收到消息时将消息发送回客户端。

客户端已经调用完write函数将消息写入内核，内核将数据发送给服务器端，客户端现在正阻塞在read函数上，等待服务器的消息回来，但是这时候服务器已经崩溃，不会响应信息，通过抓包可以看见，客户端持续的重传数据给服务器，希望服务器返回一个ack回来，重传该数据大概12次，持续将近9分钟，当客户端最后放弃重传时，read将返回一个错误，如果服务器一直崩溃，对客户端发送的数据没有响应，最后会返回一个ETIMEDOUT错误，如果是中间的路由器判定服务器不可达，将会响应一个“destination unreachable”的ICMP消息，返回的错误码是EHOSTUNREACH或ENETUNREACH，正常使用中我们不会等待9分钟，所以要设置timeout时间

**2、当处于正常连接的服务器崩溃后然后重启**

当服务器崩溃后重启时，它的TCP已经丢失了之前的所有连接信息，因此收到客户端发送的信息时将响应一个RST(复位)，由于此时客户端正阻塞于read等待服务器的回答，收到RST时会导致该系统调用返回ECONNRESET错误

**3、 当服务器主机关机时**

Unix系统关机时，init进程通常向所有的进程发送SIGTERM信号(可捕获），然后等待5到20s过后，给所有仍在运行的进程发送SIGKILL信号(不能被捕获），所有的进程都会被终止，服务器打开的描述符都会被关闭，会向客户端发送fin，客户端就会向服务器发送ack，进入半双工状态，如果这时客户端还向服务器发送数据，服务器将响应一个RST。

**4、 服务器多进程的情况**

如果是多进程的情况，父进程accpet返回后将连接套接字交由子进程处理，父进程重新阻塞在accept。当服务器子进程阻塞于read时，如果这时候客户端终止，向服务器发送FIN，子进程收到FIN后返回ack，进入CLOSE\_WAIT状态。当子进程终止时，会向服务器发送一个SIGCHID信号，如果父进程不处理这个信号，该信号默认被忽略，子进程就进入僵尸状态（僵尸进程，当父进程退出时交由init进程接管，才会被清理，实际情况中我们的父进程一直是while循环accept，不会轻易退出，大量的僵尸进程会耗尽进程资源），在我们的程序中需要捕捉这个信号，不然会占用资源。

这里说一下父进程回收子进程时最好使用waitpid，因为如果信号处理函数调用wait回收子进程时，如果没有已终止的子进程，wait会阻塞到现有的子进程第一个终止为止，这就会造成如果大量的客户端同时关闭，服务器父进程产生的子进程同一时间向父进程发送SIGCHID信号，如果这时候wait是阻塞的，还是处于信号处理流程中，那会有大量的信号不会被处理，还是会产生僵尸进程(unix中，一个信号处于信号处理函数流程中，该信号是会被阻塞的，如果一个信号在阻塞期间产生了一次或多次，这个信号在解阻塞后通常只递交一次，所以不是每个信号都能处理到，就会产生僵尸进程）。

而pid\_t waitpid(pid\_t pid, int \*statloc, int options)的参数options可以指定为WNOHANG，用于告知内核在没有终止的子进程时不阻塞。

父进程捕捉该信号时，是阻塞在accept函数中的，接收到SIGCHID信号时，可能会返回一个EINTR错误，所以我们需要处理这个错误，让父进程重新accept(这时慢系统调用（指可能永远阻塞的系统调用）的一个特点，当阻塞于某个慢系统调用的一个进程捕捉到某个信号且相应的信号处理函数返回时，可能会返回EINTR，所以正常情况下read，write这些系统调用都需要处理这个错误，不过connect不能再次调用，当connect返回EINTR时再次调用会立即返回错误)。

# 三．网络层

网络层的作用：

1. **转发**：当一个分组到达路由器的一条输入链路时，路由器必须将该分组移动到适当的输出链路。
2. **路由选择**：当分组从发送方流向接收方时，网络层必须决定这些分组所采用的路由或路径。计算这些路径的算法被称为路由选择算法

每台路由器具有一张转发表。路由器通过检查到达分组首部字段的值来转发分组，然后使用该值在该路由器的转发表中索引查询。存储在转发表项中的该首部的值指出了该分组将被转发的路由器的输出链路接口。路由选择算法决定了插入路由器的转发表中的值。路由选择算法可能是集中式的(例如，算法在某个中心场点执行，并向每台路由器下载路由选择信息)，或是分布式的(即，使用运行在每台路由器上的分布式路由选择算法的一部分)。

## 1. 虚电路和数据报网络

仅在网络层提供连接服务的计算机网络称为虚电路(VC) 网络；仅在网络层提供无连接服务的计算机网络称为数据报网络。

### 1.1. 虚电路网络

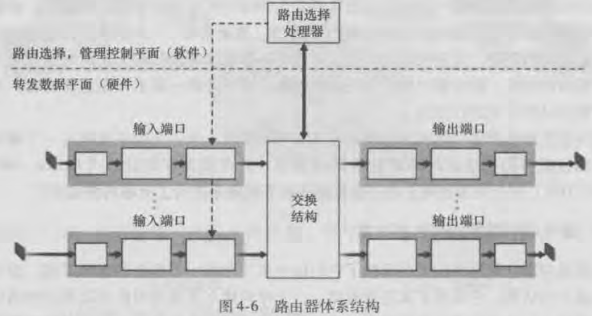
一条虚电路的组成如下：①源和目的主机之间的路径(即一系列链路和路由器)；②VC号，沿着该路径的每段链路的一个号码；③沿着该路径的每台路由器中的转发表表项。属于一条虚电路的分组将在它的首部携带一个VC号。

### 1.2. 数据报网络

在数据报网络中，每当一个端系统要发送分组，它就为该分组加上目的端系统的地址，然后将分组推进网络中。当分组从源到目的地传输，它通过一系列路由器传递。这些路由器中的每台都使用分组的目的地址来转发该分组。特别是，每台路由器有一个将目的地址映射到链路接口的转发表；当分组到达路由器时，路由器使用该分组的目的地址在转发表中查找适当的输出链路接口。然后路由器有意将分组向该输出链路接口转发。

虽然在数据报网络中的路由器不维持连接状态信息，但它们无论如何在其转发表中维持了转发状态信息然而，转发状态信息表变化的时间尺度相对要慢。实际上，在数据报网络中的转发表是通过路由选择算法进行修改的，这通常每1-5钟左右更新一次转发表。在虚电路网络中，无论何时通过路由器建立一条新的连接，或无论何时通过路由器拆除一条现有的连接，路由器中的转发表就被更新。

### 1.3. 路由器工作原理



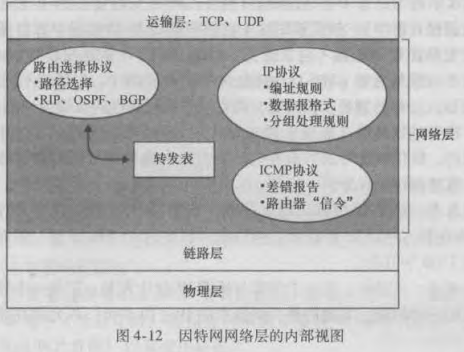
**输入端口**。输入端口执行几项关键功能：1）将一条输入的物理链路与路由器相连接；2）与位于入链路远端的数据链路层交互；3）在输入端口完成查找功能。通过查询转发表决定路由器的输出端口，到达的分组通过路由器的交换结构将转发到输出端口。控制分组(如携带路由选择协议信息的分组)从输入端口转发到路由选择处理器。

**交换结构**。交换结构将路由器的输入端口与输出端口相连接。这种交换结构完全包含在路由器中，即它是一个网络路由器中的网络！

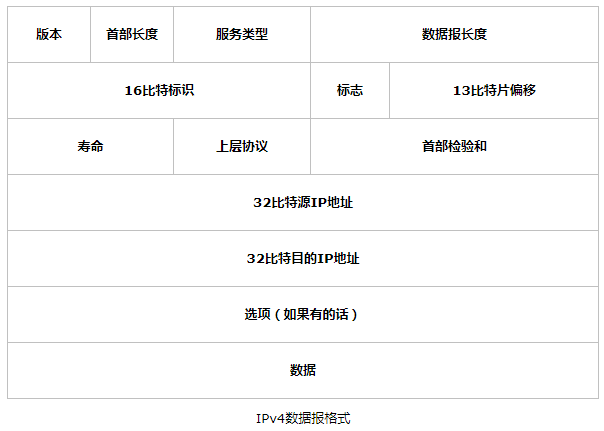
**输出端口**。输出端口存储从交换结构接收的分组，并通过执行必要的链路层和物理层功能在输入链路上传输这些分组。

**路由选择处理器**。路由选择处理器执行路由选择协议，维护路由选择表以及连接的链路状态信息，并为路由器计算转发表。

## 2. 网际协议



### 数据报格式



**1）版本（号）：**4bit，规定了数据包的IP协议版本；通过查看版本号，路由器能够确定如何解释IP数据报的剩余部分。

**2）首部长度：**因为IPV4数据报可包含一些可变数量的选项，所以需要用这4bit来确定首部的长度，以确定IP数据报的数据部分实际从哪里开始。大多数IP数据报不包含选项，所以一般IP数据报具有20字节的首部。

**3）服务类型：**8bit，服务类型包含在首部中以使不同类型的IP数据报能相互区分开来。

**4） 据报长度：**这是IP数据报的总长度（首部加上数据），以字节计，因为该字段长为16bit，所以IP数据报的理论最大长度为65535字节，然而数据报很少有超过1500字节的（因为IP数据还要靠数据链路层运输的，而链路层帧能承载的最大数据量为叫做最大运输单元（Maximum Transmission Unit，MTU））。

**5）标识、标志、片偏移：**

　　（1）分片：把IP数据报中的数据分成两个或者更多个较小的IP数据报，用单独的链路层帧封装成较小的IP数据报，每个这些较小的数据报称为片；

　　（2）为什么要分片：因为每个IP数据报封装在链路层帧中从一台路由器传输到下一台路由器，而链路层帧能承载的最大数据量（最大运输单元（Maximum Transmission Unit，MTU）是一定的，故链路层帧严格限制着IP数据报的长度；而且发送方和与目的路径上的每段链路可能使用不同的链路层协议，且每种协议可能具有不同的MTU，所以就更有可能需要分片了，以便能够使得数据报能够顺利的传递数据报；

　　（3）组装：目的主机从相同源收到一系列数据报时，需要确定哪些数据报是分片，如果是分片的话，还要进一步指导何时收到最后一个分片，如何将接收到的分片拼接起来以形成初始的数据报，故IPV4的设计者将标识、标志和片偏移字段放在IP数据报首部中，当生成一个数据报时，发送主机为该数据报设置源和目的地址的同时，再填上标识号

**标识：**16bit，源主机发送IP数据报的时候，通常为它发送的每个数据报的标识号加1，所以当某个路由器需要对某一个数据报分片时，形成的每个数据报（分片）具有初始数据报的源地址、目的地址、与标识号，这样目的主机就可以 判别哪些分片是属于一个初始数据报的；

**标志：**3bit，由于IP是一种不可靠服务，一个或者多个片可能永远到不了目的地，所以为了让目的主机绝对的相信它已经收到了初始数据报的最后一个片，最后一个片的标志比特被设置为0，而所有其他片的标志比特被设置为1；

**片偏移：**13bit，标示数据相对于初始数据报的偏移值，并且偏移值应当被规定以8字节块为单位。所以除了最后一个片的所有初始有效载荷数据的数量应当是8字节的倍数。



**6）寿命：**8bit，寿命（TTL）字段是用来确保数据报不会永远（如由于长时间的路由选择环路）在网络中循环，每当数据报由一台路由器处理时，该字段的值减1。若TTL字段减为0，则该数据报必须丢弃。

**7）协议：**8 bit，该字段仅当一个IP数据报到达其最终目的地才会有用，指示了IP数据报的数据部分应交给哪个特定的运输层协议，如：值为6应该交给TCP，而值为17表示数据部分要交给UDP。

**8）首部检验和：**16bit，首部检验和用于帮助路由器检测收到的IP数据报中的比特错误。路由器要对每个收到的IP数据报计算其首部检验和，如果数据报首部中携带的检验和与计算得到的检验和不一致，则检验出是个差错，路由器一般会丢弃检测出错误的数据报

**9）源和目的IP地址：**32bit，当某源生成一个数据报时，在这两个字段中插入发送主机的IP地址和目的地的IP地址，通常源主机DNS查找来决定目的地址。

**10）选项：**32bit，选项字段允许IP首部被扩展。首部选项意味着很少使用，因此决定对每个数据报首部不包括选项字段的信息，这样能够节约开销，但是因为有些数据报有选项，有些数据报没有选项，故导致一台路由器处理一个IP数据报所需的时间变化很大。

**11）数据（有效荷载）：3**2bit，这是数据报存在的首要理由！大多数情况下，IP数据报中的数据字段包含要交付给目的地运输层报文段（UDP或TCP）。

注：注意到一个IP数据报有总长为20字节的首部（假设无选项）如果一个数据报承载一个TCP报文，则每个（无分片的）数据报共承载了总长40字节的首部（20字节的IP数据报加上20字节的TCP首部）以及应用文报文。

### 2.2. 因特网控制报文协议

ICMP被主机和路由器用来彼此沟通网络层的信息。ICMP最典型的用途是差错报告。ICMP通常被认为是IP的一部分，但从体系结构上讲它是位于IP之上的，因为ICMP报文是承载在IP分组中的，作为IP有效载荷承载。当一台主机收到一个指明上层协议为ICMP的IP数据报时，它分解出该数据报的内容给ICMP ，就像分解出一个数据报的内容给TCP或UDP一样。

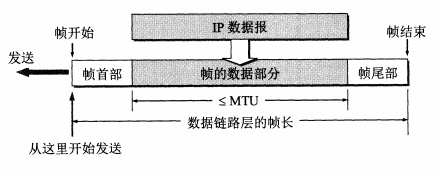


# 四．数据链路层

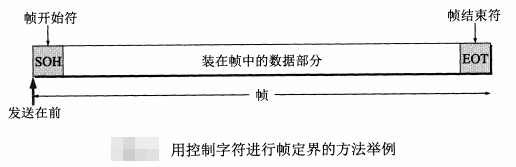
## 1. 三个基本问题

### 1.1. 封装成帧

网络层的IP数据报传送到数据链路层就成为帧的数据部分，在帧的数据部分的前面和后面分别加上首部和尾部，构成一个完整的帧。首部和尾部一个重要的作用是进行帧定界，即确定帧的界限。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**。

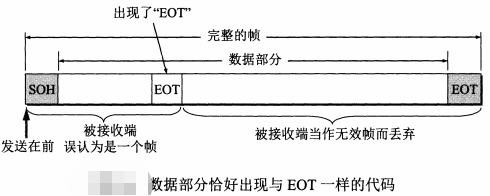


下图说明帧定界的方法：控制字符SOH表示帧的首部开始，十六进制码0x01；控制字符EOF表示帧的尾部，十六进制码为0x04。



### 1.2. 透明传输

如果传送的帧中的数据某个字节的编码恰好和SOH或EOT一样，如下图，则不再是透明传输。数据链路层会错误地找到帧边界，收下部分帧，丢弃剩下的部分。



解决：在数据中出现控制字符SOH或EOT前面插入转义字符ESC（十六进制0x1B）。接收端的数据链路层在把数据送往网络层之前删除这些转义字符。

### 1.3. 差错检测

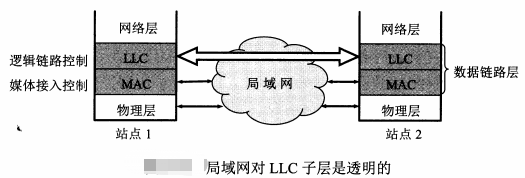
数据传输过程可能会出现比特差错：传输过程中比特1可能会变成0，反之亦然。数据链路层广泛使用了**循环冗余校验CRC**的检错技术。

在数据链路层若仅仅使用循环冗余检验CRC差错检测技术，则只能做到对帧的无差错接受，即凡是接收端数据链路层接受的帧，均无差错。**对有差错的帧是否需要重传由高层决定**。（如果高层使用TCP协议，TCP发现丢失了一些数据，经过一段时间后TCP就会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，而是当做新的数据帧来发送**）。

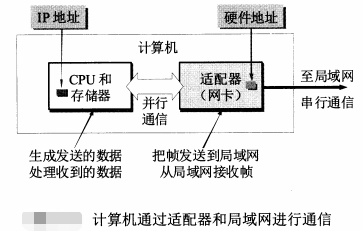
## 2. 使用广播信道的数据链路层

### 2.1. 局域网的数据链路层

局域网的数据链路层拆分为两个子层：**逻辑链路控制LLC**和**媒体接入控制MAC**子层。与接入到传输媒体有关的内容都放在MAC子层。



适配器：计算机通过适配器与外界局域网通信。适配器和局域网之间的通信是通过电缆或双绞线以并行方式传输。因此适配器的一个重要功能是进行数据串行传输与并行传输的转换。注意，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CPU。当适配器收到有差错的帧时，不必通知计算机，而直接丢弃这个帧。当适配器收到正确的帧时，使用中断来通知计算机，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。注意：计算机的硬件地址保存在适配器的ROM中，软件地址(IP地址)保存在计算机存储器中。



### 2.2. CSMA/CD协议

总线上只要有一台计算机在发送数据，总线的传输资源就被占用，因此，在同一时间只能允许一台计算机发送数据。否则各计算机之间就会相互干扰，使得所发送数据被破坏。以太网采用的协议是**CSMA/CD，载波监听多点接入/碰撞检测**：

**多点接入**：说明是总线型网络，许多计算机以多点接入的方式连接在一根总线上；

**载波监听**：用电子技术检测总线上有没有其他计算机也在发送。不管是在发送前、发送中，每个站都必须不停的检测信道。在发送前检测信道，是为了获得发送权，如果检测到已经有其他站在发送，则自己就暂时不发送数据，必须要等到信道空闲时才发送。在发送中检测信道，是为了及时发现有没有其他站的发送和本站发送的碰撞，即碰撞检测。

**碰撞检测**：边发送边监听，适配器边发送数据边检测信道上的信号电压变化情况，以便判断自己在发送数据时其他站是否也在发送数据。当几个站同时在总线上发送数据时，总线上的信号电压变化幅度会增大（互相叠加）。当适配器检测到时，就认为总线上至少有两个站同时在发送数据，表明产生了碰撞。适配器将立即停止发送。

在使用**CSMA/CD**协议时，一个站不可能同时进行发送和接收（但必须边发送边监听信道），因此使用CSMA/CD协议的以太网不可能进行全双工通信而只能进行双向交替通信（半双工通信）。

适配器每发送一个新的帧，就要执行一次CSMA/CD算法，适配器对过去发生过的碰撞并无记忆功能。

电磁波在1km电缆的传播时延约为5us。假设总线上单程端到端传播时延为τ，站A发送数据后，最迟要经过2τ的时间才能知道自己发送的数据和其他站发送的数据是否发生了碰撞。所以，每一个站在自己发送数据之后的一小段时间内，存在着遇到碰撞的可能性。这一小段时间取决于两端之间的距离。因此，以太网不能保证某一时间内一定能够把自己的数据帧成功地发送出去，这一特点称为**发送的不确定性**。

以太网的端到端的往返时间2τ称为争用期或碰撞窗口。经过争用期这段时间还没有检测到碰撞，才能肯定这次发送不会发生碰撞。协议规定，争用期时间为51.2us。对于10Mb/s以太网，在争用期内可以发送512bit，即64字节。以太网规定，一个最短帧长为64字节。如果发送的数非常少，少于64字节，那么必须加入填充字节至64字节。所以，以太网在发送数据时，如果在争用期没有发生碰撞，那么后续发送的数据一定不会发生冲突。也就是说，如果发生碰撞，则一定是发生在前64字节之内。因此，凡长度小于64字节的帧都是由于冲突而异常终止的无效帧，只要收到了这种无效帧，应该立即丢弃。

## 3. 以太网的MAC层

#### 1. MAC层的硬件地址

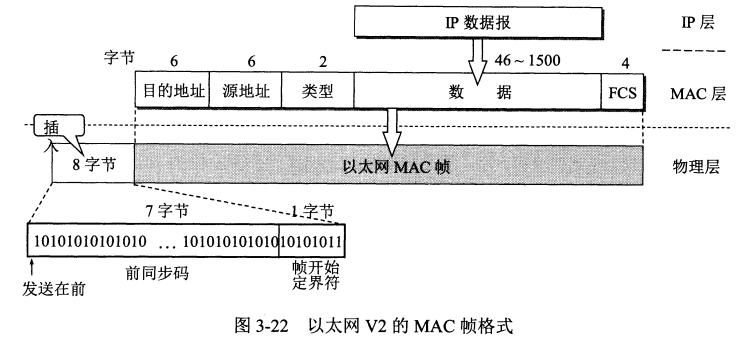
局域网中，硬件地址又称为物理地址或者MAC地址（因为这种地址用在MAC帧中），固化在适配器的ROM中，6字节共48位。

适配器有过滤功能，从网络上每收到一个MAC帧就先硬件检查MAC帧中的目的地址。如果是发往本站的帧则收下，否则就将此帧丢弃。发往本站的帧包括三种：

* 1. 单播帧：一对一，即收到的帧的MAC地址与本站的硬件地址相同；
  2. 广播帧：一对全体，即发送给本局域网上所有站点的帧（全1地址）；
  3. 多播帧：一对多，即发送给本局域网上一部分站点的帧。

以太网可设置为一种特殊的工作方式：**混杂模式**，工作在混杂模式的适配器只要听到有帧在以太网上传输就都悄悄地记录下来，而不管这些帧是发往哪个站。

#### 2. MAC帧格式



1. **目的地址和源地址**：各占6字节；
2. **类型**：2字节，标识上一层使用的什么协议；
3. **数据**：46~1500字节，最小长度64字节减去18字节的首部和尾部即为最小长度46字节。
4. **帧检验序列FCS**：4字节，使用CRC校验。

## 4. 扩展局域网

### 4.1. 在物理层扩展局域网

集线器扩展局域网。

### 4.2. 在数据链路层扩展局域网

#### 1. 网桥

网桥，根据MAC帧的目的地址对收到的帧进行转发和过滤。当网桥收到一个帧时，首先检查帧的目的MAC地址，再决定将该帧转发到哪一个接口，或者丢弃（即过滤）。

网桥依靠转发表（转发数据库或路由目录）来转发帧。

#### 2. 以太网交换机

以太网交换机实质上是一台多接口网桥。

# 五. Unix网络编程

## 1. IO复用

1. IO复用典型地应用在如下场合：  
   当客户处理多个描述字时（一般是交互式输入和网络套接口），必须使用IO复用；
2. 一个客户同时处理多个套接口（很少出现，但有可能）；
3. 一个TCP服务器既要处理监听套接口，又要处理已连接套接口，一般也要用到IO复用；
4. 一个服务器既要处理TCP又要处理UDP，一般也要使用IO复用；
5. 如果一个服务器要处理多个服务或多个协议，一般要是用IO复用。

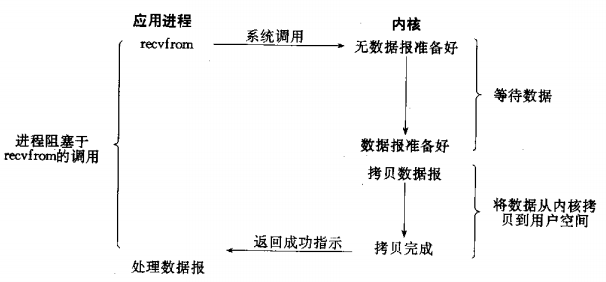
## 2. IO模型

五个IO模型：

* 阻塞IO
* 非阻塞IO
* IO复用（select和poll）
* 信号驱动IO（SIGIO）
* 异步IO（Posix.1的aio\_系列函数）

对于一个套接口上的输入操作，第一步一般是等待数据到达网络，当分组到达时，将其拷贝到内核中的某个缓冲区；第二部是将数据从内核缓冲区拷贝到应用缓冲区。

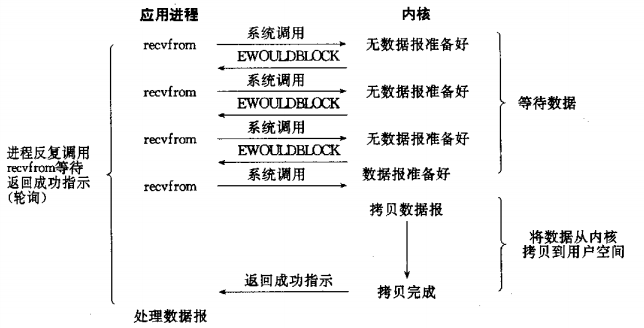
### 阻塞IO模型



UDP：数据准备好指整个数据报是否已接收；

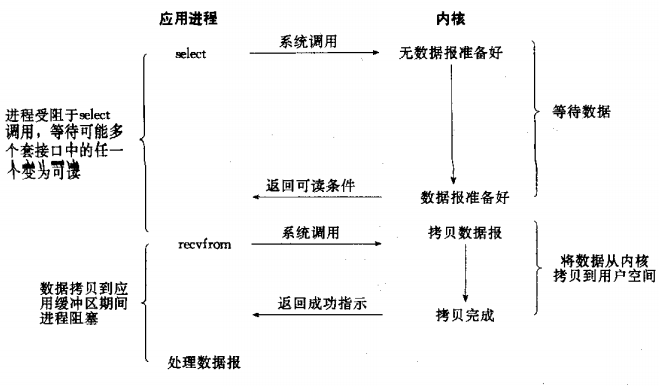
TCP：数据准备好与套接口的低潮限度等附加变量有关。

### 2.2. 非阻塞IO模型



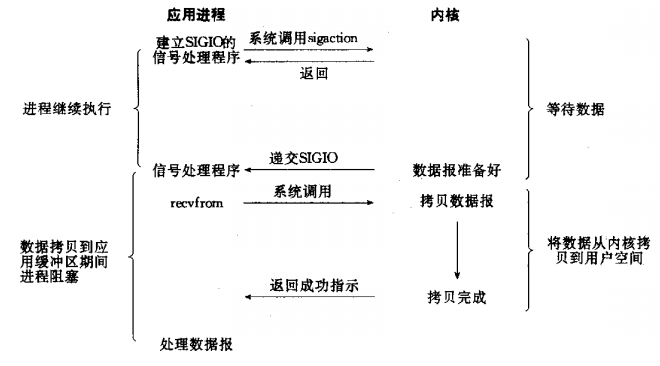
应用进程轮询，连续不断查询内核，看看某操作是否已准备好。

### 2.3. IO复用模型



调用select或poll，在这两个系统调用中的某一个上阻塞，而不是阻塞于真正的IO系统调用。阻塞于select调用，等待数据报套接口可读。

### 2.4. 信号驱动IO模型



首先，允许套接口进行信号驱动IO，并通过系统调用sigaction安装（注册）信号处理程序。此系统调用立即返回，进程继续工作，它是非阻塞的。当数据报准备好被读时，就为该进程生成一个**SIGIO**信号，然后可以在信号处理程序中调用recvfrom读取数据报。

### 2.5. 异步IO模型

异步IO模型是由内核通过我们IO操作何时完成。

