为什么要有socket，socket是什么？？

应用层通过传输层进行数据通信时，TCP和UDP会遇到同时为多个应用程序进程提供并发服务的问题。多个TCP连接或多个应用程序进程可能需要通过同一个TCP协议端口传输数据。为了区别不同的应用程序进程和连接，许多计算机操作系统为应用程序与TCP／IP协议交互提供了称为套接字(Socket)的接口，区分不同应用程序进程间的网络通信和连接。   
生成套接字，主要有3个参数：通信的目的IP地址、使用的传输层协议(TCP或UDP)和使用的端口号。Socket原意是“插座”。通过将这3个参数结合起来，与一个“插座”Socket绑定，应用层就可以和传输层通过套接字接口，区分来自不同应用程序进程或网络连接的通信，实现数据传输的并发服务。

# **TCP字节流和UDP数据报区别**

Jan 1, 2015 • Hector

两者的区别在于TCP接收的是一堆数据，而每次取多少由主机决定;而UDP发的是数据报，客户发送多少就接收多少。

拥有这些区别的原因是由于TCP和UDP的特性不同而决定的。TCP是面向连接的，也就是说，在连接持续的过程中，socket中收到的数据都是由同一台主机发出的，因此，知道保证数据是有序的到达就行了，至于每次读取多少数据自己看着办。 而UDP是无连接的协议，也就是说，只要知道接收端的IP和端口，且网络是可达的，任何主机都可以向接收端发送数据。这时候，如果一次能读取超过一个报文的数据，则会乱套。比如，主机A向发送了报文P1，主机B发送了报文P2，如果能够读取超过一个报文的数据，那么就会将P1和P2的数据合并在了一起，这样的数据是没有意义的。

#两个协议的其他区别 ##TCP(Transmission Control Protocol)传输控制协议 TCP是面向连接的、可靠的流协议。流就是指不间断的数据结构。TCP为提供可靠性传输，实行“顺序控制”或“重发控制”机制。此外还具备“流控制（流量控制）”、“拥塞控制”、提高网络利用率等众多功能。 此外，TCP作为一种面向有连接的协议，只有在确认通信对端存在才会发送数据，从而可以控制通信流量的浪费。

##UDP (User Datagram Protocol) 用户数据报协议 UDP是不具有可靠性的数据报协议。细微的处理它会交给上层的应用去完成。在UDP的情况下，虽然可以确保发送信息的大小，却不能保证信息一定会到达。因此，应用有时会根据自己的需要进行重发处理。 UDP不提供复杂的控制机制，利用IP提供面向无连接的通信服务。由于UDP面向无连接，它可以随时发送数据。再加上UDP本身的处理既简单又高效，因此经常用于以下方面：

* 包总量较少的通信(DNS、SNMP等)
* 视频、音频等多媒体通信(即时通信)
* 限定于LAN等特定网络中的应用通信
* 广播通信(广播、多播)

##TCP和UDP区分使用 可能有人会认为，鉴于TCP是可靠的传输协议，那么它一定优于UDP。其实不然。TCP和UDP的优缺点无法简单地、绝对地去做比较。那么，对这两种协议应该如何加以区分使用呢？下面做一简单说明。

TCP用于在传输层有必要实现可靠传输的情况。由于它是面向有连接并具备顺序控制、重发控制等机制的，所以它可以为应用提供可靠传输。

而在一方面，UDP主要用于那些对高速传输和实时性有较高要求的通信或广播通信。举一个IP电话进行通话的例子。如果使用TCP，数据在传送途中如果丢失会被重发，但这样无法流畅地传输通话人的声音， 会导致无法进行正常交流。而采用UDP，它不会进行重发处理。从而也就不会有声音大幅度延迟到达的问题。即使有部分数据丢失，也只是会影响某一小部分的通话。此外， 在多播与广播通信中也使用UDP而不是TCP。 因此，TCP和UDP应该根据应用的目的按需使用。

## **数据帧（Frame）：是一种信息单位，它的起始点和目的点都是数据链路层。**

## **数据包（Packet）：也是一种信息单位，它的起始和目的地是网络层。**

## **数据报（Datagram）：通常是指起始点和目的地都使用无连接网络服务的的网络层的信息单元。**

## **段（Segment）：通常是指起始点和目的地都是传输层的信息单元。**

## **数据链路层的PDU叫做Frame（帧）；**

## **网络层的PDU叫做Packet（数据包）；**

## **TCP的叫做Segment（数据段）；**

## **udp如何实现可靠性传输？**

         UDP它不属于连接型协议，因而具有资源消耗小，处理速度快的优点，所以通常音频、视频和普通数据在传送时使用UDP较多，因为它们即使偶尔丢失一两个数据包，也不会对接收结果产生太大影响。

         传输层无法保证数据的可靠传输，只能通过应用层来实现了。实现的方式可以参照tcp可靠性传输的方式，只是实现不在传输层，实现转移到了应用层。

         实现确认机制、重传机制、窗口确认机制。

         如果你不利用linux协议栈以及上层socket机制，自己通过抓包和发包的方式去实现可靠性传输，那么必须实现如下功能：

         发送：包的分片、包确认、包的重发

         接收：包的调序、包的序号确认

         目前有如下开源程序利用udp实现了可靠的数据传输。分别为

**RUDP（Reliable User Datagram Protocol）**

**RTP（Real Time Protocol）**

**UDT（UDP based Data Transfer Protocol）**

# 3.1 RUDP RUDP（Reliable User Datagram Protocol）

RUDP 提供一组数据服务质量增强机制，如拥塞控制的改进、重发机制及淡化服务器算法等，从而在包丢失和网络拥塞的情况下， RTP 客户机（实时位置）面前呈现的就是一个高质量的 RTP 流。在不干扰协议的实时特性的同时，可靠 UDP 的拥塞控制机制允许 TCP 方式下的流控制行为。

可靠用户数据报协议（RUDP）是一种基于可靠数据协议（RDP： RFC908 和 1151 （第二版））的简单分组传输协议。作为一个可靠传输协议，RUDP 用于传输 IP 网络间的电话信号。它允许独立配置每个连接属性，这样在不同的平台可以同时实施不同传输需求下的协议。

**RUDP 提供一组数据服务质量增强机制，如拥塞控制的改进、重发机制及淡化服务器[算法](http://lib.csdn.net/base/datastructure" \o "算法与数据结构知识库" \t "http://blog.csdn.net/kennyrose/article/details/_blank)等，**从而在包丢失和网络拥塞的情况下， RTP 客户机（实时位置）面前呈现的就是一个高质量的 RTP 流。在不干扰协议的实时特性的同时，可靠 UDP 的拥塞控制机制允许 TCP 方式下的流控制行为。

为了与网络 TCP 通信量同时工作，**RUDP 使用类似于 TCP 的重发机制和拥塞控制算法。**在最大化利用可用带宽上，这些算法都得到了很好的证明。

RUDP特性

客户机确认响应服务器发送给客户机的包；

视窗和拥塞控制，服务器不能超出当前允许带宽；一旦发生包丢失，服务器重发给客户机；比实时流更快速，称为“缓存溢出”。用户数据报协议（UDP）

# 3.2 RTP Real Time Protocol

         实时传输协议（RTP）为数据提供了具有实时特征的端对端传送服务，如在组播或单播网络服务下的交互式视频音频或模拟数据。应用程序通常在 UDP 上运行 RTP 以便使用其多路结点和校验服务；这两种协议都提供了传输层协议的功能。但是 RTP 可以与其它适合的底层网络或传输协议一起使用。如果底层网络提供组播方式，那么 RTP 可以使用该组播表传输数据到多个目的地。

RTP 本身并没有提供按时发送机制或其它服务质量（QoS）保证，它依赖于底层服务去实现这一过程。 RTP 并不保证传送或防止无序传送，也不确定底层网络的可靠性。 RTP 实行有序传送， RTP 中的序列号允许接收方重组发送方的包序列，同时序列号也能用于决定适当的包位置，例如：在视频解码中，就不需要顺序解码。

RTP，实时协议被用来为应用程序如音频，视频等的实时数据的传输提供端到端（end to end）的网络传输功能。传输的模型可以是单点传送或是多点传送。数据传输被一个姐妹协议——实时控制协议（RTCP）来监控，后者允许在一个大的多点传送网络上监视数据传送，并且提供最小限度的控制和识别功能。

RTP是被IETF在RFC1889中提出来的。顺带提及，RTP已经被接受为实时多媒体传送的通用标准。ITU-T跟IETF都在各自的系统中将这一协议标准化。

**1.1 为何需要RTP?**

TCP不能支持像交互视频，会议等的实时服务，这一原因是由于TCP只是一个“慢”协议，需要三次握手。就此在IP层上UDP是一个比TCP更好的选择。**但是UDP是本质上是一个不可靠协议，不支持在包丢情况下的重传机制。诚然，UDP有一些特性，比如多路复用跟校验和服务，这些都是对实时服务很有利的。为了消除UDP的缺点，RTP是作为应用层而被提出来的。**

RTP提供的各种服务包括**有效负载识别**，**序列编号**，**时间戳**和**投递监听**。RTP能够序列化包，当这些包在收端不是按顺序到达的时。序列号也能被用来识别包丢失。时间戳被用于媒体有效的播放。到达的数据一直被RTCP监听，以通知RTP层来校正其编码和传输的参数。例如，如果RTCP层检测到包丢失，它会通知RTP层减缓发送速率。

尽管RTP有助于实时媒体的有效的播放 ，但是要注意的是RTP自身并不提供任何机制来确保及时传递或提供其他服务质量（QoS）的保证，而是依靠低层服务来完成这些。同样，RTP也不保证投递或防止无序投递。RTP被设计出来主要是为了满足有多个参加者的多媒体会议的需要。RTP也同样适合于象持续数据的储存，分布式交互仿真，主动标记以及应用程序的控制和测量。

1.2 RTP特性一览

RTP提供有效负荷类型识别，乱序重排和利用时间戳的媒体有效播放。

RTCP监控服务质量，也提供在一个当前进行的会话中传送关于参加者的信息作用。

RTP对于下层协议是独立的，它能够工作在像TCP/IP，ATM，帧时延等类型的网络上。

如果被下层网络支持，RTP支持利用多路技术的对于多点的数据传输。

RTP序列号也能被用来确定包的合适位置。例如在视频解码，包无需按序解码。

### 3.3UDT

         基于UDP的数据传输协议（**UDP-based Data Transfer Protocol**，简称UDT）是一种互联网数据传输协议。UDT的主要目的是支持高速广域网上的海量数据传输，而互联网上的标准数据传输协议TCP在高带宽长距离网络上性能很差。顾名思义，UDT建于UDP之上，并引入新的拥塞控制和数据可靠性控制机制。UDT是面向连接的双向的应用层协议。它同时支持可靠的数据流传输和部分可靠的数据报传输。由于UDT完全在UDP上实现，它也可以应用在除了高速数据传输之外的其它应用领域，例如点到点技术（P2P），防火墙穿透，多媒体数据传输等等。

基于UDP的数据传输协议（UDP-based Data Transfer Protocol，简称UDT）是一种互联网数据传输协议。UDT的主要目的是**支持高速广域网上的海量数据传输**，而互联网上的标准数据传输协议TCP在高带宽长距离网络上性能很差。 顾名思义，**UDT建于UDP之上，并引入新的拥塞控制和数据可靠性控制机制。UDT是面向连接的双向的应用层协议。它同时支持可靠的数据流传输和部分可靠的数据报传输。** 由于UDT完全在UDP上实现，它也可以应用在除了高速数据传输之外的其它应用领域，例如点到点技术（P2P），防火墙穿透，多媒体数据传输等等。

#### UDT是双工的，每个UDT实体有两个部分：发送和接收。发送者根据流量控制和速率控制来发送（和重传）应用程式数据。接收者接收数据包和控制包，并根据接收到的包发送控制包。发送和接收程式共享同一个UDP端口来发送和接收。接收者也负责触发和处理任何的控制事件，包括拥塞控制和可靠性控制和他们的相对机制，例如RTT估计、带宽估计、应答和重传。UDT总是试着将应用层数据打包成固定的大小，除非数据不够这么大。和TCP相似的是，这个固定的包大小叫做MSS（最大包大小）。由于期望UDT用来传输大块数据流，我们假定只有很小的一部分不规则的大小的包在UDT session中。MSS能够通过应用程式来安装，MTU是其最优值（包括任何包头）。UDT拥塞控制算法将速率控制和窗口（流量控制）合并起来，前者调整包的发送周期，后者限制最大的位被应答的包。在速率控制中使用的参数通过带宽估计技术来更新，他继承来自基于接收的包方法。同时，速率控制周期是估计RTT的常量，流控制参数依赖于对方的数据到达速度，另外接收端释放的缓冲区的大小。

**UDT协议是什么？**是一种基于UDP的数据传输协议（UDP-based Data Transfer Protocol，简称UDT）。

**UDT协议的主要作用是什么？**UDT的主要目的是支持高速广域网上的海量数据传输，而互联网上的标准数据传输协议TCP在高带宽长距离网络上性能很差。

**那么UDT与UDP的区别又是什么？**UDT建于UDP之上，并引入新的拥塞控制和数据可靠性控制机制。UDT是面向连接的双向的应用层协议。它同时支持可靠的数据流传输和部分可靠的数据报传输。

**UDT的使用场景是什么？**由于UDT完全在UDP上实现，它也可以应用在除了高速数据传输之外的其它应用领域，例如点到点技术（P2P），防火墙穿透，多媒体数据传输等等。

##### UDT协议的主要特性有哪些？

**基于UDP的应用层协议**： 有基本网络知识的朋友都知道TCP和UDP的区别和使用场景，但是有没有一种协议能同时兼顾TCP协议的安全可靠和UDP协议的高效，那么UDT就是一种。

**面向连接的协议**：面向连接意味着两个使用协议的应用在彼此交换数据之前必须先建立一个连接，当然UDT是逻辑上存在的连接通道。这种连接的维护是基于握手、Keep-alive（保活）以及关闭连接。

**可靠的协议**：依靠包序号机制、接收者的ACK响应和丢包报告、ACK序号机制、重传机制(基于丢包报告和超时处理)来实现数据传输的可靠性。

**双工的协议**：每个UDT实例包含发送端和接收端的信息。

**单播的数据流。**

**新的拥塞算法，并且具有可扩展的拥塞控制框架**：新的拥塞控制算法不同于基于窗口的TCP拥塞控制算法(慢启动和拥塞避免)，是混合的基于窗口的、基于速率的拥塞控制算法。可扩展的拥塞控制框架开源的代码和拥塞控制的C++类架构，可支持开发者派生专用的拥塞控制算法。

**带宽估计**：UDT使用对包(PP -- Packet pair)的机制来估计带宽值。即每16个包为一组,最后一个是对包,即发送方不用等到下一个发送周期内再发送。接收方接收到对包后对其到达时间进行记录,可结合上次记录的值计算出链路的带宽(计算的方法称为中值过滤法), 并在下次ACK中进行反馈。

##### UDT一些主要特性的实现

**UDT包确认机制是基于时间的定时器实现的**。原理：

uses timer-based selective acknowledgement, which generates an acknowledgement at a fixed interval. This means that the faster the transfer speed, the smaller the ratio of bandwidth consumed by control traffic. Meanwhile, at very low bandwidth, UDT acts like protocols using cumulative acknowledgement.

The ACK interval of UDT is the same as the rate control interval (SYN).

To support this scheme, negative acknowledgement (NAK) is used to explicitly feed back packet loss. NAK is generated once a loss is detected so that the sender can **[React](http://lib.csdn.net/base/react" \o "React知识库" \t "http://blog.csdn.net/bytxl/article/details/_blank)** to congestion as quickly as possible. The loss information (sequence numbers of lost packets) will be resent after an increasing interval if there are timeouts indicating that the retransmission or NAK itself has been lost.

**UDT流量控制是基于以下三个机制实现的。**

**new congestion control 新的拥塞控制**

**DAIMD rate control 递减式速率控制**

**dynamic window control 动态窗口机制**

详细说明在这：[http://www.jenkinssoftware.com/raknet/manual/congestioncontrol.html](http://demo.netfoucs.com/u012730075/article/details/38236021" \t "http://blog.csdn.net/bytxl/article/details/_blank)

##### **UDT支持哪些数据传输类型**

**基于流的send， recv。**

**基于数据报sendmsg，recvmsg。**

**文件传输sendfile，recvfile。**

##### 下面我们结合UDT version 4版本来给大家分析下这个版本的UDT所拥有的一些新的特性。

**使用了UDP multiplexer（UDP多路复用）机制**，这样做的好处是：

therefore it is possible (and by default) all UDT sockets in one process will share one UDP port. This scheme makes it easier for firewall traversing.

UDT流控采用可配置的拥塞控制算法，你可以关闭它、配置它，改良它来实现自己需要的流控策略。

采用新的资源管理（内存管理）和共享的拥塞控制方法来支持更多并发的UDT连接。有关内存管理的介绍如下：

UDT4 has a new buffer management module that enables all UDT sockets in one process can share protocol buffer. The goodness it brings is the much less memory usage for multiple UDT connections compared to previous versions.

UDT4 can automatically resize its buffer in order to reduce memory usage while providing maximum throughput.

Because of the new memory management scheme, overlapped IO has been removed from UDT4. If your existing code uses overlapped IO, you need to modify it to use regular IO. This is the only change needed for exiting code to move from UDT3 to UDT4.

其他的特点：

不和原生socket api冲突；

线程安全；

进程间不共享句柄；

错误处理；

防火墙穿透；

安全性好；

建立连接快速；

以上就是我依据自己的记忆以及笔记对UDT协议进行的分析总结，笔记在此[UDT协议研究.mmap](http://demo.netfoucs.com/u012730075/article/details/38236021" \t "http://blog.csdn.net/bytxl/article/details/_blank) 欢迎大家下载。非常感谢谷云洪博士为我们提供了这么一个优秀的协议。

# 2. 设计目标

　　**UDT主要用在小数量的bulk源共享富裕带宽的情况下**，最典型的例子就是建立在光纤广域网上的网格计算，一些研究所在这样的网络上运行他们的分布式的数据密集程式，例如，远程访问仪器、分布式数据挖掘和高分辨率的多媒体流。  
　　**UDT的主要目标是效率、公平、稳定。**单个的或少量的UDT流应该利用任何高速连接提供的可用带宽，即使带宽变化的很剧烈。同时，任何并发的流必须公平地共享带宽，不依赖于不同的带宽瓶劲、起始时间、RTT。稳定性需要包发送速率应该一直会聚可用带宽很快，并且必须避免拥塞碰撞。  
　　**UDT并不是在瓶颈带宽相对较小的和大量多元短文档流的情况下用来取代TCP的。**

**UDT主要作为TCP的朋友，和TCP并存，UDT分配的带宽不应该超过根据MAX-MIN规则的最大最小公平共享原则。**（备注，最大最小规则允许UDT在高BDP连接下分配TCP不能使用的可用带宽）。

# 3. 协议说明

## 3.1. 概述

**UDT是双工的，每个UDT实体有两个部分：发送和接收。**发送者根据流量控制和速率控制来发送（和重传）应用程式数据。接收者接收数据包和控制包，并根据接收到的包发送控制包。发送和接收程式共享同一个UDP端口来发送和接收。  
　　接收者也负责触发和处理任何的控制事件，包括拥塞控制和可靠性控制和他们的相对机制，例如RTT估计、带宽估计、应答和重传。  
　　**UDT总是试着将应用层数据打包成固定的大小，除非数据不够这么大。**和TCP相似的是，这个固定的包大小叫做**MSS（最大包大小）**。由于期望UDT用来传输大块数据流，我们假定只有很小的一部分不规则的大小的包在UDT session中。MSS能够通过应用程式来安装，MTU是其最优值（包括任何包头）。

**UDT拥塞控制算法将速率控制和窗口（流量控制）合并起来，前者调整包的发送周期，后者限制最大的位（未？）被应答的包。在速率控制中使用的参数通过带宽估计技术来更新，他继承来自基于接收的包方法。同时，速率控制周期是估计RTT的常量，流控制参数依赖于对方的数据到达速度，另外接收端释放的缓冲区的大小。**

## 3.2. 包结构

　　UDT有两种包：数据包和控制包。他们通过包头的第一位来区分（标志位）。假如是0，表示是数据包，1表示是控制包。

### 3.2.1. 数据包

　　数据包结构如下显示：

0 1 2 3

0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1

+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

|0| Packet Sequence Number |

+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

|FF |O| Message Number |

+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

| Time Stamp |

+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

| Destination Socket ID |

+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

　　数据包以0开头，包序号是UDT数据包头中唯一的内容。它是个无符号整数，使用标志位后的31位，UDT包是基于序列的，例如，每个非重传的包都增加序号1。序号在到达最大值2^31-1的时候覆盖。紧跟在这些数据后面的是应用程式数据。

The next 32-bit field in the header is for the messaging. The first

two bits "FF" flags the position of the packet is a message. "10" is

the first packet, "01" is the last one, "11" is the only packet, and

"00" is any packets in the middle. The third bit "O" means if the

message should be delivered in order (1) or not (0). A message to be

delivered in order requires that all previous messages must be either

delivered or dropped. The rest 29 bits is the message number, similar

to packet sequence number (but independent). A UDT message may

contain multiple UDT packets.

Following are the 32-bit time stamp when the packet is sent and the

destination socket ID. The time stamp is a relative value starting

from the time when the connection is set up. The time stamp

information is not required by UDT or its native control algorithm.

It is included only in case that a user defined control algorithm may

require the information (See [Section 6](http://tools.ietf.org/search/draft-gg-udt-03" \l "section-6" \t "http://blog.csdn.net/bytxl/article/details/_blank)).

The Destination ID is used for UDP multiplexer. Multiple UDT socket

can be bound on the same UDP port and this UDT socket ID is used to

differentiate the UDT connections.

### 3.2.2. 控制包结构

      控制包结构如下：

0 1 2 3

0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1

+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

|1| Type | Reserved |

+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

| | Additional Info |

+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

| Time Stamp |

+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

| Destination Socket ID |

+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

| |

~ Control Information Field ~

| |

+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

There are 8 types of control packets in UDT and the type information

is put in bit field 1 - 15 of the header. The contents of the

following fields depend on the packet type. The first 128 bits must

exist in the packet header, whereas there may be an empty control

information field, depending on the packet type.

Particularly, UDT uses sub-sequencing for ACK packet. Each ACK packet

is assigned a unique increasing 16-bit sequence number, which is

independent of the data packet sequence number. The ACK sequence

number uses bits 32 - 63 ("Additional Info") in the control packet

header. The ACK sequence number ranges from 0 to (2^31 - 1).

TYPE 0x0: Protocol Connection Handshake

Additional Info: Undefined

Control Info:

1) 32 bits: UDT version

2) 32 bits: Socket Type (STREAM or DGRAM)

3) 32 bits: initial packet sequence number

4) 32 bits: maximum packet size (including UDP/IP headers)

5) 32 bits: maximum flow window size

6) 32 bits: connection type (regular or rendezvous)

7) 32 bits: socket ID

8) 32 bits: SYN cookie

9) 128 bits: the IP address of the peer's UDP socket

TYPE 0x1: Keep-alive

Additional Info: Undefined

Control Info: None

TYPE 0x2: Acknowledgement (ACK)

Additional Info: ACK sequence number

Control Info:

1) 32 bits: The packet sequence number to which all the

previous packets have been received (excluding)

[The following fields are optional]

2) 32 bits: RTT (in microseconds)

3) 32 bits: RTT variance

4) 32 bits: Available buffer size (in bytes)

5) 32 bits: Packets receiving rate (in number of packets

per second)

6) 32 bits: Estimated link capacity (in number of packets

per second)

TYPE 0x3: Negative Acknowledgement (NAK)

Additional Info: Undefined

Control Info:

1) 32 bits integer array of compressed loss information

(see [section 3.9](http://tools.ietf.org/search/draft-gg-udt-03" \l "section-3.9" \t "http://blog.csdn.net/bytxl/article/details/_blank)).

TYPE 0x4: Unused

TYPE 0x5: Shutdown

Additional Info: Undefined

Control Info: None

TYPE 0x6: Acknowledgement of Acknowledgement (ACK2)

Additional Info: ACK sequence number

Control Info: None

TYPE 0x7: Message Drop Request:

Additional Info: Message ID

Control Info:

1) 32 bits: First sequence number in the message

2) 32 bits: Last sequence number in the message

TYPE 0x7FFF: Explained by bits 16 - 31, reserved for user defined

Control Packet

Finally, Time Stamp and Destination Socket ID also exist in the

control packets.

　　UDT 的 ACK包使用子序列。每个ACK/ACK2包有一个无符号的16位序号，它跟数据包序列号不相关。ACK序列号使用控制包头中的32 - 63 位("Additional Info")。  
使用位16-31。应答需要从0到（2^16-1）。ACK序列号的范围从0~( 2^31 - 1 )。

　　注意，对于数据和控制包来说，能够从UDP协议头中得到实际的包大小。包大小信息能被用来得到有效的数据负载和NAK包中的控制信息字段大小。

## 3.3. 定时器

　　UDT在接收端使用4个定时器来触发不同的周期事件，每个定时器都有不同的周期，并且它们之间是相互独立的。这4个定时器包括**速率控制、应答、丢失报告（negative应答NAK）和重传/连接维护。**  
　　UDT中的定时器使用系统时间作为源。UDT接收端主动查询系统时间来检查一个定时器是否过期。对于某个定时器T来说，其拥有周期TP，将定变量t用来记录最近定时器T被配置或复位的时间。假如定时器T在系统时间t0被设置或者复位（ t = t0 ），那么任何t1（ t1 - t >= TP ）是定时器T过期的条件，定时器T过期会触发周期性事件E。

　　四个定时器是：RC定时器（原文为SND定时器）、ACK定时器、NAK定时器、EXP定时器。他们的周期分别是：**RCTP、ATP、NTP、ETP。**

**RC定时器（SND定时器）只用于发送端，用来发送基于速率的包。另外三个定时器只用于接收端。**

　　RC定时器用来触发周期性的速率控制。（见6.1）

        ACK定时器用来触发应答包（ACK）。它的周期由拥塞控制模块决定。然而，如果拥塞控制不需要触发基于时间的ACK，UDT会在0.01s内发送一个ACK。

　　NAK被用来触发丢失包应答（NAK包）。它的周期是动态的，为4 \* RTT\_+ RTTVar + SYN。其中RTTVar是RTT的方差（where RTTVar is the variance of RTT samples）。

        EXP重传定时器被用来触发一个数据包的重传和维护连接状态。它的周期是动态的，为N \* (4 \* RTT + RTTVar + SYN)，其中N为连续超时次数。为了避免不必要的超时，实现中会使用一个最小的门限值（例如：0.5s）。

        推荐的周期粒度是微秒。然而，除了RC定时器外，精准的时间保持不是必要的。

        （在这篇文章的余下部分，一个定时器变量机会用来表示定时器本身，也会用来表示它的周期，这取决于它的上下文。例如，ACK即可以用来表示ACK事件，也可以用来表示ACK周期。）

　　在每次bounded UDP接收操作（假如收到一个UDP包，一些额外的必须的数据处理时间）时查询系统时间来检查四个定时器是否已过期。UDP接收时间溢出值是实现的一个选择，这依赖于循环查询的负担和事件周期精确度之间的权衡。

　　速率控制事件更新包发送周期STP，UDT发送端使用STP来安排数据包的发送。假定一个数据包在时间t0被发送，那么下一次包发送时间是（t0+ STP）。换句话说，假如前面的包发送花费了t’时间，发送端将等待（STP-t’）来发送下一个数据包（假如 STP - t’ = 0 ，就无需等待了）。这个等待间隔需要一个高精确度的实现，推荐使用CPU时钟周期粒度。

## 3.4. 发送端算法

　　3.4.1. 数据结构和变量A． SND PKT历史窗口：一个循环数组记录每个数据包的开始时间  
　　B． 发送端丢失链表：发送段丢失列表是个连接链表，用来存储被接收方NAK包中返回的丢失包序号。这些数字以增加的顺序存储。  
　　3.4.2. 数据发送算法A． 假如发送端的丢失链表是非空的，重传第一个在list中的包，并删除该成员，到5。  
　　B． 等待有应用程式数据需要发送  
　　C． 假如未应答的包数量超过了两量窗口的大小，转到1。假如不是包装一个新的包并发送他。  
　　D．假如当前包的序号是16n，n是个整数，转第2步。  
　　E． 在SND PKT历史窗口中记录包的发送时间  
　　F． 假如这是自上次发送速率降低之后的第一个包，等外SYN时间。

　　G．等外（STP – t）时间，t是第1到第4步之间的总时间，然后转到1。

## 3.5. 接收端算法

### 3.5.1. 数据结构和变量

       A． 接收端丢失链表：是个duple连接链表，元素的值包括：丢失数据包的序号、最近丢失包的反馈时间和包已被反馈的次数。值以包序号增序的方式存储。  
　　B． 应答历史窗口：每个发送ACK的和时间一个循环数组；由于其循环的特性，意味着假如数组中没有更多空间的时候新的值将覆盖老的值。  
　　C． RCV PKT历史窗口：一个用来记录每个包到达时间的循环数组。  
　　D．对包窗口：一个用来记录每个探测包对之间的时间间隔。

　　E． LRSN：一个用来记录最大接收数据包需要的变量。LRSN被初始化为初始序号减1。

### 3.5.2. 数据接收算法

       A． 查询系统时间来检查RC、ACK、NAK、或EXP定时器是否过期。假如任一定时器过期，处理事件（本节下面介绍）并复位过期的定时器。  
　　B． 启动一个时间bounded UDP接收。假如每个包到，转1。  
　　C． 配置exp-count为1，并更新ETP为：ETP=RTT+4\*RTTVar + ATP。  
　　D．假如任何的发送数据包已被应答，复位EXP时间变量。  
　　E． 检查包头的标志位。假如是个控制包，根据类型处理他，然后转1。  
　　F． 假如当前数据包的需要是16n+1，n是个整数，记录当前包和上个在对包窗口中数据包的时间间隔。  
　　G．在PKT历史窗口中记录包到达时间  
　　H． 假如当前数据包的序号大于LRSN+1，将任何在（但不包括）这两个值之间的序号放入接收丢失链表，并在一个NAK包中将这些序号发送给发送端。假如序号小于LRSN，从接收丢失链表中删除他。

　　I． 更新LRSN，转1。

### 3.5.3. RC定时器

      到通过速率控制算法来更新STP（见3.6节）。  
  
　　过程如下：  
　　A． 按照下面的原则查找接收端所接收到的任何包之前的序号：假如接收者丢失链表是空的，ACK号码是LRSN+1，否则是在接收丢失队列中的最小序号。  
　　B． 假如应答号不大于曾被ACK2应答的最大应答号，或等于上次应答的应答号并且两次应答之间的时间间隔小于RTT+4\*RTTVar，停止（不发送应答）。  
　　C． 分配这个应答一个唯一增加的ACK序列号，推荐采用ACK序列号按步骤1增加，并且重叠在达到最大值之后。  
　　D．根据下面的算法来计算包的抵达速度：使用PKT历史窗口中的值计算最近16个包抵达间隔（AI）中值。在这16个值中，删除那些大于AI\*8或小于AI\*8的包，假如最后剩余8个值，计算他们的平均值(AI’)，包抵达速度是1/AI’（每秒包的数量），否则是0。  
　　E． 根据3.7节中的内容为每端（W）计算流量窗口。然后计算有效的流量窗口大小为：最大（W，可用接收方缓冲大小），2）。  
　　F． 根据下面的算法来计算连接容量估计。假如流量控制快启动阶段（3.7）一直继续，返回0，否则计算最近16个对包间隔（PI），这些值在对包窗口中，那么连接容量就是1/PI（每秒包的数量）。  
　　G．打包应答序列号，应答号，RTT，RTT 变量，有效的流量窗口大小并估计连接，将他们放入ACK包中，然后发送出去。

　　H． 记录ACK序列号，应答号和这个应答的开始时间，并放入历史窗口中。

### 3.5.4. 处理NAK定时器到时

       &Oslash; 查找接受方的丢失链表，找到任何上次反馈时间是（k\*（RTT+4\*RTTVar ) ）前的包，k当前这个包的反馈次数加1，假如没有反馈丢失，停止。  
　　&Oslash; 压缩第一步中得到的序号（见3.9），然后在一个NAK包中发送他们到发送方。

　　&Oslash; 假如不是停止流量控制快启动阶段。

### 3.5.5. 处理EXP定时器

       A． 假如发送端的丢失链表不是空的，停止  
　　B． 将任何未应答的包放到发送端的丢失链表中  
　　C． 假如(exp-count>16)并且自上次从对方接收到一个包以来的总时间超过3秒，或这个时间已超过3分钟了，这被认为是连接已断开，关闭UDT连接。  
　　D．假如没有数据，也就没有应答，发送一个保活包给对端，否则将任何未应答包的序号放入发送丢失列表中。  
　　E． 更新exp-count为：exp-count= exp-count+1

　　F． 更新ETP为：ETP=exp-count\*（RTT+4\*RTTVar）+ATP。

### 3.5.6. 收到应答包

      A． 更新最大的应答序号  
　　B． 更新RTT和RTTVar为：RTT = rtt， RTTVar = rv；rtt和rv是ACK包中的RTT和RTTVar值。  
　　C． 更新NTP和ETP为：NTP=RTT+4\*RTTVar；ETP=exp-count\*（RTT+4\*RTTVar）+ATP。  
　　D． 更新连接容量估计：B=（B\*7+b）/8，b是ACK包带的值。  
　　E． 更新流量窗口大小为ACK中的值。  
　　F． 发送ACK2包，并配置和ACK序号相同的应答号到对端

　　G． 复位EXP定时器

### 3.5.7. 当收到NAK包的时候

       A． 将任何NAK包中带的序号放入发送方的丢失列表中  
　　B． 通过速率控制来更新STP（见3.6）

　　C． 复位EXP定时器

### 3.5.8. 当收到ACK2包

       &Oslash; 在ACK历史窗口中根据接收到的ACK2序列号查找行营的ACK包。  
　　&Oslash; 更新曾被应答的最大应答号  
　　&Oslash; 根据ACK2的到达时间和ACK离开时间计算新的rtt值，并且更新RTT和RTTVar值为：  
　　RTTVar = (RTTVar \*3 +abs(rtt-RTT)/4  
　　RTT = (RTT \*7+rtt)/8  
　　RTT和RTTVar的初始值是0.1秒和0.05秒。  
　　&Oslash; 更新NTP和ETP为：  
　　NTP = RTT；

　　ETP = (exp-count +1)\* RTT+ATP

### 3.5.9. 当收到保活包的时候什么也不做

### 3.5.10. 当收到连接握手和关闭包的时候

      见3.8节

## 3.6. 速度控制算法

### 3.6.1. 速率控制

快启动STP被初始为最小的时间精度（1个CPU周期或1毫秒）。这是在快启动阶段，一般收到一个ACK包其携带的估计带宽大于0这个阶段就停止了。包的发送周期被配置为1/W，W是ACK携带的流量窗口的大小。

　　快启动阶段仅仅在开始一个UDT连接的时候发生，且不会在UDT连接的以后再出现。在快启动阶段之后，下面的算法就要工作了。

### 3.6.2. 当RC定时器时间到

       1． 假如在上一个RCTP时间内，没有收到一个ACK，停止  
　　2． 计算在上个RCTP时间内的丢失率，计算方法是根据总共发送的包和NAK反馈中总共丢失包的数量。假如丢失率大于0.1%，停止。  
　　3． 下个RCTP时间内发送包的增加数量如下计算：(inc)  
　　If (B  
　　Else inc = max (10^(ceil(log10((B-C)\*MSS\*8)))\*Beta/MSS,1/MSS)  
　　B是连接容量估计，C是当前的发送速度。两个都计算为每秒多少个包。MSS是以字节计算的；Beta是值为0.0000015的常量。  
　　4． 更新STP：STP=（STP\*RCTP）/（STP\*inc + RCTP）  
　　5． 计算真正的数据发送周期（rsp），从SND PKT历史窗口中得到，假如（STP）配置STP为（0.5 \* rsp）。

　　6． 假如（STP），配置STP为1.0。

### 3.6.3. 当收到NAK包时

#### 3.6.3.1. 数据结构和变量

        1． LSD：自上次速率降低后发送的最大序号  
　　2． NumNAK：自上次LSD更新以后的NAK数量  
　　3． AvgNAK：当最大序号大于LSD时两次事件之间的NAK移动的平均数。  
　　4． DR：在1到AvgNAK之间的随机平均数。

#### 3.6.3.2. 算法

       1． 假如NAK中最大的丢失序列号大于LSD：  
　　增加STP为：STP=STP\*（1+1/8）  
　　更新AvgNAK为：AvgNAK = （AvgNAK \*7 +NumNAK）/8  
　　更新DR  
　　复位 NumNAK = 0  
　　记录LSD

　　2． 否则，增加NumNAK按照1个步骤增加；假如NumNAK % DR = 0；增加STP为：STP=STP\*（1+1/8）；记录LSD。

## 3.7. 流量控制算法

　　流量控制窗口大小（W）初始值是16。

### 3.7.1. 当ACK定时器到的时候

        1． 流量控制快启动：假如没有NAK产生或W没有到达或超过15个包，并且AS>0，流量窗口大小更新为应答包的总数量。  
　　2． 否则，假如（AS>0），W更新为：（AS是包的到达速度）  
　　W= ceil (W \*0.875+AS\* (RTT +ATP) \*0.125)  
　　3． 限制W到对方最大流量窗口大小。

## 3.8. 连接建立和关闭

　　一个UDT实体首先作为一个SERVER启动，当一个客户端需要连接的时候其发送握手包。客户端在从服务端接收到一个握手响应包或时间溢出之前，应该每隔一段时间发送一个握手包（时间间隔由响应时间和系统overhead来权衡）。  
　　握手包有如下信息：  
　　1． UDT版本：这个值是兼容的目的。当前的版本是2  
　　2． 初始序号：这是发送这个UDT实体将来用于发送数据包的起始序号。他必须是个在1到（2^31-1）之间的随机值。另外，建议这个值在合理的时间历史窗口中不应该重复。  
　　3． MSS：数据包的大小（通过IP有效负载来度量）  
　　4． 最大的流量窗口大小：这是接收到握手信息的UDT实体允许的最大流量窗口大小，窗口大小通常限制为接收端的数据结构大小。  
　　服务器接收到一个握手包之后，比较MSS值和他自己的值并配置他自己的值为较小的值。结果值也在握手响应中被发送到客户端，另外更有服务器的版本信息，初始序列号，最大流量窗口大小。  
　　版本字段用来检查两端的兼容性。初始序列号和最大流量窗口大小用于初始化接收到这个握手包的UDT实体参数。  
　　服务器在第一步完成以后就准备发送或接收数据。然而，只要从同一个客户端接收任何握手包，其应该发送响应包。  
　　客户端一旦得到服务器的一个握手响应其就进入发送和接收数据状态。配置他自己的MSS为握手响应包中的值并初始化相应的参数为包中的值（序列号、最大流量窗口）。假如收到任何其他的握手信息，丢掉他。

　　假如其中的UDT实体要关闭，他将发送一个关闭信息到对端；对方收到这个信息以后将自己关闭。这个关闭信息通过UDP传输，仅仅发送一次，并不确保一定收到。假如消息没有收到，对方将根据时间溢出机制来关闭连接。

## 3.9. 丢失信息的压缩方案

　　NAK包中携带的丢失信息是个32-bit整数的数组。假如数组的中数字是个正常的序号（第1位是0），这意味着这个序号的包丢失了，假如第1位是1，意味着从这个号码开始（包括该号码）到下一个数组中的元素（包括这个元素值）之间的包（他的第1位必须是0）都丢失。  
　　例如，下面的NAK中携带的信息：  
　　0x00000002, 0x80000006, 0x0000000B, 0x0000000E

　　上面的信息表明序号为：2，6，7，8，9，10，11，14的包都丢了。

# 4. 效率和公平性

　　UDT能够充分利用当前有线网络的单独于连接容量的可用带宽 、RTT、后台共存流、给定的连接比特错误率。UDT在没有数据包丢失的情况下从0bits/s到90%带宽需要一个常量时间，这个时间是7.5秒。UDT并不适合无线网络。  
　　UDT的确满足单瓶劲网络拓扑的最大-最小公平性。在多个瓶劲情况下，根据最大最小原则他能确保较小瓶劲连接或至少一半的平等共享(it guarantees that flows over smaller bottleneck links obtain at least half of their fair share according to max-min rule)。RTT对公平性都一点影响。  
　　当和大块的TCP流共存的时候，TCP能占用比UDT更多的带宽，除了三种情况：  
　　1． 网络BDP很大，TCP不能利用他们的公平共享带宽。这种情况下，UDT将占用TCP不能利用的带宽。  
　　2． 连接容量是如此的小，从而导致UDT的带宽估计技术不能最有的工作；模拟显示这个极限连接容量大约是100kb/s。  
　　3． 在使用FIFO队列作为网络路径的网络中，假如队列大小大于BDP，TCP的共享带宽随着队列大小的增加而降低。然而，抵达UDT的共享带宽是，队列大小通常超过实际路由器/交换机提供的数量。  
　　当短（timewise）类似web的TCP流和小的并发UDT流共存的时候，UDT在TCP流上的效果很小。  
　　更多的分析在[GHG03]。  
　　5. 安全考虑UDT并没有使用特定的安全机制，相反，他依赖于应用程式提供的授权和底层提供的安全机制。  
　　然而，由于UDP是无连接的，UDT实现应该检查任何达到的包是否是预期的来源。这是从socket的API连接概念中继承而来，其连接只是接收指定来源的数据。  
　　6.UDT SOURCE CODE LINK

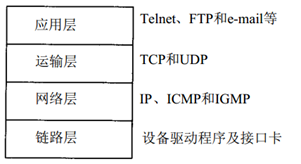
-------------------------------------UDT END-----------------------------------------------------------

### UDP构建可靠数据传输

简单来讲，要使用UDP来构建可靠的面向连接的数据传输，就要实现类似于TCP协议的超时重传，有序接受，应答确认，滑动窗口流量控制等机制，等于说要在传输层的上一层（或者直接在应用层）实现TCP协议的可靠数据传输机制，比如使用UDP数据包+序列号，UDP数据包+时间戳等方法，在服务器端进行应答确认机制，这样就会保证不可靠的UDP协议进行可靠的数据传输，不过这好像也是一个难题！

## **TCP/IP概述**

TCP/IP通常被认为是一个四层协议：

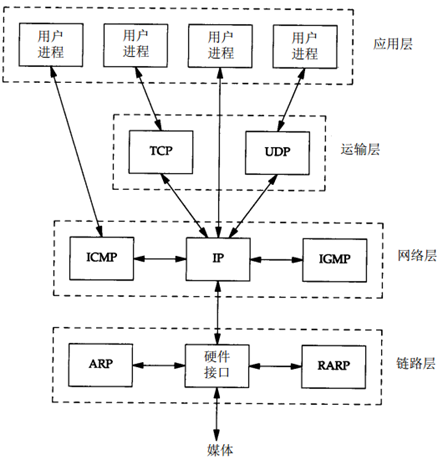


链路层。也称数据链路层或网络接口层，**包括设备驱动程序和网络接口卡**，它们一起**处理与电缆的物理接口细节**。

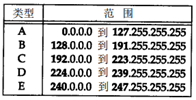
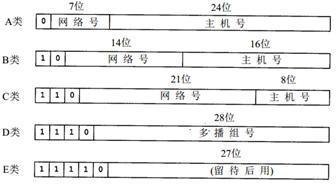
网络层。处理分组在网络中的活动，如分组的选路；网络层的协议包括**IP协议**、**ICMP**协议（Internet**互联网控制报文协议**）、**IGMP协议（Internet组管理协议）。**

传输层。主要为两台主机上的应用程序**提供端到端的通信**，包括**TCP（传输控制协议）**和**UDP（用户数据报协议）**。

应用层。处理特定的应用程序细节。

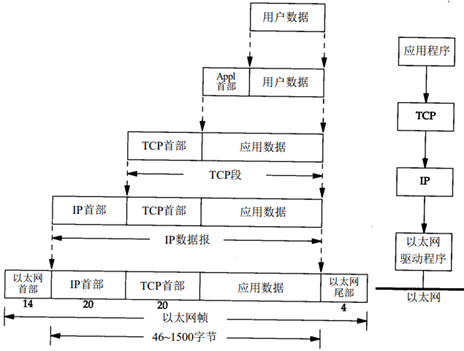
TCP/IP协议族中不同层次

互联网上的每个接口必须有唯一的IP地址，5类不同的IP地址格式和返回如下：



当应用程序用TCP传送数据时，数据被送入协议栈，然后逐个通过每一层直到被当作一串比特流送入网络，其中每一层对收到的数据都要增加一些首部信息。

## **链路层**

链路层主要有三个目的：

**为IP模块发送和接受IP数据；**

**为ARP模块发送ARP请求和接受ARP（地址解析协议）应答；**

外**RARP（反向地址转换协议）**发送RARP请求和**接受RARP应答**。

### **环回接口**

环回接口允许运行在同一台主机上的客户程序和服务器程序通过TCP/IP进行通信，IP地址127.0.0.1被分配给这个接口，并命名为localhost。

目的地址的环回地址时，照样还是传输层和网络层的所有过程，只是**当IP数据报离开网络层时把它返回给自己**，这样做简化了设计，因为环回接口可以看作是网络层下面的一个链路层。网络层把一份数据报传送给环回接口，就像传送给其它链路层一样，只不过环回接口把它返回到IP的输入队列中。

**MTU**

**以太网对数据帧的长度有一个限制，最大值是1500字节**。链路层的这个特性称作**MTU（最大传输单元）**。如果IP层数据报的长度比链路层的MTU还要大，那么IP层就需要进行分片，每一片都要小于MTU。

如果两台主机之间的通信要通过多个网络，那么每个网络的链路层就可能有不同的MTU。两台通信主机路径中的最小MTU，被称为路径MTU。两个方向上的选路不一定对称，因此路径MTU在两个方向上不一定是一致的。

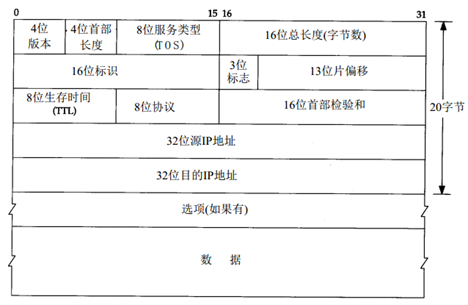
**IP协议**

IP是TCP/IP协议族中最为核心的协议**。所有的TCP、UDP、ICMP及IGMP数据都以IP数据报格式传输。**

**IP提供不可靠、无连接的数据服务**。不**可靠的是指它不能保证IP数据报能成功地到达目的地；无连接是指IP并不维护任何关于后续数据报的状态信息，每个数据报的处理是相互独立的。**

**IP数据报格式**

IP数据报格式如下图所示，普通的IP首部长为20字节，除非含有选项字段。



总长度字段是指整个IP数据报的长度，该字段长16位，所以**IP数据报的最大长度为65535字节**。

最后一个字段是任选项，是数据报中的一个可变长的可选信息，如记录路径、时间戳等，这些选项一般很少使用。

**IP路由选择**

IP路由选择是简单的，对于主机来说，如果目的主机与源主机直接相连或都在一个共享网络上，那么IP数据报就直接送到目的主机上；否则主机就把数据报发往一默认的路由器上，由路由器来转发该数据报。

IP层既可以配置成路由器的功能，也可以配置成主机的功能。**IP层在内存中有一个路由表**，当收到一份数据报并进行发送时，它都要对该表搜索一次。当数据报来自某个网络接口时，IP首先检查目的IP地址是否为本机的IP地址之一或者IP广播地址。如果是的话，数据报就被送到由IP首部协议字段所指定的协议模块进行处理。如果数据报的目的不是这些地址，如果IP层被设置为路由器的功能，那么就对数据报进行转发，否则数据报被丢弃。

**IP路由选择是逐跳进行的，IP并不知道到达任何目的的完整路径，所有IP路由选择只为数据报传输提供下一站路由器的IP地址**。IP路由选择主要完成这些功能：

**搜索路由表，寻找能与目的IP地址完全匹配的表目。**

**搜索路由表，寻找能与目的网络号相匹配的表目。**

**搜索路由表，寻找标为"默认"的表目。**

如果上面的步骤都没有成功，数据报就不能被传送。

**子网寻址**

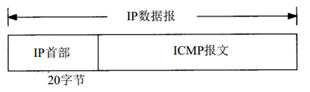
现在所有主机都要求支持子网编址。不是把IP地址看成单纯的一个网络号和一个主机号组成，而是吧主机号再分成一个子网号和一个主机号。

主机通过子网掩码来确定IP地址多少位用于子网号，多少位用于主机号。子网掩码是一个32位的值，**值为1的位留给网络号和子网号，为0的位留给主机号**。

**给定IP地址和子网掩码以后，主机就可以确定IP数据报的目的是：（1）本子网上的主机；（2）本网络中其它子网的主机；（3）其它网络上的主机。**

**ICMP**

**ICMP（Internet控制报文协议）是IP层一个组成部分，它传递差错报文以及其他需要注意的信息**，ICMP报文通常被IP层或更高层协议（TCP或UDP）使用，**一些ICMP报文把差错报文返回给用户进程**。**ICMP是在IP数据报内部被传输的。**



ICMP的一个规则是，ICMP差错报文必须包括生成该差错报文的数据报IP首部，还必须至少包括跟在该IP首部后面的前8个字节。这样就包含了UDP的首部，接收ICMP的系统就可以根据源端口号来把差错报文与某个特定的用户进程相关联。

**Ping**

Ping程序目的是为了测试另一台主机是否可达。该程序发送一份**ICMP回显请求报文**给主机，并等待返回ICMP回显应答。Ping程序还能测出到这台主机的往返时间。大多数的TCP/IP实现都在内核中直接支持Ping服务，这种服务器不是一个用户进程。

Ping程序通过在ICMP报文数据中存放请求的时间值来计算往返时间。当应答返回时，用当前时间减去存放在ICMP报文中的时间值，既是往返时间。

**Traceroute**

**Traceroute程序可以让我们看到IP数据报从一台主机传到另一台主机所经过的路由**，**Traceroute程序还可以让我们使用IP源路由选项**。

Traceroute程序的操作过程是，它发送一份TTL字段为1的IP数据报给目的主机。处理这份数据报的第一个路由器将TTL值减1，丢弃改数据报，并发回一份超时ICMP报文。这样就得到了该路径的第一个路由器地址。然后Traceroute程序发送一份TTL为2的数据报，这样我们就可以得到第二个路由器的地址。继续这个过程直至该数据报到达目的主机。

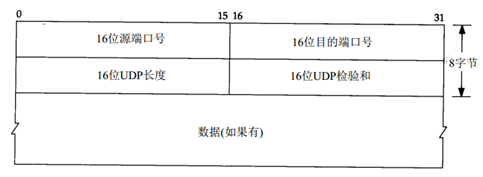
Traceroute程序发送一份UDP数据报给目的主机，但它选择一个不可能的值作为UDP端口号，将使目的主机的UDP模块产生一份"端口不可达"错误的ICMP报文，这样就可以区分接收到的ICMP报文是超时还是端口不可达，以判断什么时候结束。

**UDP**

UDP是一个简单的面向数据报的传输层协议。应用程序必须关心IP数据报的长度，如果它超过了MTU，就要对IP数据报进行分片。

**UDP首部**

UDP首部的各字段如下图：



**端口号表示发送进程和接收进程**，TCP端口号由TCP来查看，而UDP端口号由UDP来查看，TCP端口号和UDP端口号是相互独立的。

**IP数据报的最大长度时65535字节**，这是由IP首部16比特总长度字段所限制的。除去20字节的IP首部和8字节的UDP首部，UDP数据报中用户数据的最长长度为65507字节。

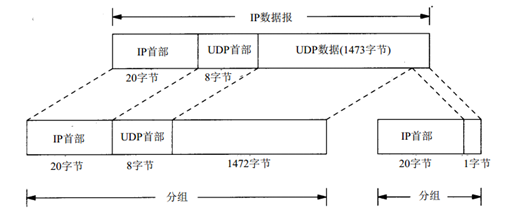
**IP分片**

IP把MTU与数据报长度进行比较，如果需要则进行分片。**分片可以发生在原始发送端主机上，也可以发生在中间路由器上。**把一份IP数据报分片以后，只有到达目的地才进行重新组装。**重新组装由目的端的IP层来完成**，**其目的是使分片和重新组装过程对传输层透明**。

IP数据报中的标识字段包含一个唯一值，该值在数据报分片时被复制到每个片中。标志字段用其中一个比特来表示"更多的片"，除最后一片外，其它每个组成数据报的片都要把该比特置1。片偏移字段指的是该片偏移原始数据报开始处的位置。

由于IP层本身没有超时重传的机制，即使只丢失一片数据也要重传整个数据报。如果对数据报分片的是中间路由器，而不是起始端系统，那么起始端系统就无法知道数据报是如何被分片的。所以要避免分片。

需要注意的是，任何传输层首部只出现在第1片数据中。



**TCP**

**TCP提供一种面向连接的、可靠的字节流服务**。TCP通过以下方式来提供可靠性：

**应用数据被分割成TCP认为最合适发送的数据块。**

TCP发送一个段后，它启动一个定时器，如果不能及时收到一个确认，将重发这个报文段。

TCP收到发自TCP连接另一端的数据时，将发送一个确认。

TCP将保持它首部和数据的校验和。

TCP对收到的数据进行重新排序，将收到的数据以正确的顺序交给应用层。

TCP的接收端会丢弃重复的数据。

TCP还能提供流量控制。

**TCP提供一种面向连接的、可靠的字节流服务**

**在一个TCP连接中，仅有两方进行彼此通信。广播和多播不能用于TCP**

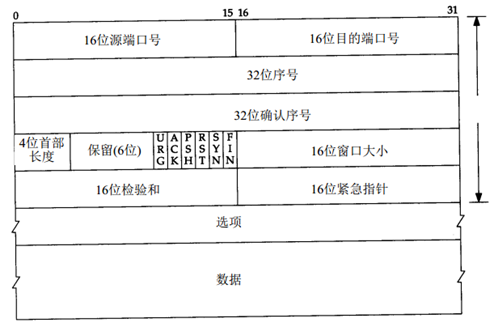
**TCP使用校验和，确认和重传机制来保证可靠传输**

**TCP使用累积确认**

**TCP使用滑动窗口机制来实现流量控制，通过动态改变窗口的大小进行拥塞控制**

**TCP首部**

TCP首部的数据格式如下图：



序号用来标识从TCP发送端想TCP接收端发送的数据字节流，它表示在这个报文段中的第一个数据字节，TCP用需要对每个字节进行计数。确认序号包含发送确认的一端所期望收到的下一个序号，确认序号是上次已成功收到的数据字节序号加1。

TCP的流量控制由连接的每一端通过声明的窗口大小来提供，**这个值是接收端正期望接收的字节。窗口大小是一个16位的字段，所以窗口大小最大是65535字节**，新的窗口扩大选项允许这个值按比例变化以提供更大的窗口。

**流量控制**

**TCP使用滑动窗口协议的形式进行流量控制**。该协议允许发送方在停止并等待确认前可以连续发送多个分组，可以加速数据的传输。

使用TCP的滑动窗口协议时，接收方不必确认每一个收到的分组，ACK是累积的，它们表示接收方已经正确收到了一直到确认序号减1的所有字节。

在应用程序读取了TCP缓冲区的数据后，TCP在需要的时候会发送一个ACK，它并不确认任何新数据，只是用来增加窗口的右边沿，因此被称为窗口更新。

滑动窗口的动态性总结如下：

1. 发送方不必发送一个全窗口大小的数据。

2. 来自接收方的一个报文段确认数据并把窗口向右边滑动。

3. 窗口的大小可以减小，但是窗口的右边沿却不能向左移动。

4. 接收方在发送一个ACK前不必等待窗口被填满。

TCP需要支持一种被称为**慢启动的算法**，该算法通过观察到新分组进入网络的速率应该与另一端返回确认的速率相同而进行工作。**慢启动为发送方的TCP增加了另一个窗口：拥塞窗口，拥塞窗口被初始化为1个报文段，每收到一个ACK，拥塞窗口就增加一个报文段。发送方取拥塞窗口与滑动窗口中的最小值作为发送上限。拥塞窗口是发送方使用的流量控制，而滑动窗口则是接收方使用的流量控制。**

**超时与重传**

对每个连接，TCP管理4个不同的定时器：

1. **重传定时器**用于当希望收到另一端的确认。

2. **坚持定时器**使窗口大小信息保持不断流动。

3. **保活定时器**可以检测到一个空闲连接的另一端何时崩溃或重启。

4. **2MSL定时器**测量一个连接处于TIME\_WAIT状态的时间。

TCP在对端ACK超时后按照一定的时间间隔进行重试，在多次重试仍超时后最终会放弃并发送一个复位信号。每次重试的时间间隔是一种被称为**指数退避**的倍乘关系，直至一个最大值，如重传间隔每次重传时增加1倍直至64秒。

TCP超时与重传中最重要的部分就是对一个给定连接的往返时间（RRT）的测量。由于路由器和网络流量均会变化，因此我们认为这个时间可能经常会发生变化，TCP应该跟踪这些变化并相应地改变其超时时间。

**TCP在收到一个失序的报文段时，立即需要产生一个ACK，这个重复的ACK目的在于让对方知道收到一个失序的报文段**。发送端端不知道的ACK是由一个丢失的报文段引起的，还是由于仅仅出现了几个报文段的重新排序，**如果一连串收到3个或3个以上重复ACK，就非常可能是一个报文段丢失了，于是发送端就重传丢失的数据报文段，而无需等待超时定时器溢出，这就是快速重传算法**。

**三次握手与四次挥手**

所谓三次握手(Three-way Handshake)，是指建立一个 TCP 连接时，需要客户端和服务器总共发送3个包。

**三次握手的目的是连接服务器指定端口，建立 TCP 连接，并同步连接双方的序列号和确认号，交换 TCP 窗口大小信息。**在 socket 编程中，客户端执行 connect() 时。将触发三次握手。

**1. 第一次握手(SYN=1, seq=x):**

客户端发送一个 TCP 的 SYN 标志位置1的包，指明客户端打算连接的服务器的端口，以及初始序号 X,保存在包头的序列号(Sequence Number)字段里。

发送完毕后，客户端进入 `**SYN\_SEND**` 状态。

**2. 第二次握手(SYN=1, ACK=1, seq=y, ACKnum=x+1):**

服务器发回确认包(ACK)应答。即 SYN 标志位和 ACK 标志位均为1。服务器端选择自己 ISN 序列号，放到 Seq 域里，同时将确认序号(Acknowledgement Number)设置为客户的 ISN 加1，即X+1。

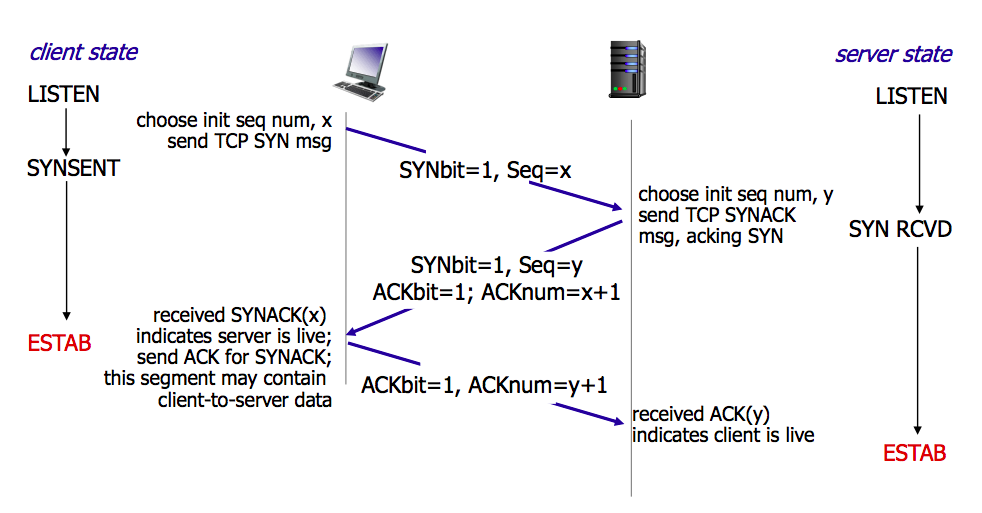
发送完毕后，服务器端进入 `**SYN\_RCVD**` 状态。

**3. 第三次握手(ACK=1，ACKnum=y+1)**

客户端再次发送确认包(ACK)，SYN 标志位为0，ACK 标志位为1，并且把服务器发来 ACK 的序号字段+1，放在确定字段中发送给对方，并且在数据段放写ISN的+1

发送完毕后，客户端进入 `ESTABLISHED` 状态，当服务器端接收到这个包时，也进入 `**ESTABLISHED**` 状态，TCP 握手结束。

三次握手的过程的示意图如下：



T**CP的连接的拆除需要发送四个包**，**因此称为四次挥手(Four-way handshake)**，也叫做改进的三次握手。客户端或服务器均可主动发起挥手动作，在 socket 编程中，任何一方执行 close() 操作即可产生挥手操作。

**第一次挥手(FIN=1，seq=x)**

假设客户端想要关闭连接，客户端发送一个 FIN 标志位置为1的包，表示自己已经没有数据可以发送了，但是仍然可以接受数据。

发送完毕后，客户端进入 FIN\_WAIT\_1 状态。

**第二次挥手(ACK=1，ACKnum=x+1)**

服务器端确认客户端的 FIN 包，发送一个确认包，表明自己接受到了客户端关闭连接的请求，但还没有准备好关闭连接。

发送完毕后，服务器端进入 CLOSE\_WAIT 状态，客户端接收到这个确认包之后，进入 FIN\_WAIT\_2 状态，等待服务器端关闭连接。

**第三次挥手(FIN=1，seq=y)**

服务器端准备好关闭连接时，向客户端发送结束连接请求，FIN 置为1。

发送完毕后，服务器端进入 LAST\_ACK 状态，等待来自客户端的最后一个ACK。

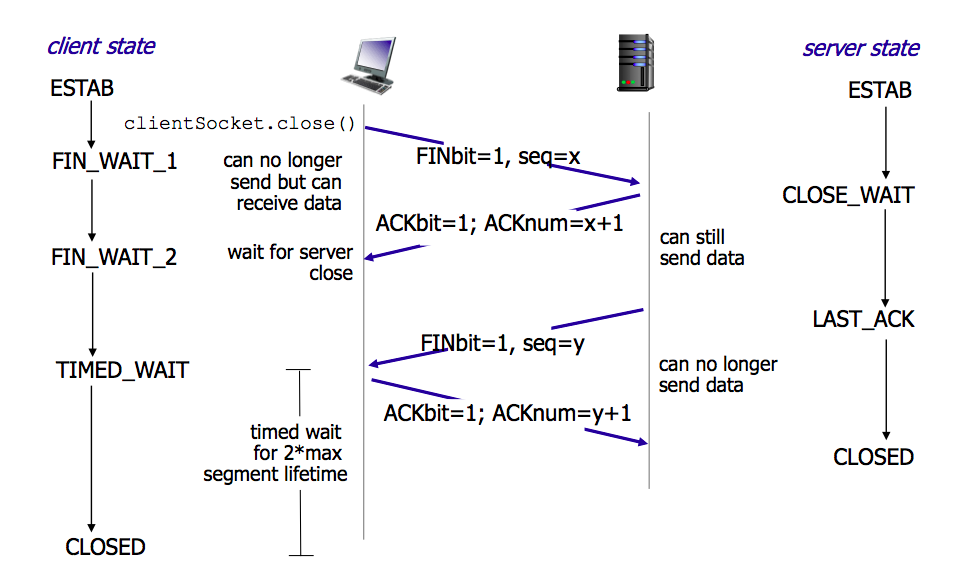
**第四次挥手(ACK=1，ACKnum=y+1)**

客户端接收到来自服务器端的关闭请求，发送一个确认包，并进入 TIME\_WAIT状态，等待可能出现的要求重传的 FIN包。

服务器端接收到这个确认包之后，关闭连接，进入 CLOSED 状态。

客户端等待了某个固定时间（两个**最大段生命周期**，2MSL，2 Maximum Segment Lifetime）之后，没有收到服务器端的 FIN ，认为服务器端已经正常关闭连接，于是自己也关闭连接，进入 CLOSED 状态。

四次挥手的示意图如下：



**SYN攻击**

**什么是 SYN 攻击（SYN Flood）？**

在三次握手过程中，服务器发送 SYN-ACK 之后，收到客户端的 ACK 之前的 TCP 连接称为半连接(half-open connect)。此时服务器处于 SYN\_RCVD 状态。当收到 ACK 后，服务器才能转入 ESTABLISHED 状态.

SYN 攻击指的是，攻击客户端在短时间内伪造大量不存在的IP地址，向服务器不断地发送SYN包，服务器回复确认包，并等待客户的确认。由于源地址是不存在的，服务器需要不断的重发直至超时，这些伪造的SYN包将长时间占用未连接队列，正常的SYN请求被丢弃，导致目标系统运行缓慢，严重者会引起网络堵塞甚至系统瘫痪。

SYN 攻击是一种典型的 DoS/DDoS 攻击。

**如何检测 SYN 攻击？**

检测 SYN 攻击非常的方便，**当你在服务器上看到大量的半连接状态时，特别是源IP地址是随机的，基本上可以断定这是一次SYN攻击**。在 Linux/Unix 上可以使用系统自带的 netstats 命令来检测 SYN 攻击。

**如何防御 SYN 攻击？**

SYN攻击不能完全被阻止，除非将TCP协议重新设计。我们所做的是尽可能的减轻SYN攻击的危害，常见的防御 SYN 攻击的方法有如下几种：

**缩短超时（SYN Timeout）时间**

**增加最大半连接数**

**过滤网关防护**

**SYN cookies技术**

**8.TCP为什么不是两次连接？而是三次握手？**

**如果A与B两个进程通信，如果仅是两次连接。可能出现的一种情况就是：A发送完请报文以后，由于网络情况不好，出现了网络拥塞，即B延时很长时间后收到报文，即此时A将此报文认定为失效的报文。B收到报文后，会向A发起连接。此时两次握手完毕，B会认为已经建立了连接可以通信，B会一直等到A发送的连接请求，而A对失效的报文回复自然不会处理。依次会陷入B忙等的僵局，造成资源的浪费。**

o**四次挥手释放连接时，等待2MSL的意义？**

o第 一，**为了保证A发送的最有一个ACK报文段能够到达B**。这个ACK报文段有可能丢失，因而使处在LAST-ACK状态的B收不到对已发送的FIN和ACK 报文段的确认。B会超时重传这个FIN和ACK报文段，而A就能在2MSL时间内收到这个重传的**ACK+FIN报文段**。接着A重传一次确认。

o第二，**报文可能会被混淆**，意思是说，其他时候的连接可能会被当作本次的连接。就是防止上面提到的已失效的连接请求报文段出现在本连接中，A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。

为了提高信道的利用率TCP协议不使用停止等待协议，而是使用连续ARQ协议，意思就是可以连续发出若干个分组然后等待确认，而不是发送一个分组就停止并等待该分组的确认。其中TCP的**流量控制与拥塞控制**是TCP在数据传输过程俩个重点机制，为TCP有效数据传输立下汗马功劳，这部分也是面试网络协议重点所在，下面从以下俩大方面总结一下 【1】流量控制 【2】拥塞控制 1、流量控制 **流量控制：指点对点通信量的控制，是端到端的问题**。

## 1、流量控制

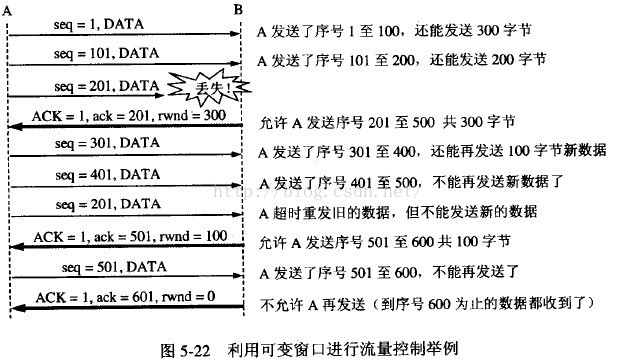
**流量控制：指点对点通信量的控制，是端到端的问题**。**流量控制所要做的就是抑制发送端发送数据的速率，以便使接收端来得及接收。**

在TCP中的数据流主要分为俩大类：成块数据流和交互数据流，（其中交互数据流主要是数据字节长度比IP首部/TCP首部都短，属于小分组），主要针对这俩种数据流类型做了不同的处理

### 1.1、**滑动窗口机制**

主要针对成块数据流，

**利用滑动窗口机制可以很方便地在TCP连接上实现对发送方的流量控制。**  
 设A向B发送数据。在连接建立时，B告诉了A：“我的接收窗口是 **rwnd = 400** ”(这里的 rwnd 表示 receiver window) 。因此，发送方的发送窗口不能超过接收方给出的接收窗口的数值。请注意，TCP的窗口单位是字节，不是报文段。TCP连接建立时的窗口协商过程在图中没有 显示出来。再设每一个报文段为100字节长，而数据报文段序号的初始值设为1。大写ACK表示首部中的确认位ACK，小写ack表示确认字段的值ack。



从图中可以看出，B进行了三次流量控制。第一次把窗口减少到 rwnd = 300 ，第二次又减到了 rwnd = 100 ，最后减到 rwnd = 0 ，即不允许发送方再发送数据了。这种使发送方暂停发送的状态将持续到主机B重新发出一个新的窗口值为止。B向A发送的三个报文段都设置了 ACK = 1 ，只有在ACK=1时确认号字段才有意义。  
注意： TCP为每一个连接设有一个**持续计时器(persistence timer)**。只要TCP连接的一方收到对方的零窗口通知，就启动持续计时器。若持续计时器设置的时间到期，就发送一个零窗口控测报文段（携1字节的数 据），那么收到这个报文段的一方就重新设置持续计时器

### 1.2 **交互数据流的Nagle算法(等待确认)**

其实Nagle更多的应该是提高传输效率，

**由于TCP的数据传输分为交互数据流和成块数据流**，**交互数据流一般是一些交互式应用程序的命令，所以这些数据很小，而考虑到TCP报头和IP报头的总和就有40字节，在局域网通常不会引起麻烦啊**，但是在广域网上这些小分组往往会增加拥塞的可能性。

**Nagle[算法](http://lib.csdn.net/base/31" \o "算法与数据结构知识库" \t "https://yq.aliyun.com/articles/_blank)（针对小分组）**：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。当发送方接收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再发送出去，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。

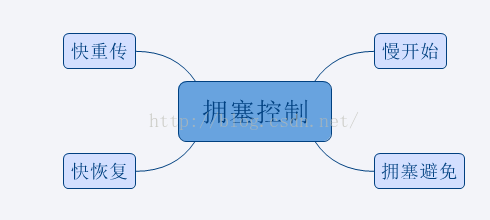
**该算法的优越之处在于他的自适应性：确认到达越快，数据也发送的越快。**

## **拥塞控制**

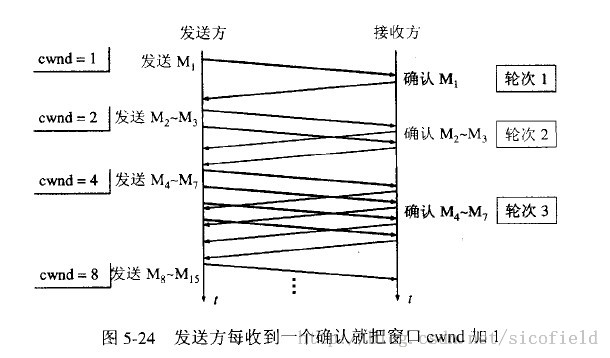
在某段时间，若对网络中的某一资源的需求超过了该资源所能提供的可用部分，网络的性能就要变化，这种情况叫做拥塞。

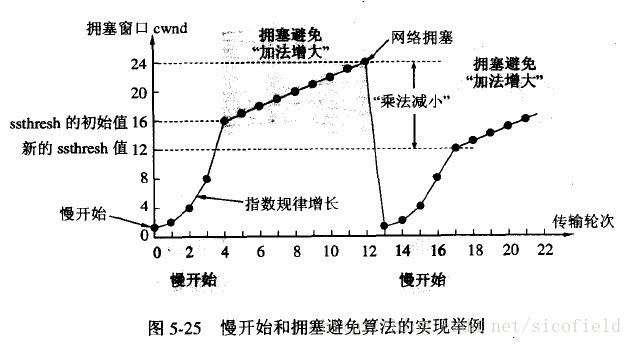
**拥塞控制就是防止过多的数据注入网络中，这样可以使网络中的路由器或链路不致过载**。**拥塞控制是一个全局性的过程**，和流量控制不同，流量控制指点对点通信量的控制。

通常有以下方式来避免拥塞



### 2.1、慢开始与拥塞避免

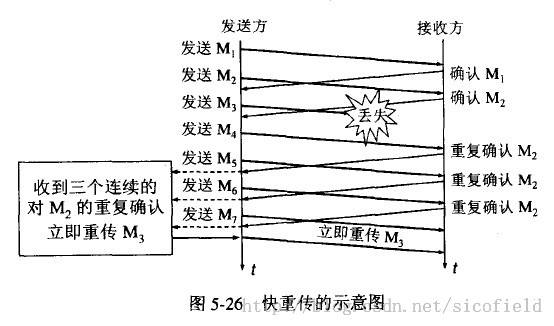
**发送方维持一个叫做拥塞窗口cwnd（congestion window）的状态变量**。拥塞窗口的大小取决于网络的拥塞程度，并且动态地在变化。**发送方让自己的发送窗口等于拥塞窗口，另外考虑到接受方的接收能力，发送窗口可能小于拥塞窗口**。慢开始算法的思路就是，不要一开始就发送大量的数据，先探测一下网络的拥塞程度，也就是说**由小到大逐渐增加拥塞窗口的大小。** 这里用报文段的个数的拥塞窗口大小举例说明慢开始算法，实时拥塞窗口大小是以字节为单位的。如下图：  
  


        当然收到单个确认但此确认多个数据报的时候就加相应的数值。所以一次传输轮次之后拥塞窗口就加倍。这就是乘法增长，和后面的拥塞避免算法的加法增长比较。  
 为了防止cwnd增长过大引起网络拥塞，还需设置一个**慢开始门限ssthresh**状态变量。ssthresh的用法如下：  
（1）**当cwnd<ssthresh时，使用慢开始算法。**  
（2）**当cwnd>ssthresh时，改用拥塞避免算法。**  
（3）**当cwnd=ssthresh时，慢开始与拥塞避免算法任意。**  
       拥塞避免算法让拥塞窗口缓慢增长，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。这样拥塞窗口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。**如下图：  


对上图的解释如下：  
 （1）当TCP连接进行初始化时，把拥塞窗口cwnd置为1。前面已说过，为了便于理解，图中的窗口单位不使用字节而使用报文段的个数。慢开始门限的初始值设置为16个报文段，即 cwnd = 16 。  
（2）. 在执行慢开始算法时，拥塞窗口 cwnd 的初始值为1。以后发送方每收到一个对新报文段的确认ACK，就把拥塞窗口值另1，然后开始下一轮的传输（图中横坐标为传输轮次）。因此拥塞窗口cwnd 随着传输轮次按指数规律增长。当拥塞窗口cwnd增长到慢开始门限值ssthresh时（即当cwnd=16时），就改为执行拥塞避免算法，拥塞窗口按线 性规律增长。  
（3）. 假定拥塞窗口的数值增长到24时，网络出现超时（这很可能就是网络发生拥塞了）。更新后的ssthresh值变为12（即变为出现超时时的拥塞窗口数值 24的一半），拥塞窗口再重新设置为1，并执行慢开始算法。当cwnd=ssthresh=12时改为执行拥塞避免算法，拥塞窗口按线性规律增长，每经过 一个往返时间增加一个MSS的大小。

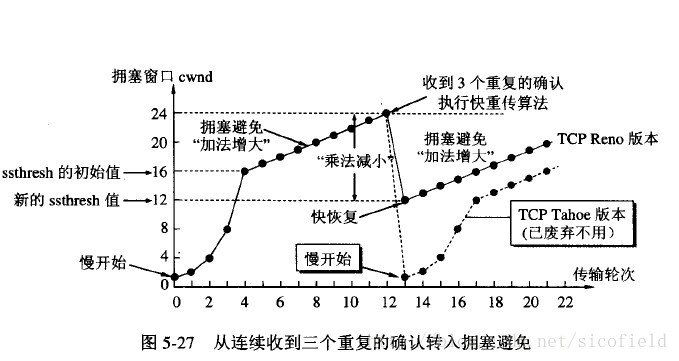
### 2.2、**快重传和快恢复**

**快重传要求接收方收到一个失序的报文段之后就立即发出重复确认，而不要等到自己发送的数据时捎带确认。快重传 算法规定，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**，而不必继续等待设置的重传计时器时间到期（快体现在这里）。如下图：



**而快速重传以后，因为走的不是慢启动而是拥塞避免算法，所以这又叫做快速恢复算法。**

主要有有以下两个要点:  
①当发送方连续收到三个重复确认时，就执行“乘法减小”算法，把ssthresh门限减半。但是接下去并不执行慢开始算法。  
②考虑到如果网络出现拥塞的话就不会收到好几个重复的确认，所以发送方现在认为网络可能没有出现拥塞。所以此时不执行慢开始算法，而是将cwnd设置为ssthresh的大小，然后执行拥塞避免算法。如下图：



**TCP/IP协议的工作流程如下：**

●在源主机上，应用层将一串应用数据流传送给传输层。

●**传输层将应用层的数据流截成分组，并加上TCP报头形成TCP段，送交网络层。**

●**在网络层给TCP段加上包括源、目的主机IP地址的IP报头，生成一个IP数据包**，并将IP数据包送交链路层。

●**链路层在其MAC帧的数据部分装上IP数据包，再加上源、目的主机的MAC地址和帧头**，并根据其目的MAC地址，将MAC帧发往目的主机或IP路由器。

●在目的主机，链路层将MAC帧的帧头去掉，并将IP数据包送交网络层。

●**网络层检查IP报头，如果报头中校验和与计算结果不一致**，则丢弃该IP数据包；若校验和与计算结果一致，则去掉IP报头，将TCP段送交传输层。

●**传输层检查顺序号，判断是否是正确的TCP分组，然后检查TCP报头数据**。若正确，则向源主机发确认信息；若不正确或丢包，则向源主机要求重发信息。

●在**目的主机，传输层去掉TCP报头，将排好顺序的分组组成应用数据流送给应用程序**。这样目的主机接收到的来自源主机的字节流，就像是直接接收来自源主机的字节流一样。

1、OSI的中文全称是开放式系统互联（ 国际标准化组织），

它们分别是（ 应用层(Application layer) 表示层(Presentation layer) 会话层(Session layer) 传输层(Transport layer)

网络层(Network layer) 数据链路层(Data link layer) 物理层(Physical layer) ）。

2、**集线器hub工作在OSI参考模型的（物理）层**；**网卡工作在OSI参考模型的（物理）层**；

**网桥工作在数据链路层，路由器router工作在OSI参考模型的（网络）层**；**交换机Switch工作在OSI参考模型的（数据链路）层。**

3、机器A的IP地址为202.96.128.130，子网掩码为255.255.255.128，则该IP地址的网络号是（202.96.128），

主机号是（130 ）。

子网掩码的作用就是用来判断任意两个IP地址是否属于同一子网络，这时只有在同一子网的计算机才能"直接"互通。

IP地址分网络号和主机号，要将一个网络划分为多个子网，因此网络号将要占用原来的主机位，如对于一个C类地址，它用21位来标识网络号，要将其划分为2个子网则需要占用1位原来的主机标识位。此时网络号位变为22位为主机标示变为7位。同理借用2个主机位则可以将一个C类网络划分为4个子网……那计算机是怎样才知道这一网络是否划分了子网呢？这就可以从子网掩码中看出。子网掩码和IP地址一样有32bit，确定子网掩码的方法是其与IP地址中标识网络号的所有对应位都用"1"，而与主机号对应的位都是"0"。如分为2个子网的C类IP地址用22位来标识网络号，则其子网掩码为：11111111 11111111 11111111 10000000即255.255.255.128。于是我们可以知道，A类地址的缺省子网掩码为255.0.0.0,B类为255.255.0.0,C类为255.255.255.0。下表是C类地址子网划分及相关子网掩码：

子网位数 子网掩码 主机数 可用主机数

1 255.255.255.128 128 126

2 255.255.255.192 64 62

3 255.255.255.224 32 30

4 255.255.255.240 16 14

5 255.255.255.248 8 6

6 255.255.255.252 4 2

你可能注意到上表分了主机数和可用主机数两项，这是为什么呢？因为但**当地址的所有主机位都为"0"时，这一地址为线路（或子网）地址**，而**当所有主机位都为"1"时为广播地址。**

同时我们还可以使用可变长掩码（VLSM）就是指一个网络可以用不同的掩码进行配置。这样做的目的是为了使把一个网络划分成多个子网更加方便。在没有VLSM的情况下，一个网络只能使用一种子网掩码，这就限制了在给定的子网数目条件下主机的数目。例如你被分配了一个C类地址，网络号为192.168.10.0,而你现在需要将其划分为三个子网,其中一个子网有100台主机,其余的两个子网有50台主机。我们知道一个C类地址有254个可用地址，那么你如何选择子网掩码呢？从上表中我们发现，当我们在所有子网中都使用一个子网掩码时这一问题是无法解决的。此时VLSM就派上了用场，我们可以在100个主机的子网使用255.255.255.128这一掩码，它可以使用192.168.10.0到192.168.10.127这128个IP地址，其中可用主机号为126个。我们再把剩下的192.168.10.128到192.168.10.255这128个IP地址分成两个子网，子网掩码为255.255.255.192。其中一个子网的地址从192.168.10.128到192.168.10.191,另一子网的地址从192.168.10.192到192.168.10.255。子网掩码为255.255.255.192每个子网的可用主机地址都为62个，这样就达到了要求。可以看出合**理使用子网掩码，可以使IP地址更加便于管理和控制.**

4、**ARP的中文意思是（地址解析协议）**，请用简单语言说明其的工作原理。

　 1. 首先，每台主机都会在自己的**ARP缓冲区 (ARP Cache)中建立一个 ARP列表**，以表示IP地址和MAC地址的对应关系。

　　2. 当源主机需要将一个数据包要发送到目的主机时，会首先检查自己 ARP列表中是否存在该 IP地址对应的MAC地址，

如果有﹐就直接将数据包发送到这个MAC地址；如果没有，**就向本地网段发起一个ARP请求的广播包**，查询此目的主机对应的MAC地址。此ARP请求数据包里包括源主机的IP地址、硬件地址、以及目的主机的IP地址。

　　3. 网络中所有的主机收到这个ARP请求后，会检查数据包中的目的IP是否和自己的IP地址一致。如果不相同就忽略此数据包；如果相同，该主机首先将发送端的MAC地址和IP地址添加到自己的ARP列表中，如果ARP表中已经存在该IP的信息，则将其覆盖，然后给源主机发送一个 ARP响应数据包，告诉对方自己是它需要查找的MAC地址；

　　4. 源主机收到这个ARP响应数据包后，将得到的目的主机的IP地址和MAC地址添加到自己的ARP列表中，并利用此信息开始数据的传输。如果源主机一直没有收到ARP响应数据包，表示ARP查询失败。

5、**DNS是指（域名系统Domain Name System）**。请用简单语言描述其工作原理。

当 DNS 客户机需要查询程序中使用的名称时，它会查询 DNS 服务器来解析该名称。

客户机发送的每条查询消息都包括3条信息，以指定服务器应回答的问题。

　　1 指定的 DNS 域名，表示为完全合格的域名 (FQDN) 。

　　2 指定的查询类型，它可根据类型指定资源记录，或作为查询操作的专门类型。

　　3 DNS域名的指定类别。

6、TCP和UDP的区别

**TCP提供的是面向连接的、可靠的数据流传输，而UDP提供的是非面向连接的、不可靠的数据报传输。通过校验和，确认和重传实现可靠传输，通过序号，标志位实现有序到达，通过滑动窗口机制实现流量控制，通过动态改变拥塞窗口大小实现拥塞控制。**

**简单的说，TCP注重数据安全，而UDP数据传输快点，但安全性一般，适合数据高速传输，还有广播以及多播场景下的传输。**

7、**网关的作用。**

**通过它可以访问外网**

二，网络命令

1、ipconfig的作用是什么？

**显示当前的TCP/IP配置的设置值**

2、运行net share　返回的结果是什么？

列出共享资源相关信息 如 IPC$

3、net use 和net user分别是指什么？

net user 用于用户管理，添加，删除网络使用用户。

net use 用于网络设备管理，例如添加磁盘

4、如何在命令行下面查看当前系统开放的服务？

在命令行下执行net services 命令

5、除以上命令，还有哪些，请写出你知道的命令。

taskill

taslist

net view显示计算机列表

netstat

ftp

telnet

三，系统端口及服务

　1、关掉以下服务，会出现什么样的情况，并请说明你的看法。

Automatic Updates

不能自动更新

Plug and Play

禁用会导致USB不能使用.

Remote Registry Service

防范通过浏览网页来修改你的注册表

Computer Browser

无法通过该服务维护网络上计算机的最新列表以及提供这个列表给请求的程序。

2．端口及相对的服务

FTP（21 文件传输FTP服务　　　）

Terminal Services　的端口是（　3389　　　）

23端口是（TELNET）开放的默认端口

25端口是（E-mail SMTP）开放

109端口是（　　POP2　）开放

1433端口是（　　SQL Server　　　）开放

四，网络协议

ICMP：

是Internet Control Message Protocol（**Internet控制消息协议**）的缩写。

它是TCP/IP协议族的一个子协议，用于在IP主机、路由器之间传递控制消息。

**控制消息是指网络通不通、主机是否可达、路由是否可用等网络本身的消息。**

**这些控制消息虽然并不传输用户数据，但是对于用户数据的传递起着重要的作用。**

TFTP：

Trivial File Transfer Protocol,是TCP/IP协议族中的一个用来在客户机与服务器之间进行**简单文件传输的协议**

提供不复杂、开销不大的文件传输服务。

SMTP:简单邮件传输协议，发送邮件

Pop3：对应于SMTP，接收邮件

HTTP：

**HTTP超文本传输协议，是一个属于应用层的面向对象的协议**，**由于其简捷、快速的方式，适用于分布式超媒体信息系统，**

它于1990年提出，经过几年的使用与发展，得到不断地完善和扩展。

DHCP：动态主机配置协议， 是一种让系统得以连接到网络上，并获取所需要的配置参数手段

　　第二套：

网络知识考查

日期：2008/3/3

姓名：Amxking

一，填空题。

1，浏览器与WWW服务器之间传输信息时使用的协议是（http ）。

2，在星型局域网结构中，连接文件服务器与工作站的设备是（交换机 ）。

3，在安装完成Linux系统后，系统自动创建的管理员帐号是（ root ）。

4，单位分得合法IP地址202.112.68.40 掩码为255.255.255.248，其中，路由器的外口和ISP之间占据了2个，

若使用202.112.68.41和202.112.68.42，掩码为255.255.255.252

问：1，则可供使用的合法IP还有多少哪些？

还可用的有 202.112.68.44/29, 202.112.68.45/29 ,202.112.68.46/29

问：2，使用内部IP进行地址转换，若用一台主机连接内外两个网络，请说出2中不同的网络接法；并进行比较？

1）主机接一块网卡绑定两个不同子网的地址，运行代理软件，内部网络将网关设置指向该主机。

2）主机插2块网卡，分别连接内外网，主机起到网关和地址转换作用。

1）中方案若内网盗用主机合法IP可以绕过主机

2）不能绕过主机。

问：3，Internet上保留了哪些内部IP有可以供使用？

　可以使用10.0.0.0 或 172.16直172.31 或192.168.0 直192.168.255

5，如何规划防火墙，将内部业务服务器和部分PC机与Internet隔离？

可以构建一个非军事区，将内部业务服务器通过内网路由器对内提供服务。

部分PC通过外网路由连接INTERNET，Internet上限制内部的部分pc机访问Internet。

在非军事区和内网之间设置路由器或代理服务器作为防火墙，限制外部的访问。

6，在我国，目前可供选择大的用户选择的接入方式有哪些，各自的接入速率为多少？

DDN 最高2M

ISDN 64K\*2 （2B＋D）

帧中继 最高2M

X.25 64K

[[[DDH　　（　最高2M 　）

ADSL　　（　非对称数字用户线，下行速率从512Kbit/s到8Mbit/s，而上行速率则从64Kbit/s到640Kbit/s　　　）

ISDN　　　（综合业务数字网最高速度可达到64Kbps或128Kbps　　）]]]

7，被路由器隔离的2个子网能否公用一台DHCP服务器？（　不能 　）

8，用户通过什么命令可以看到自己申请到的本机IP地址？用何命令可以重新向DHCP服务器申请IP？用何命令可以释放IP？

ipconfig /all

ipconfig /release释放ip

ipconfig /renew 获取新ip

8，ADSL使用的多路复用技术是（a　）

A.频分多路复用 B.时分多路复用

C.码分多址 D.空分多址

二，问答题。

1,写出下面网络命令的作用。

Netstat　–p　　　（只打印给出名字的协议的统计数字和协议控制块信息 ）

Net view (显示当前域或网络上的计算机上的列表)

如何在命令行下显示windows的服务　（ net services 　　）

2, **网桥的作用。**

**是一个局域网与另一个局域网之间建立连接的桥梁**

3,用一条命令实现：将远程主机C盘映射为自己的F盘

　net use f: \\远程主机IP\c$ "密码" /user:"用户名"

5，防火墙的端口防护是指？

指通过对防火墙的端口开关的设置，关闭一些非必需端口，达到一定安全防护目的的行为。

1、建立连接协议（三次握手）

（1）客户端发送一个带SYN标志的TCP报文到服务器。这是三次握手过程中的报文1。

（2） 服务器端回应客户端的，这是三次握手中的第2个报文，这个报文同时带ACK标志和SYN标志。因此它表示对刚才客户端SYN报文的回应；同时又标志SYN给客户端，询问客户端是否准备好进行数据通讯。

（3） 客户必须再次回应服务段一个ACK报文，这是报文段3。

2、连接终止协议（四次挥手）

　 　由于TCP连接是全双工的，因此每个方向都必须单独进行关闭。这原则是当一方完成它的数据发送任务后就能发送一个FIN来终止这个方向的连接。收到一个 FIN只意味着这一方向上没有数据流动，一个TCP连接在收到一个FIN后仍能发送数据。首先进行关闭的一方将执行主动关闭，而另一方执行被动关闭。

　（1） TCP客户端发送一个FIN，用来关闭客户到服务器的数据传送（报文段4）。

　（2） 服务器收到这个FIN，它发回一个ACK，确认序号为收到的序号加1（报文段5）。和SYN一样，一个FIN将占用一个序号。

　（3） 服务器关闭客户端的连接，发送一个FIN给客户端（报文段6）。

　（4） 客户段发回ACK报文确认，并将确认序号设置为收到序号加1（报文段7）。

CLOSED: 这个没什么好说的了，表示初始状态。

LISTEN: 这个也是非常容易理解的一个状态，表示服务器端的某个SOCKET处于监听状态，可以接受连接了。

SYN\_RCVD: 这个状态表示接受到了SYN报文，在正常情况下，这个状态是服务器端的SOCKET在建立TCP连接时的三次握手会话过程中的一个中间状态，很短暂，基本上用netstat你是很难看到这种状态的，除非你特意写了一个客户端测试程序，故意将三次TCP握手过程中最后一个ACK报文不予发送。因此这种状态时，当收到客户端的ACK报文后，它会进入到ESTABLISHED状态。

SYN\_SENT: 这个状态与SYN\_RCVD遥想呼应，当客户端SOCKET执行CONNECT连接时，它首先发送SYN报文，因此也随即它会进入到了SYN\_SENT状态，并等待服务端的发送三次握手中的第2个报文。SYN\_SENT状态表示客户端已发送SYN报文。

ESTABLISHED：这个容易理解了，表示连接已经建立了。

FIN\_WAIT\_1: 这个状态要好好解释一下，其实FIN\_WAIT\_1和FIN\_WAIT\_2状态的真正含义都是表示等待对方的FIN报文。而这两种状态的区别是：FIN\_WAIT\_1状态实际上是当SOCKET在ESTABLISHED状态时，它想主动关闭连接，向对方发送了FIN报文，此时该SOCKET即进入到FIN\_WAIT\_1状态。而当对方回应ACK报文后，则进入到FIN\_WAIT\_2状态，当然在实际的正常情况下，无论对方何种情况下，都应该马上回应ACK报文，所以FIN\_WAIT\_1状态一般是比较难见到的，而FIN\_WAIT\_2状态还有时常常可以用netstat看到。

FIN\_WAIT\_2：上面已经详细解释了这种状态，实际上FIN\_WAIT\_2状态下的SOCKET，表示半连接，也即有一方要求close连接，但另外还告诉对方，我暂时还有点数据需要传送给你，稍后再关闭连接。

TIME\_WAIT: 表示收到了对方的FIN报文，并发送出了ACK报文，就等2MSL后即可回到CLOSED可用状态了。如果FIN\_WAIT\_1状态下，收到了对方同时带FIN标志和ACK标志的报文时，可以直接进入到TIME\_WAIT状态，而无须经过FIN\_WAIT\_2状态。

CLOSING: 这种状态比较特殊，实际情况中应该是很少见，属于一种比较罕见的例外状态。正常情况下，当你发送FIN报文后，按理来说是应该先收到（或同时收到）对方的ACK报文，再收到对方的FIN报文。但是CLOSING状态表示你发送FIN报文后，并没有收到对方的ACK报文，反而却也收到了对方的FIN报文。什么情况下会出现此种情况呢？其实细想一下，也不难得出结论：那就是如果双方几乎在同时close一个SOCKET的话，那么就出现了双方同时发送FIN报文的情况，也即会出现CLOSING状态，表示双方都正在关闭SOCKET连接。

CLOSE\_WAIT: 这种状态的含义其实是表示在等待关闭。怎么理解呢？当对方close一个SOCKET后发送FIN报文给自己，你系统毫无疑问地会回应一个ACK报文给对方，此时则进入到CLOSE\_WAIT状态。接下来呢，实际上你真正需要考虑的事情是察看你是否还有数据发送给对方，如果没有的话，那么你也就可以close这个SOCKET，发送FIN报文给对方，也即关闭连接。所以你在CLOSE\_WAIT状态下，需要完成的事情是等待你去关闭连接。

LAST\_ACK: 这个状态还是比较容易好理解的，它是被动关闭一方在发送FIN报文后，最后等待对方的ACK报文。当收到ACK报文后，也即可以进入到CLOSED可用状态了。

最后有2个问题的回答，我自己分析后的结论（不一定保证100%正确）

1、 为什么建立连接协议是三次握手，而关闭连接却是四次握手呢？

这是因为服务端的LISTEN状态下的SOCKET当收到SYN报文的建连请求后，它可以把ACK和SYN（ACK起应答作用，而SYN起同步作用）放在一个报文里来发送。但关闭连接时，当收到对方的FIN报文通知时，它仅仅表示对方没有数据发送给你了；但未必你所有的数据都全部发送给对方了，所以你可以未必会马上会关闭SOCKET,也即你可能还需要发送一些数据给对方之后，再发送FIN报文给对方来表示你同意现在可以关闭连接了，所以它这里的ACK报文和FIN报文多数情况下都是分开发送的。

2、 为什么TIME\_WAIT状态还需要等2MSL后才能返回到CLOSED状态？

这是因为：虽然双方都同意关闭连接了，而且握手的4个报文也都协调和发送完毕，按理可以直接回到CLOSED状态（就好比从SYN\_SEND状态到ESTABLISH状态那样）；但是因为我们必须要假想网络是不可靠的，你无法保证你最后发送的ACK报文会一定被对方收到，因此对方处于LAST\_ACK状态下的SOCKET可能会因为超时未收到ACK报文，而重发FIN报文，所以这个TIME\_WAIT状态的作用就是用来重发可能丢失的ACK报文。

**4：进程间通讯的方式有哪些，各有什么优缺点**

进程间通信

Linux 进程间通信（IPC）以下以几部分发展而来：

早期UNIX进程间通信、基于System V进程间通信、基于Socket进程间通信和POSIX进程间通信。

UNIX进程间通信方式包括：管道、FIFO、信号。

System V进程间通信方式包括：System V消息队列、System V信号灯、System V共享内存、

POSIX进程间通信包括：posix消息队列、posix信号灯、posix共享内存。

现在linux使用的进程间通信方式：

（1）管道（pipe）和有名管道（FIFO）

（2）信号（signal）

（3）消息队列

（4）共享内存

（5）信号量

（6）套接字（socket)

**4.TTL是什么？有什么用处，通常那些工具会用到它？（ping? traceroute? ifconfig? netstat?）**

答:

简:TTL是Time To Live，目前是hup count，当包每经过一个路由器它就会被减去一，如果它变成0，路由器就会把包丢掉。IP网络往往带有环(loop)，比如子网A和子网B有两个路由器相连，它就是一个loop。**TTL的主要目的是防止包在有回路的网络上死转，因为包的TTL最终后变成0而使得此包从网上消失(此时往往路由器会送一个ICMP包回来，traceroute就是根据这个做的**)。ping会送包出去，所以里面有它，但是ping不一定非要不可它。**traceroute则是完全因为有它才能成的**。ifconfig是用来配置网卡的，netstat -rn 是用来列路由表的，所以都用不着它。

**5.路由表示做什么用的？在linux环境中怎么来配置一条默认路由？**

详:**路由表是用来决定如何将包从一个子网传送到另一个子网的，换局话说就是用来决定从一个网卡接收到的包应该送的哪一张网卡上的。**路由表的每一行至少有目标网络号、netmask、到这个子网应该使用的网卡。当路由器从一个网卡接收到一个包时，它扫描路由表的每一行，用里面的netmask和包里的目标IP地址做并逻辑运算(&)找出目标网络号，如果此网络号和这一行里的网络号相同就将这条路由保留下来做为备用路由，如果已经有备用路由了就在这两条路由里将网络号最长的留下来，另一条丢掉，如此接着扫描下一行直到结束。如果扫描结束任没有找到任何路由，就用默认路由。确定路由后，直接将包送到对应的网卡上去。在具体的实现中，路由表可能包含更多的信息为选路由算法的细节所用。题外话：路由算法其实效率很差，而且不scalable，解决办法是使用IP交换机，比如MPLS。

在Linux上可以用“route add default gw <默认路由器IP>”来配置一条默认路由。

**6.在网络中有两台主机A和B，并通过路由器和其他交换设备连接起来，已经确认物理连接正确无误，怎么来测试这两台机器是否连通？如果不通，怎么来判断故障点？怎么排除故障？**

答:测试这两台机器是否连通：从一台机器ping另一台机器

**如果ping不通，用traceroute可以确定是哪个路由器不能连通**，然后再找问题是在交换设备/hup/cable等。

**3. 网络编程的一般步骤**

**对于TCP连接**：

1.服务器端1）创建套接字create；2）绑定端口号bind；3）监听连接listen；4）接受连接请求accept，并返回新的套接字；5）用新返回的套接字recv/send；6）关闭套接字。

2.客户端1）创建套接字create; 2）发起建立连接请求connect; 3）发送/接收数据send/recv；4）关闭套接字。

TCP总结：

Server端：create -- bind -- listen-- accept-- recv/send-- close

Client端：create------- conncet------send/recv------close.

**对于UDP连接：**

1.服务器端:1）创建套接字create；2）绑定端口号bind；3）接收/发送消息recvfrom/sendto；4）关闭套接字。

2.客户端:1）创建套接字create；2）发送/接收消息sendto/recvfrom；3）关闭套接字.

UDP总结:

Server端：create----bind ----recvfrom/sendto----close

Client端：create---- sendto/recvfrom----close.

**5. TCP的重发机制是怎么实现的？**

1.滑动窗口机制，确立收发的边界，能让发送方知道已经发送了多少（已确认）、尚未确认的字节数、尚待发送的字节数；让接收方知道（已经确认收到的字节数）。

2.选择重传，用于对传输出错的序列进行重传。

**6. TCP和UDP的区别？**

1）TCP面向连接（三次握手机制），通信前需要先建立连接；UDP面向无连接，通信前不需要建立连接；

2）TCP保障可靠传输（按序、无差错、不丢失、不重复）；UDP不保障可靠传输，使用最大努力交付；

3）TCP面向字节流的传输，UDP面向数据报的传输。

**TCP---传输控制协议,提供的是面向连接、可靠的字节流服务。**当客户和服务器彼此交换数据前，必须先在双方之间建立一个TCP连接，之后才能传输数据。TCP提供超时重发，丢弃重复数据，检验数据，流量控制等功能，保证数据能从一端传到另一端。

**UDP---用户数据报协议，是一个简单的面向数据报的运输层协议。UDP不提供可靠性**，它只是把应用程序传给IP层的数据报发送出去，但是并不能保证它们能到达目的地。由于UDP在传输数据报前不用在客户和服务器之间建立一个连接，且没有超时重发等机制，故而传输速度很快

**8.TCP为什么不是两次连接？而是三次握手？**

如果A与B两个进程通信，如果仅是两次连接。可能出现的一种情况就是：A发送完请报文以后，由于网络情况不好，出现了网络拥塞，即B延时很长时间后收到报文，即此时A将此报文认定为失效的报文。B收到报文后，会向A发起连接。此时两次握手完毕，B会认为已经建立了连接可以通信，B会一直等到A发送的连接请求，而A对失效的报文回复自然不会处理。依次会陷入B忙等的僵局，造成资源的浪费。

**9. connect方法会阻塞，请问有什么方法可以避免其长时间阻塞？**

可以考虑采用**异步传输机制**，同步传输与异步传输的主要区别在于同步传输中，如果调用recvfrom后会一致阻塞运行，从而导致调用线程暂停运行；异步传输机制则不然，会立即返回。

**管道**包括三种:1)**普通管道PIPE**, 通常有种限制,一是半双工,只能单向传输;二是只能在父子进程间使用. 2)**流管道s\_pipe**: 去除了第一种限制,可以双向传输. 3)**命名管道:name\_pipe**, 去除了第二种限制,可以在许多并不相关的进程之间进行通讯.

**系统IPC**的三种方式类同,都是使用了内核里的标识符来识别

**管道**: 优点是所有的UNIX实现都支持, 并且在最后一个访问管道的进程终止后,管道就被完全删除;缺陷是管道只允许单向传输或者用于父子进程之间

**系统IPC**: 优点是功能强大,能在毫不相关进程之间进行通讯; 缺陷是关键字KEY\_T使用了内核标识,占用了内核资源,而且只能被显式删除,而且不能使用SOCKET的一些机制,例如select,epoll等.

socket可以跨网络通讯，其他进程间通讯的方式都不可以，只能是本机进程通讯。

**4.sendMessage与postMessage区别？**

不同点：**sendMessage发送完毕以后需要等待处理完才返回**；而**postMessage发送消息后立即返回。**

Do not post the WM\_QUIT message using PostMessage; use thePostQuitMessage function.

**postMessage将消息放置到消息队列中，不等待线程处理消息就立即返回。**

**sendMessage发送指定的消息到窗口，并会调用窗口过程，直到窗口过程处理完毕后才返回。**

TCP/IP是应用在INTERNET上的非国际标准体系结构（国际标准：OSI），也就是说我平常的上网必须遵循这种TCP/IP协议集才可以上网。

    TCP/IP是采用分层体系结构，它与开放系统互连OSI模型的层次结构相似，它可分为四层，由低到高依次为：数据链路层，网络层（既IP层），传输层（既TCP层）和应用层。

1.链路层

在TCP/IP参考模型中，最低层名称很多，一般有链路层，网络访问层，主机－主机层等。**该层的主要功能是连接上一层的IP数据报，通过网络向外发送，或者接收和处理来自网络上的物理帧，并抽取IP数据传送到上一层**----网络层。

ARP协议（正向地址解析协议）：将网络层地址转换为链路层地址。

RARP协议（逆向地址解析协议）：将链路层地址转换为网络层地址。

2.网络层(IP层）

**网路层主要解决计算机之间的通信问题，它负责管理不同设备之间的数据交换**，它是INTERNET通信子网的最高层，它所提供的是不可靠的无连接数据报机制（无连接服务的含义：发送端简单地把信息包发送到网络上，在传送信息包之前发送端和接收端没有沟通的过程，也没有对方来确认，因而不知道目的地是否接收到。无连接服务和面向连接服务相对），无论传输是否正确，不做验证，不发确认，也不保证分组的正确顺序。

IP层主要有以下协议：

IP协议（网络协议）：使用IP地址确定收发端，提供端到端的“数据报”传递，也是TCP/IP协议簇中处于核心地位的一个协议。

ICMP协议(网络控制报文协议）：处理路由，协助IP层实现报文传送的控制机制，提供错误和信息报告。

3.传输层协议（TCP协议）

传输层主要是确保所有传送到某个系统数据正确无误地到达该系统，提供端到端的可靠性传输，该层主要协议有:

TCP协议：传输控制协议，提供可靠的面向连接的数据传输服务。

UDP协议：用户数据报协议，采用无连接数据报传送方式，一次传输少量信息的情况，如数据查询等，当通信子网相当可靠时，UDP协议的优越性尤为可靠。

4.应用层

应用层时将应用程序的数据传送给传输层，以便进行信息交换。它主要为各种应用程序提供了使用的协议，标准的应用层协议主要有：

**FTP文件传输协议**：为文件传输提供了途径，它允许数据从一台主机传送到另一台主机上（咱们用的QQ传送文件就是用到这个协议），也可以从FTP服务器上下载文件，或者向FTP服务器上传文件。

**HTTP超文本传输协议**：用来访问在WWW服务器上的各种页面。

**DNS域名服务系统**：用于实现从主机域名到IP地址之间的转换。

**TELNET虚拟终端服务**：实现互联网中的工作站登陆到远程服务器的能力。

**SMTP简单邮件传输协议**：实现互联网中电子邮件的传输功能。

**NFS网络文件系统**：用于实现网络中不同主机之间的文件共享。

RIP路由信息协议：用于网络设备之间交换路由信息。

TCP/IP 是INTERNET事实上的标准，并且在最近几年已经成为专门网络所选择的协议，在最新的网络操作系统中，TCP/IP已经被成为默认协议。

**（1）OSI七层模型**

OSI中的层 功能 TCP/IP协议族  
应用层 文件传输，电子邮件，文件服务，虚拟终端 TFTP，HTTP，SNMP，FTP，SMTP，DNS，Telnet  
表示层 数据格式化，代码转换，数据加密 没有协议  
会话层 解除或建立与别的接点的联系 没有协议  
传输层 提供端对端的接口 TCP，UDP  
网络层 为数据包选择路由 IP，ICMP，RIP，OSPF，BGP，IGMP  
数据链路层 传输有地址的帧以及错误检测功能 SLIP，CSLIP，PPP，ARP，RARP，MTU  
物理层 以二进制数据形式在物理媒体上传输数据 ISO2110，IEEE802，IEEE802.2



三 Time\_Wait状态解释

根据TCP协议，主动发起关闭的一方，会进入TIME\_WAIT状态(TCP实现必须可靠地终止连接的两个方向(全双工关闭))，持续2\*MSL(Max Segment Lifetime)，缺省为240秒.

TIME\_WAIT的等待时间为2MSL，即最大段生存时间.如果 TIME\_WAIT 状态保持时间不足够长(比如小于2MSL)，第一个连接就正常终止了。第二个拥有相同相关五元组的连接出现(因为连接终止前发起的一方可能需要重发 ACK，所以停留在该状态的时间必须为MSL的2倍。)，而第一个连接的重复报文到达，干扰了第二个连接。TCP实现必须防止某个连接的重复报文在连接终 止后出现，所以让TIME\_WAIT态保持时间足够长(2MSL)，连接相应方向上的TCP报文要么完全响应完毕，要么被丢弃。建立第二个连接的时候，不 会混淆。

为什么TIME\_WAIT 状态 停留2MSL（max segment lifetime）时间

也就是TCP/IP设计者本来是这么设计的

主要有两个原因

1。防止上一次连接中的包，迷路后重新出现，影响新连接

（经过2MSL，上一次连接中所有的重复包都会消失）

2。可靠的关闭TCP连接

在主动关闭方发送的最后一个 ack(fin) ，有可能丢失，这时被动方会重新发

fin, 如果这时主动方处于 CLOSED 状态 ，就会响应 rst 而不是 ack。所以

主动方要处于 TIME\_WAIT 状态，而不能是 CLOSED 。

四   Close\_Wait状态解释

CLOSE\_WAIT状态的生成原因

通过TCP的状态图我们可以看出只有被动关闭的一端才有CLOSE\_WAIT状态，当收到Fin并发送了Ack后

服务器状态就变成了CLOSE\_WAIT状态，如果我们的服务器一直处于CLOSE\_WAIT状态的话，说明套接字是被动关闭的！，并且没有发送Fin信令，原因往往是没有调用TCP的CloseSocket。

解决CLOSE\_WAIT的方法:

1 一般原因都是TCP连接没有调用关闭方法。需要应用来处理网络链接关闭。

2 对于Web请求出现这个原因，经常是因为Response的BodyStream没有调用Close.

比如Widnows下:

使用HttpWebRequest 一定要保证GetRequestStream和GetResponse对象关闭，否则容易造成连接处于CLOSE\_WAIT状态

3 TCP的KeepLive功能，可以让操作系统替我们自动清理掉CLOSE\_WAIT的连接。

但是KeepLive在Windows操作系统下默认是7200秒，也就是2个小时才清理一次。往往满足不了要求。可以调小该数值。

Windows下的调整方法为

HKEY\_LOCAL\_MACHINE/CurrentControlSet/Services/Tcpip/Parameters下的以下三个参数：

KeepAliveInterval,设置其值为1000   www.2cto.com

KeepAliveTime,设置其值为300000(单位为毫秒，300000代表5分钟)

TcpMaxDataRetransmissions,设置其值为5

Close\_Wait引发的问题:

Close\_Wait会占用一个连接，网络可用连接小。数量过多，可能会引起网络性能下降，并占用系统非换页内存。 尤其是在有连接池的情况下(比如HttpRequest)

会耗尽连接池的网络连接数，导致无法建立网络连接

修改2MSL的值通常是机器全局的配置修改。你可以在socket级别使用SO\_REUSEADDR socket选项解决TIME\_WAIT的问题，这使得即使一个有着同样地址和端口的socket存在，也可以创建一个新的socket，新的socket最终将会劫持旧的socket。你可以使用SO\_REUSEADDR选项，在一个有着同样端口的socket已经处于TIME\_WAIT状态的时候创建新的socket，但这样做可能会造成一些问题，比如拒绝服务攻击或者数据窃取。在Windows平台下，有另一个socket选项SO\_EXCLUSIVEADDRUSE，使用它可以避免SO\_REUSEADDR选项的[缺陷](http://msdn.microsoft.com/en-us/library/ms740621(v=vs.85" \t "https://segmentfault.com/a/_blank)

还有另外一种方式断开TCP连接。通过终止(abort)连接并发送一个RST代替FIN，这可以通过设置socket的SO\_LINGER选项为 0 来实现。这样会使未处理的数据直接被丢弃并且连接被RST中断，而不是使用FIN的时候那样，未处理的数据继续完成传输。认识到当连接被RST中断的时候，任何在终端之间未处理的数据都将会被直接丢弃是非常重要的，通常这个RST代表了一个错误消息"connection has been reset by the peer"。远程终端知道连接是被中断还是进入了TIME\_WAIT状态。

大多数服务器端一般执行被动关闭，服务器不会进入TIME\_WAIT状态。

服务端为了解决这个TIME\_WAIT问题，可选择的方式有三种：

    Ø  保证由客户端主动发起关闭（即做为B端）

    Ø  关闭的时候使用RST的方式

    Ø  对处于TIME\_WAIT状态的TCP允许重用