File name	**DLMS 协议培训**	Date	2022-4-11
Archive No.		Version	1.0

DLMS 协议培训

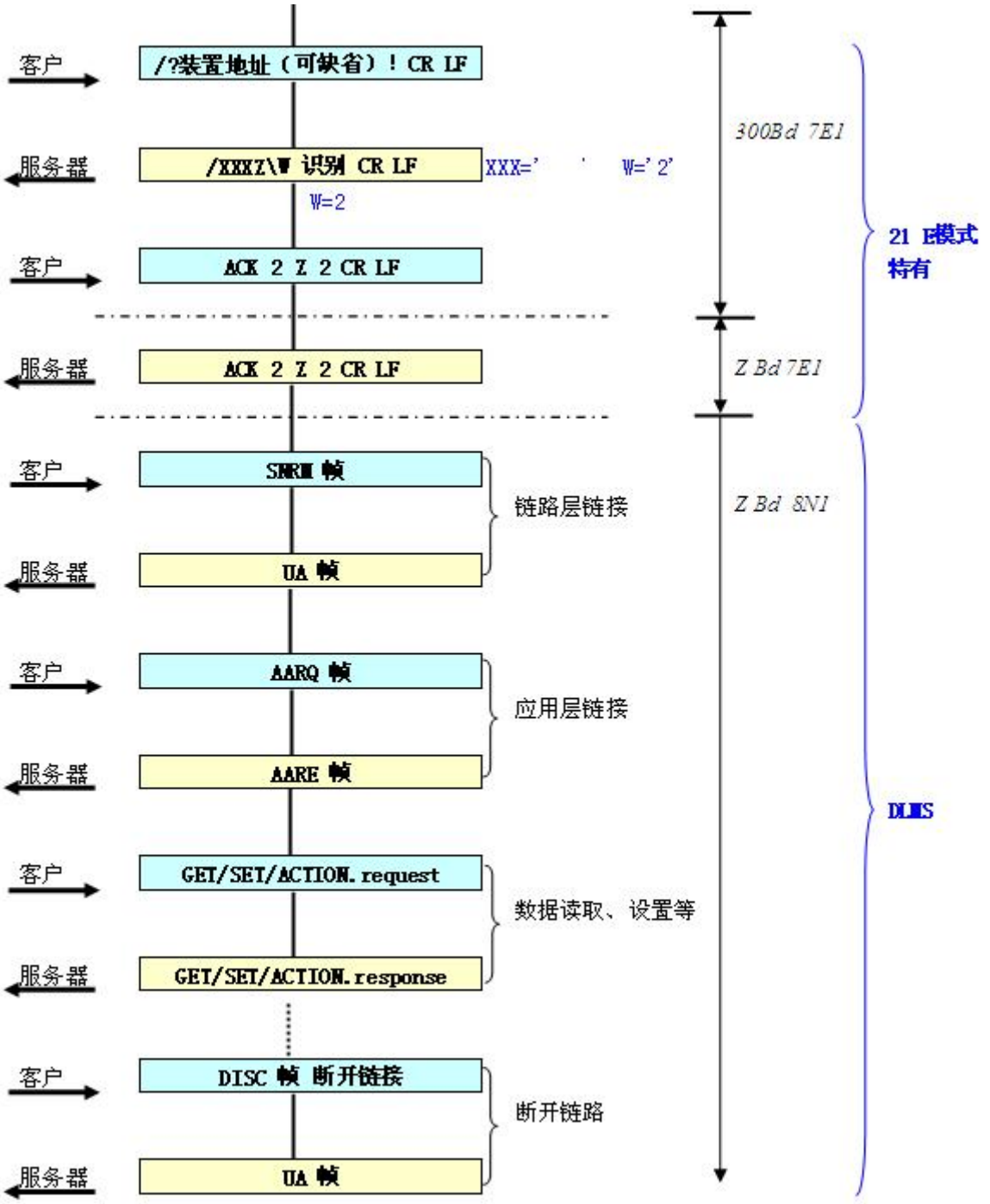
CONTENTS

1. 前言

本文档适合于初次接触 DLMS 协议的人员,想要更进一步学习的请查看蓝皮书、绿皮书（本文当中如无特声明，蓝皮书是指：《Blue_Book_Edition_13.pdf》，绿皮书是指《Green-Book-Ed-10-V1.0.pdf》）。

2. IEC 62056-21 E 模式通信流程

DLMS/COSEM 协议中将 Communication_profile 分为 TCP_profile 和 HDLC_profile 两种，而使用 HDLC_profile 的又分为 E 模式和直接 HDLC，两者的唯一的区别是 E 模式有波特率 300bps 转到 Zpbs 的握手过程，而直接 HDLC 直接进入波特率 Z 下通信。下图为电能表在 IEC62056-21 E 模式下正常工作时的通信流程：



整个通信过程为C/S模式，表计充当服务端，HHU/PC为客户端。每一次通信过程由客户端发起，服务端应答。

3. HDLC 帧格式

IEC62056-21 E模式中通信链路帧采用HDLC帧格式，除信息域按其指定格式外，其他域均为16进制传送，其格式如下：

标志	帧格式	目的地址	源地址	控制	HCS	信息	FCS	标志

3.1. 标志域

标志域的长度为一字节，值为7EH。当两个或多个帧连续传输时，这一个标志既要用作前一帧的结束标志，又要用作下一个帧的开始标志，如图11所示。

注：当两个传输的字符之间的时段没有超过指定的最大内部字节周期时，帧可以连续传输。

7E	帧 I	7E	帧 I+1	7E	帧 I+2	7E

3.2. 帧格式域

帧格式域的长度为两个字节，它由三个子域组成：Frame_type子域(4 bit)，分段位(S, 1 bit)和帧长度子域(11 bit)，见图12。

MSB LSB

1	0	1	0	S	L	L	L	L	L	L	L	L	L	L	L

格式类型子域的值为1010（二进制）。

分段位S表示是否有后续帧，如果服务端给客户端传送的数据能在一帧内传送完，那么S=0，如果有后续帧那么S=1。

长度子域的值是除两个7E标志之外的8位位组数。在一般情况下，帧长度不会超过256，因此帧格式域第一个字节为 A0或者A8 ，第二个字节表示该帧的长度。

3.3. 地址域

这个帧有两个地址域：一个目的HDLC地址和一个源HDLC地址。根据数据的传输方向，客户机端地址和服务器地址都可以是目标地址或源地址。

客户机端地址总是用一个字节表示。扩展地址的使用把客户机地址的范围限制在128。

在服务器端，为了能在一个物理设备内寻址一个以上的逻辑装置并且支持多站配置，可以将HDLC地址分为两部分。 一部分称为“**高端HDLC地址**”用于逻辑设备（一个物理设备内可独立寻址的实体）寻址，而第二部分——“**低端HDCL地址**”将用于物理设备（多站配置的一个物理设备）寻址。高端HDLC地址总是存在，而低端HDCL地址在不需要时可不用。

HDLC地址扩展机制应用于以上两种地址域。这种地址扩展说明可变长度的地址域，但是考虑到该协议，一个完整的HDLC地址域的长度被限制为一字节，两字节或四字节如下：

- 一字节：只有高端HDLC地址存在。
- 两字节：一字节高端HDLC地址和一字节低端HDLC地址。
- 四字节：两字节高端HDLC地址和两字节低端HDLC地址。

上述三种情况在下图说明。

一字节地址结构：

LSB

高端HDLC地址	1

两字节地址结构：

LSB LSB

高端HDLC地址	0	低端HDLC地址	1

第一字节　　第二字节

四字节地址结构：

	LSB		LSB		LSB		LSB
高端HDLC高字节	0	高端HDLC低字节	0	低端HDLC高字节	0	低端HDLC高字节	1

第一字节　　第二字节　　第三字节　　第四字节

这种可变长度HDLC地址结构为每字节保留了一位来标示所给字节是最后一字节还是有字节跟随。这意味着一字节地址的地址范围是0…0x7F，两字节地址的地址范围是0…0x3FFF。

单独的，多播及广播的寻址可由高端HDLC地址和低端HDLC地址方便提供。

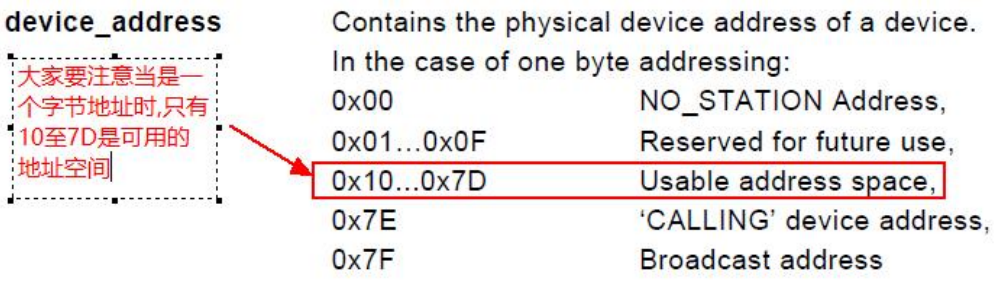
例如，一个HDLC帧以下列地址从客户机端发送到服务器端：

客户HDLC地址 = $3A_H$ = 00111010_B
服务器HDLC地址（用四字节寻址）
低端HDLC地址= $3FFF_H$ = 0011111111111111_B　ALL_STATION（广播）地址
高端HDLC地址 = 1234_H = 0001001000110100_B

消息地址域应包含以下8位字组：

服务器地址				客户机地址
高端HDLC,高字节	高端HDLC,低字节	低端HDLC,高字节	低端HDLC,低字节	HDLC 地址
LSB	LSB	LSB	LSB	LSB
0 1 0 0 1 0 0　0	0 1 1 0 1 0 0　0	1 1 1 1 1 1 1　0	1 1 1 1 1 1 1　1	0 1 1 1 0 1 0　1
第一字节	第二字节	第三字节	第四字节	第五字节
目的地址				源地址
4868FEFF				75

<Blue-Book-Ed14--V1.0.pdf>蓝皮书描述地址第216页



3.4. 控制域格式

命令/应答帧控制字段的编码方式为模式8，如ISO/IEC 13239的5.5及下表规定。

表7 控制字段格式

	MSB							LSB
I	R	R	R	P/F	S	S	S	0
RR	R	R	R	P/F	0	0	0	1
RNR	R	R	R	P/F	0	1	0	1
SNRM	1	0	0	P	0	0	1	1
DISC	0	1	0	P	0	0	1	1
UA	**0**	**1**	**1**	**F**	**0**	**0**	**1**	**1**
DM	**0**	**0**	**0**	**F**	**1**	**1**	**1**	**1**
FRMR	**1**	**0**	**0**	**F**	**0**	**1**	**1**	**1**
UI	**0**	**0**	**0**	**P/F**	**0**	**0**	**1**	**1**

I Information (a HDLC frame type) 信息帧，有帧序。

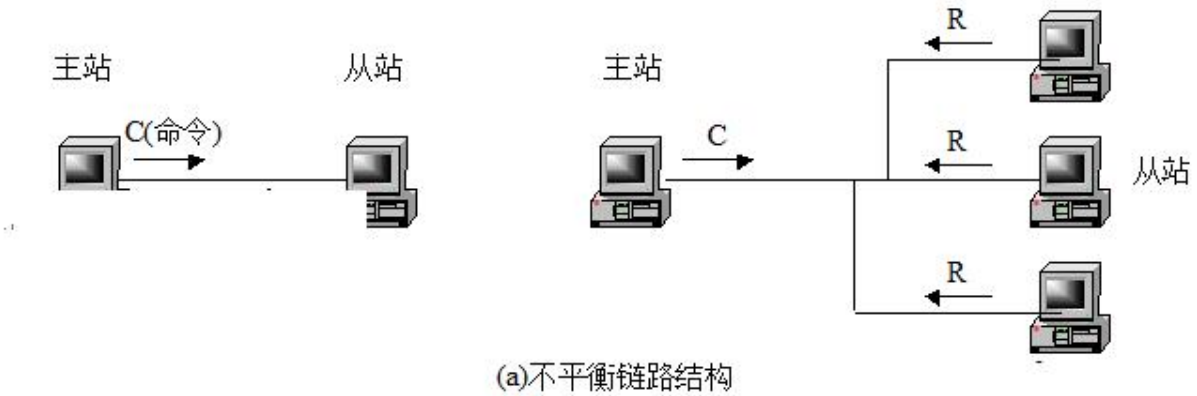
RR Receive Ready (a HDLC frame type) 接收准备帧，如果PC端没有收到，可以发此帧，等表的应答。

RNR Receive Not Ready (a HDLC frame type) 接收未准备。

SNRM Set Normal Response Mode (a HDLC frame type) 建立链路层联接

注:NRM链路,即正常响应方式链路,他具有哪些特点,我摘要如下(来自中兴通讯一份内训资料,权威的需参照国家标准)

常响应方式（NRM）适用于不平衡链路结构(如下图)，即用于点-点和点-多点的链路结构中，特别是点-多点链路。这种方式中，由主站控制整个链路的操作，负责链路的初始化、数据流控制和链路复位等。从站的功能很简单，它只有在收到主站的明确允许后，才能发出响应。从站电表处理被动状态



(a)不平衡链路结构

DISC Disconnect (a HDLC frame type)断开链路

UA Unnumbered Acknowledge (a HDLC frame type) 无编号确认的

DM Disconnected Mode (a HDLC frame type)断开模式

FRMR: Frame Reject (a HDLC frame type) 帧拒绝应答。

UI Unnumbered Information (a HDLC frame type)未确认信息

关于表应答断开链接特别注意:

如果表收到53请求断开.如果此时表在收此帧之前处于连接状态,那么应答为73 即UA帧, 当表收到此帧时表认为已经处于断开状态那么应答为DM即1F,

这里RRR是接收序列号N(R)，SSS是发送序列号N(S)，P/F是查询/结束位。

链路层链接好即 SNRM UA 帧后，RRR 和 SSS 均为 000，发送一帧 I 帧 SSS 加 1，接收到一帧 I 帧 RRR 加 1，客户端和服务端都是如此。P/F 标志位中 P 是对客户端而言的，需要响应 P=1，那么广播帧时 P=0；F 是对服务端而言的，F 表示发送是否结束，也就是是不是没有后续帧，F=1 表示有后续帧，因此当客户端收到服务端发送来的帧格式域中 S=1 和此处的 F=1 的帧时，需回应 RR 帧等待接收未接收完的数据。

例如：

//SNRM 帧 100P0011,因需要响应，P=1，所以控制字为 93

客户端:7E A0 0A 48 68 FE FF 75 93 D8 F8 7E

//UA 帧 011P0011,因需要响应，F=1，所以控制字为 73

服务端:7E A0 21 75 48 68 FE FF 73 7C 16 81 80 12 05 01 80 06 01 80 07 04 00 00 00 01 08 04 00 00 00 01 53 3B 7E `

//AARQ帧（属于I类型）RRR=0，SSS=0，P=1，所以控制字为10

客户端:7E A0 46 48 68 FE FF 75 10 05 C1 E6 E6 00 60 35 A1 09 06 07 60 85 74 05 08 01 01 8A 02 07 80 8B 07 60 85 74 05 08 02 01 AC 0A 80 08 41 42 43 44 45 46 47 48 BE 10 04 0E 01 00 00 00 06 5F 04 00 00 00 14 00 00 BD BF 7E

//AARE帧（属于I类型）RRR=1，SSS=0，F=1，所以控制字为30

服务端:7E A0 52 75 48 68 FE FF 30 95 39 E6 E7 00 61 41 A1 09 06 07 60 85 74 05 08 01 01 A2 03 02 01 00 A3 05 A1 03 02 01 00 88 02 07 80 89 07 60 85 74 05 08 02 01 AA 0A 80 08 41 42 43 44 45 46 47 48 BE 0F 04 0D 08 00 06 5F 04 00 00 00 14 21 34 00 07 14 53 7E

//GET.request（属于I类）RRR=1，SSS=1，P=1，所以控制字为32

客户端:7E A0 1C 48 68 FE FF 75 32 0E 3B E6 E6 00 C0 01 81 00 07 00 00 63 01 00 FF 00 02 DE 82 7E

//GET.response（属于I类）RRR=2，SSS=1，F=0，所以控制字为52；（S=1，所以帧格式为A8）

服务端:7E A8 8C 75 48 68 FE FF 52 CE 26 E6 E7 00 C4 01 81 00 01 22 02 06 02 02 09 0C 07 D3 06 09 FF 09 1D 0D FF FF FF FF 04 06 40 00 00 00 10 00 00 10 00 00 02 06 00 00 00 00 10 00 00 10 00 00 02 06 00 00 00 00 10 00 00 10 00 00 02 06 00 00 00 00 10 00 00 10 00 00 02 06 00 00 00 00 10 00 00 10 00 00 02 06 00 00 00 00 10 00 00 10 00 00 02 06 00 00 00 00 10 00 00 10 00 00 02 06 00 00 00 83 9E 7E

//RR 帧 由于上面服务端回的帧中 F=0 S=0 即有后续帧，所以此处客户端应回应 RR 帧等待服务端的后续帧 RRR=2，P=0，所以控制字为 51；

客户端:7E A0 0A 48 68 FE FF 75 51 14 E4 7E

//GET.response 的后续帧 RRR=2，SSS=2，F=0，所以控制字为 54；（S=1，所以帧格式为 A8）

服务端:7E A8 8C 75 48 68 FE FF 54 F8 43 00 10 00 00 10 00 00 02 06 00 00 00 00 10 00 00 10 00 00 02 06 00 00 00 00 10 00 00 10 00 00 02 06 00 00 00 00 10 00 00 10 00 00 02 06 00 00 00 00 10 00 00 10 00 00 02 06 00 00 00 00 10 00 00 10 00 00 02 06 00 00 00 00 10 00 00 10 00 00 02 06 00 00 00 00

10 00 00 10 00 00 02 06 00 00 00 00 10 00 00 10 00 00 02 06 00 00 00 00 10 00 00 10 00 00 02 F9 31 7E

//RR 帧 由于上面服务端回的帧中 F=0 S=0 即有后续帧，所以此处客户端应回应 RR 帧等待服务端的后续帧

RRR=2，P=0，所以控制字为 71

客户端:7E A0 0A 48 68 FE FF 75 71 16 C5 7E

//GET.response 的后续帧 RRR=2，SSS=3，F=0，所以控制字为 56；（S=1，所以帧格式为 A8）

服务端:7E A8 8C 75 48 68 FE FF 56 EA 60 06 00 00 00 00 10 00 00 10 00 00 02 06 00 00 00 00 10 00 00 10 00 00 02 06 00 00 00 00 10 00 00 10 00 00 02 06 00 00 00 00 10 00 00 10 00 00 02 06 00 00 02 02 09 0C FF FF FF FF FF 0F 08 34 FF FF FF FF 04 06 40 02 02 09 0C FF FF FF FF FF 0F 0E 1F FF FF FF FF 04 06 40 10 00 00 10 00 00 02 06 00 00 00 00 10 00 00 10 00 00 02 06 00 00 00 00 10 00 00 10 00 00 02 06 00 00 00 00 10 00 00 14 24 7E

//RR 帧 由于上面服务端回的帧中 F=0 S=0 即有后续帧，所以此处客户端应回应 RR 帧等待服务端的后续帧

RRR=3，P=0，所以控制字为 91

客户端:7E A0 0A 48 68 FE FF 75 91 18 22 7E

//GET.response 的后续帧 RRR=2，SSS=4，F=0，所以控制字为 58；

服务端:7E A0 75 75 48 68 FE FF 58 40 72 10 00 00 02 06 00 00 00 00 10 00 00 10 00 00 02 06 00 00 00 00 10 00 00 10 00 00 02 06 00 00 00 00 10 00 00 10 00 00 02 06 00 00 00 00 10 00 00 10 00 00 02 06 00 00 00 00 10 00 00 10 00 00 02 06 00 02 02 09 0C FF FF FF FF FF 11 23 0C FF FF FF FF 04 06 40 00 00 10 00 00 10 00 00 A2 1B 7E

客户端:7E A0 0A 48 68 FE FF 75 B1 1A 03 7E

服务端:7E A0 0A 75 48 68 FE FF 51 39 E7 7E

客户端:7E A0 0A 48 68 FE FF 75 53 06 C7 7E

服务端:7E A0 0A 75 48 68 FE FF 73 29 E5 7E

3.5. 帧序算法

关于帧序的算法，一般我们是参数上述例程，分别在接收和发送端加1来实现，在正常通讯中似乎是没有问题，这主要是表做了兼窗口．同时以前我们没有用到过分段传输的情况．2019年6月康宣采用K公司的软件对T15VD的表进行分段升级，我们才发现以前的帧序的算法不够正确．我们先来看以下的报文临控：

序号	帧方向	帧功能	帧序	旧算法	对方值	新算法 (接收域为对方的发送域+1，发送域为对方接收域)	
1	PC->表	SNRM					
2	表->PC	UA					
3	PC->表	AARQ	10				
4	表->PC	AARE	30		表收3， 表发0		
5	PC->表	ACTION REQUEST	32	32发送+1	PC收3 PC发2		
6	表->PC	Response	52	表接收+1，	表收5 表发2	5=001+001=010；并入PD位0101即为5. 2=0011最低位置1即为2	OK

7	PC->表	乱数据帧	--				
8	表->PC	RR	D1	?	表收D	因为停止了监控所以,不知帧到了D1.	
9	PC->表	分段升级第一段	7C	54	PC收7 PC发C	7: ?不知,因为停止了监控 C: 1101(D)最低位置1即为C	
10	表->PC	RR	F1	表收+1	表收F	F:C110+1=111 并入PF=1111 1:RR帧发送域是置1的	
11	PC->表	分段升级第二段 (最后一段)	7E	旧算法遇到分段就要特殊处理了.收不变,发送+1	PC收7 PC发E	7:?不知,因为停止的监控 E:F最低位置0为E	
12	表->PC	Response	16		表收1 表发6	1:111+1=000 并入PF帧,即为1. 6:7=0111低位置0110=6	
13	PC->表	分段升级第一段	90		PC收9 PC发0	9: 6+1 0110-011+1=100,并入PF即为1001=9 0:0001最低位置0为0000	
14	表->PC	RR	31		表收3	3:0+1 0010并入PF即0011=3	
15	PC->表	分段升级第二段	92		PC收9 PC发2	9:6+1=0110-011+1=100并入PF即为9 2:3-1 0011最低位置0 为2	
16	表->PC	RR	51		表收5 表发1	5:2 0010-001+1=010,并入PF位即为0101=5	
17	PC->表	分段升级第三段 (最后一段)	94		PC收9 PC发4	9:6+1 0110-011+1=100,并入PF即为1001=9 4: 5-0101最低位置0即为4	
18	表->PC	Response	78		表收7 表发8	7: 4 0100-010+1=011,并入PF后即为0111=7 8:9-1001最低位置0即为8	
19	PC->表		B6		PC收B PC发6	B: 100+1=101并入PF后即为1011=B 6:7 0111最低位置0即为6	
20	表->PC	RR	91		表收9	9:0110-011+1=100,并入PF后即为9	
21	PC->表		B8		PC收B PC发8	B:8 1000-100+1=101并入PF后即为1011=B 8:9 1001最低位置0=8	
22	表->PC	RR	B1		表收B	B:1000 100+1=101,并入PF后即	

						为1011=B	
23	PC->表		BA		PC收B PC发A	B: 1000 100+1=101, 并入PF后即为1011=B A:B--1011最低位置0=A	
24	表->PC		DA		表收D表发A	D:A+1 101+1=110, 并入PF后即为1101为D A:B-1最低位置0=A	
25	PC->表		DC		PC收D PC发C	D:A+1 101+1=110, 并入PF后即为1101为D C:D-1最低位置0=C	
26	表->PC	RR	F1		表收F	F:C+1 =1100 110+1=111并入PF后即为F	
27	PC->表		DE		PC收D PC发E	D:A+1 101+1=110并入PF后即为1101为D E:F-1最低位置0=E	
28	表->PC	RR	11		表收1	1:E+1=111 +1=000, 并入PF后即为0001	
29	PC->表		D0		PC收D PC发0	D:A+1 101+1=110, 并入PF后即为1101为D 0:1-1最低位置0	
30	表->PC		3C		表收3 表发C	B:0+1=001并入PF后即为0011为3 C:D-1..	

3.6. 头校验序列（HCS）域

HCS的长度是两个字节。

HCS计算除开始标志和HCS本身外的头的字节数。

HCS的计算方法跟帧校验序列（FCS）类似。不包含信息域的帧，仅含FCS（在这种情况下，HCS被看作FCS）。HCS（和FCS）的计算方法采用CRC校验算法，不等式为X**0+X**5+X**12+X**16。

3.7. 信息域

信息域可以是任意字节序列。

3.8. 帧校验序列（FCS）域

FCS 域的长度是两个字节，用来计算除开始标志和 FCS 本身 外的完整的帧长度。不包含信息域的帧只包含 FCS（这里 HCS 被看作 FCS）。

4. 链路层链接与断开

4.1. IEC1107 模式 E

常用于通过红外口通讯

看报文:

Baud Rate :300 , StopBits: 1, Parity: Even, DataBits: 7

发送: /?!

接收: /DZG0\220300HD11FW451 //注意表返回的信息它的通讯波特率要求为 0X30,即 300

/DZG5\220300HD11FW451 //注意表返回的信息它的通讯波特率要求为 0X35,即 9600

发送切换波特率指令: 202

那么切换波特率到 300 8,N,1 以后收到表返回的数据

接收: 202

4.2. 链路层的链接

链路层的连接包括客户端发送的 SNRM 帧和服务端发送的 UA 帧。

SNRM(Set normal response mode)/UA信息交换不但允许建立连接，而且允许协商一些数据链路参数。该协商的HDLC参数子集包含两部分：

- WINDOW_SIZE 参数（最大窗体数，即一次最多可传输的帧数，这个参数不能大于7）
- MAXIMUM_INFORMATION_FIELD_LENGTH 参数（信息域的最大字节数）

这些参数的缺省值如下：

- 默认WINDOW_SIZE=1
- 默认 MAXIMUM_INFORMATION_FIELD_LENGTH=128(80H)

协商规则如下：

协商从SNRM帧开始。该帧可能包含信息域。当信息域存在时，它应采用下面格式(见ISO/IEC13239 5.5.3.2条)：(见绿皮书第137页)

信息域编码的格式举例(参数值为缺省值):

81	80	12	05	01	80	06	01	80	07	04	00	00	00	01	08	04	00	00	00	01

这里：

81_H	格式标识符
80_H	组标识符
12_H	组长(18字节)
05_H	参数标识符(最大信息字段长度，发送)
01_H	参数长度（1字节）
80_H	参数值
06_H	参数标识符(最大信息字段长度，接收)
01_H	参数长度（1字节）
80_H	参数值
07_H	参数标识符(窗口大小,发送)
04_H	参数长度（4字节）
00_H	参数值(值高字节)
00_H	参数值
00_H	参数值
01_H	参数值(值低字节)
08_H	参数标识符(窗口大小，接收)
04_H	参数长度
00_H	参数值(值高字节)

00_H 参数值
00_H 参数值
01_H 参数值(值低字节)

具体看报文(不带参数协商的 SNRM 帧)
发送: 7E A0 0A 00 02 FE FF 09 93 2E 6F 7E
接收: 7E A0 23 09 00 02 FE FF 73 7A 0B 81 80 14 05 02 01 94 06 02 01 74 07 04 00 00 00 01 08 04 00 00 00 01 29 F8 7E

4.3. 链路层的断开

断开链路层包括客户端发送的DISC帧和服务端回应的UA/DM帧。

接收到DISC时没有断开链路，服务端回UA，然后链路断开；接收到DISC时链路已经断开则服务端回应DM帧。

DISC帧、UA帧和DM帧固定，只要将地址域及HCS域更换，如下：

DISC帧为 7E A0 0A 48 68 FE FF 75 53 06 C7 7E

UA帧为 7E A0 0A 75 48 68 FE FF 73 29 E5 7E

DM帧为 7E A0 0A 75 48 68 FE FF 1F 98 AD 7E

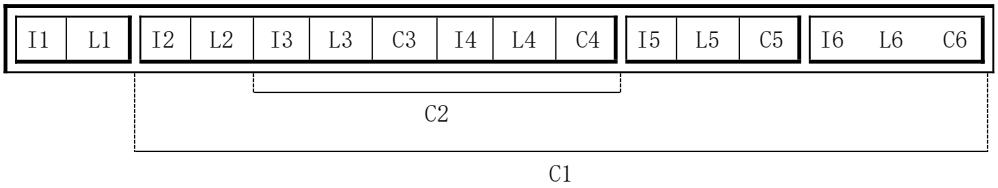
5. 应用层的链接

应用层链接由客户端发起的 AARQ 帧和服务端回应的 AARE 帧构成。主要协商引用机制（逻辑名引用还是短名引用，瑞银暂定为逻辑名引用）、支持的应用操作（读、写、方法）、用户级别等。对于 ASN.1 语义 AARQ、AARE（除用户信息单元中的 Initial 部分）采用的是 BER 编码规则，其余均采用 A-XDR 编码规则。

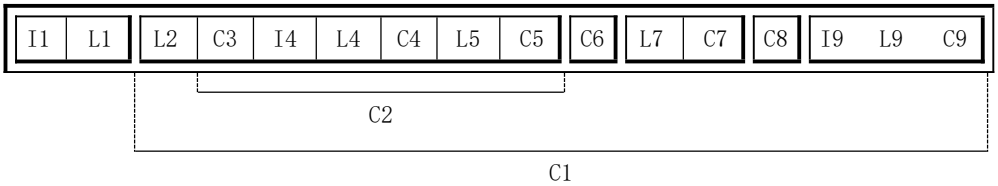
5.1. 编码规则

BER 编码由三部分构成：类型，长度及值，如下图。

那么表达多项内容时则由一系列字节构成，如下：



A-XDR 编码规则与 BER 编码类似，只是 A-XDR 编码省略了冗余的类型和长度部分，比如类型为 integer8 时，从类型便可看出长度为 1 个字节，因此可以省略掉长度部分。A-XDR 编码结构如下：



5.2. AARQ APDU 和 AARE APDU 规范

在COSEM中,AARQ APDU在BER中被编码,而且包含一个AXDR编码为DLMS-Initiate.request的pdu，作为用户信息单元内部的 OCTETSTRING。

AARQ和AARE APDU-s规范如下（实际为AARQ AARE帧中信息域的参数，OPTIONAL表示该参数可缺省）

5.3. AARQ-pdu 的编码例子

5.3.1. NS（无安全认证）的 AARQ 报文

7EA02E0002FEFF0B10F8E6E6E600601DA109060760857405080101BE10040E01000000065F1F0400001819FFFF3ECC7E

CTT: 601DA109060760857405080101BE10040E01000000065F1F040060FEDFFFFF

5.3.2. LLS(低安全级别) 的 AARQ 报文

具体见报文.

LLS 发送: 7E A0 47 00 02 FE FF 09 10 F4 30 E6 E6 00 60 36 A1 09 06 07 60 85 74 05 08 01 01 8A 02 07 80 8B 07 60 85 74 05 08 02 01 AC 0A 80 08 32 32 32 32 32 32 32 32 BE 10 04 0E 01 00 00 00 06 5F 1F 04 00 00 18 19 FF FF A6 4E 7E

5.4. AARE-pdu 的编码例子

具体见报文

NS（无安全认证的 AARQ 的应答）:

7EA0300B0002FEFF30146EE6E700611FA109060760857405080101A203020101A305A10302010DBE06040040E010600E5607E

LLS 接收: 7E A0 3A 09 00 02 FE FF 30 15 8F E6 E7 00 61 29 A1 09 06 07 60 85 74 05 08 01 01 A2 03 02 01 00 A3 05 A1 03 02 01 00 BE 10 04 0E 08 00 06 5F 1F 04 00 00 18 19 01 94 00 07 45 17 7E

6. 应用层的断开释放(RLRQ 与 RLRE)

详见绿皮书 P375 页.

6.1. 报文示例

RLRQ 的 APDU,绿皮书 P455 页.

RLRE 的 APDU,绿皮书 456 页.

WRAPPER[1] Sent 000100100001000562038000100

WRAPPER[1] Rec 000100010010000563038000100

7. 读操作(Get)

分为Get. Request和Get.Response。

7.1. Get. Request

数据请求帧

7.1.1. 正常抄读:

7E A0 1C 00 02 FE FF 09 54 99 30 E6 E6 00 C0 01 C1 00 01 00 00 60 01 01 FF 02 00 32 BC 7E

C0: Get request.命令

01：requestType 请求的类别 01 表示 Normal Get,有以下三种类型

```
Get-Request ::= CHOICE
{
get-request-normal [1] IMPLICIT Get-Request-Normal,
get-request-next [2] IMPLICIT Get-Request-Next,
get-request-with-list [3] IMPLICIT Get-Request-With-List
}
/// <summary>
        /// Normal Get.
        /// </summary>
        Normal = 1,
        /// <summary>
        /// Next data block.
        /// </summary>
        NextDataBlock = 2,
        /// <summary>
        /// Get request with list.
        /// </summary>
        WithList = 3
```

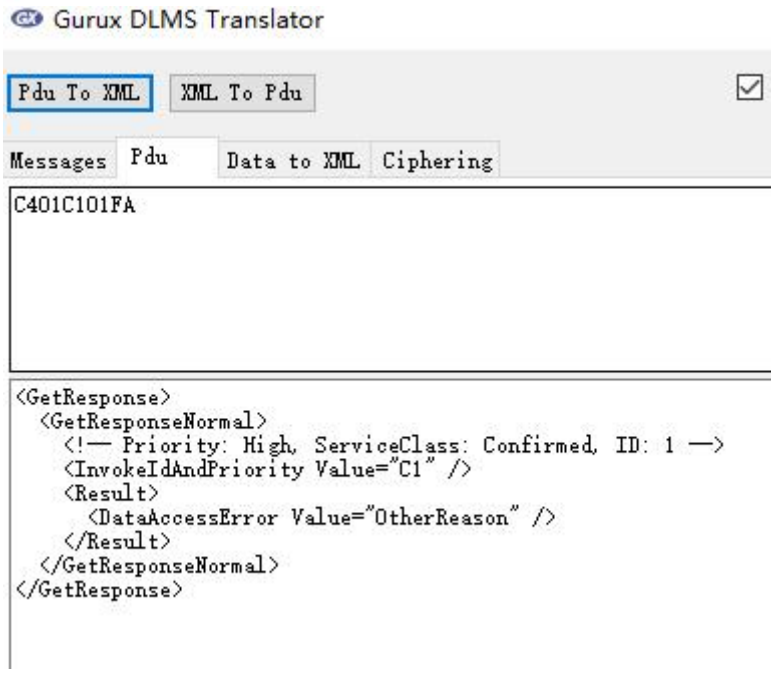
C1: Add Invoke Id And Priority. 引用 ID 与优先级，以前公司用的是 81，后都统一用 C1

7.2. Get. Response

数据响应帧。

7.2.1. 正常抄读的应答

```
Get-Response ::= CHOICE
{
    get-response-normal                [1] IMPLICIT    Get-Response-Normal,
    get-response-with-datablock        [2] IMPLICIT    Get-Response-With-Datablock
    get-response-with-list             [3] IMPLICIT    Get-Response-With-List
}

Get-Response-Normal ::= SEQUENCE
{
    invoke-id-and-priority             Invoke-Id-And-Priority,
    result                             Get-Data-Result
}
Get-Data-Result ::= CHOICE
{
   data                                    [0] Data,
   data-access-result                      [1] IMPLICIT Data-Access-Result
}
```



正常应答：7E A0 1D 09 00 02 FE FF 74 AE 2F E6 E7 00 C4 01 C1 00 0A 08 45 33 30 30 35 2D 53 41 B0 CD 7E

错误应答：7E A0 14 0B 00 02 FE FF 52 F5 22 E6 E7 00 C4 01 C2 01 FA 59 07 7E

8. 写操作(Set)

分为 Set. Request 和 Set.Response。

8.1. Set. Request

8.1.1. 写正常数据:

7E A0 26 00 02 FE FF 09 32 B4 09 E6 E6 00 C1 01 C1 00 01 00 00 60 01 01 FF 02 00 0A 08 45 33

30 30 35 2D 53 41 A6 E5 7E (0-0:96.1.1*255 Meter model E3005-SA)

C1 表明文写操作-----对应表的应答则为 C5

01:表写操作的类型:

01 表示: Writing the value of a single attribute without block transfer

04 表示: Writing the value of a list of attributes without block transfer

C1: InvokeIdAndPriority

00 01 00 00 60 01 01 FF 02: Cosem-Attribute-Descriptor,

00: Selective-Access-Descriptor OPTIONAL . 为 0.即不带 Selective-Access-Descripto, ***特别强调这个这个字节不是用来区分是方法还是写数据的标志.***

8.2. Set. Response

数据设置响应。

Set-Response ::= CHOICE

{

set-response-normal [1] IMPLICIT Set-Response-Normal,

set-response-datablock [2] IMPLICIT Set-Response-Datablock,

set-response-last-datablock [3] IMPLICIT Set-Response-Last-Datablock,

set-response-last-datablock-with-list [4] IMPLICIT Set-Response-Last-Datablock-With-List,

set-response-with-list [5] IMPLICIT Set-Response-With-List

}

写正常数据应答:

7E A0 14 09 00 02 FE FF 52 A3 2A E6 E7 00 C5 01 C1 01 0D 49 60 7E(0-0:96.1.1*255 Meter model E3005-SA 没有写成功返回一个错误)

9. 方法操作(Action)

分为 ACTION Request 和 ACTION.Response。

9.1. ACTION. Request

请求的格式绿皮书的定义为:

Action-Request ::= CHOICE

{

action-request-normal [1] IMPLICIT Action-Request-Normal,

action-request-next-pblock [2] IMPLICIT Action-Request-Next-Pblock,

action-request-with-list [3] IMPLICIT Action-Request-With-List,

action-request-with-first-pblock [4] IMPLICIT Action-Request-With-First-Pblock,

action-request-with-list-and-first-pblock [5] IMPLICIT Action-Request-With-List-And-First-Pblock,

action-request-with-pblock [6] IMPLICIT Action-Request-With-Pblock

}

以 Action-Request-Normal 为例介绍结构:

```
Action-Request-Normal ::= SEQUENCE
{
invoke-id-and-priority Invoke-Id-And-Priority,
cosem-method-descriptor Cosem-Method-Descriptor,
method-invocation-parameters Data OPTIONAL//需要特别注意此参数是可选的.无数据时为 00,有数据时先填 01 后需再跟数据.
}
```

9.1.1. 执行正常数据

action-request-normal[1]
发送:7E A0 1F 00 02 FE FF 09 32 C7 9E E6 E6 00 C3 01 C1 00 09 00 00 0A 00 01 FF 01 01 12 00 01 DF A2 7E(S16VEVF 表需量与 EOB 复位注意组帧方法与写的区别,属性 ID 与方法 ID 的位置)
特别注意红体字 01 不是执行方法的标志位.而是指方法有引用的参数,当为 00 时表没有引用的参数.

9.2. ACTION. Response

应答格式如下:

```
Action-Response ::= CHOICE
{
action-response-normal [1] IMPLICIT Action-Response-Normal,
action-response-with-pblock [2] IMPLICIT Action-Response-With-Pblock,
action-response-with-list [3] IMPLICIT Action-Response-With-List,
action-response-next-pblock [4] IMPLICIT Action-Response-Next-Pblock
}
```

9.2.1. 正常应答

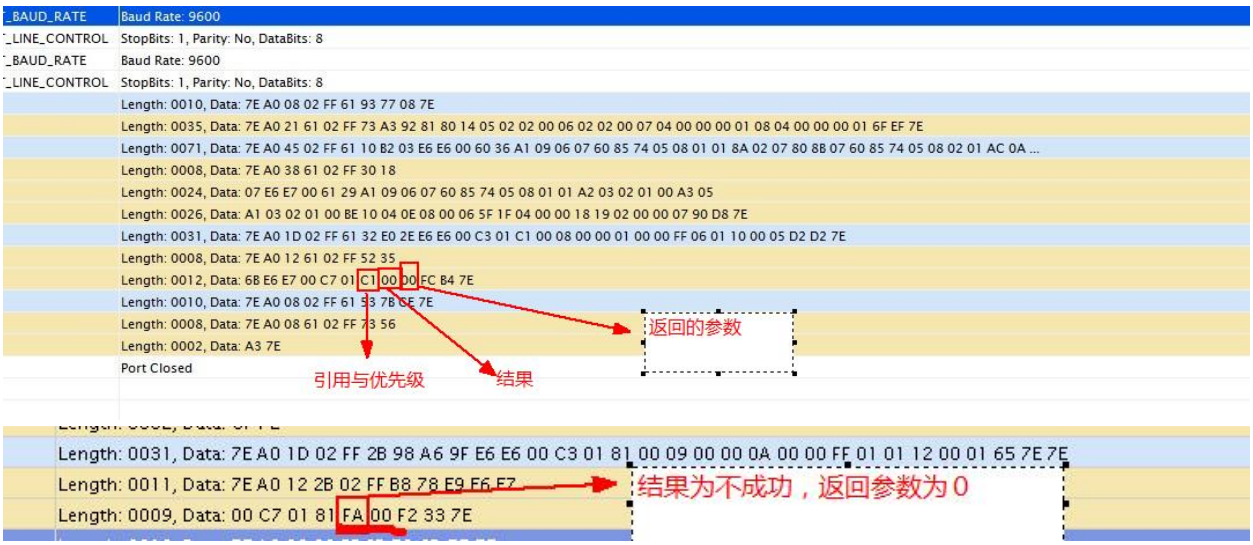
action-response-normal [1]
应答: 7E A0 14 09 00 02 FE FF 52 A3 2A E6 E7 00 C7 01 C1 00 00 FC B4 7E(注意判断是否成功.)

```
</xsd:complexType>

<xsd:complexType name="Action-Response-With-Optional-Data">
   <xsd:sequence>
      <xsd:element name="result" type="Action-Result"/>
      <xsd:element name="return-parameters" minOccurs="0" type="Get-Data-Result"/>
   </xsd:sequence>
</xsd:complexType>
```



```
<xsd:complexType name="Action-Response-Normal">
   <xsd:sequence>
      <xsd:element name="invoke-id-and-priority" type="Invoke-Id-And-Priority"/>
      <xsd:element name="single-response" type="Action-Response-With-Optional-Data"/>
   </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="Action-Response-With-Pblock">
   <xsd:sequence>
      <xsd:element name="invoke-id-and-priority" type="Invoke-Id-And-Priority"/>
      <xsd:element name="pblock" type="DataBlock-SA"/>
```

如下图的报文分析：



9.2.2. 非正常应答

如下报文：7EA0150B0002FEFF5220BDE6E700C701C10C010054717E

--Action-Response-With-Optional-Data
0C Action-Result type-unmatched (12),

---return-parameters
01 Data-Access-Result

00 success

54717E

type-unmatched (12),
scope-of-access-violated

```
Action-Response-With-Optional-Data ::= SEQUENCE
{
result       Action-Result,
return-parameters    Get-Data-Result OPTIONAL
}

Get-Data-Result ::= CHOICE
{
data [0] Data,
data-access-result [1] IMPLICIT Data-Access-Result
}
```

10. 帧类型标志

如何查找这个帧类型(命令)字节,那么如果是 PC 端发出的只需要检查 E6 E6 00 后面的一个字节即可,如果是表应答那么只需要检查 E6 E7 00 后面的一个字节即可.

命令说明	HEX 码	绿皮书第 P316 页
Glo get request.加密抄读请求	0xC8	with global ciphering
Glo get response.加密抄读应答	0xCC	绿皮书第 8-3 版第 215 页有 Table 40 – Example: glo-get-request xDLMS APDU 引用 C8
Glo set request. 加密写请求	0xC9	
Glo set response. 加密写应答	0xCD	
Glo method request.加密执行方法请求	0xCB	
Glo method response. 加密执行方法的应答	0xCF	
Get request. 抄读请求	0xC0	
Get response. 抄读应答	0xC4	
Set request. 写请求	0xC1	
Set response. 写应答	0xC5	
Action request. 执行方法请求	0xC3	
Action response. 执行方法的应答	0xC7	
AARQ request.	0x60	
AARE request.	0x61	
Disconnect request for HDLC framing.断开链路请求	0x53	
ded-get-request	0XD0	with dedicated ciphering
ded-set-request	0XD1	

ded-event-notification-request	0XD2	
ded-action-Request	0XD3	
ded-get-response	0XD4	
ded-set-response	0XD5	
ded-action-response	0XD7	

11. TCP 的格式

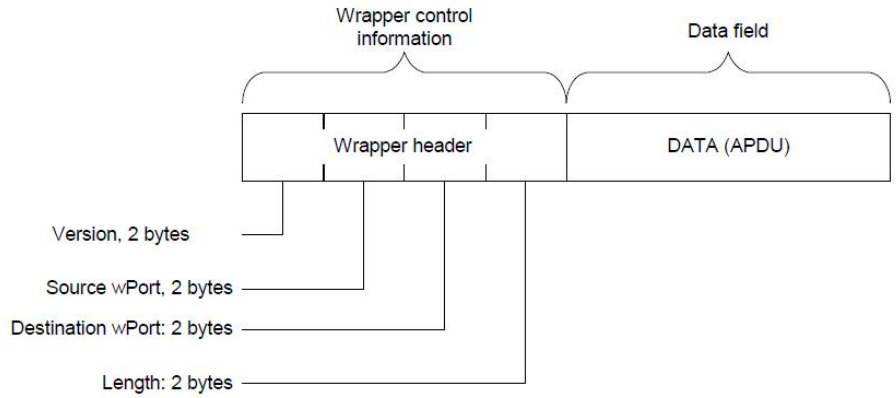
概述

TCP 格式是将 DLMS 的 APDU 加上 TCP 格式的头部组成。

7.3.3.2 The wrapper protocol data unit (WPDU)

The WPDU consists of two parts:

- the wrapper header part, containing the wrapper control information; and
- the data part, containing the DATA parameter – an xDLMS APDU – of the corresponding UDP-DATA.xxx service invocation.



NOTE The maximum length of the APDU should be eight bytes less than the maximum length of the UDP datagram.

Figure 29 – The wrapper protocol data unit (WPDU)

11.1. TCP 头格式

由 8 字节组成，依次如下：

版本：固定为 1，U16 类型

源地址：U16 类型，如果是客户端发出的则填 CLIENT 地址，如果是服务端发则填服务端地址。

目的地址：U16 类型，如果是客户端发出的则填服务端地址，如果是服务端发则填 CLIENT 地址。

数据长度：U16 类型，即要发送的 APDU 的长度。

有关地址取值见图:

7.3.3.4 Reserved wrapper port numbers (wPort)

Reserved wPort Numbers are specified in Table 2:

Table 2 – Reserved wrapper port numbers in the UDP-based DLMS/COSEM TL

Client side reserved addresses	
	Wrapper Port Number
No-station	0x0000
Client Management Process	0x0001
Public Client	0x0010
Open for client SAP assignment	0x02 ...0x0F
	0x11... 0xFF
Server side reserved addresses	
	Wrapper Port Number
No-station	0x0000
Management Logical Device	0x0001
Reserved	0x0002...0x000F
Open for server SAP assignment	0x0010...0x007E
All-station (Broadcast)	0x007F

11.2. 报文举例

PC 通过 TCP 格式发出的 AARQ:

00010004000000386036a1090607608574050801018a0207808b076085740508020 1ac0a8008323232323232323232be10040e01000000065f1f0400001819ffff

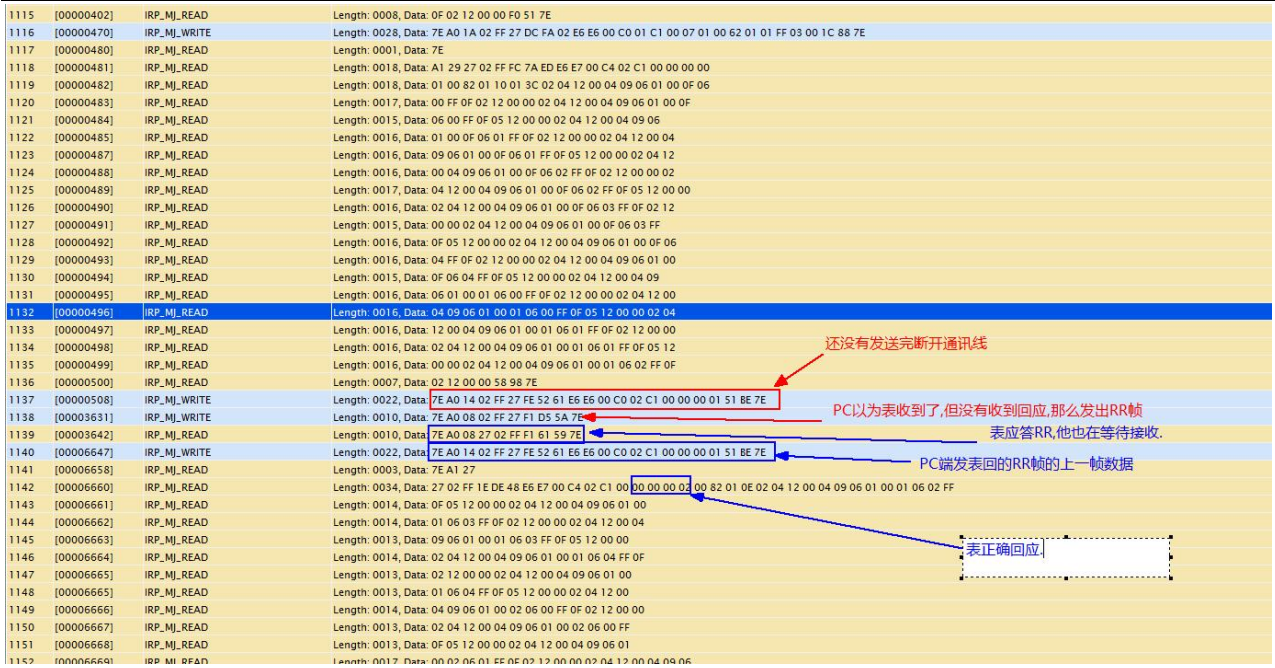
AMI 系统通过 TCP 格式发出的 AARQ:

00010011000100386036a1090607608574050801018a0207808b076085740508020 1ac0a8008323232323232323232be10040e01000000065f1f04000018190194

12. RR 帧的处理

以下示例以 Q8 表截图.

12.1. PC 发出的数据表未接收,PC 等不到表回应,发出 RR 帧



12.2. PC 发出的数据表收到, PC 等不到表的回应, 发出 RR 帧