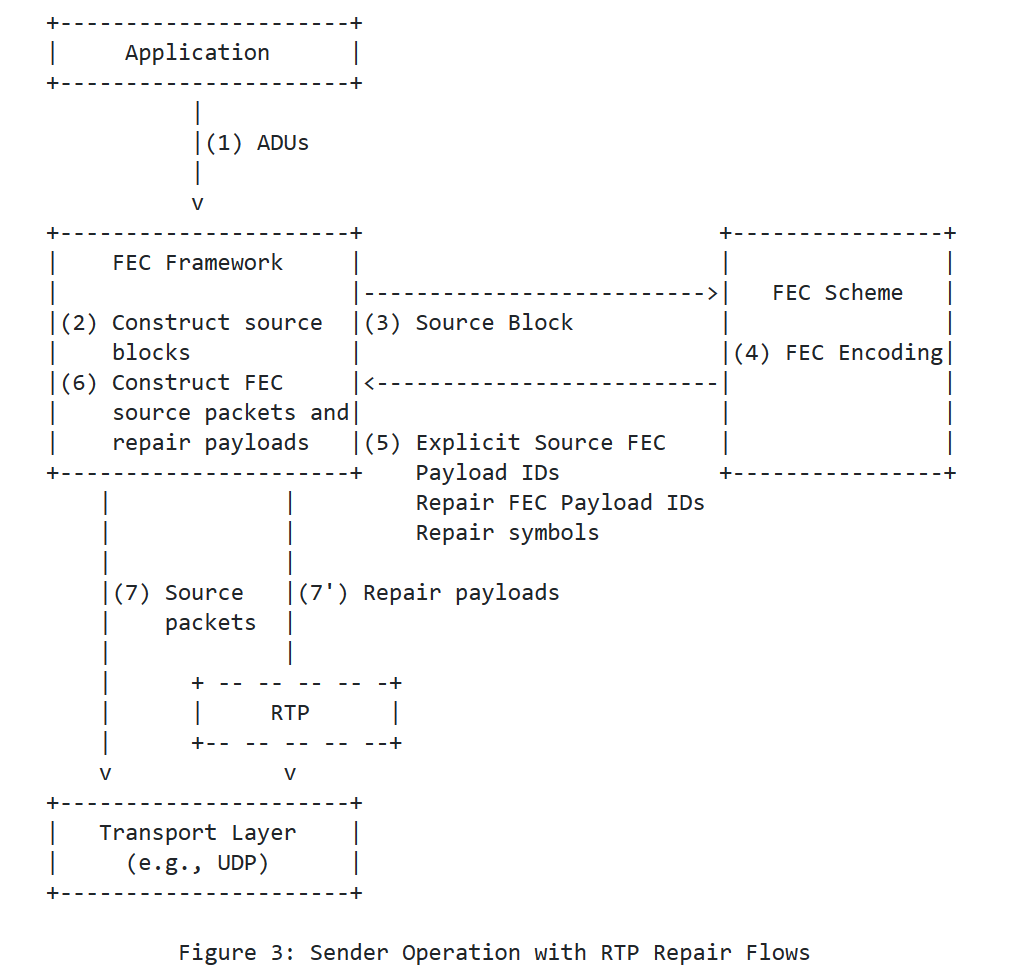
前面我们已经介绍过FEC的基础原理，包括XOR和RS FEC的。这一篇我们将介绍如何通过RTP承载FEC数据，从传输角度来介绍如何实现。介绍还是以启发的方式，希望能让大家了解为什么要这样设计。

## **1. FEC的传输**

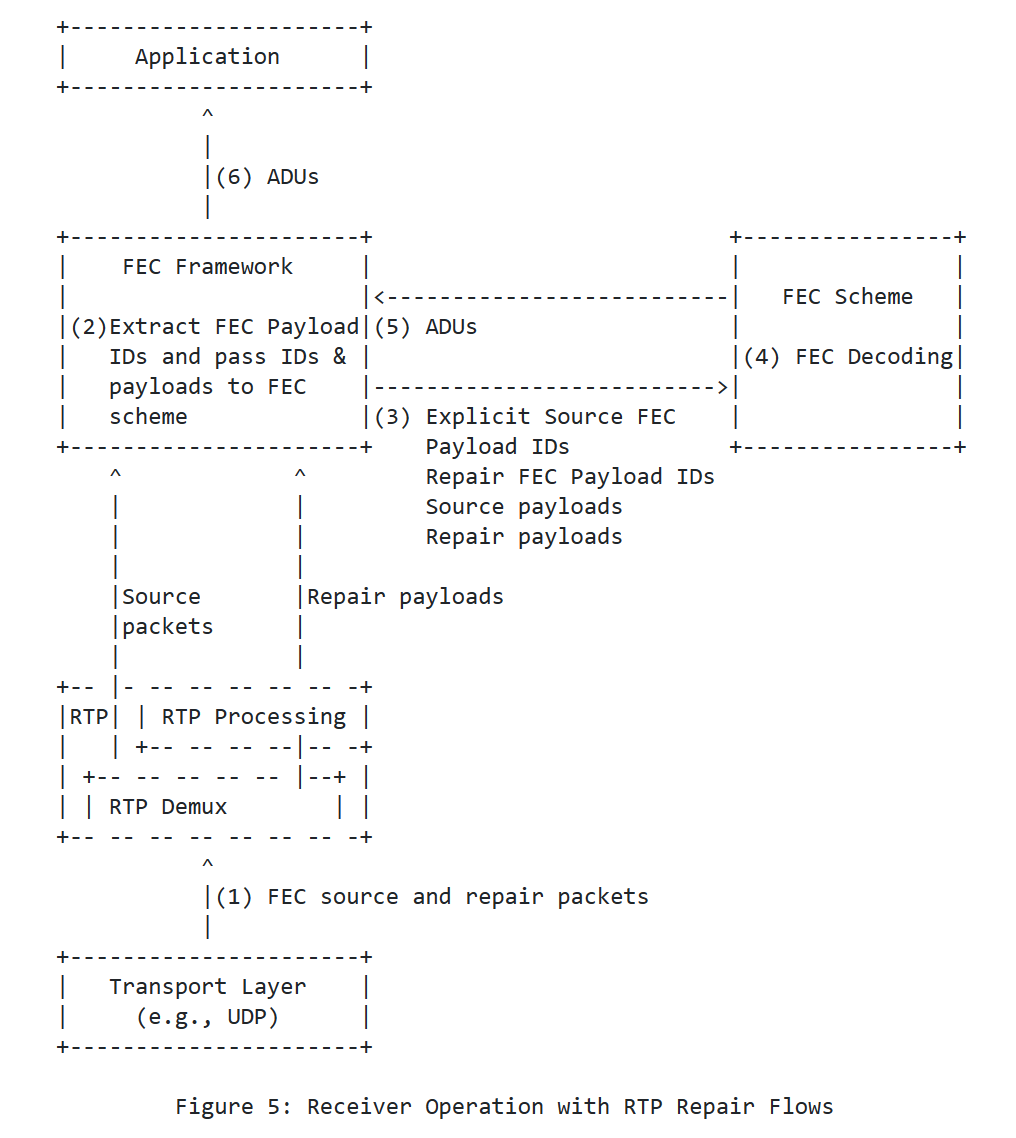
### **1.1 处理流程**

我们可以从一些标准框架里面得到一些设计灵感。如[RFC 6363 - Forward Error Correction (FEC) Framework (ietf.org)](https://link.zhihu.com/?target=https://datatracker.ietf.org/doc/html/rfc6363" \t "https://zhuanlan.zhihu.com/p/_blank)就描述了一个通用的FEC框架。当然，也很简单。

我们可以看下，通过RTP传输FEC方式的流程：



[接收端](https://zhida.zhihu.com/search?content_id=222282440&content_type=Article&match_order=1&q=%E6%8E%A5%E6%94%B6%E7%AB%AF&zhida_source=entity" \t "https://zhuanlan.zhihu.com/p/_blank)的流程：



### **1.1 RTP[封装](https://zhida.zhihu.com/search?content_id=222282440&content_type=Article&match_order=1&q=%E5%B0%81%E8%A3%85&zhida_source=entity" \t "https://zhuanlan.zhihu.com/p/_blank)的问题**

除了基本流程，我们需要知道的东西太多了。我们需要把编码信息通知到接收端，否则无法完成[解码](https://zhida.zhihu.com/search?content_id=222282440&content_type=Article&match_order=1&q=%E8%A7%A3%E7%A0%81&zhida_source=entity" \t "https://zhuanlan.zhihu.com/p/_blank)。通知通道可以包括：

* 带外方式：SDP协商
* 带内方式：信息放到FEC头部

FEC[编码算法](https://zhida.zhihu.com/search?content_id=222282440&content_type=Article&match_order=1&q=%E7%BC%96%E7%A0%81%E7%AE%97%E6%B3%95&zhida_source=entity" \t "https://zhuanlan.zhihu.com/p/_blank)相关参数，一般来说，通用的能力，通话中不变的我们可以通过SDP协商方式，如使用的是ULP FEC还是Flex FEC，还是RS FEC。而随时可以变的FEC编码参数我们通过带内传输。常变的参数如：FEC保护哪些包，这些包的sequence number分别多少（后面我们可以看到有两个很重要的FEC参数：SN base和mask）。

我们还需要考虑FEC的rtp包PT、ssrc、sn、ts怎么填，怎么和其他包区分开。以及是否支持多路流一起FEC？我们的设计需要考虑到，接收端是否能将这些包区分开来，清楚地划分[原始包](https://zhida.zhihu.com/search?content_id=222282440&content_type=Article&match_order=1&q=%E5%8E%9F%E5%A7%8B%E5%8C%85&zhida_source=entity" \t "https://zhuanlan.zhihu.com/p/_blank)、FEC包、他们在编码中的位置、他们的属于哪个流。这其实就是Multiplexing和Demultiplexing的设计逻辑了。

## **2. 几个FEC传输方案**

以上，我们抛出了一些问题，下面我们来看下几个成熟方案如何解决这个问题。

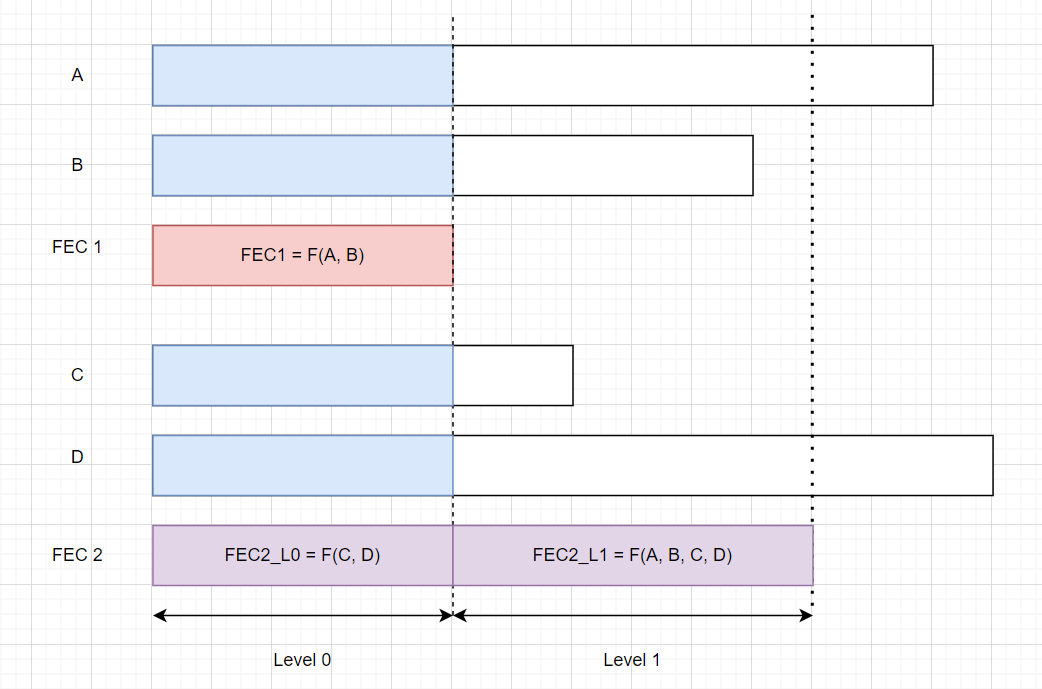
### **2.1 ULP FEC - RFC5019**

[!NOTE] ULP，即Uneven Level Protection，即不均匀保护。WebRTC里面默认已经实现。  
[RFC 5109 - RTP Payload Format for Generic Forward Error Correction (ietf.org)](https://link.zhihu.com/?target=https://datatracker.ietf.org/doc/html/rfc5109" \t "https://zhuanlan.zhihu.com/p/_blank)

### **ULP FEC原理**

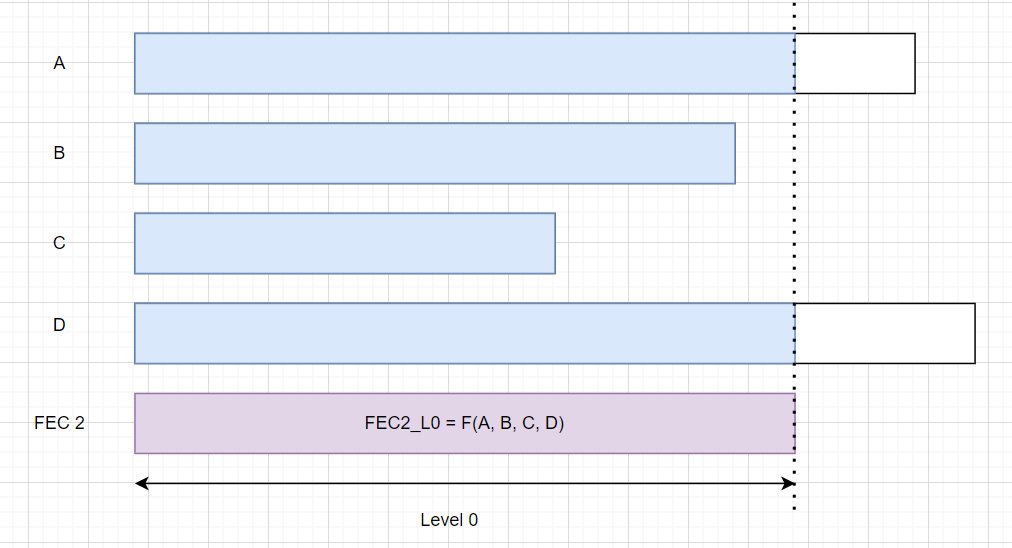
这里的假设是非常美好的，认为某些数据的前面部分更重要（如[I帧](https://zhida.zhihu.com/search?content_id=222282440&content_type=Article&match_order=1&q=I%E5%B8%A7&zhida_source=entity" \t "https://zhuanlan.zhihu.com/p/_blank)的SPS、PPS），因此赋予更高的保护权重，在带宽不足的情况下可以只保护更重要的数据。如下图：

* Level 0保护两个数据，只保护前面部分重要数据，如FEC1只有 level0，保护了A、B的前部分数据；FEC2的level 0保护了C、D的前部分数据。
* Level 1保护4个数据（保护强度不如level 0），如FEC2的level 1保护了A、B、C、D的后面部分数据。



ULP

实际使用中，我们用的更多的是只有一个level的情况，只保护一个报文前面的数据在WebRTC里面不是那么有效：



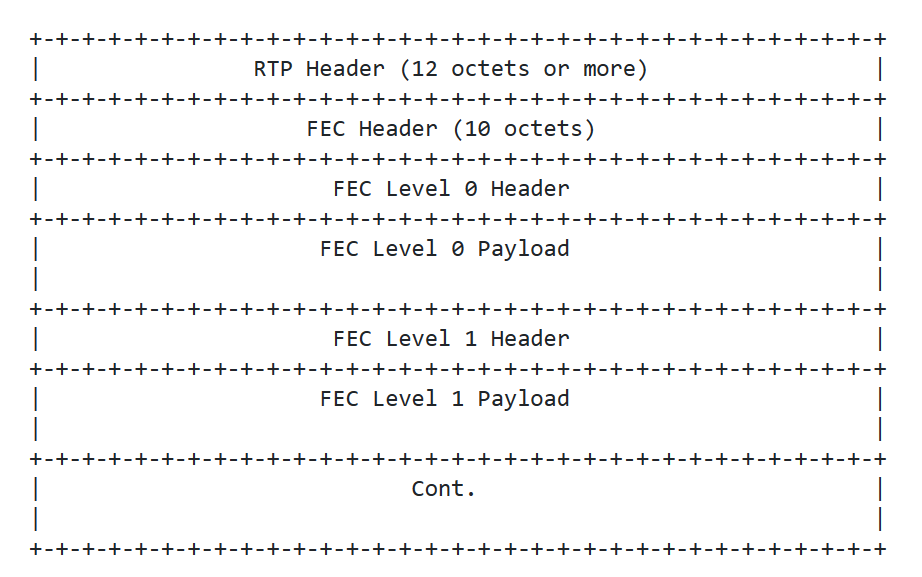
ULP只有一个level

### **ULP FEC的RTP**

基于以上的介绍，我们可以思考下如何基于RTP通道承载FEC数据。对于ULP FEC来说，编码基本参数较少，都是通过带内方式传输。

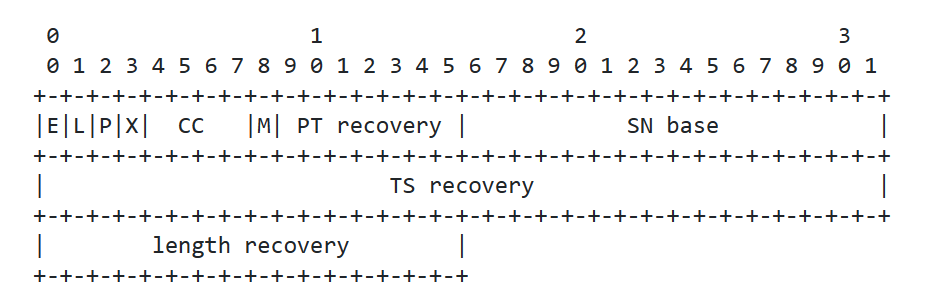
一个通用的ULP FEC的RTP报文格式如下，主要包含了：

* RTP头部（RTP报文必须）
* FEC头部（FEC level 0~level n共用的信息）
* FEC level n的头部+FEC level n的payload



RTP format of ULP FEC

一个level n通用的信息，我们会放到FEC头部。如RTP头部信息、报文长度、timestamp、保护的sequence base等。  
我们知道，不同的原始包，其RTP头部不同、timestamp和length都不同，如果所有包的数据都放到原原本本地放到这里，那么会浪费带宽。所以，这里存储的是一个经过FEC编码处理的数据，所以下面命名都以xxx recovery形式。如P（padding）、X、CC（CSRC Count）、M（Mark）、PT、timestamp、[报文长度](https://zhida.zhihu.com/search?content_id=222282440&content_type=Article&match_order=2&q=%E6%8A%A5%E6%96%87%E9%95%BF%E5%BA%A6&zhida_source=entity" \t "https://zhuanlan.zhihu.com/p/_blank)，都是经过FEC处理的，如XOR。  
如果有一个原始包丢失，只要其他的包收到了，我们可以根据这些recovery字段（XOR结果）以及收到的包恢复原始数据。

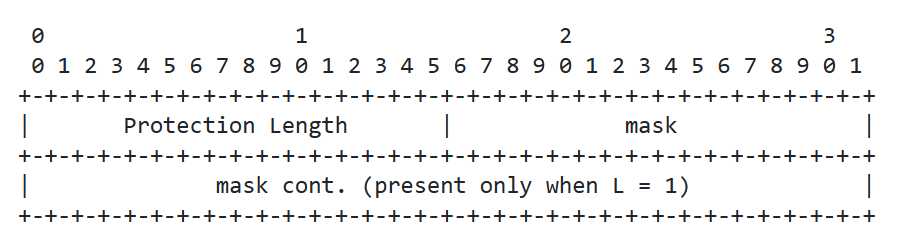


Header of ULP FEC

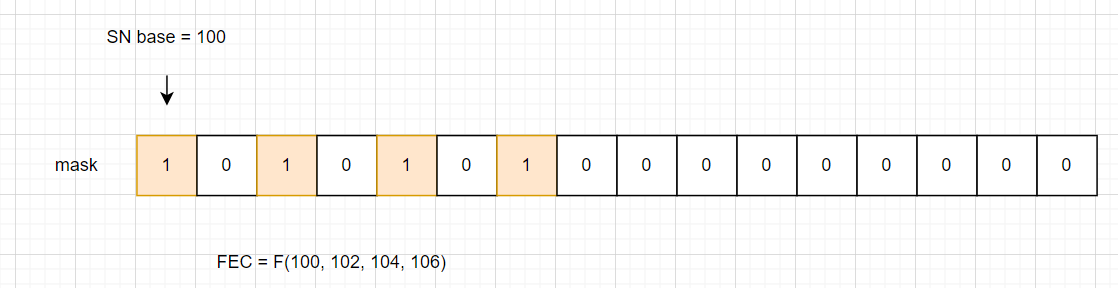
另外，这里的E是extension标志，用于扩展，我们暂不讨论。L（long），标识保护的报文是否需要更多：

* L=0，默认。最多16保护16个原始数据
* L=1，表示最多保护48个原始数据，默认为0的目的在于在level header中减少传输字节数

不同的保护level，也有各自不同的头部，主要是其level保护的长度，以及其保护的mask。在FEC头部已经包含了SN base，结合mask就可以得到是保护的那几个包。（FEC头部中L=0，则mask为16bits；当L=1，mask为48bits）。



FEC level头部的mask和FEC头部的SN base可以得知当前FEC保护了哪些原始包：



ULP FEC一般媒体包不改变，FEC和媒体包使用相同的SSRC和sequence number。但是他们的PT不同，可以根据PT，让不支持FEC的设备丢弃FEC包即可（当然还要得益于系统码的优良特性）。

FEC的发送有两种方式：

* 独立的一路流，即和原始媒体包有不同的PT。
* 通过增加RED方式multiplexing

独立的流传输实际上也是可以，但是我们需要知道它绑定了哪条流。一个可行的结果是fec信息放到application里面，这样按照当前的标准一定能知道ulpfec属于哪路流：

v=0

o=adam 289083124 289083124 IN IP4 [host.example.com](http://host.example.com/)

s=ULP FEC Seminar

t=0 0

c=IN IP4 224.2.17.12/127

a=group:FEC 1 2

a=group:FEC 3 4

m=audio 30000 RTP/AVP 0

a=mid:1

m=application 30002 RTP/AVP 100

a=rtpmap:100 ulpfec/8000

a=mid:2

m=video 30004 RTP/AVP 31

a=mid:3

m=application 30004 RTP/AVP 101

c=IN IP4 224.2.17.13/127

a=rtpmap:101 ulpfec/8000

a=mid:4

而使用RED方式，就比较简单，这里使用RED承载ULPFEC，RED里面包含了PCM（0）、DVI（5）、ULPFEC（100）：

m=audio 12345 RTP/AVP 121 0 5 100

a=rtpmap:121 red/8000/1

a=rtpmap:100 ulpfec/8000

a=fmtp:121 0/5/100

WebRTC实现，其FEC包和[媒体包](https://zhida.zhihu.com/search?content_id=222282440&content_type=Article&match_order=4&q=%E5%AA%92%E4%BD%93%E5%8C%85&zhida_source=entity" \t "https://zhuanlan.zhihu.com/p/_blank)会打上一个RED头部，参考[RFC 2198: RTP Payload for Redundant Audio Data (rfc-editor.org)](https://link.zhihu.com/?target=https://www.rfc-editor.org/rfc/rfc2198" \t "https://zhuanlan.zhihu.com/p/_blank)的实现。因此，为什么要加RED，还是因为有要遵守标准。如果自己写个RFC然后大家都能认（如RFC2198本身，它就增加了一个RED PT，并且绑定到m行），就没必要这么麻烦了。

ULP FEC有个缺陷，它和原始包使用相同的RTP SSRC以及sequence number，这就导致了FEC丢包导致视频组帧问题。对于不支持Generic Frame Description的[视频编码](https://zhida.zhihu.com/search?content_id=222282440&content_type=Article&match_order=1&q=%E8%A7%86%E9%A2%91%E7%BC%96%E7%A0%81&zhida_source=entity" \t "https://zhuanlan.zhihu.com/p/_blank)来说，需要通过sequence number和timestamp的连续性来判断组帧，ULP FEC丢包，我们不知道是丢的原始包还是FEC包！

关于ULP FEC的组帧问题，可以看下这个[issue讨论](https://link.zhihu.com/?target=https://gitlab.freedesktop.org/gstreamer/gst-plugins-good/-/issues/581" \t "https://zhuanlan.zhihu.com/p/_blank)。

### **ULP的协商**

RFC5109注册了audio/ulpfec、video/ulpfec、text/ulpfec用于协商。SDP介绍相对简单，这里不多介绍，简单举个例子：

a=audio ...

...

a=rtpmap: 110 ulpfec/48000

...

### **2.2 Flex FEC - RFC 8627**

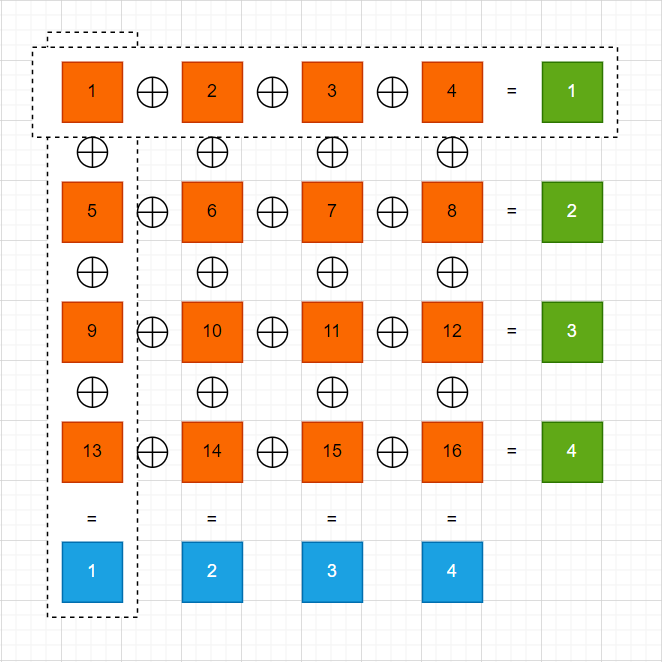
[!NOTE] Flex FEC是一个更灵活（flex）的方案，目前在WebRTC里面也是主要推荐的。  
[RFC 8627 - RTP Payload Format for Flexible Forward Error Correction (FEC) (ietf.org)](https://link.zhihu.com/?target=https://datatracker.ietf.org/doc/html/rfc8627" \t "https://zhuanlan.zhihu.com/p/_blank)

### **Flex FEC的三个FEC[编码方法](https://zhida.zhihu.com/search?content_id=222282440&content_type=Article&match_order=1&q=%E7%BC%96%E7%A0%81%E6%96%B9%E6%B3%95&zhida_source=entity" \t "https://zhuanlan.zhihu.com/p/_blank)**

之前的文章我们也提过，Flex FEC有四种FEC方法：

* 1D非交织（none-interleaved）保护
* 1D交织（interleaved）保护
* 2D（交织+非交织）保护
* flexible mask

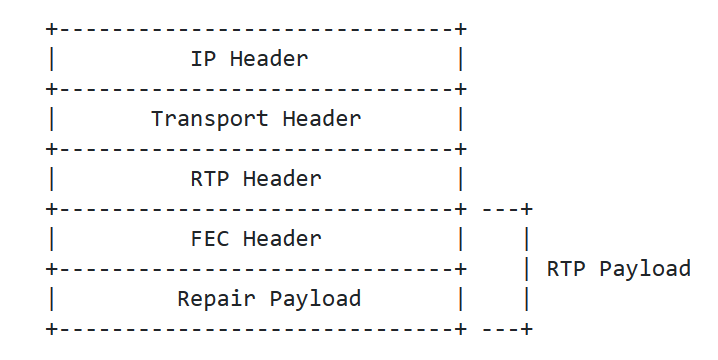
我们说的flex其实就是指保护的mask是否是灵活的。这里重新贴下之前的图，非flexible的mask，一般都是固定的，因此在传输一些[元信息](https://zhida.zhihu.com/search?content_id=222282440&content_type=Article&match_order=1&q=%E5%85%83%E4%BF%A1%E6%81%AF&zhida_source=entity" \t "https://zhuanlan.zhihu.com/p/_blank)时可以节省 一些字节。



### **Flex FEC的RTP**

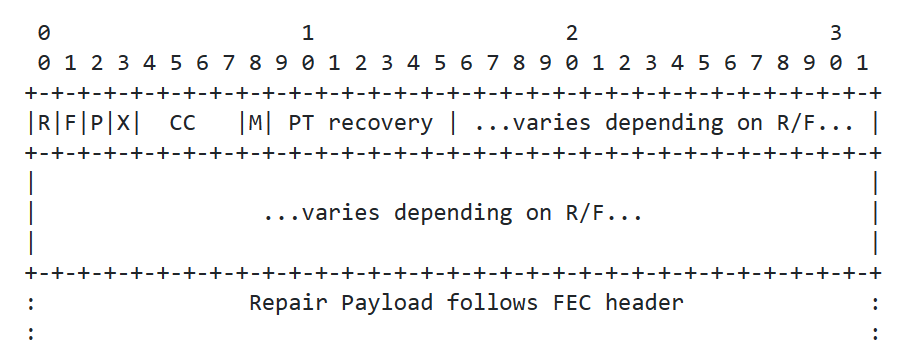
一个Flex FEC的RTP封装如下，主要包含：

* RTP头部
* Flex FEC头部
* FEC的payload



Format of FEC Repair Packets

一个Flex FEC头部如下：



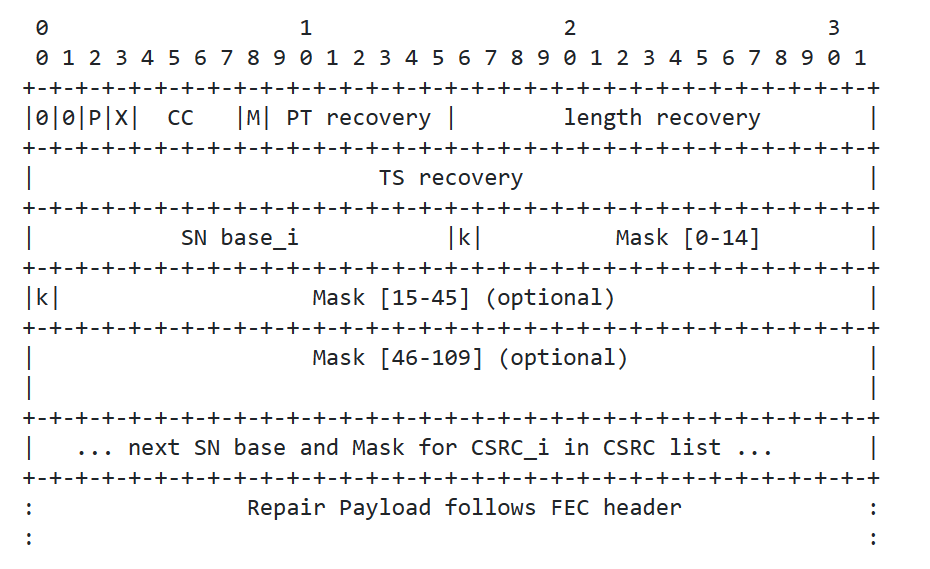
这里的P、X、CC、M、PT recovery和ULP FEC类似，这里不多解释。主要介绍下R和F，R和F的选择也会导致FEC header的内容有所不同。

* R表示retransmission，表示这个是否是原始包的重传。
* F表示fixed，0表示flexible，1表示fixed。上面提到的1D交织、非交织、2D都属于fixed。

因此有以下搭配：

|  |  |  |
| --- | --- | --- |
| R | F | 含义 |
| 0 | 0 | flexible，灵活的mask |
| 0 | 1 | fixed，rows和cols固定 |
| 1 | 0 | 原始包的重传 |
| 1 | 1 | 预留，暂时不用 |

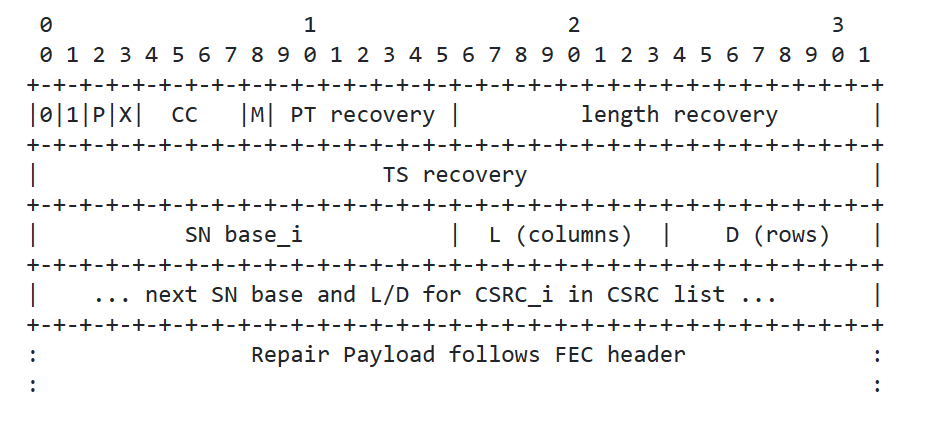
我们先看flexible mask，R=0，F=0，它的FEC头部完整内容：



FEC Header for F=0

有些字段上面已经介绍过，这里便不多介绍。SN base和ULP的一样，但是这里的mask最多可以支持到110位。mask[m-n]前面的一个bit K，表示当前的mask是否需要。通过这种方式就可以支持mask扩展。这个比ULP FEC的L位更好，能支持更长的mask。

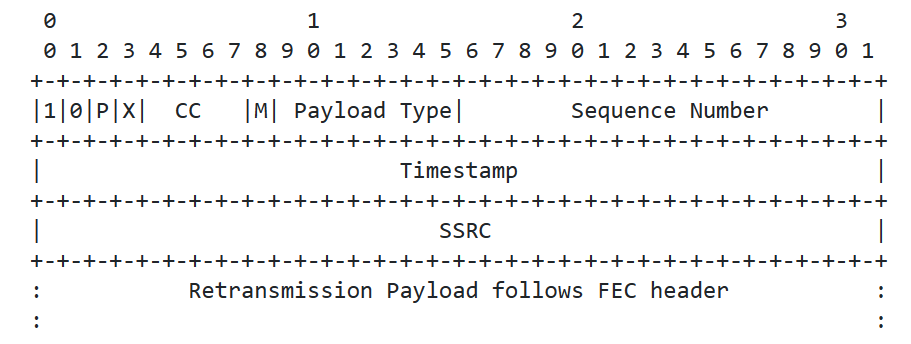
再看下Fixed mask，R=0，F=1，它的FEC完整内容：



FEC Header for F=1

这里不需要mask，只需要行和列个数。原理可以参考上面介绍的交织和非交织方式。

这个标准里还提供了一个重传的格式（R=1），可以选择使用，需要注的是这里的PT等都填的是原始包的信息，不是FEC处理后的：



对于Flex FEC来说，就不存在ULP FEC中的组帧问题。以为他的SSRC、sequence number都是独立的，OpenH264也不存在组帧问题。

### **Flex的协商**

这个标准里面注册了audio/flexfec、video/flexfec、text/flexfec、application/flexfec，SDP协商不多介绍，只举个例子：

a=audio ...

...

a=rtpmap: 110 flexfec/48000

...

### **2.3 RS FEC方案**

通过分析上面的ULP FEC和Flex FEC，其实我们已经知道一个FEC头部需要哪些东西了。ULP和Flex方式中，它们的[编码方案](https://zhida.zhihu.com/search?content_id=222282440&content_type=Article&match_order=1&q=%E7%BC%96%E7%A0%81%E6%96%B9%E6%A1%88&zhida_source=entity" \t "https://zhuanlan.zhihu.com/p/_blank)已经固定了，也就是说我们已经约定了发送端和接收端的编解码方法，只需要传输mask等就可以。

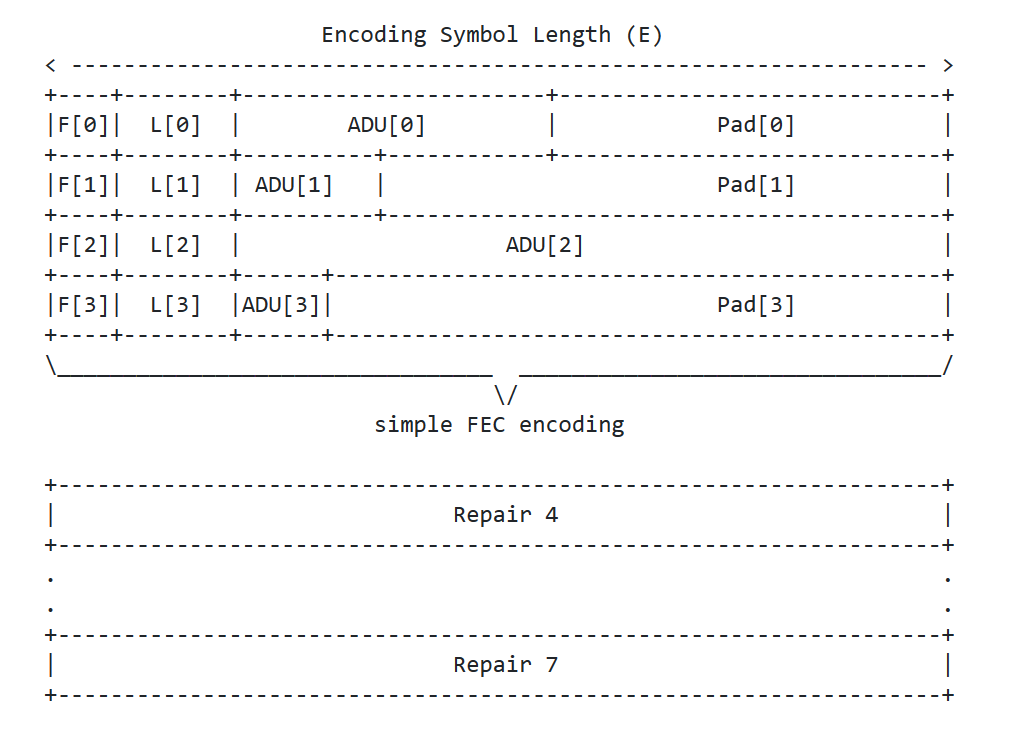
RS FEC的方案，可以类比Flex FEC的flexible方式，特殊的地方在于还需要收发双方：

* 约定RS FEC算法，是[范德蒙矩阵](https://zhida.zhihu.com/search?content_id=222282440&content_type=Article&match_order=1&q=%E8%8C%83%E5%BE%B7%E8%92%99%E7%9F%A9%E9%98%B5&zhida_source=entity" \t "https://zhuanlan.zhihu.com/p/_blank)还是柯西矩阵
* 矩阵生成方式是什么样的
* 有限域用的[本原多项式](https://zhida.zhihu.com/search?content_id=222282440&content_type=Article&match_order=1&q=%E6%9C%AC%E5%8E%9F%E5%A4%9A%E9%A1%B9%E5%BC%8F&zhida_source=entity" \t "https://zhuanlan.zhihu.com/p/_blank)是哪个

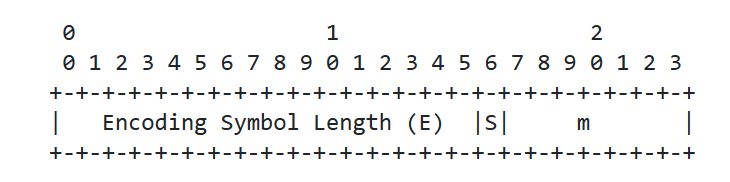
### **一个简单的RS FEC方案（非RTP）**

这里介绍一个简单的方案，希望大家有所启发：  
[RFC 6865 - Simple Reed-Solomon Forward Error Correction (FEC) Scheme for FECFRAME (ietf.org)](https://link.zhihu.com/?target=https://datatracker.ietf.org/doc/html/rfc6865" \t "https://zhuanlan.zhihu.com/p/_blank)[RFC 5510 - Reed-Solomon Forward Error Correction (FEC) Schemes (ietf.org)](https://link.zhihu.com/?target=https://datatracker.ietf.org/doc/rfc5510/" \t "https://zhuanlan.zhihu.com/p/_blank)

这个方案支持多路流的FEC方案，每个原始数据前面会增加上流ID和流长度信息。



对于这个方案里，收发双发需要知道编码信息（FSSI，FEC scheme-Specific Information），这里包括最大编码长度（如在IP网络中会被限制到MTU以内）、严格标志以及有限域大小m即GF(2^m)。

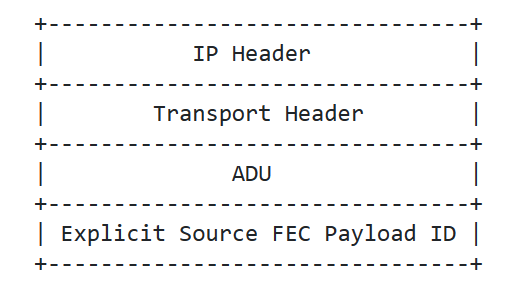


这个FSSI可以通过SDP协商：

a=fec-repair-flow: encoding-id=8; fssi=E:1400,S:0,m:8

这是一个比较标准的做法，但是实际使用中我们的参数可能永远不会变，实际上可能都不需要做这么复杂的协商。

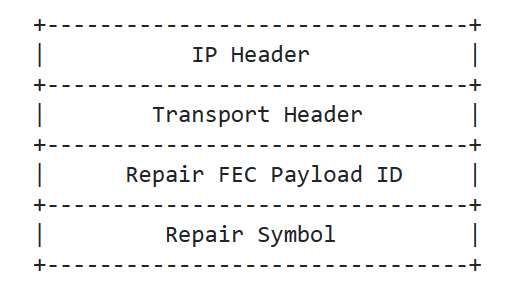
一个原始包的格式如下：



每个原始包后面需要带一个Source FEC payload ID（payload ID用于标识payload的信息，以及关于这个payload的FEC编码信息），包含：

* Source Block Number，类比RTP的sequence number
* Encode Symbol ID(ESI)，表示编码了多少个符号，对于GF(2^8)就是多少个字节。
* Source Block Length，k，编码数据长度。

一个FEC包的格式如下，注意它的payload ID是放在FEC payload（repaire symbol）前面：



总结起来：

* FEC编码基本信息（FSSI）通过SDP协商，主要包含编码最大长度、有限域大小
* 增加payload id概念，保存payload基本信息和FEC信息，payload id也需要

## **3. 总结**

以上便是关于FEC如何打包的介绍，希望对大家有所启发、能够看得懂代码、能够动手改造。篇幅先知，不能包含所有的知识点，详细的信息大家可以点击链接看看标准。