长安汽车以太网报文封装策略

更改历史

版本	更改描述	更改日期	更改	校核
V1.0	初版	2019.10.30	黄杰	
V1.1	修改 5.1 数据报文头定义的数据类型和 5.2 分片包报文头大小，与报文头各字段的数据长度匹配	2019.12.17	黄杰	

目录

1. 目的

定义以太网报文封装策略。

2. 数据类型

基本数据类型及取值范围

类型	字节/位	取值范围
char	1/8	-128~127
unsigned char	1/8	0~255
short	2/16	-32768~32767
unsigned short	2/16	0~65535
int	4/32	-2147483648~2147483647
Unsigned int	4/32	0~4294967295

3. 主机字节序和网络字节序

不同的 CPU 有不同的字节序类型，这些字节序是指“整数”在内存中保存的顺序，叫做主机字节序。

最常见的有两种：

- Little endian：将低序字节存储在起始地址
- Big endian：将高序字节存储在起始地址

LE little-endian（小端）

- 地址低位存储值的低位
- 地址高位存储值的高位

BE big-endian（大端）

- 地址低位存储值的高位
- 地址高位存储值的低位

内存中双字 0x01020304(DWORD) 的存储方式

内存 4000 4001 4002 4003

LE 04 03 02 01

BE 01 02 03 04

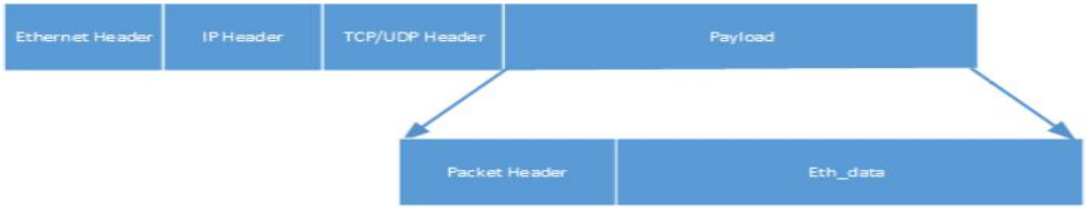
网络字节序

网络字节顺序是 TCP/IP 中规定好的一种数据表示格式，它与具体的 CPU 类型、操作系统等无关，从而可以保证数据在不同主机之间传输时能够被正确解释，网络字节序采用大端方式。

当 short 类型或者 int 类型的数据在网络上传输时，需要转换为网络字节序，接收到该类型的数据时需要转换为主机字节序。

4. 数据报文封装

将数据报文头和数据内容封装如下：



5. 报文封装策略

此处的报文封装，指的是以太网 payload 的数据封装，将典型的 CAN 信号值封装在以太网数据包中传输。

因 CAN 的带宽及负载长度有限，所以信号中是按位 bit 来进行数据封装，这样在发送端和接收端都需要按位 bit 来赋值和取值；以太网采用字节 byte 方式来进行数据封装，避免繁琐的 bit 运算，并使用数据结构的方式对应整个数据负载，赋值和取值都依据于数据结构字段。

原则：

- 信号长度小于 8bits 的，以太网中以 8bits/1byte 表示
- 信号长度大于 8bits 且小于 16bits 的，以太网中以 16bits/2bytes 表示
- 信号长度大于 16bits 且小于 32bits 的，以太网中以 32bits/4byte 表示
- 信号长度为 64bits 的，以太网中以 2 个 32bits 表示，高 32bits 在前，低 32bits 在后

CAN 信号与以太网数据长度示意表

Signal name	Chinese name	CAN Signal length bits	ETH signal length bits	Network byte order	Signal type
SAS_SteeringAngle	转向角度	16	16	Yes	short
EMS_IBATT_QUIESCENT	蓄电池传感器静态电流信号	8	8	No	Unsigned char
ABS_BrakePedalStatus	制动踏板状态	2	8	No	Unsigned char
TCU_TransFluidTemp	传动液温度	8	8	No	Unsigned char
AC_RearReqTemp	后排温度调节_自动	5	8	No	Unsigned char

5.1 数据结构定义原则

相邻的字段长度尽量以 4 字节对齐

数据结构定义：（该定义为推荐）

```
Typedef struct eth_data {
  Short sas_steeringangle,
  Unsigned char ems_ibatt_quiescent,
  Unsigned char abs_brakepedalstatus,
  Unsigned char tcu_transfluidtemp,
  Unsigned char ac_rearreqtemp,
}ETH_DATA;
```

1. 数据报文头定义

该报文头定义类似于 IP 头，主要目的是识别该数据帧的用途，携带 CRC 校验的信息等，同时也方便将来扩展。

头定义如下：

```
Typedef struct packet_header {
  Unsigned char version;
  Unsigned char crc_len
  Unsigned short packet_count;
  Unsigned int length;
  Unsigned int message_type;
  Unsigned int crc_value;
  Unsigned short frag_set;
  Unsigned short frag_offset;
}PACKET_HEADER;
```

每字段含义如下表：

<table>
<tr><th>字段名</th><th>字段长度（Bytes）</th><th>含义</th></tr>
<tr><td>version</td><td>1</td><td>版本号，默认填充 0</td></tr>
<tr><td>crc_len</td><td>1</td><td>CRC 算法的长度 8/16/32，0x01 表示 8bit、0x10 表示 16bit、0x11 表示 32bit</td></tr>
<tr><td>packet_count</td><td>2</td><td>帧计数器，0 至 FFFF 循环计数</td></tr>
<tr><td>length</td><td>4</td><td>数据帧的字节长度</td></tr>
<tr><td>message_type</td><td>4</td><td>数据帧的数据类型，根据 message_type 的值，
可区分数据包内容，比如 100ms 还是 200ms 的数据</td></tr>
<tr><td>crc_value</td><td>4</td><td>该帧的 crc 值，目前默认使用 32bit 算法，算法如下：
<table>
<tr><td>CRC result width:</td><td>32 bits</td></tr>
<tr><td>Polynomial:</td><td>F4’AC’FB’13h</td></tr>
<tr><td>Initial value:</td><td>FFFFFFFFh</td></tr>
<tr><td>Input data reflected:</td><td>Yes</td></tr>
<tr><td>Result data reflected:</td><td>Yes</td></tr>
<tr><td>XOR value:</td><td>FFFFFFFFh</td></tr>
</table>
</td></tr>
</table>

frag_set	2	分片标志位，1 为分片，0 为未分片包
frag_offset	2	分片包偏移量，需除以 8，最大支持 512k 分片， 在定义每包最大长度时，其长度需是 8 的整数倍

5.2 分片包处理流程

如果发送数据的长度+包头长度超过 MAX_PACKET_SIZE，就需要分包，否则直接发送。当需要分包处理时，需定义最大包大小 MAX_PACKAGE_SIZE，该值需是 8 的整数倍，比如定义为 1472.

举例说明如下：

发送 1000byte 的数据封装方式：

message_type	length	frag_set	frag_offset	payload	comments
0x100	1000	0	0	1000 字节信息	

发送 4000 字节的数据封装方式

MAX_PACKAGE_SIZE 为 1472，包头大小为 20，所以每包 payload 大小为 1452，4000 字节需要拆分为 3 个包来发送

第一个包：

message_type	length	frag_set	frag_offset	payload	comments
0x200	1452	1	0	1452 字节信息	

第二个包：

message_type	length	frag_set	frag_offset	payload	comments
0x200	1452	1	181	1452 字节信息	

第三个包：

message_type	length	frag_set	frag_offset	payload	comments
0x200	1096	0	362	1096 字节信息	