一、重传机制

在TCP中，发送端的数据到达接收端后，接收端主机会返回一个确认应答消息。

常见的重传机制：

超时重传

快速重传

SACK

D-SACK

1.超时重传：发送数据时设定一个定时器，当没有收到ACK且超过指定的时间就会重发该数据

TCP在数据包丢失、确认应答丢失两种情况会发生超时重传

超时时间该设置为多少？

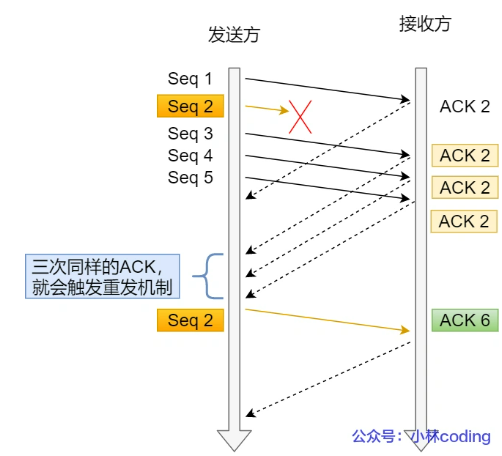
RRT（Round-Trip Time 往返时延）：数据发送时刻到接收到确认的时刻的差值，也就是包的往返时间

RTO（Retransmission Timeout 超时重传时间）：RTO较大会导致重发效率低，RTO过小会导致没有丢包就重发，会增加网络拥塞导致更多的超时。因此重传时间RTO应略大于报文往返时间RRT。实际上因为网络是时常变化的，RRT经常波动变化，因此RTO是一个动态变化的值。

超时重发的数据再次超时又需要重传的时候，TCP的策略是超时间隔加倍，每次超时重传时会将下次超时时间间隔设为之前的两倍。

超时重传的问题是超时周期可能相对较长，可以用快速重传机制解决超时重发的时间等待。快速重传机制不以时间为驱动，而是以数据驱动重传。

2.快速重传的工作机制：



第一份Seq1送到了，回复ACK2

Seq2没有送到，Seq3、Seq4、Seq5都送到了，但还是回复ACK2

发送端收到了三个ACK2的确认，知道了Seq2还没有收到，就会在定时器过期之前重传丢失的Seq2

收到Seq2后因为Seq3、Seq4、Seq5都收到了，于是回ACK6

快速重传的工作方式是当收到三个相同的ACK报文时会在定时器过期前重传丢失的报文段。

快速重传解决了超时时间的问题，但存在重传的时候重传一个还是重传所有的问题。如Seq2和Seq3都丢失了，后续收到的都是ACK2，但发送方并不清楚要重传Seq2报文还是Seq2之后的所有报文。

只重传Seq2，对于丢失的Seq3还得在后续收到三个重复的ACK3才能触发重传，重传的效率低。

重传Seq2之后的所有报文，对已经被接收过的报文做了无用功，浪费资源。

于是有了SACK

3. SACK（Selective Acknowledgment 选择性确认）方法：需要在TCP头部选项字段里加一个SACK的部分，它可以将已收到的数据的信息发送给发送方，这样发送方知道数据的接收情况，可以只重传丢失的数据。

4.D-SACK（Duplicate SACK）：使用SACK告诉发送方有哪些数据被重复接收了

好处：

让发送方知道是发送的包丢了还是回应的ACK包丢了

知道是不是发送方的数据包被网络延迟了

知道网络中是不是把发送方的数据包给复制了

二、滑动窗口：

引入窗口概念的原因：发送数据后等待确认应答再发送下一个数据，往返时间越长通信的效率越低。引入窗口即使在往返时间较长的情况下也不会降低网络通信的效率。

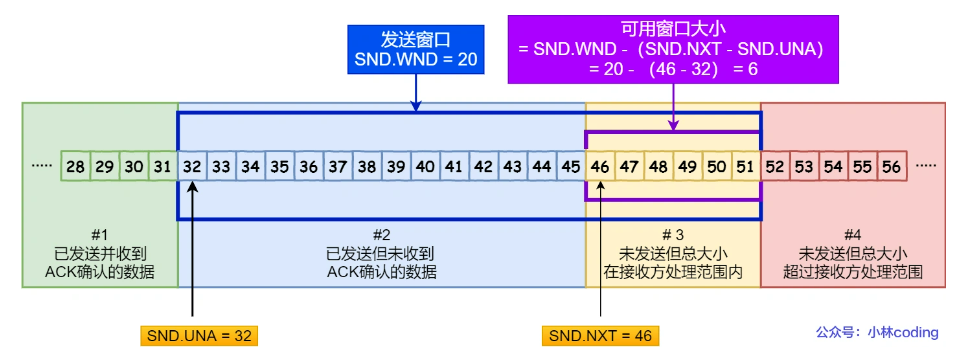
窗口大小就是指无需等待确认应答而可以继续发送数据的最大值。

累计应答（累计确认）：发送方收到了一个ACK确认应答就意味着这个应答之前的所有数据接收方都收到了，即使前面的ACK丢失了也不会进行数据重发，可以通过下一个确认应答进行确认。

窗口大小由哪一方决定？

通常是由接收方的窗口大小来决定的，TCP头里有一个Window字段就是窗口大小，这个字段告诉发送端接收端还有多少缓冲区可以接收数据，于是发送端就可以根据这个接收端的处理能力来发送数据。

发送端的窗口：



#1已经发送并收到ACK的数据

#2已经发送但未收到ACK的数据

#3未发送但总大小在接收方处理范围内

#4未发送且总大小在接收方处理范围外

使用三个指针（两个绝对指针（指特定的序列号），一个相对指针（需要做偏移））来跟踪四个传输类别：

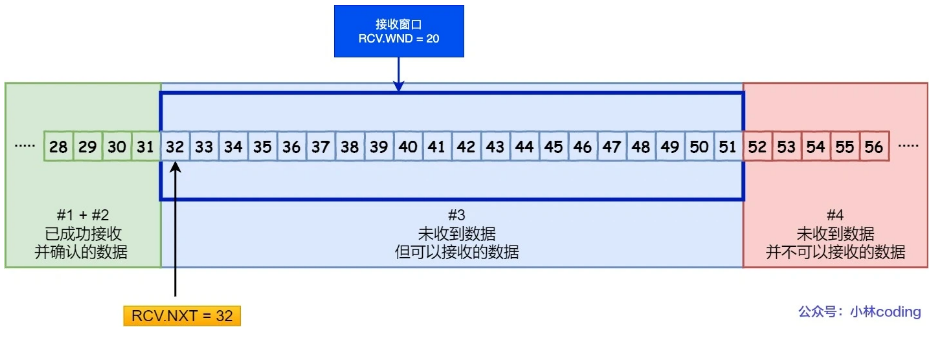
SND.WND：表示发送窗口的大小（大小由接收方指定）

SND.UNA：是一个绝对指针，指向发送且没有收到ACK的第一个字节的序列号

SND.NXT：是一个绝对指针，指向未发送但可发送的第一个字节的序列号

可用窗口大小=SND.WND – (SND.NXT – SND.UNA)

接收方的滑动窗口:



根据处理的情况划分为三个部分:

#1+#2已成功接收并确认的数据

#3未收到但可以接收的数据

#4未收到且不可接收的数据

三个部分用两个指针进行划分:

RCV.WND: 表示接收窗口的大小,它会通过给发送方

RCV.NXT: 是一个指针,指向期望发送方发送来的下一个数据字节的序列号

接收方和发送方的窗口大小是相等的吗?

并不是完全相等,接收方的窗口大小约等于发送方窗口的大小,因为滑动窗口并不是一成不变的,在同步窗口大小的时候存在时延,所以接收窗口和发送窗口是约等于的关系.

三. 流量控制: TCP提供一种机制可以让发送方根据接收方的实际接收能力控制发送的数据量

发送方发送数据要考虑接收方处理能力,如果接收方处理不过来就会导致触发重传机制,导致网络流量的无端浪费.

四. 操作系统缓冲区与滑动窗口的关系

发送窗口和接收窗口中存放的字节数都是放在操作系统内存缓冲区中的,而操作系统的缓冲区会被操作系统调整. 应用进程无法即使读取缓冲区内容时会对缓冲区造成影响.

操作系统缓冲区如何影响发送窗口和接收窗口?

一种情况: 发送的数据被接收后没有被进程全部读取, 在缓存区中占用空间的字节会导致窗口收缩相应的大小,并通过ACK中的Window字段通知发送方减小发送窗口,当窗口收缩至0时触发窗口关闭.

另一种情况: 服务端系统资源非常紧张时操作系统可能会直接减少接收缓冲区大小,这是接收窗口的大小被减小了,而发送端还没有收到消息就发送了下一条报文,因为这个是按上一次约定的滑动窗口的大小发送的,因此可能会超过当前的接收窗口的大小,这是会出现数据包丢失的现象.

为了防止这种现象,TCP规定不允许同时减少缓存又收缩窗口,而是采用提前收缩窗口,过一段时间再减少缓存,以避免丢包情况.

五. 窗口关闭:如果窗口大小为0时,就会阻止发送方给接收方传递数据,直到窗口变为非0.

窗口关闭潜在的风险: 发生窗口关闭时,接收方处理完数据后会向发送方通告一个窗口非0的ACK报文,如果这个报文在网络中丢失,发送方会一直等待接收方非0的窗口通知,而ACK报文是不会重传的,因此接收方也一直等待发送方的数据,会造成死锁.

TCP为了解决这个问题,为每个连接设置一个持续定时器,只要TCP连接一方收到对方的零窗口通知就启动持续计时器,如果持续计时器超时就发送窗口探测报文,对方在确认这个探测报文时给出自己现在的接收窗口的大小. 如果接收窗口为0就会重置持续计时器,如果窗口非0就打破了死锁. 窗口探测的次数一般为3次,如果3次过后接收窗口还是0就会发送RST来中断连接.

六. 糊涂窗口综合症: 接收方没有及时取出接收窗口里的数据导致发送方的发送窗口越来越小, 接收方几个字节的通告也会导致发送方发送,发送的数据量少不经济.

解决方法:

让接收方不通告小窗口给发送方: 当窗口大小小于min(MSS, 缓存空间/2) 也就是小于MSS与1/2缓存大小中的最小值时就会向发送方通告窗口为0, 组织发送方发送数据.

让发送方避免发送小数据: 使用Nagle算法: 只有满足(窗口大小>=MSS并且数据大小>=MSS)或(收到之前发送数据的ACK)才发送数据

如果接收方不能满足不通告小窗口,即使使用Nagle算法无法避免糊涂窗口综合症,因为如果接收方ACK回复很快也会导致发送的数据量小.

接收方不通告小窗口给发送方 + 发送方开启Nagle算法 才能避免糊涂窗口综合症

七. 拥塞控制

流量控制是为了发送方的数据填满接收方的缓存,并不知道网络中发生了什么. 但网络中可能会因为其他主机之间的通信使得网络拥堵. 在网络出现拥堵时,如果继续发送大量的数据包可能会导致数据包时延, 丢失等,这时TCP会重传数据,但重传会导致网络里负担更重.

拥塞控制的目的就是避免发送方的数据填满整个网络

定义了拥塞窗口. 拥塞窗口cwnd是发送方维护的一个状态变量, 是根据网络的拥塞程度动态变化的,加入了拥塞窗口之后,发送窗口swnd=min(cwnd, rwnd),也就是拥塞窗口和接收窗口中的最小值.

拥塞窗口的变化规则:

只要网络中没有出现拥塞cwnd就会增大

网络中出现拥塞cwnd就会减小

怎么知道网络中出现拥塞?

发送方没有在规定时间内接收到ACK应答报文,也就是发生了超时重传就认为网络出现了拥塞

拥塞控制主要是四个算法: 慢启动, 拥塞避免, 拥塞发生, 快速回复

1.慢启动:TCP在刚建立完成后有一个慢启动过程,一点一点的提高发送数据包的数量,当发送方每收到一个ACK,拥塞窗口cwnd的大小就会加1. 如开始cwnd为1个MSS,收到一个ACK后变为2个MSS,然后收到两个ACK后变为4个MSS. 慢启动算法的发包个数是指数性增长的,直到慢启动门限ssthresh(slow start threshold), 当cwnd<ssthread时使用慢启动算法,当cwnd>=ssthresh时使用拥塞避免算法.

2.拥塞避免算法:

当cwnd超过慢启动门限ssthresh就会进入拥塞避免算法,ssthresh大小一般是65535字节

进入拥塞避免算法后, 每当收到一个ACK时cwnd增加1 / cwnd

当8个ACK确认应答到来时,每个ACK使cwnd增加1/8,总共增加1. 拥塞避免算法使cwnd变成线性增长

慢启动算法和拥塞避免算法都是增长阶段,随着增长网络进入了拥塞状况,出现了丢包现象,对丢失的数据包进行重传. 当触发了重传机制就进入了拥塞发生算法.

3. 拥塞发生

网络出现拥塞后重传机制主要有两种:超时重传, 快速重传

这两种重传使用的拥塞发生算法是不同的

发生超时重传的拥塞发生算法: ssthresh设为 cwnd / 2, cwnd恢复初值, 接着重新开始慢启动. 这种方法会突然减少数据流造成网络卡顿.

发生快速重传的拥塞发生算法: TCP认为快速重传不严重,因为大部分包没丢, 这时cwnd = cwnd / 2, ssthresh = cwnd, 然后进入快速恢复算法

4. 快速恢复

快速重传和快速恢复算法一般同时使用,快速恢复之前cwnd和ssthresh已经被更新了,然后:

1.cwnd = ssthresh + 3(3的意思是有3个数据包被收到了)

2.重传丢失的数据包

3.如果再收到重复的ACK,那么cwnd增加1

4.如果收到新数据的ACK,把ACK置为第一步中的ssthresh的值, 因为该ACK确认了新数据,该恢复过程已经结束,可以恢复到拥塞避免状态