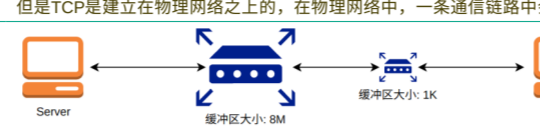


TCP拥塞控制

为什么需要拥塞控制

在TCP建立连接，即三次握手时，会告知对方自己的接收窗口大小，通常在64KB左右。也就是说，假设MSS为1460，要填满接收方的窗口需要发送44个包。当接收方缓冲区已满时，会告知发送方自己的窗口已经收缩，使发送方减缓发送频率。所以基本不用担心缓冲区溢出的问题

但是TCP是建立在物理网络之上的，在物理网络中，一条通信链路中会有无数的交换机以及路由器，每台物理设备的配置不同，所能处理和承受的数据帧数量也不相同



这是一个很极端的例子，以此说明在物理通信链路中路由器的配置差别可能会非常大

此时如果Server发送MSS大小为1460的报文，则会被路由器丢弃，接收方永远收不到该报文

是否能够收集通信链路中所有路由器和交换机的最大缓冲区大小，然后取最低的那个数值作为MSS大小？

不行

网络是动态变化的，就好比公路一样，可能只有早高峰很堵，其它时间都很通畅

其次，就算路由器能动态计算出自己剩余的缓冲区大小，并将此数值传递给发送方需要时间，而在这段时间内，无法保证剩余缓冲区大小不会变化

试探

武林高手在对决时，一开始并不知道对方的功力，所以一开始会试探，出几个小招看对方能不能接住，在摸清底细之后才会放开

TCP协议确定拥塞点的方式也是试探，利用各种算法使得获取的拥塞点尽可能的接近真实的拥塞点，而不是得到一个非常准确的数值

TCP拥塞控制

当连接建立时，双方并不知道对方的网络情况，如果一口气发太多包很有可能就遭遇拥塞，导致大面积的丢包，所以发送方将拥塞窗口的初始值定义的非常小，例如2个MSS

如果发出去的包都得到了确认，表示还没有到达拥塞点，可以增大拥塞窗口

在这个时间段内，由于基数很低，所以增速可以快一些，RFC建议每收到N个确认包，拥塞窗口就增加N个MSS，过程为线性增长

即便是线性增长，但是基数太小，传输速度依然不高，所以这个过程也称之为慢启动

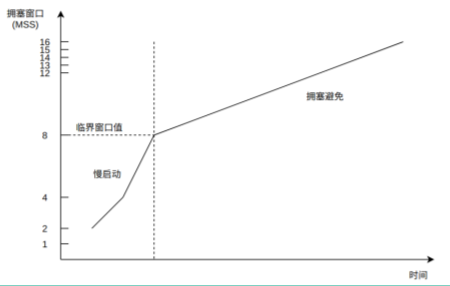
慢启动过程持续一段时间后，拥塞窗口会到一个比较大的值，此时传输速度较快，并且碰到拥塞点的概率就会增加

所以不能再翻倍增加窗口大小，而是缓慢增加。RFC建议在每个往返时间增加一个MSS

该过程称为拥塞避免

拥塞窗口与接收方的接收窗口相等，此时再增加拥塞窗口没有意义，因为接收方无法接收超过接收窗口大小的数据包

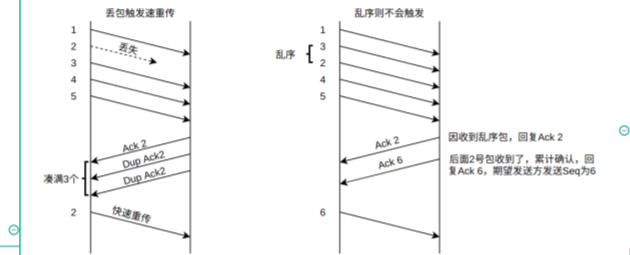
在拥塞窗口增加时发生了拥塞，那么此时的拥塞窗口大小就会作为参考拥塞点被记录



网络由于物理层的复杂而变得复杂，某台路由器运行时轻微抖动，校验和校验失败等问题，都可能会导致少量的包丢失

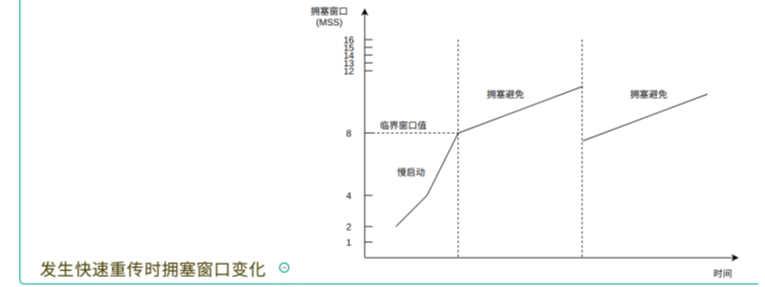
当数据包持续发送并且之间出现一个包丢失时，接收方就会发现收到的包并不连续，所以此时接收方每收到一个包就Ack一次期望的Seq号，以此提醒发送方重传

当发送方收到3个或以上重复确认(Dup Ack)时，就会意识到相应的包已经丢失了，从而立即重传该包，整个过程称为快速重传



如果发生了快速重传，同样会影响到拥塞窗口的大小，尽管可能是因为轻微网络抖动导致的

此时拥塞窗口将降低为发生拥塞时还没被确认的数量的1/2，并继续保持拥塞避免阶段

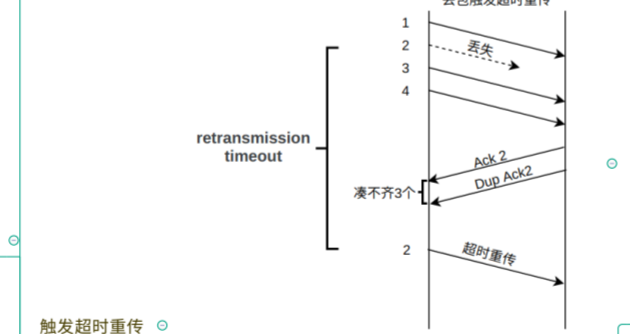


重传的研究

触发快速重传的必要条件是丢包以后存在3个或以上的数据包正确发送至接收方，如果丢包后面没有3个或以上的数据包继续传输的话，将会触发超时重传

所以，小文件的传输在丢包时可能造成的后果要远远大于大文件传输时的丢包后果

在后面可以看到，超时重传带来的后果远远超过快速重传

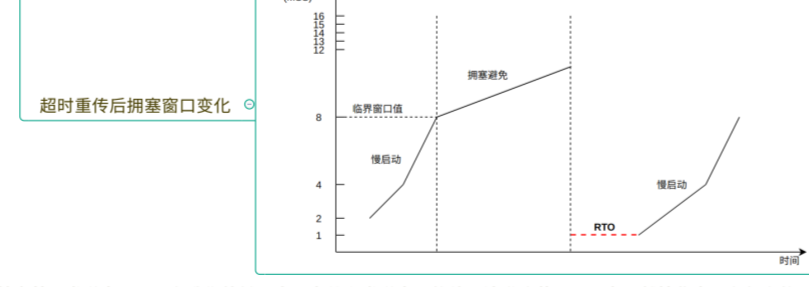


当2号包丢失后，由于后续只有3号、4号两个包的发送，所以无法触发发送方进行快速重传。只能由发送方的定时器到期时进行重新发送

从发出原始包到重传该包的这段时间称之为RTO，重传超时时间。实际上，这个值相当大，单位通常以秒计算

超时重传会直接导致重新进入慢启动过程

当发生超时重传后，RFC建议将拥塞窗口缩减到1个MSS，然后重新进入慢启动以及拥塞避免过程



拥塞窗口其实就是发送窗口，现在我们能够明白，为什么发送窗口的值无法准确获取。因为虽然接收窗口在每次的TCP报文传输过程中会告知对方，但是这并不代表发送方能够发送接收方窗口大小的数据，其中一个很重要的影响因素就是物理层通信链路的不确定性。同时，这也是为什么TCP滑动窗口的大小不会与接收方的接收窗口大小相等的原因

