TCP 和 UDP 的区别

 # TCP 协议与 UDP 协议的区别

首先咱们弄清楚，TCP 协议和 UCP 协议与 TCP/IP 协议的联系，很多人犯糊涂了，一直都是说 TCP 协议与 UDP 协议的区别，我觉得这是没有从本质上弄清楚网络通信！

TCP/IP 协议是一个协议簇。里面包括很多协议的。UDP 只是其中的一个。
之所以命名为 TCP/IP 协议，因为 TCP/IP 协议是两个很重要的协议，就用他两命名了。

TCP/IP 协议集包括应用层，传输层，网络层，网络访问层。

其中应用层包括

1）超文本传输协议(HTTP):万维网的基本协议
2）文件传输(TFTP 简单文件传输协议)
3）远程登录(Telnet),提供远程访问其它主机功能，它允许用户登录 internet 主机，并在这台主机上执行命令
4) 网络管理(SNMP 简单网络管理协议), 该协议提供了监控网络设备的方法, 以及配置管理, 统计信息收集，性能管理及安全管理等
5）域名系统(DNS)，该系统用于在 internet 中将域名及其公共广播的网络节点转换成 IP 地址

其次网络层包括

1）Internet 协议(IP)
2）Internet 控制信息协议(ICMP)
3）地址解析协议(ARP)
4）反向地址解析协议(RARP)

最后说网络访问层

网络访问层又称作主机到网络层(host-to-network)
网络访问层的功能包括 IP 地址与物理地址硬件的映射，以及将 IP 封装成帧，基于不同硬件类型的网络接口，网络访问层定义了和物理介质的连接
当然我这里说得不够完善，TCP/IP 协议本来就是一门学问，每一个分支都是一个很复杂的流程，但我相信每位学习软件开发的同学都有必要去仔细了解一番。

下面我着重讲解一下 TCP 协议和 UDP 协议的区别

TCP（Transmission Control Protocol，传输控制协议）是面向连接的协议
1）也就是说，在收发数据前，必须和对方建立可靠的连接
一个 TCP 连接必须要经过三次“对话”才能建立起来，其中的过程非常复杂，只简单的描述下这三次对话的简单过程：
1）主机 A 向主机 B 发出连接请求数据包：“我想给你发数据，可以吗？”，这是第一次对话
2) 主机 B 向主机 A 发送同意连接和要求同步（同步就是两台主机一个在发送，一个在接收，协调工作）的数据包：“可以，你什么时候发？”，这是第二次对话
3）主机 A 再发出一个数据包确认主机 B 的要求同步：“我现在就发，你接着吧！”
这是第三次对话。三次“对话”的目的是使数据包的发送和接收同步，经过三次“对话”之后，

主机 A 才向主机 B 正式发送数据。

TCP 三次握手过程

1 主机 A 通过向主机 B 发送一个含有同步序列号的标志位的数据段给主机 B，向主机 B 请求建立连接，通过这个数据段，主机 A 告诉主机 B 两件事：我想要和你通信，你可以用哪个序列号作为起始数据段来回应我

2 主机 B 收到主机 A 的请求后，用一个带有确认应答(ACK)和同步序列号(SYN)标志位的数据段响应主机 A，也告诉主机 A 两件事：我已经收到你的请求了，你可以传输数据了，你要用那个序列号作为起始数据段来回应我

3 主机 A 收到这个数据段后,再发送一个确认应答，确认已收到主机 B 的数据段："我已收到回复，我现在要开始传输实际数据了，这样 3 次握手就完成了，主机 A 和主机 B 就可以传输数据了

3 次握手的特点

没有应用层的数据

SYN 这个标志位只有在 TCP 建立连接时才会被置 1

握手完成后 SYN 标志位被置 0

TCP 建立连接要进行 3 次握手,而断开连接要进行 4 次

1 当主机 A 完成数据传输后，将控制位 FIN 置 1，提出停止 TCP 连接的请求

2 主机 B 收到 FIN 后对其作出响应，确认这一方向上的 TCP 连接将关闭，将 ACK 置 1

3 由 B 端再提出反方向的关闭请求，将 FIN 置 1

4 主机 A 对主机 B 的请求进行确认，将 ACK 置 1，双方向的关闭结束

由 TCP 的三次握手和四次断开可以看出，TCP 使用面向连接的通信方式，大大提高了数据通信的可靠性，使发送数据端和接收端在数据正式传输前就有了交互，为数据正式传输打下了可靠的基础

名词解释

1）ACK　TCP 报头的控制位之一，对数据进行确认，确认由目的端发出

用它来告诉发送端这个序列号之前的数据段都收到了

比如，认号为 X，表示前 X-1 个数据段都收到了，有当 ACK=1 时，认号才有效

当 ACK=0 时，认号无效，时会要求重传数据，证数据的完整性

2）SYN　同步序列号，CP 建立连接时将这个位置 1

3）FIN　发送端完成发送任务位，TCP 完成数据传输需要断开时，出断开连接的一方将这位置 1

TCP 的包头结构

源端口 16 位

目标端口 16 位

序列号 32 位

回应序号 32 位

TCP 头长度 4 位

reserved 6 位

控制代码 6 位

窗口大小 16 位
偏移量 16 位
校验和 16 位
选项 32 位(可选)
这样我们得出了 TCP 包头的最小长度，为 20 字节。

 # UDP（User Data Protocol，用户数据报协议）
（1） UDP 是一个非连接的协议，传输数据之前源端和终端不建立连接，当它想传送时就简单地去抓取来自应用程序的数据，并尽可能快地把它扔到网络上。
在发送端，UDP 传送数据的速度仅仅是受应用程序生成数据的速度、计算机的能力和传输带宽的限制。
在接收端，UDP 把每个消息段放在队列中，应用程序每次从队列中读一个消息段。
（2） 由于传输数据不建立连接，因此也就不需要维护连接状态，包括收发状态等，因此一台服务机可同时向多个客户机传输相同的消息。
（3） UDP 信息包的标题很短，只有 8 个字节，相对于 TCP 的 20 个字节信息包的额外开销很小。
（4） 吞吐量不受拥挤控制算法的调节，只受应用软件生成数据的速率、传输带宽、源端和终端主机性能的限制。
（5）UDP 使用尽最大努力交付，即不保证可靠交付，因此主机不需要维持复杂的链接状态表（这里面有许多参数）。
（6）UDP 是面向报文的。发送方的 UDP 对应用程序交下来的报文，在添加首部后就向下交付给 IP 层。既不拆分，也不合并，而是保留这些报文的边界，因此，应用程序需要选择合适的报文大小。

UDP 的包头结构
源端口 16 位
目的端口 16 位
长度 16 位
校验和 16 位

小结 TCP 与 UDP 的区别
1.基于连接与无连接
2.对系统资源的要求（TCP 较多，UDP 少）
3.UDP 程序结构较简单
4.流模式与数据报模式
5.TCP 保证数据正确性，UDP 可能丢包，TCP 保证数据顺序，UDP 不保证

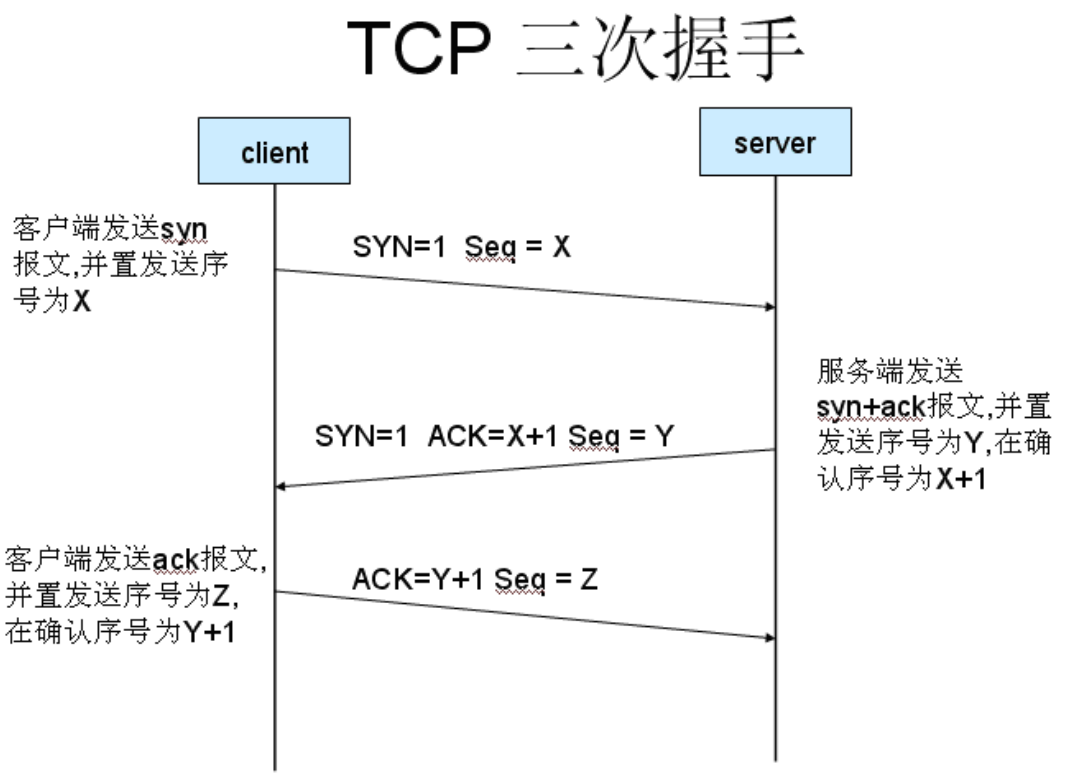
TCP 充分实现了数据传输时各种控制功能，可以进行丢包的重发控制，以对次序乱掉的分包进行顺序控制。而这些在 UDP 中都没有。此外，TCP 作为一种面向有连接的协议，只有在确认通信对端存在时才会发送数据，从而可以控制通信流量的浪费。TCP 通过检验和、序列号、确认应答、重发控制、连接管理以及窗口控制等机制实现可靠性传输。

1、TCP 面向连接（如打电话要先拨号建立连接）。UDP 是无连接的，即发送数据之前不需要建立连接

2、TCP 提供可靠的服务。也就是说，通过 TCP 连接传送的数据，无差错，不丢失，不重复，且按序到达。UDP 尽最大努力交付，即不保证可靠交付
3、TCP 面向字节流，实际上是 TCP 把数据看成一连串无结构的字节流。UDP 是面向报文的，UDP 没有拥塞控制，因此网络出现拥塞不会使源主机的发送速率降低（对实时应用很有用，如 IP 电话，实时视频会议等）
4、每一条 TCP 连接只能是点到点的。UDP 支持一对一，一对多，多对一和多对多的交互通信
5、TCP 首部开销 20 字节。UDP 的首部开销小，只有 8 个字节
6、TCP 的逻辑通信信道是全双工的可靠信道，UDP 则是不可靠信道

TCP3 次握手，4 次挥手过程

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

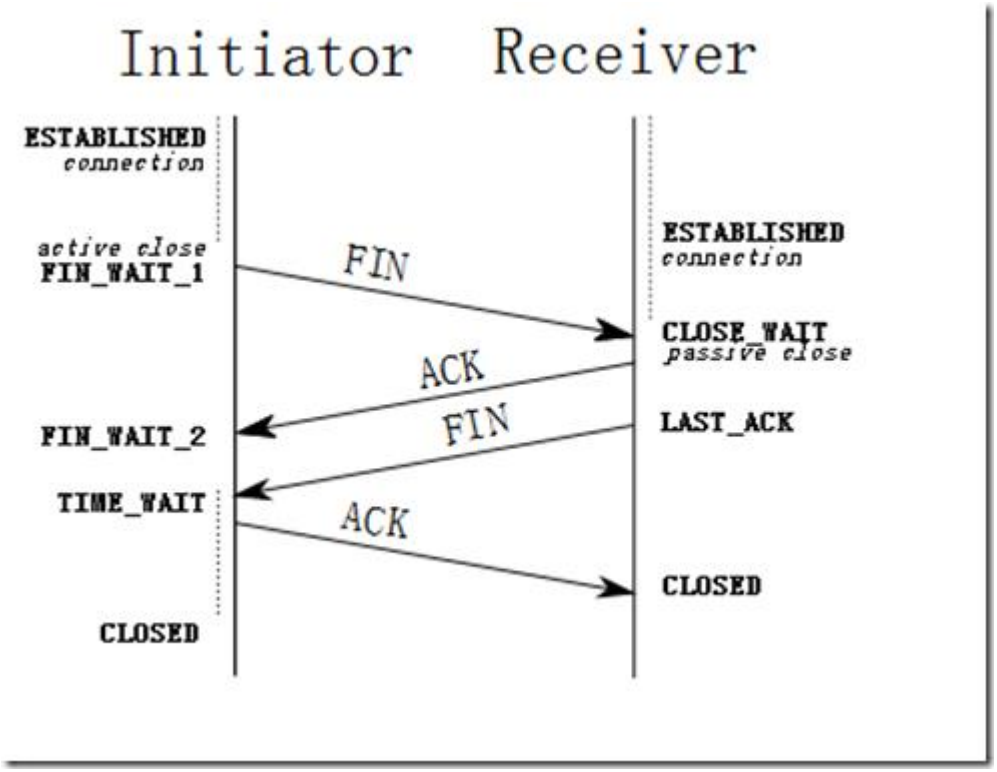
向上没有数据流动，一个 TCP 连接在收到一个 FIN 后仍能发送数据。首先进行关闭的一方将执行主动关闭，而另一方执行被动关闭。

（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）。



握手，挥手过程中各状态介绍

3 次握手过程状态：

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_SENT: 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。（发送端）

SYN_RCVD: 这个状态与 SYN_SENT 遥想呼应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。（服务器端）

ESTABLISHED：这个容易理解了，表示连接已经建立了。

4 次挥手过程状态：(可参考上图)

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。(主动方)

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(ACK 信息)，稍后再关闭连接。(主动方)

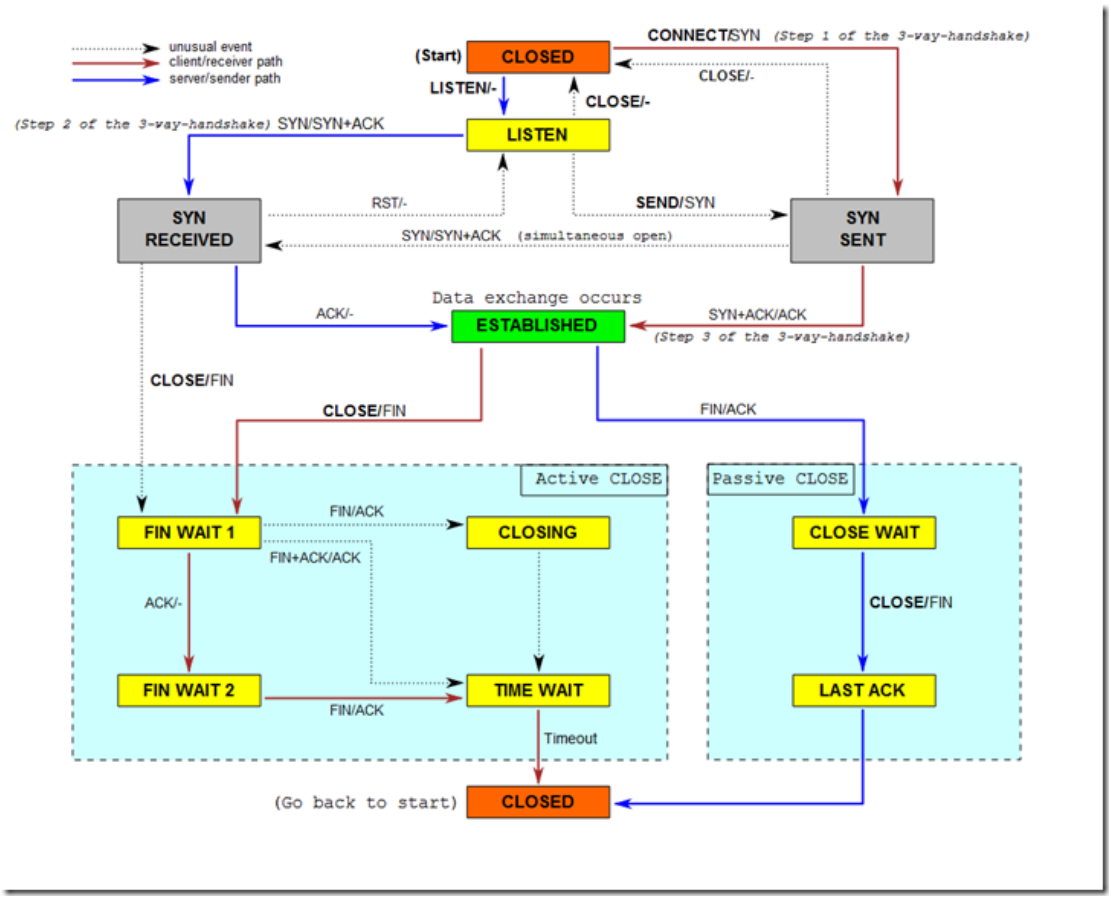
TIME_WAIT: 表示收到了对方的 FIN 报文，并发送出了 ACK 报文，就等 2MSL 后即可回到 CLOSED 可用状态了。如果 FIN_WAIT_1 状态下，收到了对方同时带 FIN 标志和 ACK 标志的报文时，可以直接进入到 TIME_WAIT 状态，而无须经过 FIN_WAIT_2 状态。(主动方)

CLOSING（比较少见）：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。(被动方)

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。(被动方)

CLOSED: 表示连接中断。


unusual event
client/receiver path
server/sender path
(Start)
CLOSED
CONNECT/SYN
(Step 1 of the 3-way-handshake)
CLOSE/-
LISTEN/-
CLOSE/-
(Step 2 of the 3-way-handshake) SYN/SYN+ACK
LISTEN
SYN RECEIVED
RST/-
SEND/SYN
SYN/SYN+ACK (simultaneous open)
SYN SENT
Data exchange occurs
ACK/-
ESTABLISHED
SYN+ACK/ACK
(Step 3 of the 3-way-handshake)
CLOSE/FIN
CLOSE/FIN
FIN/ACK
Active CLOSE
Passive CLOSE
FIN WAIT 1
FIN/ACK
CLOSING
FIN+ACK/ACK
CLOSE WAIT
ACK/-
CLOSE/FIN
FIN WAIT 2
TIME WAIT
FIN/ACK
LAST ACK
Timeout
(Go back to start)
CLOSED