# 传输层：TCP 协议

## 一、概述

TCP 和 UDP 处在同一层——运输层，但是它们有很多的不同。TCP 是 TCP/IP 系列协议中最复杂的部分，它具有以下特点：

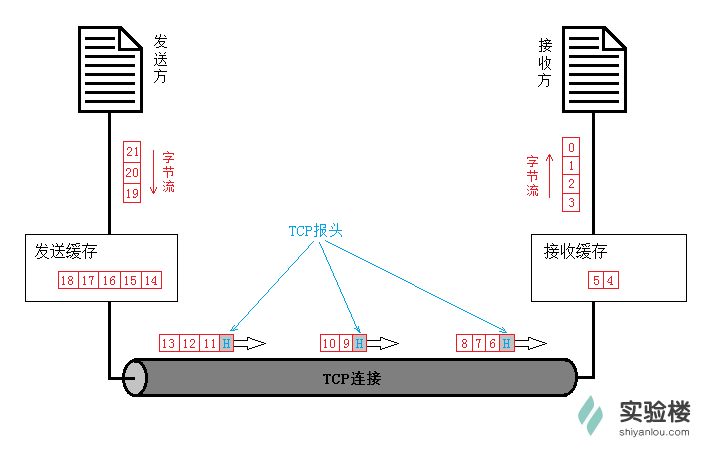
(1) TCP 提供 **可靠的** 数据传输服务，TCP 是 **面向连接的** 。应用程序在使用 TCP 通信之前，先要建立连接，这是一个类似“打电话”的过程，通信结束后还要“挂电话”。

(2) TCP 连接是 **点对点** 的，一条 TCP 连接只能连接两个端点。

(3) TCP 提供可靠传输，无差错、不丢失、不重复、按顺序。

(4) TCP 提供 **全双工** 通信，允许通信双方任何时候都能发送数据，因为 TCP 连接的两端都设有发送缓存和接收缓存。

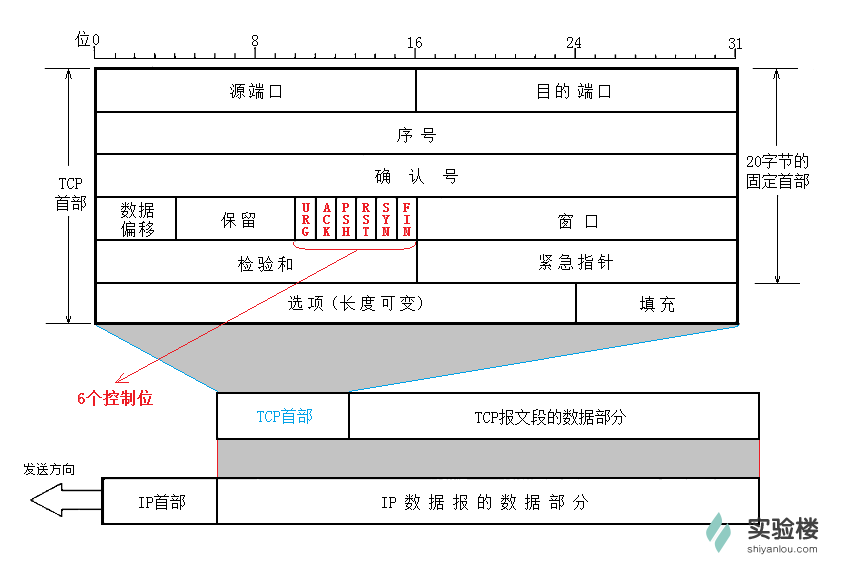
(5) TCP 面向 **字节流** 。TCP 并不知道所传输的数据的含义，仅把数据看作一连串的字节序列，它也不保证接收方收到的数据块和发送方发出的数据块具有大小对应关系。



## 二、TCP 报文段结构

TCP 是面向字节流的，而 TCP 传输数据的单元是 **报文段** 。一个 TCP 报文段可分为两部分：报头和数据部分。数据部分是上层应用交付的数据，而报头则是 TCP 功能的关键。

TCP 报文段的报头有前 20 字节的固定部分，后面 4n 字节是根据需要而添加的字段。如图则是 TCP 报文段结构：



20 字节的固定部分，各字段功能说明：

1.**源端口和目的端口**:各占 2 个字节，分别写入源端口号和目的端口号。这和 UDP 报头有类似之处，因为都是运输层协议。

2.**序号**:占 4 字节序，序号范围[0，2^32-1]，序号增加到 2^32-1 后，下个序号又回到 0。 TCP 是面向字节流的，通过 TCP 传送的字节流中的每个字节都按顺序编号，而报头中的序号字段值则指的是本报文段数据的**第一个字节的序号**。

3.**确认序号**:占 4 字节，期望收到对方下个报文段的第一个数据字节的序号。

4.**数据偏移**:占 4 位，指 TCP 报文段的报头长度，包括固定的 20 字节和选项字段。

5.**保留**:占 6 位，保留为今后使用，目前为 0。

6.**控制位**:共有 6 个控制位，说明本报文的性质，意义如下： **URG 紧急**:当 URG=1 时，它告诉系统此报文中有紧急数据，应优先传送(比如紧急关闭)，这要与**紧急指针**字段配合使用。 **ACK 确认**:仅当 ACK=1 时**确认号**字段才有效。建立 TCP 连接后，所有报文段都必须把 ACK 字段置为 1。 **PSH 推送**:若 TCP 连接的一端希望另一端立即响应，PSH 字段便可以“催促”对方，不再等到缓存区填满才发送。 **RET 复位**:若 TCP 连接出现严重差错，RST 置为 1，断开 TCP 连接，再重新建立连接。 **SYN 同步**:用于建立和释放连接，稍后会详细介绍。 **FIN 终止**:用于释放连接，当 FIN=1，表明发送方已经发送完毕，要求释放 TCP 连接。

7.**窗口**:占 2 个字节。窗口值是指发送者自己的**接收窗口**大小，因为**接收缓存**的空间有限。

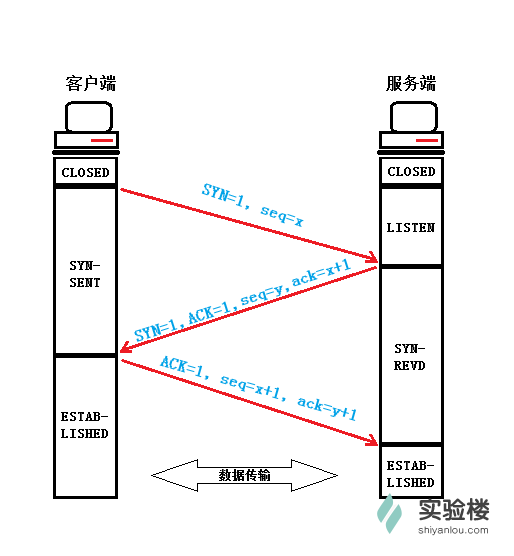
8.**检验和**:2 个字节。和 UDP 报文一样，有一个检验和，用于检查报文是否在传输过程中出差错。

9.**紧急指针**:2 字节。当 URG=1 时才有效，指出本报文段紧急数据的字节数。

10.**选项**:长度可变，最长可达 40 字节。具体的选项字段，需要时再做介绍。

## 三、连接的建立与释放

刚才说过，TCP 是面向连接的，在传输 TCP 报文段之前先要创建连接，发起连接的一方被称为客户端，而响应连接请求的一方被称为服务端，而这个创建连接的过程被称为 **三次握手** ：



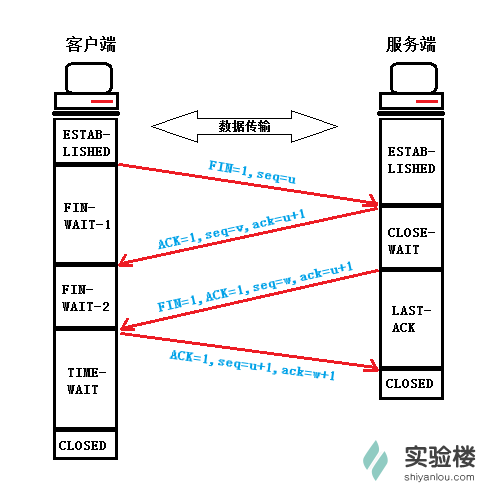
(1) 客户端发出请求连接报文段，其中报头控制位 SYN=1，初始序号 seq=x。客户端进入 SYN-SENT(同步已发送)状态。

(2) 服务端收到请求报文段后，向客户端发送确认报文段。确认报文段的首部中 SYN=1，ACK=1，确认号是 ack=x+1，同时为自己选择一个初始序号 seq=y。服务端进入 SYN-RCVD(同步收到)状态。

(3) 客户端收到服务端的确认报文段后，还要给服务端发送一个确认报文段。这个报文段中 ACK=1，确认号 ack=y+1，而自己的序号 seq=x+1。**这个报文段已经可以携带数据，如果不携带数据则不消耗序号，则下一个报文段序号仍为 seq=x+1**。

至此 TCP 连接已经建立，客户端进入 ESTABLISHED(已建立连接)状态，当服务端收到确认后，也进入 ESTABLISHED 状态，它们之间便可以正式传输数据了。

当传输数据结束后，通信双方都可以释放连接，这个释放连接过程被称为 **释放连接** :



(1) 此时 TCP 连接两端都还处于 ESTABLISHED 状态，客户端停止发送数据，并发出一个 FIN 报文段。首部 FIN=1，序号 seq=u（u 等于客户端传输数据最后一字节的序号加 1）。客户端进入 FIN-WAIT-1(终止等待 1)状态。

(2) 服务端回复确认报文段，确认号 ack=u+1，序号 seq=v（v 等于服务端传输数据最后一字节的序号加 1），服务端进入 CLOSE-WAIT(关闭等待)状态。现在 TCP 连接处于**半开半闭状态**，服务端如果继续发送数据，客户端依然接收。

(3) 客户端收到确认报文，进入 FIN-WAIT-2 状态，服务端发送完数据后，发出 FIN 报文段，FIN=1，确认号 ack=u+1，然后进入 LAST-ACK(最后确认)状态。

(4) 客户端回复确认确认报文段，ACK=1，确认号 ack=w+1（w 为半开半闭状态时，收到的最后一个字节数据的编号） ，序号 seq=u+1，然后进入 TIME-WAIT(时间等待)状态。

注意此时连接还没有释放，需要时间等待状态结束后(4 分钟) 连接两端才会 CLOSED。设置时间等待是因为，有可能最后一个确认报文丢失而需要重传。

## 四、TCP 可靠传输的实现

(1) TCP 报文段的长度可变，根据收发双方的缓存状态、网络状态而调整。

(2) 当 TCP 收到发自 TCP 连接另一端的数据，它将发送一个确认。

(3) 当 TCP 发出一个段后，它启动一个定时器，等待目的端确认收到这个报文段，如果不能及时收到一个确认，将重发这个报文段。这就是稍后介绍的**超时重传**。

(4) TCP 将保持它首部和数据的检验和。如果通过检验和发现报文段有差错，这个报文段将被丢弃，等待超时重传。

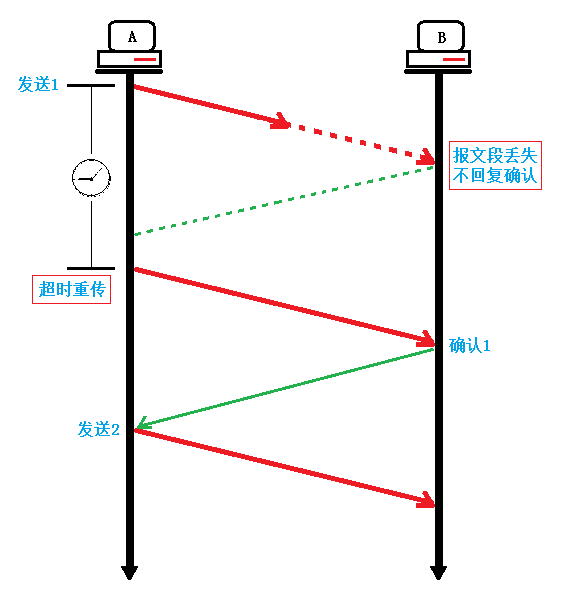
(5) TCP 将数据按字节排序，报文段中有序号，以确保顺序的正确性。

(6) TCP 还能提供流量控制。TCP 连接的每一方都有收发缓存。TCP 的接收端只允许另一端发送接收端缓冲区所能接纳的数据。这将防止较快主机致使较慢主机的缓冲区溢出。

可见超时重发机制是 TCP 可靠性的关键，只要没有得到确认报文段，就重新发送数据报，直到收到对方的确认为止。

## 五、超时重传

TCP 规定，接收者收到数据报文段后，需回复一个确认报文段，以告知发送者数据已经收到。而发送者如果一段时间内(**超时计时器**)没有收到确认报文段，便重复发送：



为了实现超时间重传，需要注意：

1.发送者发送一个报文段后，**暂时保存该报文段的副本**，为发生超时重传时使用，收到确认报文后删除该报文段。

2.确认报文段也需要序号，才能明确是发出去的那个数据报得到了确认。

3.超时计时器比传输往返时间略长，但具体值是不确定的，根据网络情况而变。

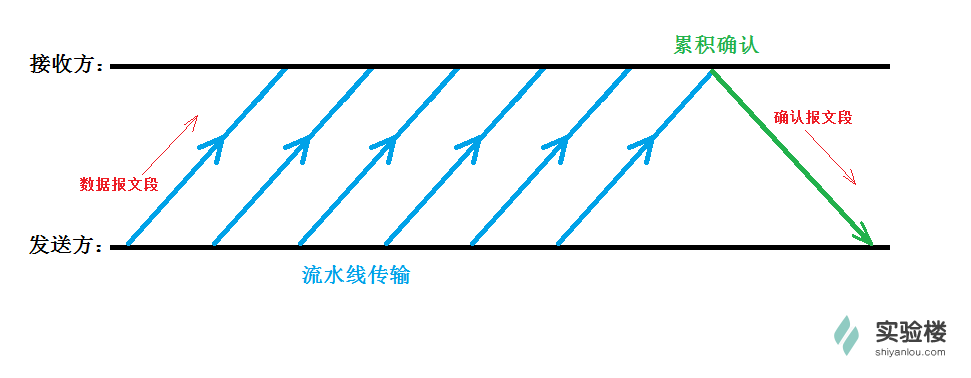
## 六、连续 ARQ 协议

也许你也发现了，按上面的介绍，超时重传机制很费时间，每发送一个数据报都要等待确认。

在实际应用中的确不是这样的，真实情况是，采用了**流水线传输**：发送方可以连续发送多个报文段(连续发送的数据长度叫做窗口)，而不必每发完一段就停下来等待确认。

实际应用中，接收方也不必对收到的每个报文都做回复，而是采用**累积确认**方式：接收者收到多个连续的报文段后，只回复确认最后一个报文段，表示在这之前的数据都已收到。

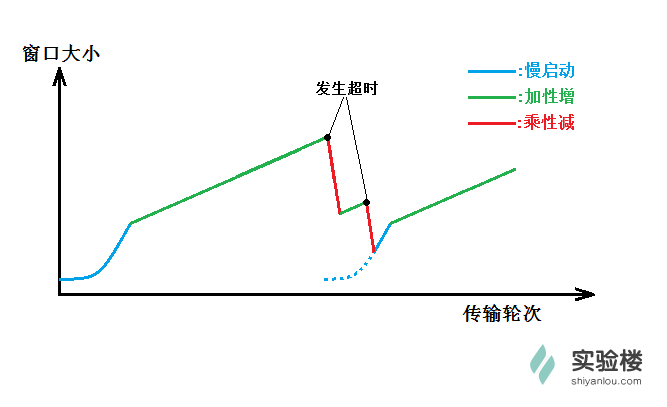
这样，传输效率得到了很大的提升。



## 七、流量控制和拥塞控制

由于接收方缓存的限制，发送窗口不能大于接收方接收窗口。在报文段首部有一个字段就叫做**窗口(rwnd)**，这便是用于告诉对方自己的接收窗口，可见窗口的大小是可以变化的。

那么窗口的大小是如何变化的呢？TCP 对于拥塞的控制总结为“慢启动、加性增、乘性减”，如图所示：



(1) **慢启动** ：初始的窗口值很小，但是按指数规律渐渐增长，直到达到**慢开始门限(ssthresh)**。

(2) **加性增** ：窗口值达到慢开始门限后，每发送一个报文段，窗口值增加一个单位量。

(3) **乘性减** ：无论什么阶段，只要出现超时，则把窗口值减小一半。

## 八、tcpdump 抓取 TCP 报文段

上一节实验，我们用 tcpdump 抓取并阅读了 UDP 报文，那么这次我们尝试抓取 TCP 报文段。当然首先要安装 tcpdump(安装方法参考上一节实验)。

针对这次实验，需要下载从 github 下载代码，是基于 TCP 的聊天小程序，分为 server(服务端)和 client(客户端)：

cd Desktop

git clone http://git.shiyanlou.com/shiyanlou/tcp\_ip\_6

cd tcp\_ip\_6

gcc -o server server.c

gcc -o client client.c

编译完成后先不要运行，先打开 tcpdump，使用命令安装并运行 tcpdump：

sudo apt-get update

sudo apt-get install tcpdump

sudo tcpdump -vvv -X -i lo tcp port 7777

最小化运行 tcpdump 的终端，然后另开一个终端，运行 server 程序：

cd Desktop/tcp\_ip\_6

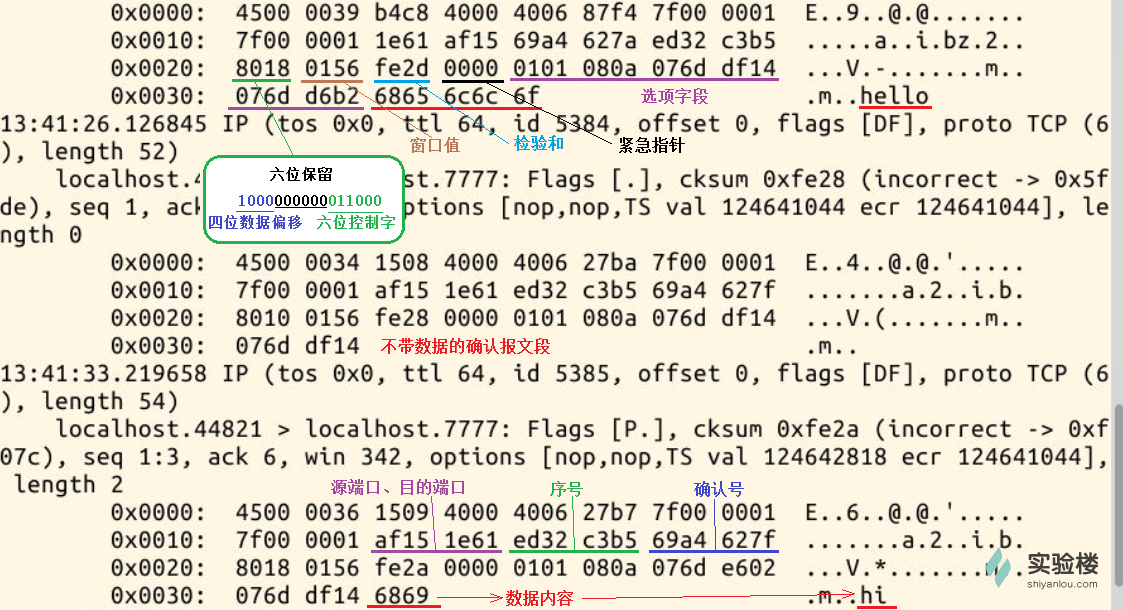
./server 127.0.0.1

然后再打开第三个终端，运行 client 程序：

cd Desktop/tcp\_ip\_6

./client 127.0.0.1

现在，使用 client 和 server 聊天，轮流互发几条简短的消息(比如 hello、hi、wei 之类的)便可以关闭 client 和 server，回到运行 tcpdump 的终端查看抓取的报文段内容：



通过抓取的报文，还可以清晰的看到建立连接三次握手和断开连接四次握手的过程。

## 九、作业

1、按实验步骤，使用 tcpdump 抓取 TCP 报文段，观察三次握手建立连接和四次握手释放连接的过程，并截图。

2、使用实验给出的聊天程序收发消息，用 tcpdump 抓取包含聊天内容的 TCP 报文段，解读聊天内容，并截图。