**ICMP**

**【internet control message protocol】用于在路由和主机之间处理错误和控制信息。这些信息由TCP/IP程序自己生成并处理，而不是用户来处理。**

**IGMP**

**【internet group management protocol】用于多播，是IPV4的可选组件。**

**ARP**

**【address resolution protocol】ARP映射一个IPV4地址到一个硬件地址上。通常用于广播网络，比如Ethernet,token ring和FDDI，但是在point-to-point网络中不需要。**

**RARP**

**【reverse address resolution protocol】RARP映射一个硬件地址到一个IPV4的地址上。通常用于无盘节点网络(diskless node)，比如X终端。**

**UDP**

**它是一个简单的传输层协议。描述与RFC【678】。UDP作为一个应用层序，它将一个datagram写道一个UDP套接字，该datagram是封装好的IPV4或IPV6的datagram，然后在送到其目的地。但没有任何保证来使得UDP datagram最终到达其目的地。**

**网络编程中的UDP缺乏可靠性。如果我们想确认datagram到达目的地，我们必须在程序中添加大量的特性：从目的地确认其达到，延时，重传等等。**

**每个UDP datagram有一个长度，我们可以将其看作一个record。如果datagram正确到达其最终目的地(也即，包到达时没有校验码错误)，然后datagram的长度就传递给接收程序。我们已经提及过，TCP是一个字节流(type-stream)协议，没有任何记录边界(record boundary)，这点跟UDP不一样。**

**UDP提供一种无连接服务(connectionless service)，它并不需要在UDP服务器端和客户端长期保持连接关系。例如，一个UDP客户端可以创建一个套接字并发送一个数据报到一个给定的服务器，接着在同一个套接字中再发送另一个数据报给另一个不同的服务器。**

**TCP**

**描述于RFC【793】。TCP提供个客户端和服务器端连接。一个TCP客户端跟一个给定的服务器端建立一个连接，并在该连接上交换数据，接着终端该连接。**

**TCP还提供了可靠性。当TCP发送数据到其它目的端，它需要一个返回的确认。如果没有收到确认，TCP就自动重传数据并等待一个更长的时间。在多次重传后，TCP会放弃传输。总共的传输时间通常为4到10分钟(依赖于具体实现)。TCP有一个算法在客户端和服务器端动态建立RTT【round-trip time】，这样它就会知道一个确认要等待多长的时间。比如，一个LAN上的RTT可能为几毫秒，而在WAN上则为几秒钟。furthermore,TCP can measure an RTT of 1 seconds between a client and server and then 30 seconds later measure an RTT of 2 seconds on the same connection, caused by variations in the network traffic.【TCP可以在客户端和服务器端建立一个1秒钟的RTT，而在30秒后，于同一个连接上又建立一个2秒钟的RTT，这根据网络拥塞情况而定】。**

**TCP同时还序列化(sequences)数据，通过关联一个序号到每个所发送的字节上。例如，假定一个程序写入2048个字节到一个TCP套接字上，使得TCP分2部分发送，第一部分包括序号为1～1024的数据，第二部分包括1025～2048的数据(一个segment为TCP发送给IP的数据单元)。如果segment到达时是无序的，接收端的TCP在将数据传送给接收程序时，会基于序号重排这2段数据。如果TCP接收到了重复的数据，它将侦测该数据是重复的并将该重复数据丢弃。**

***UDP没有提供任何可靠性.没有确认，没有序号，不建立RTT，没有延迟或重传.如果一个UDP数据报在网络中重复，那么这2份拷贝都能传送给接收主机.同样，如果一个UDP客户端发送了2个数据报到同一个目的地，那么，网络会对这2个数据报排序，但是接收的时候可能就无序了.UDP程序必须处理所有这些情况.***

**TCP提供了流控制【flow control】.TCP告诉其对等端要接受多少个字节，这叫做advertised window(AW).在任何时候，AW为接收缓冲区中当前可用的空间，它保证发送端不能使得接收端的缓冲区溢出(overflow).AW会动态的调整大小：当从发送端接收数据时，AW会减少；当当程序从缓冲区中读取数据时，AW又会增长.AW是由可能为0值的：TCP套接字接收缓冲区已经已经满了，必须等待应用程序来读取缓冲区中的数据，而要接收对等端发送过来的数据，必须得在这之后.**

***UDP没有提供流控制.很容易使得一个快速的发送者以一定的频率来传送数据，这样，UDP接收者就不能保持同步.***

**最后，一个TCP连接是全双工的(full-duplex).这意味着，一个应用层序可以在任何时候、在给定的连接上、双向的接收和发送数据.也意味着TCP必须跟踪一些状态信息，比如序号和收发两端数据流的的AW大小.**

**TCP连接：connect, accept和close**

**【三次握手机制】**

1. **服务器必须准备接收一个过来的连接.通常通过调用socket, bind和listen来实现.这过程为一个passive open.**
2. **客户端通过connect调用来关联一个active open.这使得客户端TCP发送一个SYN段给服务器，意在告知服务器，客户端即将在该连接中发送的数据的初始序号.通常，这里并不会有数据跟SYN一起发送：SYN中只是包含一个IP头，一个TCP头以及可能的一些TCP选项.**
3. **服务器端必须确认客户端的SYN，而且也要发送自己的SYN，在该SYN中包含服务器要发送给客户端的数据的初始序号.服务器端将其SYN和客户端SYN的ACK放在同一个段中发送.**
4. **客户端必须确认服务器端的SYN.**

**这个过程中最小的数据包交换次数为3次，因此得名.**

**Client Server**

**socket|connect(阻塞)(active open)| socket|bind|listen|accept(阻塞)**

**connect返回 accept返回|read(阻塞)**

**【这个地方有用一个『打电话』的比喻来说明这个过程，但有点复杂.】**

**TCP选项**

**每个SYN可以包含一些TCP选项.常用的一些选项如下：**

* **MSS选项. 该选项使得TCP声明其maximum segment size.这样，在该连接中，将期望接收TCP每个段的最大量数据.在7.9中将看到如何通过设置套接字的TCP\_MAXSEG选项.**
* **Window scale选项. TCP所能设置的告知给其它TCP的最大的window数目为65535个，因为相应的TCP头的字段长度为16比特.但是高速连接(high-speed connection)(45Mb/s或更快，在RFC【1323】定义)需要更大的window数，并以此来获取最大可能的吞吐量(throughput).这个新的选项提供了一个最大的window数目，多达65535×2^14.如果TCP通信的两端有一个使用了该选项，那么另一个也必须使用该选项.在7.5中，通过SO\_RCVBUF来设置该选项.**
* **Timestamp选项. 在高速网络连接中需要该选项来阻止可能的数据错误，数据错误是由于丢而复得数据所致.对于网络编程，不需要考虑该选项.**

**TCP连接中断**

**中断一个TCP连接需要4个步骤.**

**一端的应用层序首先调用close，我们说这是active close.通过发送一个FIN段来中断TCP连接，这意味着他完成了数据传输.**

**另一端接收FIN，即passive close.接收的FIN由TCP来确认.接收到的FIN作为一个文件结束符传输到应用层序，因为FIN的接收意味着应用层序将不再从该连接上接收任何其它数据.**

**然后，应用层序接收到文件结束符，并调用close来关闭其套接字.这也会导致TCP发送一个FIN.**

**系统上的TCP接收到最后一个FIN.(The TCP on the system that receives this final FIN(the end that did the active close) acknowledge the FIN.**

**因为在两端都需要一个FIN和一个ACK，通常这需要4个数据段(segment).但在一些情况下，第一个步骤中的FIN跟数据一起发送，同理，步骤2和步骤3中的被动关闭(passive close)数据段也可以跟数据绑在一起发送.**

****

**一个FIN和一个SYN一样，在序列数中占一个字节的空间.所以每个FIN的ACK就是FIN的序列号加1.**

**什么是Half-close？——Section 6.6**

**每个FIN的发送都是在套接字关闭后.我们假定应用层序调用close来使得这一切发生，但是我们注意到当一个UNIX进程终止时(不管是自愿终止(调用exit或main返回)的还是被动终止(收到一个信号，终止了进程))，all open descriptors(这是什么？进程描述符？) are closed,这也会导致一个FIN发送到一个已经打开的TCP连接上.**

**客户端和服务器端都能进行主动关闭.通常是客户端进行主动关闭，但是一些协议(如HTTP)就是服务器端执行主动关闭.**

**TCP状态装换图**

**TCP的建立和终止可以用一个状态转换图来说明，如图：  
**

**在该图中有11个状态…**

**对于该图而言，如果一个连接的目的只是发送一个数据段并返回一个回复数据段，那么，如果用TCP来实现的话，就需要8个数据段的传递.如果用UDP的话，就只需要2个数据包的传递：请求包和应答包.换成UDP的话，会失去TCP所提供给应用层序的可靠性.TCP提供的另一个特性就是拥塞控制，这点必须由UDP程序自己来处理.但如果是传输量少的话，很多应用还是选择UDP，因为它只需要少量的数据交换，避免了TCP连接在建立和中断时的数据交换.**

**TIME\_WAIT状态**

**The duration that this endpoint remains in this state is twice the MSL(maximum segment lifetime), sometimes called 2MSL.**

**所有的TCP应用都必须为MSL选择一个值.在RFC【1122】中建议的值为2分钟，而在BSD的实现中，一直都为30秒，这样，TIME\_WAIT状态的持续期就为1～4分钟.MSL为任何给定IP报文在网络上的最长存活时间.**

**因特网上的包丢失往往都是源自路由异常.路由出问题后，往往需要几秒或几分钟来选择路由路径.在此期间，就可能形成路由环(routing loop)，即路由A发送一个包给路由B，而B又将该包丢回给A，这样一个包就可能在这样的环中.此时，假定丢失的包是一个TCP数据段(segment)，发送端的TCP超时并重传该处于环中的包，该包最终通过选择的路径到达了终点.但过一段时间后(up to MSL seconds after the lost packet started on its journey)，路由环错误被更正，处于环中的数据包又会发送给终点.原有的包叫做lost duplicate或者wandering duplicate.TCP必须处理这些重复的包.**

**提供TIME\_WAIT这个状态有2个理由：**

1. **为了实现TCP『全双工』连接中断的可靠性**
2. **消除网络上过期的包拷贝.**

**端口数(1~65535.端口数为16位整数)**

**IANA中的著名端口: 1～1023**

**IANA注册的端口： 1024~49151**

**BSD保留端口： 1~1023**

**BSD ephemeral ports: 1024~5000**

**BSD servers(nonprivileged): 5001~65535**

**Solaris: rresvport: 1~1023**

**Solaris ephemeral: 32768~65535**

**套接字对(socket pair)**

**An IP address and a port number are often called a *socket.* 对UDP和TCP都如此.**