# 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$10^{-5}$），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

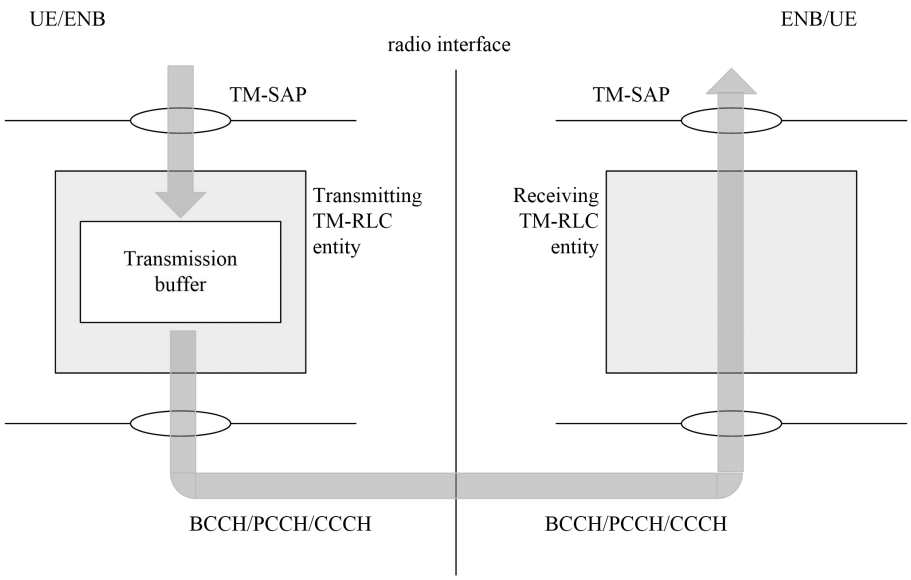



**Figure 4.2.1.1.1-1: Model of two transparent mode peer entities**

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。

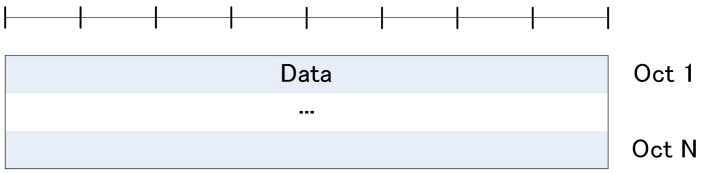



**Figure 6.2.1.2-1: TMD PDU**

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。

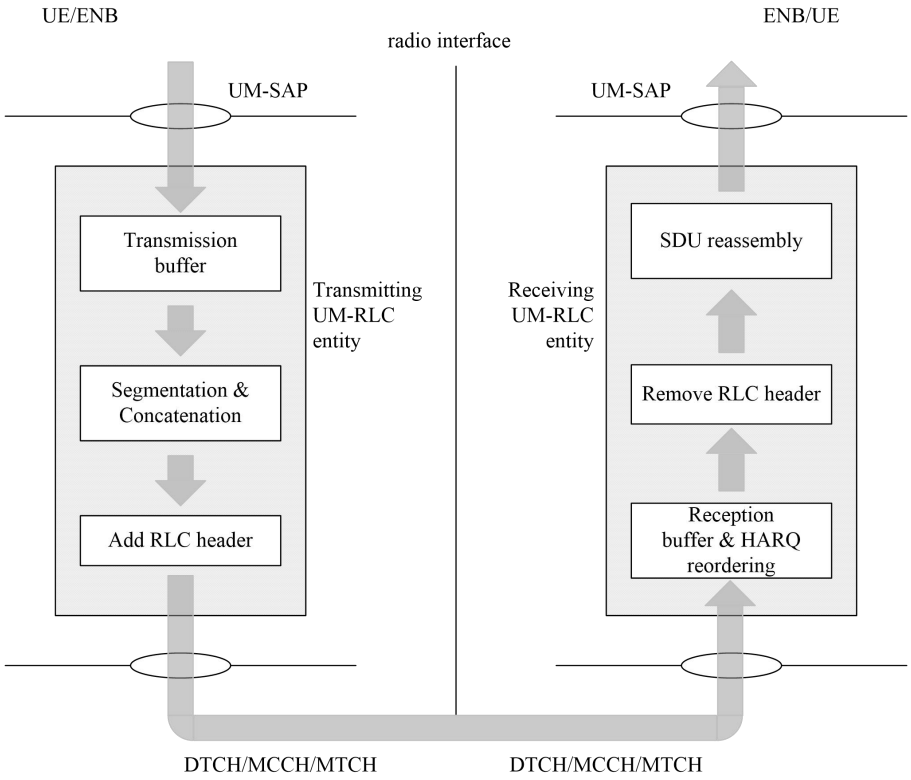



**Figure 4.2.1.2.1-1: Model of two unacknowledged mode peer entities**

### 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：

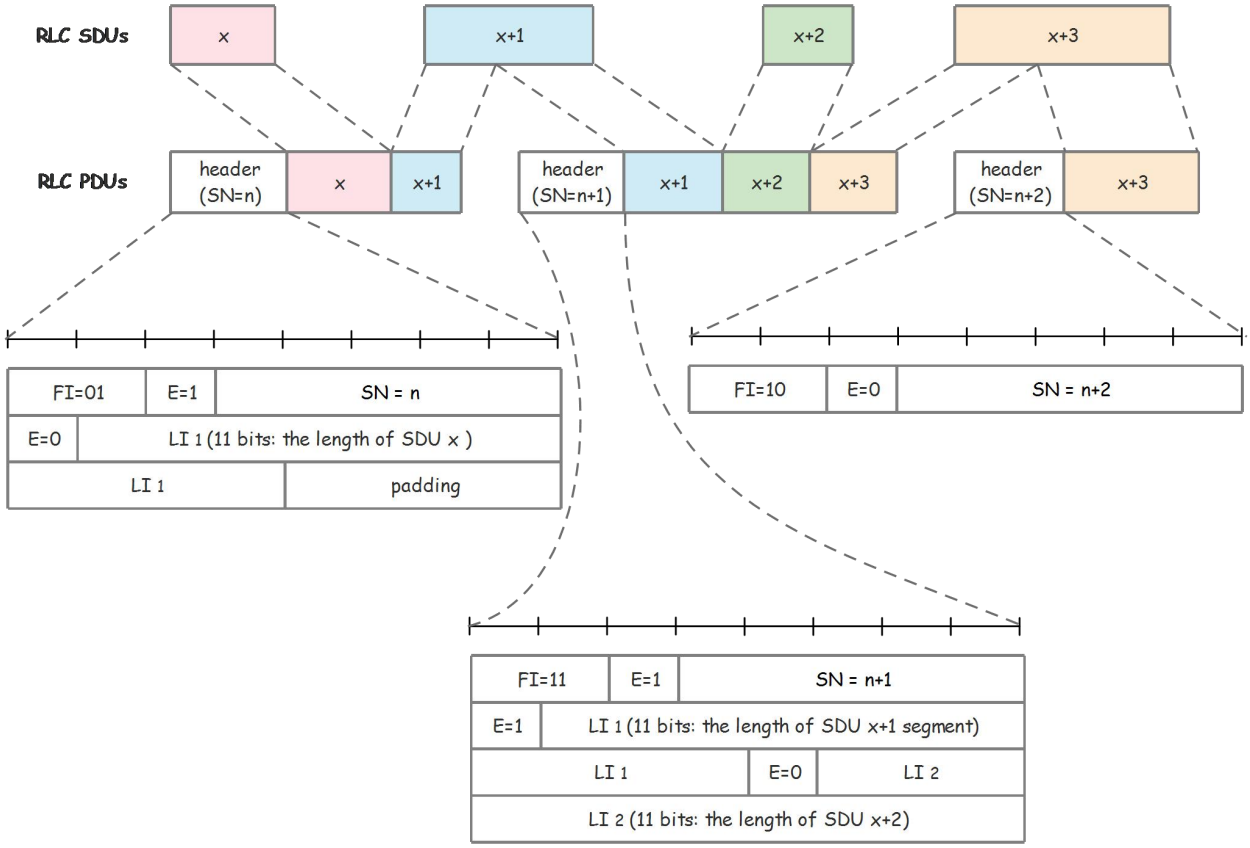
- VT(US)：该变量保存了下一个新生成的 UMD PDU 将被赋予的 SN 值。该变量初始化时为 0，并在 UM 实体发送一个 SN = VT(US)的 UMD PDU 给 MAC 层时更新。简单地说，“VT(US) – 1”等于最近一个已经发送的 UMD PDU 的 SN 值。

#### 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。



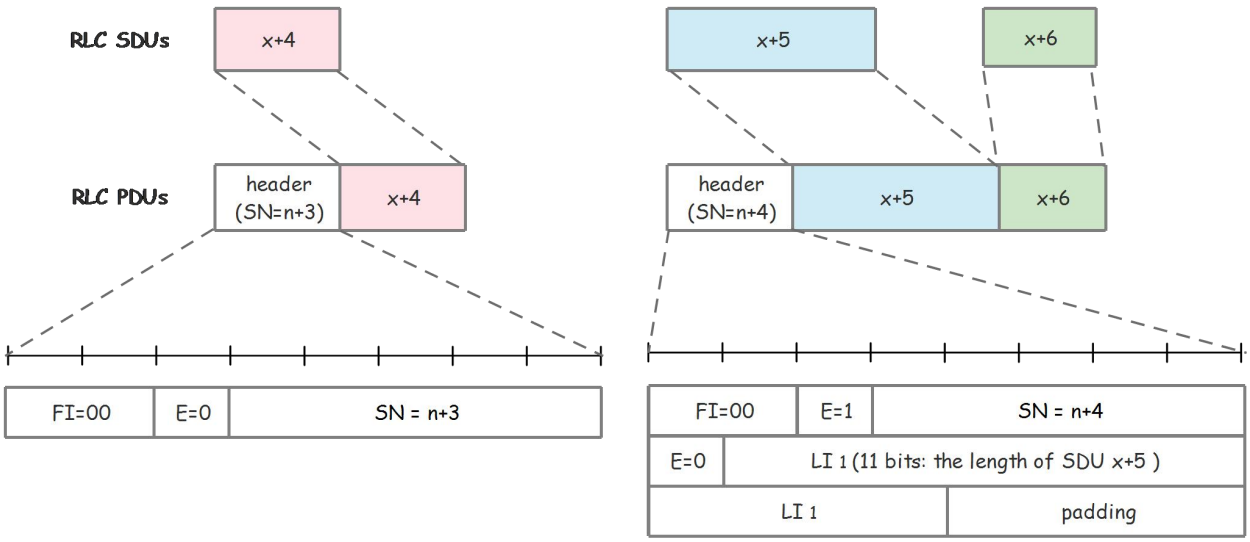



图 1-1：UM 分段和串联功能举例

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）

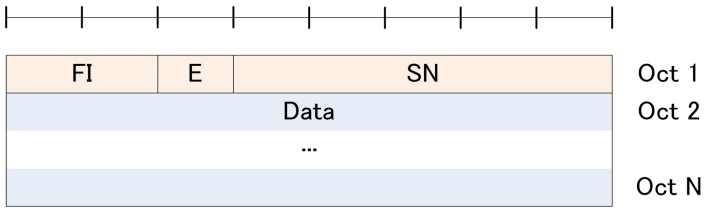



**Figure 6.2.1.3-1: UMD PDU with 5 bit SN (No LI)**

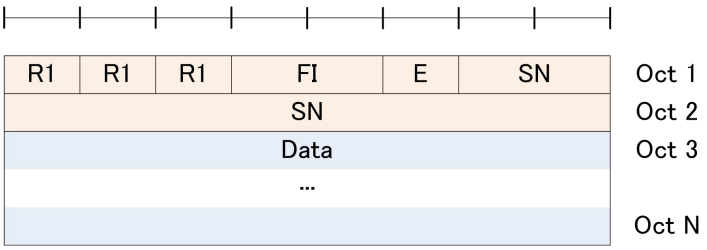



**Figure 6.2.1.3-2: UMD PDU with 10 bit SN (No LI)**

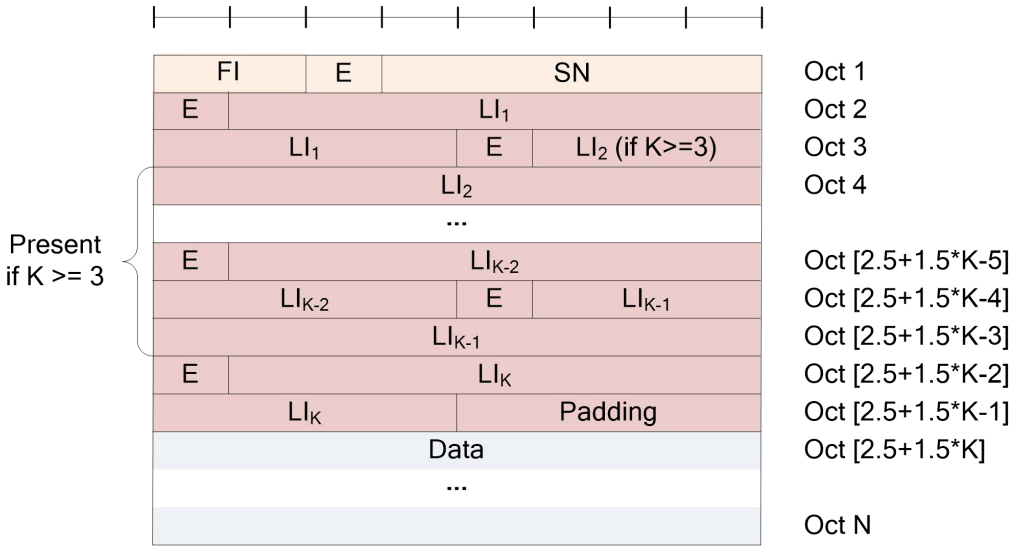



**Figure 6.2.1.3-3: UMD PDU with 5 bit SN (Odd number of LIs, i.e. K = 1, 3, 5, …)**

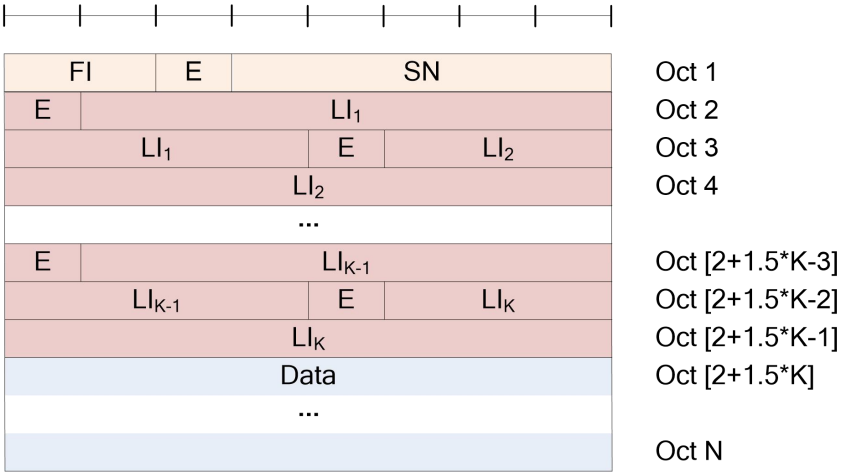



**Figure 6.2.1.3-4: UMD PDU with 5 bit SN (Even number of LIs, i.e. K = 2, 4, 6, …)**

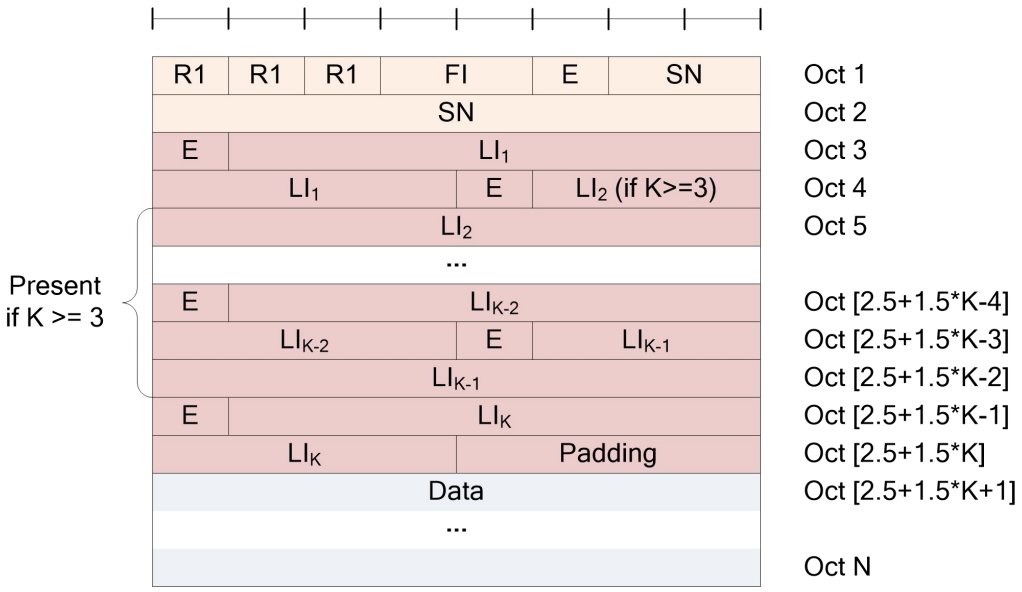



**Figure 6.2.1.3-5: UMD PDU with 10 bit SN (Odd number of LIs, i.e. K = 1, 3, 5, …)**

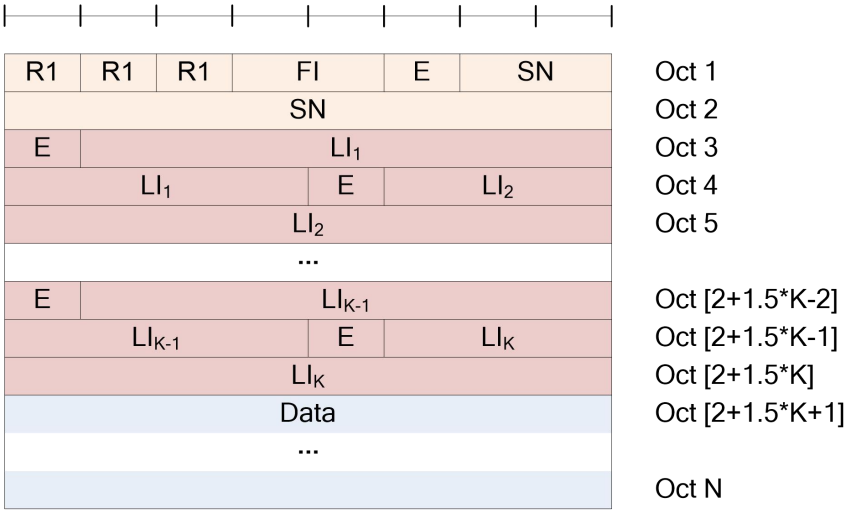



**Figure 6.2.1.3-6: UMD PDU with 10 bit SN (Even number of LIs, i.e. K = 2, 4, 6, …)**

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 $LI_1$）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 $LI_1$）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 $LI_2$）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）

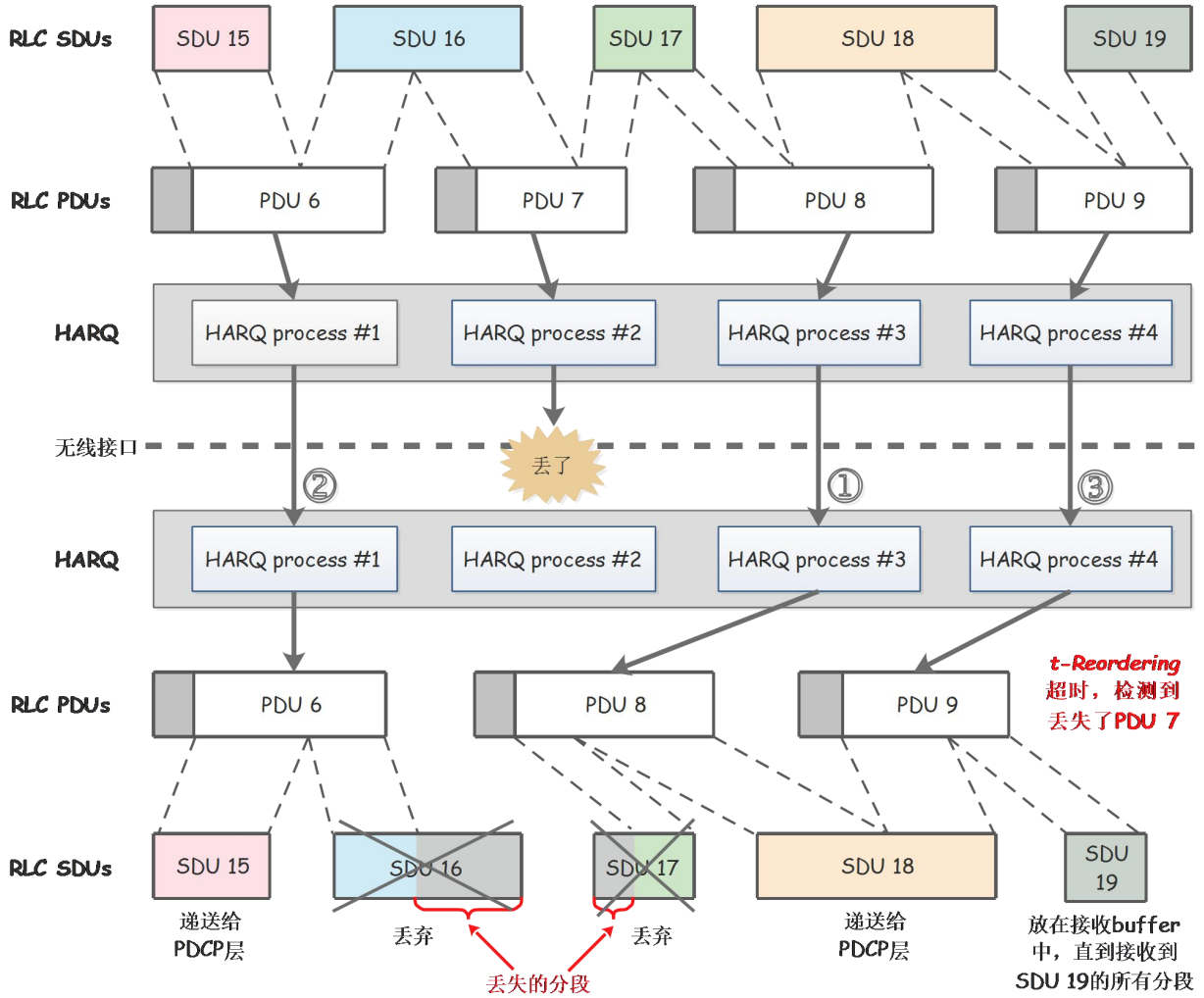


图 1-2：UM 模式中，PDU 丢失检测及处理

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。

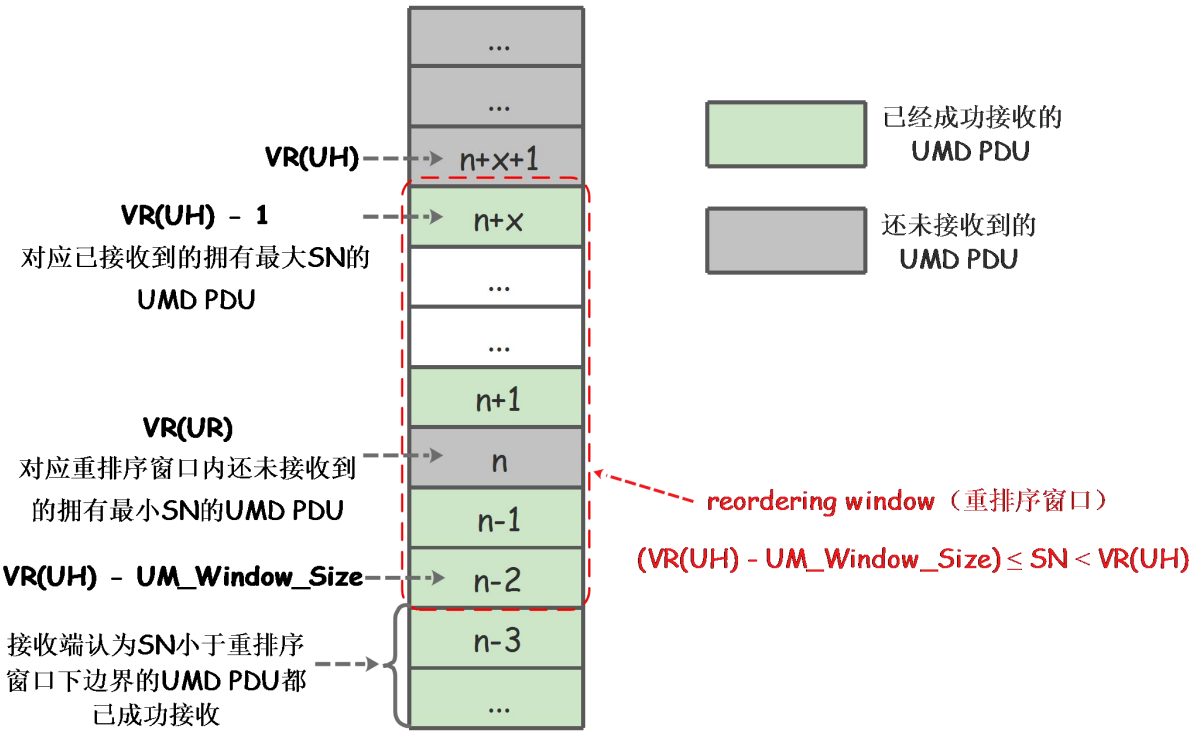



图 1-3：除 VR(UX)外的 UM 接收端相关变量

#### 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。

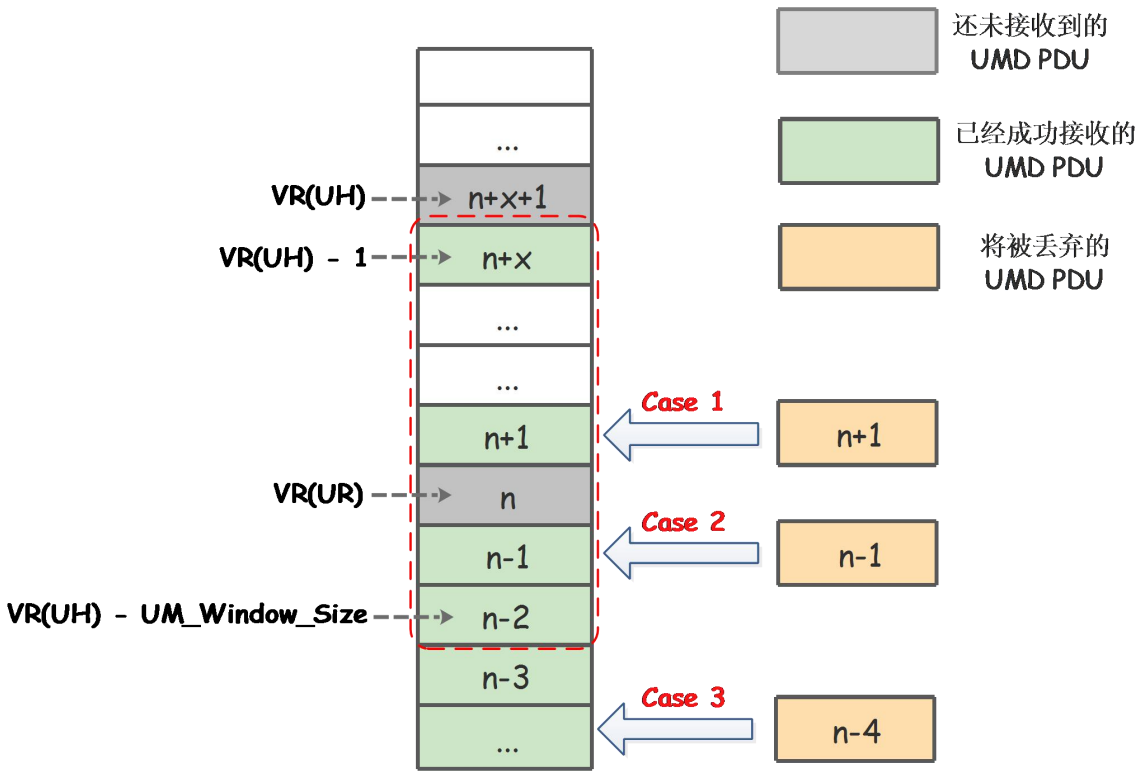



图 1-4：将被丢弃的 UMD PDU 举例

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。

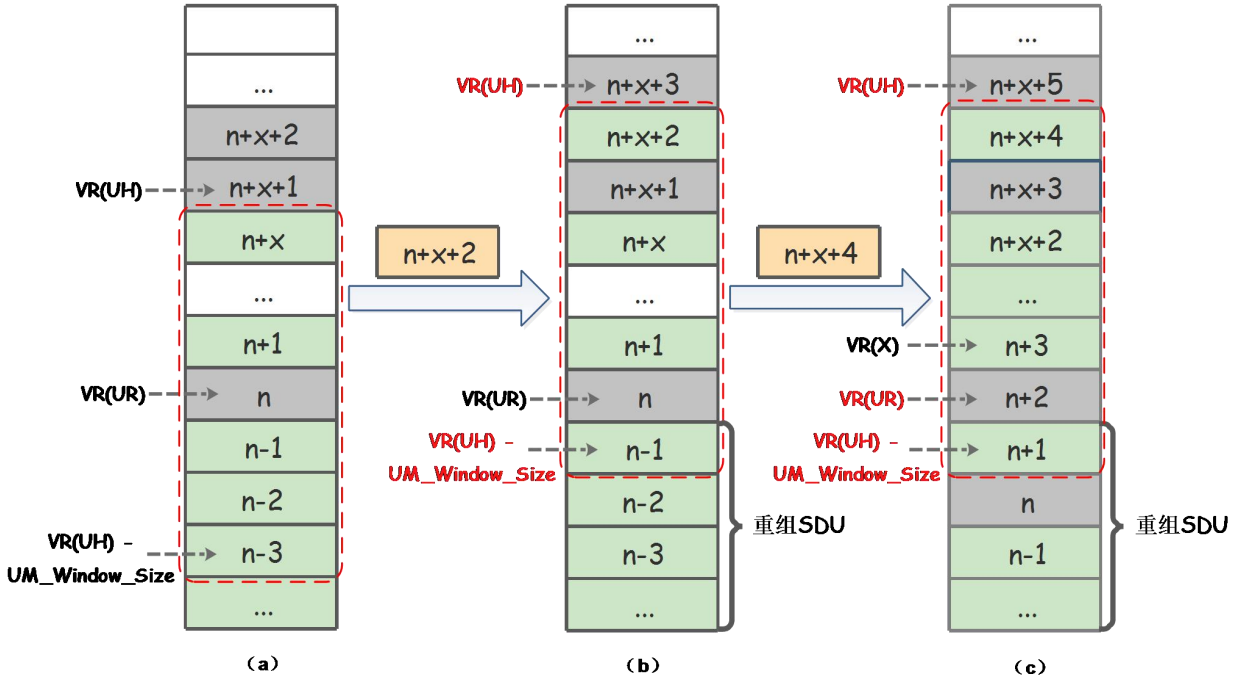



图 1-5：步骤 1 及步骤 2 举例

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。

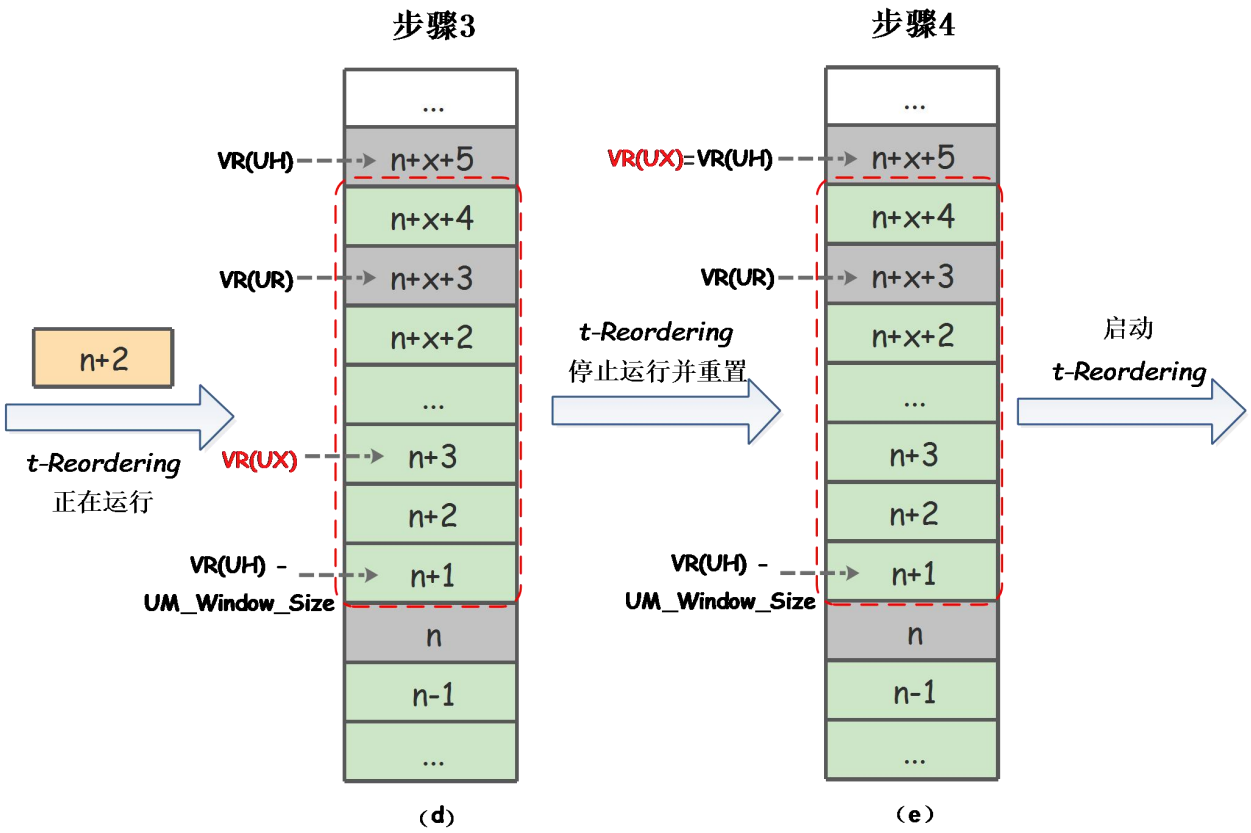



图 1-6：步骤 3 及步骤 4 举例

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。

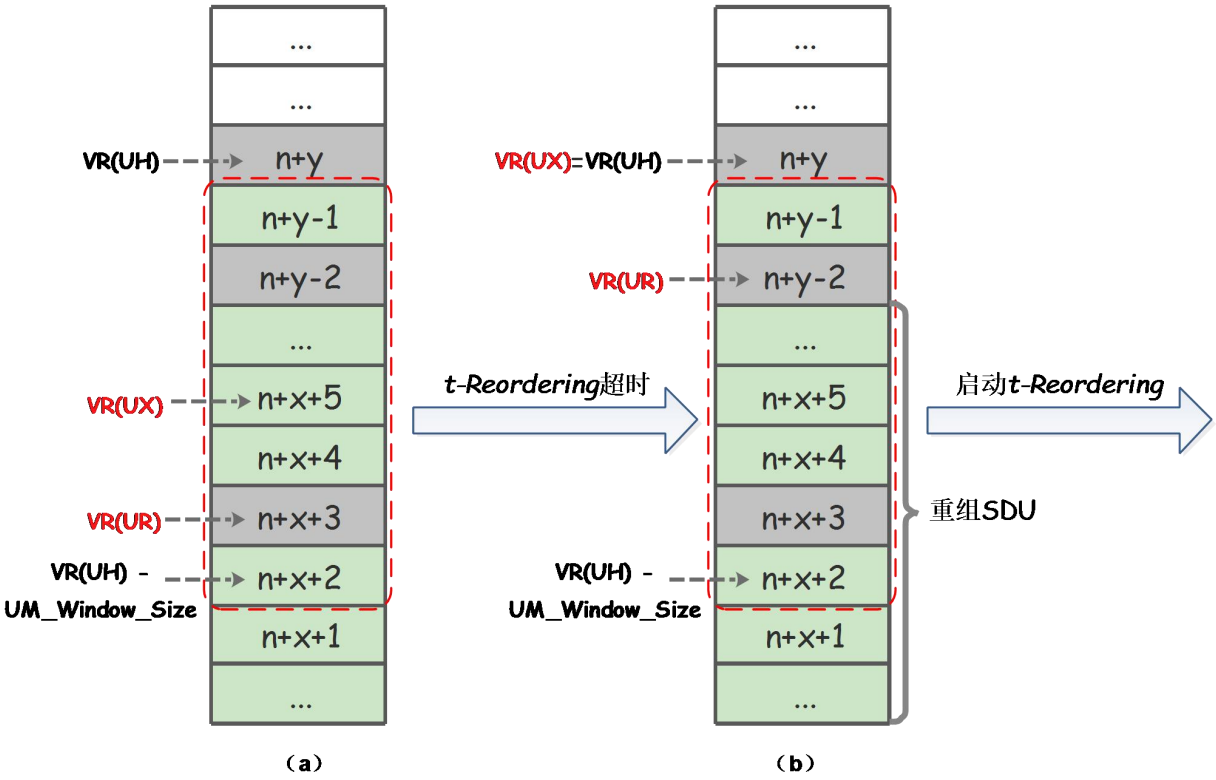



图 1-7：*t-Reordering* 超时处理举例

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。