# 第1章 RLC

RLC层位于PDCP层和MAC层之间。它通过SAP（Service Access Point）与PDCP层进行通信，并通过逻辑信道与MAC层进行通信。每个UE的每个逻辑信道都有一个RLC实体（RLC entity）。RLC实体从PDCP层接收到的数据，或发往PDCP层的数据被称作RLC SDU（或PDCP PDU）。RLC实体从MAC层接收到的数据，或发往MAC层的数据被称作RLC PDU（或MAC SDU）。

RLC层主要负责（见36.322）：

* **分段/串联和重组RLC SDU**（concatenation/segmentation/reassembly，只适用于UM和AM模式）：RLC PDU的大小是由MAC层指定的，其大小通常并不等于RLC SDU的大小，所以在发送端需要分段/串联RLC SDU以便其匹配MAC层指定的大小。相应地，在接收端需要对之前分段的RLC SDU进行重组，以便恢复出原来的RLC SDU并按序递送（in-sequence delivery）给上层。
* 通过**ARQ**来进行纠错（只适用于AM模式）：MAC层的HARQ机制的目标在于实现非常快速的重传，其反馈出错率大概在1%左右。对于某些业务，如TCP传输（要求丢包率小于），HARQ反馈的出错率就显得过高了。对于这类业务， RLC层的重传处理能够进一步降低反馈出错率。
* 对RLC data PDU进行**重排序**（reordering，只适用于UM和AM模式）：MAC层的HARQ操作可能导致到达RLC层的报文是乱序的，所以需要RLC层对数据进行重排序。重排序是根据序列号（Sequence Number，SN）的先后顺序对RLC data PDU进行排序的。
* **复包检重测**（duplicate detection，只适用于UM和AM模式）：出现重复包的最大可能性为发送端反馈了HARQ ACK，但接收端错误地将其解释为NACK，从而导致了不必要的MAC PDU重传。
* 对RLC data PDU进行**重分段**（resegmentation，只适用于AM模式）：当RLC data PDU（注意：这里不是SDU）需要重传时，可能需要进行重分段。例如，当MAC层指定的大小小于需要重传的原始RLC data PDU的大小时，就需要对原始RLC data PDU进行重分段。

按序递送（in-sequence delivery）指的是RLC实体的接收端必须按序将重组（reassembly）好的SDU发送给PDCP层，也就是说，SDU n必须在SDU n+1之前发送给PDCP层。其基本思想是将接收到的RLC PDU（假设其SN = x）放在接收buffer（reception buffer）中，直到较小SN（小于x）的所有PDU都已成功接收并递送给PDCP层。只有当拥有较小SN的所有RLC PDU都用于重组SDU后，下一个RLC PDU才会被使用。例如对于类似VoLTE的流应用，要求接收到的数据的顺序与它们被发送时的顺序是一致的，否则可能造成声音的紊乱。

RLC层的功能是由RLC实体来实现的。一个RLC实体可以配置成以下3种模式之一：

* Transparent Mode（TM）：对应TM RLC实体，简称TM实体。该模式可以认为是空的RLC，因为这种模式下只提供数据的透传（pass through）功能。
* Unacknowledged Mode（UM）：对应UM RLC实体，简称UM实体。该模式提供除重传和重分段外的所有RLC功能，因此提供了一种不可靠的传输服务。
* Acknowledged Mode（AM）：对应AM RLC实体，简称AM实体。通过出错检测和重传，AM模式提供了一种可靠的传输服务。该模式提供了所有的RLC功能。

除TM模式对应的逻辑信道外，每个逻辑信道对应的RLC实体的模式（在UM模式和AM模式之间进行选择）是在无线承载建立时，eNodeB通过相关RRC消息的RLC-Config字段来配置的。在36.331中，搜索“RLC-SAP”，能看到各种RRC消息所使用的RLC模式（以及SRB）。

每种模式支持的RLC层功能见表1-1（见36.322的4.4节）。

表1-1：每种模式支持的RLC功能

|  |  |  |  |
| --- | --- | --- | --- |
| **RLC功能** | **TM** | **UM** | **AM** |
| 传输上层PDU | Yes | Yes | Yes |
| 使用ARQ进行纠错 | No | No | Yes |
| 对RLC SDU进行分段、串联和重组 | No | Yes | Yes |
| 对RLC data PDU进行重分段 | No | No | Yes |
| 对RLC data PDU进行重排序 | No | Yes | Yes |
| 重复包检测 | No | Yes | Yes |
| RLC SDU丢弃处理 | No | Yes | Yes |
| RLC重建 | Yes | Yes | Yes |
| 协议错误检测 | No | No | Yes |

## 1.1 TM模式



Figure 4.2.1.1.1-1: Model of two transparent mode peer entities

TM模式下，RLC实体只进行透传，不对RLC SDU进行分段和串联，也不添加任何头部信息。TM模式通过逻辑信道BCCH、PCCH和DL/UL CCCH来接收/发送RLC PDU。

在eNodeB或UE侧，一个TM实体只能接收或发送数据，而不能同时收发数据，即TM实体只提供单向的数据传输服务。

从36.322的Figure 4.2.1.1.1-1可以看出，在发送端，一个TM实体只由一个保存RLC SDU的传输buffer组成。当MAC层告诉该TM实体有一个传输机会时，TM实体会将传输buffer中的1个RLC SDU直接发送给MAC层，而不做任何修改。在接收端，一个TM实体直接将从MAC层接收到的RLC PDU发送给PDCP层。

对于TM实体来说，一个RLC SDU等同于一个RLC PDU。协议中，TM实体传输的PDU被称为TMD PDU。TMD PDU只由一个Data域组成，并不包含任何RLC头部。见36.322的Figure 6.2.1.2-1。



Figure 6.2.1.2-1: TMD PDU

只有那些无需RLC配置的RRC消息会使用TM模式，如系统消息SI、Paging消息以及使用SRB0的RRC消息。DRB不支持TM模式，只支持UM模式或AM模式。

## 1.2 UM模式

UM模式不提供重传和重分段功能，其提供的是一种不可靠的服务。UM模式常用于实时性要求较高的业务，如VoIP等，这种业务允许有一定的错包或丢包，但对延迟较为敏感，同时要求按序传输，并丢弃重复报文。点到多点的服务，如MBMS，由于没有可用的反馈路径而不能使用AM模式，也使用UM模式。

UM模式通过逻辑信道DL/UL DTCH、MCCH或MTCH来接收/发送RLC PDU。

与TM模式类似，一个UM实体只能接收或发送数据，而不能同时收发数据。UM实体只提供单向的数据传输服务。



Figure 4.2.1.2.1-1: Model of two unacknowledged mode peer entities

### 1.2.1 发送端

UM实体在发送端需要做2件事：（1）将来自上层（PDCP层）的RLC SDU缓存在传输buffer（transmission buffer）中；（2）在MAC层通知其发送RLC PDU时，分段/串联RLC SDU以生成RLC PDU，并赋予合适的SN值，然后将生成的RLC PDU发给MAC层。

UM实体在发送端会维护如下变量：

* VT(US)：该变量保存了下一个新生成的UMD PDU将被赋予的SN值。该变量初始化时为0，并在UM实体发送一个SN = VT(US)的UMD PDU给MAC层时更新。简单地说，“VT(US) – 1”等于最近一个已经发送的UMD PDU的SN值。

**1.2.1.1 分段和串联**

UM实体发送RLC PDU的前提条件是：MAC层通知UM实体发送一个RLC PDU，即通知UM实体有一个传输机会。MAC层同时会告诉UM实体在这次传输机会中，可以传输的RLC PDU的总大小。

由MAC层指定的RLC PDU的大小通常并不等于RLC SDU的大小，所以在发送端需要分段/串联RLC SDU以便生成的RLC PDU匹配MAC层指定的大小。

生成一个RLC PDU时，是按照RLC SDU到达UM实体的顺序来逐个将RLC SDU放入RLC PDU中的。





图1-1：UM分段和串联功能举例

图1-1是UM分段和串联功能的一个例子。UM实体的传输buffer里有7个RLC SDU，对应SDU x至SDU x+6。当MAC层通知该UM实体发送一个RLC PDU，并指定该PDU的大小时，UM实体发现该大小只能容纳整个SDU x和部分SDU x+1（以及生成的RLC header）。此时UM实体会将SDU x+1分段，并将分段的前半部分与SDU x串联起来，再加上相应的RLC header后，生成SN = n的RLC PDU，并将其发给MAC层。

在MAC层通知的下一个传输机会里，UM实体发现MAC层指定的大小只能容纳下SDU x+1的后半部分、整个SDU x+2以及部分SDU x+3（以及生成的RLC header）。此时该实体会将SDU x+3分段，并将SDU x+1的后半部分、SDU x+2以及SDU x+3的前半部分串联起来，再加上相应的RLC header后，生成SN = n+1的RLC PDU，并将其发给MAC层。

在接下来的传输机会里，UM实体发现MAC层指定的大小正好能容纳下SDU x+3的后半部分及其相应的RLC header。此时该实体会将SDU x+3的后半部分加上相应的RLC header后，生成SN = n+2的RLC PDU，并发给MAC层。

在生成SN = n+3的PDU时，MAC层指定的大小正好能容纳下SDU x+4及其相应的RLC header。此时无需分段，也不需要串联，将SDU x+4加上相应的RLC header后，生成SN = n+3的RLC PDU，然后将其发给MAC层即可。

在接下来的传输机会里，MAC层指定的大小正好能容纳下SDU x+5、SDU x+6及其相应的RLC header。此时无需分段，只需要将SDU x+5和SDU x+6串联起来，加上相应的RLC header后，生成的SN = n+4的RLC PDU，然后将其发给MAC层即可。

**1.2.1.2 UMD PDU**

结合图1-1的例子，我们进一步介绍UMD PDU的结构及其相关字段的作用。

UMD PDU由2部分组成：Data域和header（头部）。

Data域由一个或多个“Data field element”组成。一个“Data field element”对应一个RLC SDU或一个RLC SDU分段。Data field element是按照RLC SDU到达RLC实体的先后顺序映射到Data域的。例如：图1中SN = n的RLC PDU的Data域包含了2个“Data field element”，分别对应SDU x和SDU x+1的分段。SN = n+1的RLC PDU的Data域包含了3个“Data field element”，分别对应SDU x+1的分段、SDU x+2和SDU x+3的分段。

一个RLC PDU的Data域按照顺序由“0个或1个SDU分段 + 0个或多个SDU + 0个或1个SDU分段”组成。也就是说，SDU分段只可能出现在Data域的最开始或者最后。

header由固定部分（每个PDU都有）和可能存在的扩展部分组成。

固定部分（fixed part）由一个FI（2比特）、一个E（1比特）和一个SN（5比特或10比特）字段组成，其本身是字节对齐（byte-aligned）的。

SN唯一指定了一个UMD PDU。SN的长度由RRC层下发的*sn-FieldLength*字段指定，其值可以为5比特，也可以为10比特。如果使用5比特的SN，则固定部分长为1字节；如果使用10比特的SN，则固定部分长为2字节（多出来的3比特是预留的，用R1表示）。

扩展部分（extension part）由1个或多个“E（1比特） + LI（11比特）”组成。只有当Data field element的个数多于1个时，才存在扩展部分（extension part）。除了最后一个Data field element外，其它Data field element都有一个对应的“E + LI”。如果LI的个数为奇数，则最后一个LI之后需要添加4比特的padding。可以看出，扩展部分中“E + LI”的个数等于Data field element的个数减去1。

关于UMD PDU的结构，可参见36.322的6.2.1.3节。（或见下图。图中最左边的比特对应MSB，最右边的比特对应LSB）



Figure 6.2.1.3-1: UMD PDU with 5 bit SN (No LI)



Figure 6.2.1.3-2: UMD PDU with 10 bit SN (No LI)



Figure 6.2.1.3-3: UMD PDU with 5 bit SN (Odd number of LIs, i.e. K = 1, 3, 5, …)



Figure 6.2.1.3-4: UMD PDU with 5 bit SN (Even number of LIs, i.e. K = 2, 4, 6, …)



Figure 6.2.1.3-5: UMD PDU with 10 bit SN (Odd number of LIs, i.e. K = 1, 3, 5, …)



Figure 6.2.1.3-6: UMD PDU with 10 bit SN (Even number of LIs, i.e. K = 2, 4, 6, …)

E是Extension bit的意思，该字段用于指示“固定部分”或“E + LI”之后紧接着的是Data域，还是一个“E + LI”。值为0表示之后紧跟着Data域，值为1表示之后紧跟着一个“E + LI”。

LI是Length Indicator的意思，该字段用于指示对应的Data field element（SDU或SDU分段）的长度（以字节为单位）。

FI是Framing Info的意思，该字段用于指示在Data域的开始或结束位置的Data field element是不是一个RLC SDU分段。该字段长为2比特，高比特位表示Data域的第一个字节是否是一个RLC SDU的第一个字节，低比特位表示Data域的最后一个字节是否是一个RLC SDU的最后一个字节（0对应“是”，1对应“否”）。这2比特可以分别对应不同的RLC SDU，也可以对应同一RLC SDU。

图1-1假设使用5比特的SN域。SN = n的RLC PDU由2个Data field element（SDU x和SDU x+1的分段）组成，扩展部分包含1个“E + LI”，因此固定部分的E设置为1，表示固定部分之后紧接着1个“E + LI”。扩展部分的“E + LI”中的E值为0，表示其后紧接着Data域，LI（对应LI1）指示了Data域中SDU x的长度。由于Data域的第一个字节对应SDU x的第一个字节，Data域的最后一个字节并不对应SDU x+1的最后一个字节，所以FI的值设置为01。

SN = n+1的RLC PDU由3个Data field element（SDU x+1的分段、SDU x+2和SDU x+3的分段）组成，扩展部分包含2个“E + LI”，因此固定部分的E设置为1，表示固定部分之后紧接着1个“E + LI”。扩展部分的第一个“E + LI”中的E值为1，表示其后紧接着1个“E + LI”，LI（对应LI1）指示了Data域中SDU x+1的分段的长度。扩展部分的第二个“E + LI”中的E值为0，表示其后紧接着Data域，LI（对应LI2）指示了Data域中SDU x+2的长度。由于Data域的第一个字节并不对应SDU x+1的第一个字节，Data域的最后一个字节也不对应SDU x+3的最后一个字节，所以FI的值设置为11。

SN = n+2的RLC PDU由1个Data field element（SDU x+3的分段）组成，扩展部分包含0个“E + LI”，因此固定部分的E设置为0，表示固定部分之后紧接着Data域。由于Data域的第一个字节并不对应SDU x+3的第一个字节，Data域的最后一个字节对应SDU x+3的最后一个字节，所以FI的值设置为10。

接着对比SN = n+3和SN = n+4的PDU可以看出，虽然二者的FI都设置为00，但SN = n+3的PDU的FI字段的2比特对应的是同一SDU，而SN = n+4的PDU的FI字段的2比特分别对应不同的SDU。

包括UMD PDU以及后面将介绍到的AMD PDU、AMD PDU segment和STATUS PDU在内，其对应的RLC PDU总是字节对齐的，并且PDU的最后不存在padding。

从上面的介绍可以看出，RLC header里并不指定Data域中最后一个Data field element的大小。这是因为RLC PDU的长度是由MAC指定的，该长度会在MAC PDU中对应该RLC PDU的subheader中的L字段中体现（见36.321的6.1.2节和6.2.1节）。“RLC PDU的长度 - RLC PDU header的长度 - 所有LI之和”即为最后一个Data field element的长度。

当UM实体要发送一个新的UMD PDU给MAC层时，它会将该PDU的SN设置成VT(US)，然后将VT(US)加1。

### 1.2.2 接收端

UM实体在接收端主要做几件事：（1）对分段的RLC SDU进行重组（reassembly），以便恢复出原来的RLC SDU并发往PDCP层；（2）对RLC PDU进行重排序（reordering）；（3）检测并丢弃重复包（duplicate detection）。

如果接收端收到的RLC PDU是乱序的，则需要先进行重排序。由于MAC层使用多个HARQ process来处理HARQ，因此乱序到达是不可避免的（可参见11.2节的介绍）。乱序到达的RLC PDU会先保存在接收buffer中，直到之前的RLC PDU都已成功接收并递送给PDCP层。（同样适用于AM模式）

如图1-2所示，MAC层的HARQ处理导致了RLC PDU以 PDU 8、PDU 6和PDU 9的顺序到达接收端的RLC层。此时接收端会对接收到的PDU进行重排序处理，并将PDU以PDU 6、PDU 8和PDU 9的顺序保存在接收buffer中。

在重排序期间，通过校验收到的RLC PDU的SN值，接收端可以知道是否收到了重复包。重复包将被丢弃，从而保证上层不会收到重复的数据。（同样适用于AM模式）

接收端需要检测MAC层是否丢失了某个RLC PDU，并避免过度的重排序延迟。简单地说，接收端只会等待还未收到的RLC PDU一段时间，等不到就不等了。重排序定时器*t\_Reordering*决定了在多长时间内等待一个还未收到的PDU。每个UM实体只有一个*t\_Reordering*。使用该定时器的目的是为了检测MAC层是否丢失了某个RLC PDU，如果在*t\_Reordering*指定的时间内没有收到该PDU，则接收端认为该PDU已经丢失了，且UM实体不会再去尝试接收这些已经丢失了的PDU。（同样适用于AM模式。但不同的是，AM模式会要求对端重传丢失了的PDU，而不是直接丢弃）

只有当一个RLC SDU的所有分段都存在于接收buffer里时，才能从保存的RLC PDU中重组出该SDU。只要有部分分段没有被接收到，该RLC SDU的所有数据都会被丢弃。也就是说，如果某个RLC SDU的一个或几个分段所在的RLC PDU丢失了，而导致包含其某个分段的另一个RLC PDU无法重组出该RLC SDU，则收到的RLC SDU分段将被丢弃（但使用收到的PDU成功重组出的其它SDU要发送给PDCP层）。（同样适用于AM模式。但不同的是，AM模式会要求对端重传丢失了的PDU）



图1-2：UM模式中，PDU丢失检测及处理

如图1-2所示，当接收端收到PDU 8时，会启动定时器*t\_Reordering*。由于在该定时器超时之前，PDU 7还没收到，因此接收端认为PDU 7丢失了。接收端在重组RLC SDU时，由于PDU 7的丢失导致了SDU 16和SDU 17的部分分段丢失，从而无法重组出完整的SDU 16和SDU 17，因此SDU 16和SDU 17的已接收分段将被丢弃。SDU 15和SDU 18是完全接收的，所以会被递送给PDCP层。SDU 19的已接收分段会继续放在接收buffer中，直到接收到SDU 19的所有分段（最终也有可能被丢弃）。

重排序和重复包检测功能并不会应用在使用MCCH或MTCH的UM实体上，因为这些信道在MAC层并不使用HARQ操作。

**1.2.2.1 接收端相关变量**

在详细介绍UM模式接收端处理流程之前，我们需要先介绍接收端的相关变量。UM实体在接收端会维护如下变量：

VR(UR)： UE接收状态变量（UM receive state variable）。该变量保存了等待重排序的最早一个UMD PDU的SN值。该变量初始化时为0。接收端认为SN小于VR(UR)的UMD PDU都已被成功接收（即使没有成功接收，也认为小于VR(UR)的UMD PDU已经丢失而不再去接收了），VR(UR)对应重排序窗口内还未接收到的拥有最小SN的UMD PDU。

VR(UX)：UM *t-Reordering*状态变量（UM *t-Reordering* state variable）。该变量保存了触发*t-Reordering*的UMD PDU的SN值的下一个SN。当启动*t-Reordering*（同时会更新VR(UX)）时，说明有小于VR(UX)的UMD PDU还未接收到，此时需要等待这些UMD PDU以便进行重排序。

VR(UH)：UM最高接收状态变量（UM highest receiving state variable）。该变量保存的SN值等于所有已经接收到的UMD PDU中，拥有最高SN的那个UMD PDU的SN值，再加1。该变量初始化时为0。该变量对应重排序窗口的上边界（不包含VR(UH)）。简单地说，“VR(UH) – 1”等于已接收的拥有最高SN的UMD PDU的SN值。

Reordering window：重排序窗口。如果一个SN满足(VR(UH) – UM\_Window\_Size) ≤ SN < VR(UH)，则该SN位于重排序窗口内；否则该SN位于重排序窗口外。UM RLC接收实体不会去接收“SN < (VR(UH) – UM\_Window\_Size)”的UMD PDU，也就是说，接收端认为SN小于重排序窗口下边界的UMD PDU都已经成功接收。重排序窗口指定了在不向前移动该窗口的前提下，能够接收的PDU的数量。如果重排序窗口向前移动，则位于窗口之外的任一PDU，不管其状态如何，都需要进行重组，并将生成的SDU按序发往PDCP层。

UM\_Window\_Size：对应重排序窗口的大小。当使用5比特SN时，UM\_Window\_Size = 16；当使用10比特SN时，UM\_Window\_Size = 512；当UM RLC接收实体用于MCCH或MTCH时，UM\_Window\_Size = 0。

图1-3是除UR(X)外的UM接收端相关变量的一个例子。



图1-3：除VR(UX)外的UM接收端相关变量

**1.2.2.2 丢弃处理**

当UM实体从MAC层收到一个UMD PDU时，它会判断是丢弃该PDU还是将其放入接收buffer中。

当接收端收到一个SN = x的UMD PDU时，在以下3种情况下该PDU会被丢弃：

**情况1**：如果VR(UR) < x < VR(UH)，且之前已经成功接收到SN = x的UMD PDU，则该PDU是重复包，予以丢弃。如图1-4的case 1。

**情况2**：由于接收端认为SN < VR(UR)的PDU都已经成功接收，所以当(VR(UH) – UM\_Window\_Size) ≤ x < VR(UR)时，该PDU也会被丢弃。如图1-4的case 2。

**情况3**：接收端不会去接收重排序窗口之外，且SN < (VR(UH) – UM\_Window\_Size)的UMD PDU，所以x < (VR(UH) – UM\_Window\_Size)的PDU将被丢弃。如图1-4的case 3。



图1-4：将被丢弃的UMD PDU举例

而在其它情况下，接收端会将UMD PDU放入接收buffer中，并按照下一节的介绍进行处理。

**1.2.2.3 接收buffer中的UMD PDU的处理**

当一个SN = x的UMD PDU放入接收buffer时，接收端会按如下步骤进行操作。

**步骤1**：如果x位于重排序窗口之外（此时x一定是超出了重排序窗口的上边界，即x ≥ VR(UH)），

* 将VR(UH)设置成x + 1。
* 由于VR(UH)的更新，重排序窗口也将相应地向前移动，此时必定会有一些UMD PDU移到了重排序窗口之外。因此需要对重排序窗口之外的UMD PDU进行重组，移除RLC header，并将重组后的还未曾递送过的RLC SDU按SN递增的顺序递送给PDCP层。
* 如果此时VR(UR)也移到了重排序窗口之外，则将VR(UR)设置成(VR(UH) – UM\_Window\_Size)，即设置成重排序窗口的下边界。其中一种场景发生在接收端一直没有收到SN = VR(UR)的UMD PDU，但不断地收到新的位于重排序窗口上边界之外的UMD PDU，从而VR(UH)不断前移，直至VR(UR)落到重排序窗口之外。

**步骤2**：如果在接收buffer中包含了一个SN = VR(UR)的UMD PDU，则

* 将VR(UR)更新成“SN > 当前VR(UR)”且还未接收到的第一个UMD PDU的SN值。可以认为“SN < 更新后的VR(UR)”的UMD PDU都已成功接收；
* 对“SN < 更新后的VR(UR)”的UMD PDU进行重组，移除RLC header，并将重组后的还未曾递送过的RLC SDU按SN递增的顺序递送给PDCP层。



图1-5：步骤1及步骤2举例

如图1-5所示，接收buffer的情况以及相关变量的取值如(a)所示。此时接收端收到一个位于重排序窗口之外的SN = n+x+2 > VR(UH) = n+x+1的UMD PDU，因此VR(UH)从n+x+1更新为n+x+3，并导致SN = n-3和SN = n-2的UMD PDU移到了重排序窗口之外。接收端需要对SN = n-3和SN = n-2的UMD PDU（可能还包括一些SN < n-2的PDU）进行重组，移除RLC header，并将重组后的还未曾递送过的RLC SDU按SN递增的顺序递送给PDCP层。此时VR(UR)并未在重排序窗口外，且在接收buffer中并不存在一个SN = VR(UR)的UMD PDU，所以无需更新VR(UR)。此时的变量取值见图1-5的(b)。

接下来，接收端又收到一个位于重排序窗口之外的SN = n+x+4 > VR(UH) = n+x+3的UMD PDU，此时VR(UH)从n+x+3更新为n+x+5，并导致SN = n-1和SN = n的UMD PDU移到了重排序窗口之外。在步骤1中，接收端需要对SN = n-1和SN = n的UMD PDU（可能还包括一些SN < n-1的PDU）进行重组，移除RLC header，并将重组后的还未曾递送过的RLC SDU按SN递增的顺序递送给PDCP层。由于此时VR(UR)被移到了重排序窗口外，所以将VR(UR)设置成(VR(UH) – UM\_Window\_Size)，即n+1。在步骤2中，接收端发现接收buffer中有一个SN = VR(UR) = n+1的UMD PDU，因此将VR(UR)更新成“SN > 当前VR(UR) = n+1”，但还未接收到的第一个UMD PDU的SN值，即n+2，并对SN < n+2的UMD PDU进行重组，移除RLC header，并将重组后的还未曾递送过的RLC SDU按SN递增的顺序递送给PDCP层。此时变量的取值见图1-5的(c)。

**步骤3**：如果*t-Reordering*正在运行，

* 如果VR(UX) <= VR(UR)，或者
* 如果VR(UX)落在了重排序窗口之外并且VR(UX)不等于VR(UH)
  + 停止并重置*t-Reordering*。

*t-Reordering*正在运行，意味着有SN < VR(UX)的UMD PDU还未接收到。新收到SN = x的UMD PDU后，接收端需要重新判断是否需要继续运行*t-Reordering*：

* 接收端认为小于VR(UR)的UMD PDU都已接收到，这也意味着VR(UX) <= VR(UR)时，小于VR(UX)的UMD PDU都已接收到，所以此时没有运行*t-Reordering*的必要了。
* 由于VR(UX)不可能大于VR(UH)，所以当VR(UX)落在了重排序窗口之外并且VR(UX)不等于VR(UH)时，VR(UX)必定落在了重排序窗口的下边界之外。而UM实体不会去接收“SN < (VR(UH) – UM\_Window\_Size)”的UMD PDU，所以此时也没有运行*t-Reordering*的必要了。

**步骤4**：如果*t-Reordering*没有运行（包括步骤3介绍的场景导致的停止运行），并且VR(UH) > VR(UR)，则启动*t-Reordering*，并将VR(UX)设置成VR(UH)。

VR(UH) > VR(UR)意味着此时至少有一个SN < VR(UH)的UMD PDU还未接收到，所以此时应启动*t-Reordering*，并在该定时器指定的时间内去接收还未收到的PDU。



图1-6：步骤3及步骤4举例

假设VR(UX)以及其它变量的取值如图1-6的(c)所示。在图1-6中，*t-Reordering*正在运行，接收端又收到了一个SN = n+2的UMD PDU，此时VR(UR)从n+2更新成n+x+3，并导致“VR(UX) = n+3 <= VR(UR)”。接收端认为小于VR(UX)的UMD PDU都已成功接收，因此会停止并重置*t-Reordering*。此时变量的取值如图1-6的(d)所示。（VR(UX)位于重排序窗口之外的处理方式与此类似，这里就不再举例了）

由于步骤3的处理，*t-Reordering*没有运行，并且VR(UH) = n+x+5 > VR(UR) = n+x+3，意味着至少有一个SN < VR(UH) （这里对应SN = n+x+3）的UMD PDU还没有被接收到，因此会将VR(UX)设置成VR(UH)，并启动*t-Reordering*。此时变量的取值如图1-6的(e)所示。

**1.2.2.4 *t-Reordering*超时处理**

如果*t-Reordering*超时，则接收端会

* 将VR(UR)更新成“SN > VR(UX)，但还未接收到”的第一个UMD PDU的SN值。
* 对“SN < 更新后的VR(UR)”的UMD PDU进行重组，移除RLC header，并将重组后的还未曾递送过的RLC SDU按SN递增的顺序递送给PDCP层。
* 如果此时VR(UH) > VR(UR)，启动*t-Reordering*，并将VR(UX)设置成VR(UH)。

*t-Reordering*超时，说明VR(UX) > VR(UR)，且“VR(UX)在重排序窗口之内，或VR(UX)=VR(UH)”。也说明了“至少有一个SN < VR(UX)的UMD PDU在该*t-Reordering*指定的时间内没有被接收到”，此时接收端不再去尝试接收SN < VR(UX)的PDU，并认为那些没有收到的PDU已经丢失了。

可以看出，UM模式中，是通过*t-Reordering*超时来判断SN < VR(UX)的某个UMD PDU是否丢失了的。



图1-7：*t-Reordering*超时处理举例

如图1-7的(a)所示，*t-Reordering*超时的时候，还有一个SN = VR(UR) = n+x+3的PDU没有接收到。此时接收端不再去接收该PDU，并将VR(UR)更新成“SN > 当前VR(UX) = n+x+5，但还未接收到”的第一个UMD PDU的SN值，这里对应n+y-2。并对SN < n+y-2的PDU进行重组，移除RLC header，并将重组后的还未曾递送过的RLC SDU按SN递增的顺序递送给PDCP层。由于此时VR(UR) = n+y-2 < VR(UH) = n+y，因此要将VR(UX)设置成VR(UH)，并启动*t-Reordering*。此时变量的取值如图1-7的(b)所示。