# **一、FEC应用简析**

  前面的NACK方案能在延迟较小的情况下快速补偿丢包，但是随着丢包率上升以及延迟增加，纯NACK方案逐渐无法满足传输需求了。那么在已知高延迟高丢包的环境下，我们可以通过提前[冗余](https://so.csdn.net/so/search?q=%E5%86%97%E4%BD%99&spm=1001.2101.3001.7020" \t "https://blog.csdn.net/qw225967/article/details/_blank)去做数据补充，让我们可以快速有效的抵抗丢包，这就是FEC（前向纠错）在较差环境下抗丢包的优势，当然它的劣势就是会固定地增加带宽来抵抗丢包，因此我们更多考虑将他使用在超过20%丢包以上的场景上。

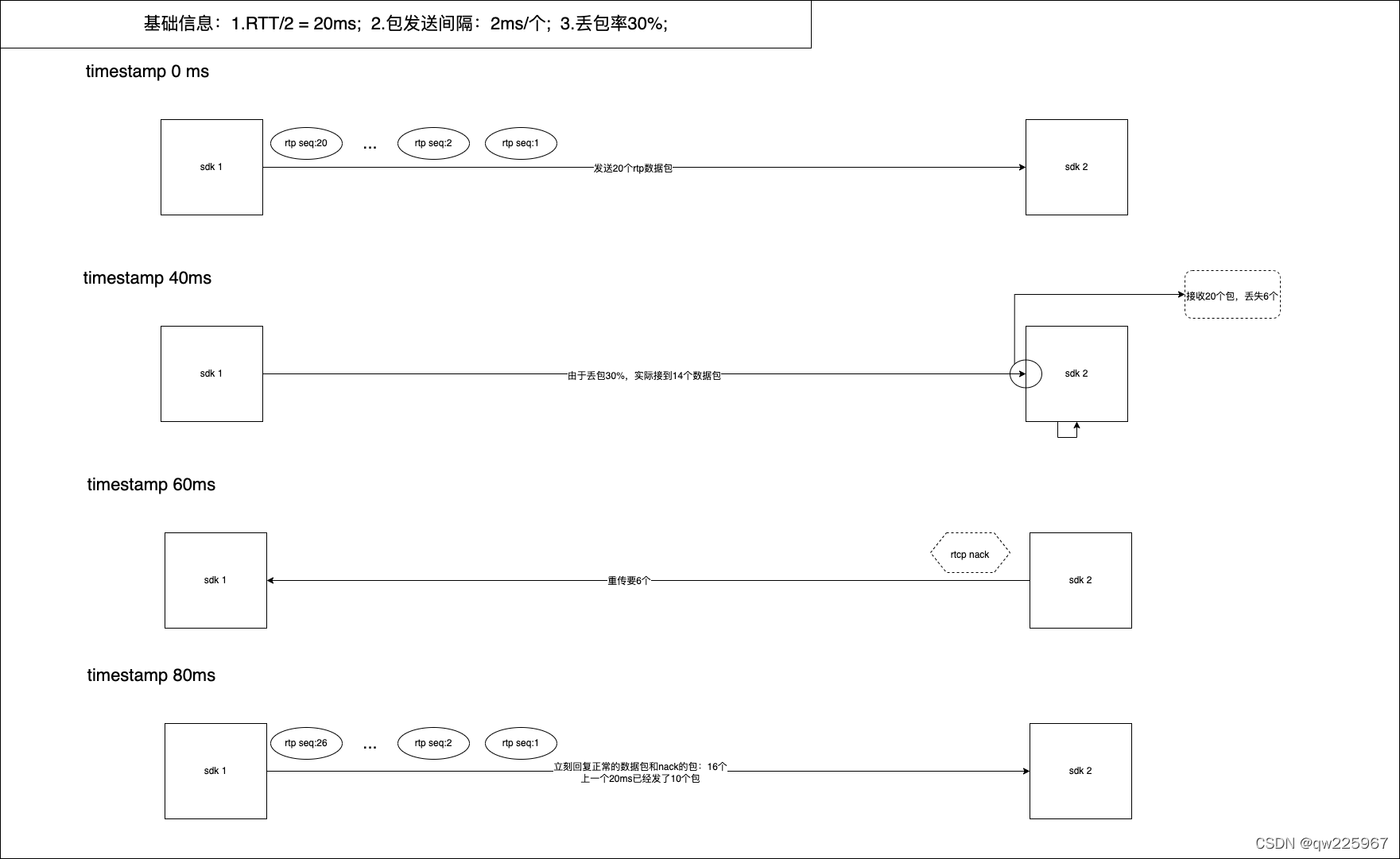
## **1.1 FEC原理简述**

  我这里有一篇文章简单分析了固定冗余的FEC算法原理，从中可以了解FEC的基本逻辑：[FEC原理](https://blog.csdn.net/qw225967/article/details/123405345)

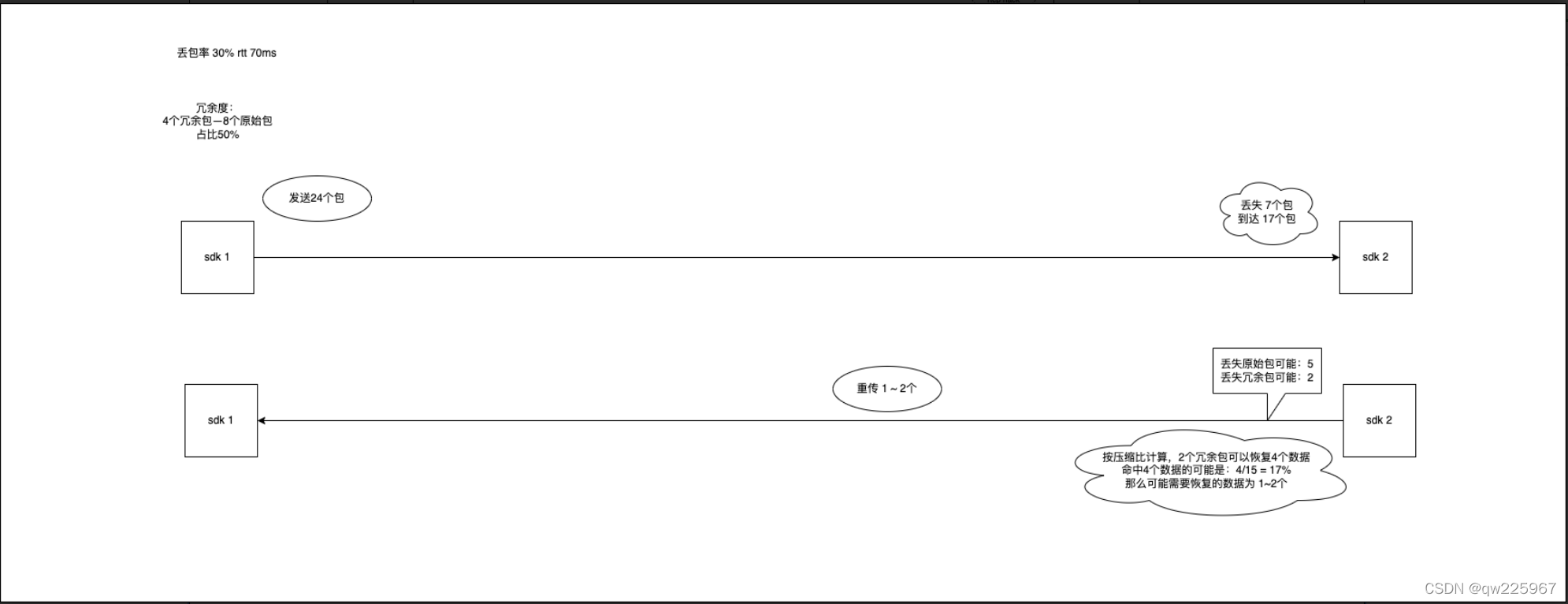
## **1.2 FEC编码的不同**

  各类FEC编码都是为了优化冗余比以及算力的，这里有一些记录可以看一下：[RS与FEC编码](https://blog.csdn.net/petib_wangwei/article/details/38441095)

# **二、FEC丢包模型**

  根据上次nack中提到的丢包模型[NACK丢包模型](https://blog.csdn.net/qw225967/article/details/123206637)，我们来计算一下这里使用fec可以达到的效果：  


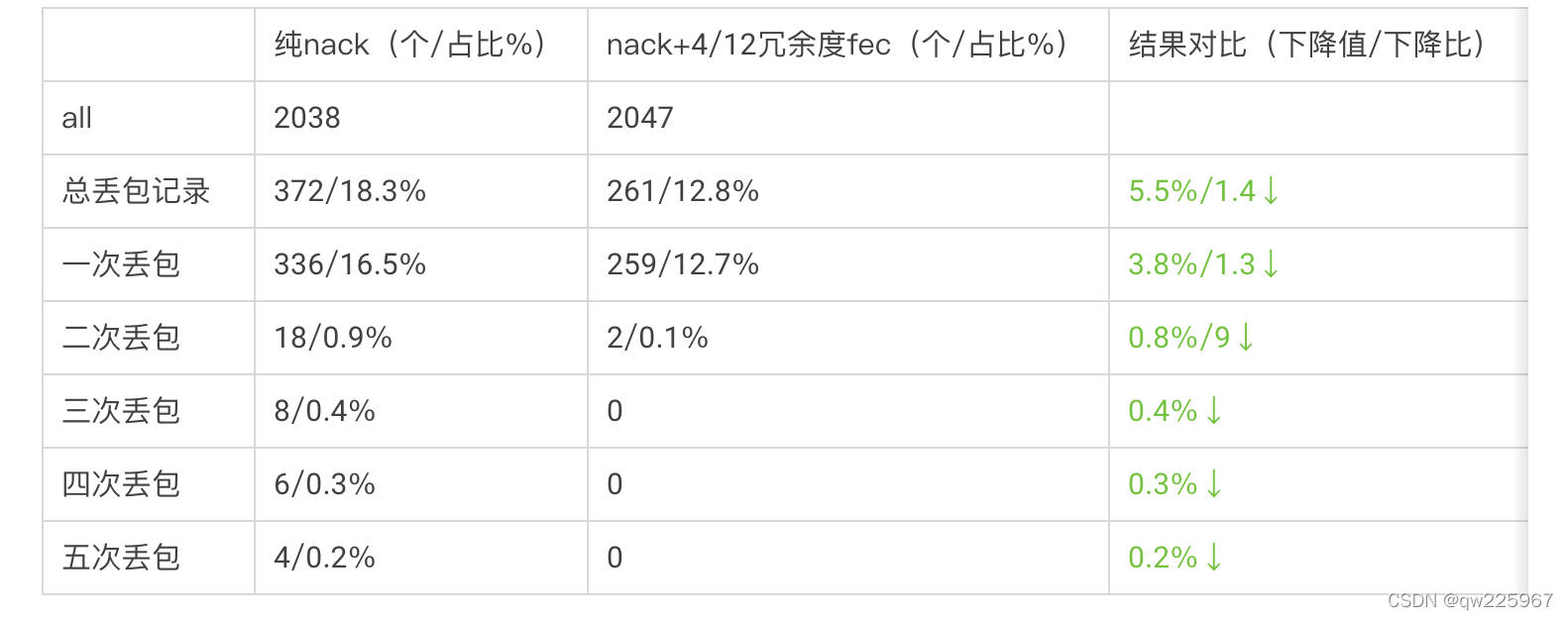
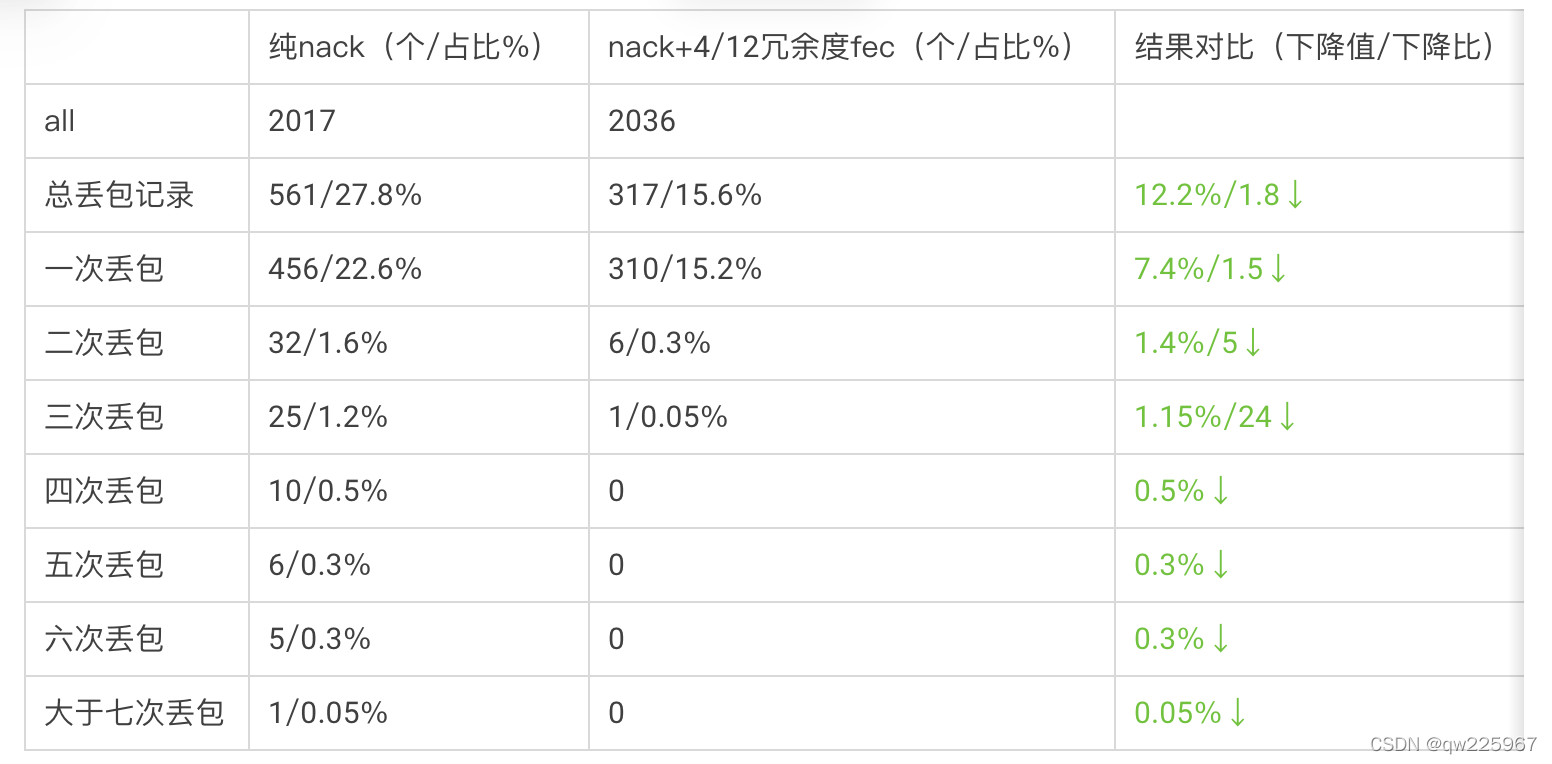
  上图提到重传问题，我们在这里加入冗余比较低的基于范德蒙矩阵的fec（webrtc中的fec冗余比会远大于这个，而且动态效果也会更好，意味着会比现在带宽减少较多——webrtc的ulpfec一次最多支持48个媒体包，[webrtc冗余度介绍](https://blog.csdn.net/qq_16135205/article/details/89843062)）



  上图中弱网条件为：30%丢包 + 70ms rtt的情况。  
  我们仿真使用的是基于范德蒙固定矩阵的FEC算法，设置了8个包打4个冗余度的情况进行实验（后续可以试试别的冗余度论证）。  
那么计算过程可以归纳为：  
  1.当发送24个包时，会有16个原始数据 + 8个冗余数据。到达对端时可能会丢失7个包；  
  2.丢失的数据中，有可能有5个数据包是原始数据、2个数据包是冗余数据；  
  3.而冗余数据每1一个可以压缩2个原始数据，那么可恢复的可能就是75%；  
  4.那么理论上5个原始数据中有3-4个包可以有效恢复，也就是1 ~ 2个包会发生重传的情况。

  因此计算出来第一次重传在30%丢包的情况下可能会明显降低至 12%左右（30% \* 2/5 = 12%）下降约18%，下降比1 ：1.7；

# **三、测试对比**

  根据上述的理论分析，我们对基于范德蒙矩阵的fec进行仿真，对比重传发生概率的下降比。  
****环境：20% 丢包 + 70ms rtt****  
  
****环境：30% 丢包 + 70ms rtt****  


# **四、结果与分析**

  对上述理论进行分析：  
  1.FEC提前进行冗余压缩并不是全部包都做冗余，而是存在对应的冗余比率，我们仿真使用的基于范德蒙矩阵的FEC冗余度仍然存在浪费，换到webrtc原生的FEC会更能适应真正的网络环境。即使如此，我们仿真的结果依然能反馈出FEC可以有效降低重传概率；  
  2.同时观察多组数据发现，FEC+NACK的方案还可以有效降低重传的次数，理论上会在高丢包情况下反应得更明显一些。而且在高丢包环境下，FEC的优势是：  
     ■ 无需等待重传的延迟；  
    ■ 单个FEC包恢复多个，降低带宽消耗；  
  3.根据上述降重传次数的情况，我们还可以分析出在高延迟环境下，FEC比NACK延迟更低，因为随着RTT的上涨，重传周期会越来越大。

  所以，我们可以得知：  
  1.FEC可以有效降低第一次丢包的占比，明显降低丢包的概率；  
  2.FEC可以减少重传次数，随着丢包率越大，这样的表现越明显；  
  3.根据第二条结论，我们可以知道在高延迟的情况下，FEC对丢包补偿的效果体现越明显。