

TCP v.s UDP

首先 UDP 协议是面向无连接的，也就是说不需要在正式传递数据之前先连接起双方。然后 UDP 协议只是数据报文的搬运工，不保证有序且不丢失的传递到对端，并且UDP 协议也没有任何控制流量的算法，总的来说 UDP 相较于 TCP 更加的轻便。

**UDP：**

* UDP 相比 TCP 简单的多，不需要建立连接，不需要验证数据报文，不需要流量控制，只会把想发的数据报文一股脑的丢给对端
* 虽然 UDP 并没有 TCP 传输来的准确，但是也能在很多实时性要求高的地方有所作为

**TCP**

总结一下这一章节的内容：

* 建立连接需要三次握手，断开连接需要四次握手
* 滑动窗口解决了数据的丢包、顺序不对和流量控制问题
* 拥塞窗口实现了对流量的控制，保证在全天候环境下最优的传递数据

**连续 ARQ**

在连续 ARQ 中，发送端拥有一个**发送窗口**，可以在没有收到应答的情况下持续发送窗口内的数据，这样相比停止等待 ARQ 协议来说减少了等待时间，提高了效率。

### 累计确认

连续 ARQ 中，接收端会持续不断收到报文。如果和停止等待 ARQ 中接收一个报文就发送一个应答一样，就太浪费资源了。通过累计确认，可以在收到多个报文以后统一回复一个应答报文。报文中的 ACK 标志位可以用来告诉发送端这个序号之前的数据已经全部接收到了，下次请发送这个序号后的数据。

但是累计确认也有一个弊端。在连续接收报文时，可能会遇到接收到序号 5 的报文后，并未接收到序号 6 的报文，然而序号 7 以后的报文已经接收。遇到这种情况时，ACK 只能回复 6，这样就会造成发送端重复发送数据的情况。

## 滑动窗口

在上面小节中讲到了发送窗口。在 TCP 中，两端其实都维护着窗口：分别为发送端窗口和接收端窗口。

发送端窗口是由接收窗口剩余大小决定的。接收方会把当前接收窗口的剩余大小写入应答报文，发送端收到应答后根据该值和当前网络拥塞情况设置发送窗口的大小，所以发送窗口的大小是不断变化的。

**滑动窗口是一个很重要的概念，它帮助 TCP 实现了流量控制的功能。**接收方通过报文告知发送方还可以发送多少数据，从而保证接收方能够来得及接收数据，防止出现接收方带宽已满，但是发送方还一直发送数据的情况。

### Zero 窗口

在发送报文的过程中，可能会遇到对端出现零窗口的情况。在该情况下，发送端会停止发送数据，并启动 persistent timer 。该定时器会定时发送请求给对端，让对端告知窗口大小。在重试次数超过一定次数后，可能会中断 TCP 链接。

## 拥塞处理

拥塞处理和流量控制不同，后者是作用于接收方，保证接收方来得及接受数据。而前者是作用于网络，防止过多的数据拥塞网络，避免出现网络负载过大的情况。

拥塞处理包括了四个算法，分别为：慢开始，拥塞避免，快速重传，快速恢复。

### 慢开始算法

慢开始算法，顾名思义，就是在传输开始时将发送窗口慢慢指数级扩大，从而避免一开始就传输大量数据导致网络拥塞。想必大家都下载过资源，每当我们开始下载的时候都会发现下载速度是慢慢提升的，而不是一蹴而就直接拉满带宽。

慢开始算法步骤具体如下

1. 连接初始设置拥塞窗口（Congestion Window） 为 1 MSS（一个分段的最大数据量）
2. 每过一个 RTT 就将窗口大小乘二
3. 指数级增长肯定不能没有限制的，所以有一个阈值限制，当窗口大小大于阈值时就会启动拥塞避免算法。

**拥塞避免算法**

拥塞避免算法相比简单点，每过一个 RTT 窗口大小只加一，这样能够避免指数级增长导致网络拥塞，慢慢将大小调整到最佳值。

在传输过程中可能定时器超时的情况，这时候 TCP 会认为网络拥塞了，会马上进行以下步骤：

* 将阈值设为当前拥塞窗口的一半
* 将拥塞窗口设为 1 MSS
* 启动拥塞避免算法

**快速重传**

快速重传一般和快恢复一起出现。一旦接收端收到的报文出现失序的情况，接收端只会回复最后一个顺序正确的报文序号。如果发送端收到三个重复的 ACK，无需等待定时器超时而是直接启动快速重传算法。具体算法分为两种：

**TCP Taho 实现如下**

* 将阈值设为当前拥塞窗口的一半
* 将拥塞窗口设为 1 MSS
* 重新开始慢开始算法

**TCP Reno 实现如下**

* 拥塞窗口减半
* 将阈值设为当前拥塞窗口
* 进入快恢复阶段（重发对端需要的包，一旦收到一个新的 ACK 答复就退出该阶段），这种方式在丢失多个包的情况下就不那么好了
* 使用拥塞避免算法

### TCP New Ren 改进后的快恢复

**TCP New Reno** 算法改进了之前 **TCP Reno** 算法的缺陷。在之前，快恢复中只要收到一个新的 ACK 包，就会退出快恢复。

在 **TCP New Reno** 中，TCP 发送方先记下三个重复 ACK 的分段的最大序号。

假如我有一个分段数据是 1 ~ 10 这十个序号的报文，其中丢失了序号为 3 和 7 的报文，那么该分段的最大序号就是 10。发送端只会收到 ACK 序号为 3 的应答。这时候重发序号为 3 的报文，接收方顺利接收的话就会发送 ACK 序号为 7 的应答。这时候 TCP 知道对端是有多个包未收到，会继续发送序号为 7 的报文，接收方顺利接收并会发送 ACK 序号为 11 的应答，这时发送端认为这个分段接收端已经顺利接收，接下来会退出快恢复阶段。

UDP------------------------

## 面向无连接

首先 UDP 是不需要和 TCP 一样在发送数据前进行三次握手建立连接的，想发数据就可以开始发送了。

并且也只是数据报文的搬运工，不会对数据报文进行任何拆分和拼接操作。

具体来说就是：

* 在发送端，应用层将数据传递给传输层的 UDP 协议，UDP 只会给数据增加一个 UDP 头标识下是 UDP 协议，然后就传递给网络层了
* 在接收端，网络层将数据传递给传输层，UDP 只去除 IP 报文头就传递给应用层，不会任何拼接操作

## 不可靠性

首先不可靠性体现在无连接上，通信都不需要建立连接，想发就发，这样的情况肯定不可靠。

并且收到什么数据就传递什么数据，并且也不会备份数据，发送数据也不会关心对方是否已经正确接收到数据了。

再者网络环境时好时坏，但是 UDP 因为没有拥塞控制，一直会以恒定的速度发送数据。即使网络条件不好，也不会对发送速率进行调整。这样实现的弊端就是在**网络条件不好的情况下可能会导致丢包，但是优点也很明显，在某些实时性要求高的场景（比如电话会议）就需要使用 UDP 而不是 TCP**。

## 高效

虽然 UDP 协议不是那么的可靠，但是正是因为它不是那么的可靠，所以也就没有 TCP 那么复杂了，需要保证数据不丢失且有序到达。

因此 UDP 的头部开销小，只有八字节，相比 TCP 的至少二十字节要少得多，在传输数据报文时是很高效的。