工控预警系统驱动开发报告

—以 s7comm 协议为例

0. 前言

工控预警系统后端驱动在网络协议层面的开发主要分为三个模块，如图 1

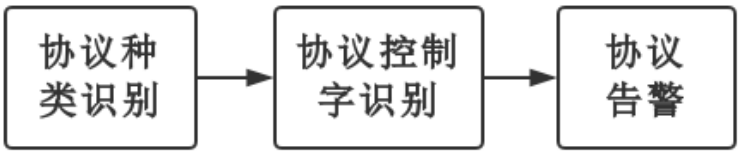


图 1 工控预警系统后端模块

所示，由低到高分别是协议种类识别、协议控制字识别和协议告警。

本系统**可识别**的工控协议有 12 种：

西门子 s7comm（西门子 PLC 系列协议）

MODBUS（施耐德系列协议）

ETHERNET/IP（罗克韦尔系列协议）

DNP3.0（分布式网络协议，主要用于电力行业）

IEC104（输配电通讯协议）

OMRON（欧姆龙 PLC 协议）

ESIO

S-BUS

BACNET（楼宇自动控制网络数据通讯协议）

COAP（轻量应用层协议）

RTPS（实时流传输协议）

HARTIP（高速可寻址远程传感器协议）

其中可**深度解析**的工控协议有 7 种：

西门子 s7comm（西门子 PLC 系列协议）

MODBUS（施耐德系列协议）

DNP3.0（分布式网络协议，主要用于电力行业）

IEC104（输配电通讯协议）

OMRON（欧姆龙 PLC 协议）

RTPS（实时流传输协议）

HARTIP（高速可寻址远程传感器协议）

需要对它们进行完整三步操作。本报告**以 s7comm 协议为例**，分步骤说明详细开发过程。

1. 协议种类识别

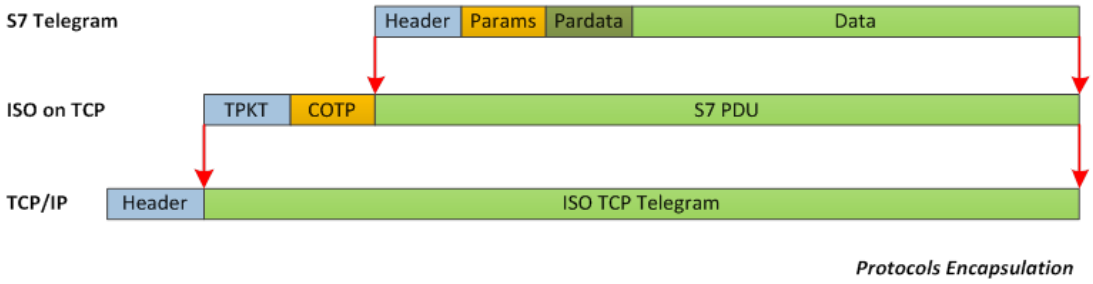


图 2 s7comm 协议层次

S7comm 协议的层次如图 2 所示，一条完整的 s7comm 报文是由 tcp/ip+tpkt+cotp+s7comm 协议组成。若 cotp 层之后报文**第一位等于 0x32**，则该报文属于 s7comm 协议。

2. 协议控制字识别

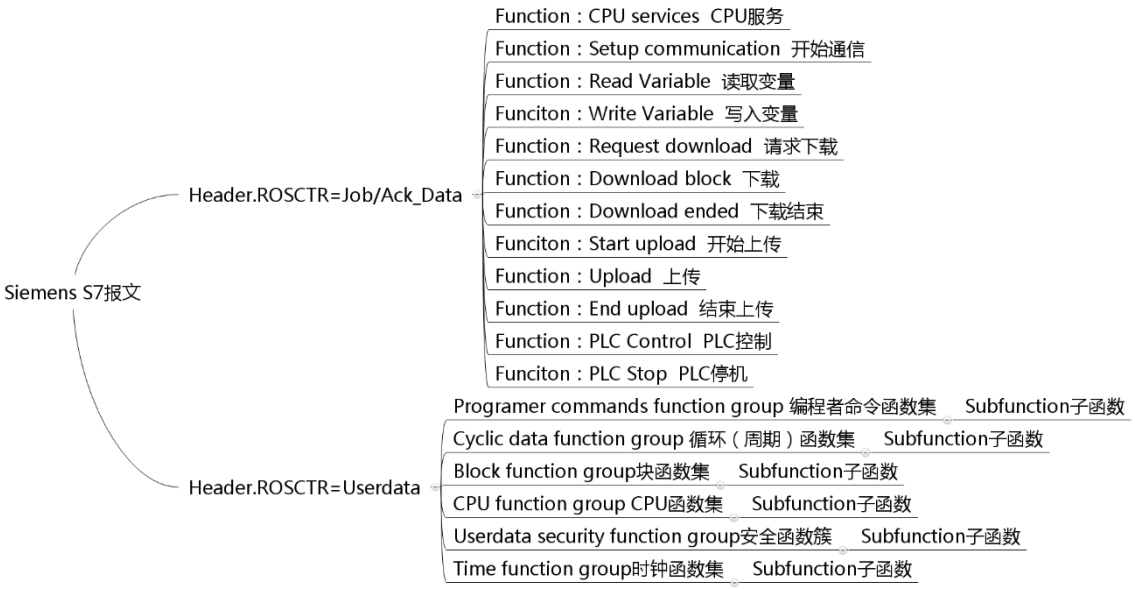


图 3 s7comm 功能码分类

如图所示，s7comm 报文由 Header.ROSCTR 字段的不同，可分类两大类。

① 第一类：

首段

字段	协议编号	远程操作服务控制	冗余识别	参考码	参数段长度	数据段长度	错误类型	错误代码
位数	1	2	3-4	5-6	7-8	9-10	11	12

参数段

字段	函数	未知 1	错误	未知 2	参数
位数	13	14	15-16	×	×

报文结构如图所示，根据第 13 位的数值可以判断控制字（功能码），不同的数值代表了不同的功能码，罗列如下：

0x00 - CPU services　CPU 服务

0xF0 - Setup communication　通信设置

0x04 - Read Variable　读取变量

0x05 - Write Variable　写入变量

0x1A - Request download　请求下载

0x1B - Download block　下载块

0x1C - Download ended　下载结束

0x1D - Start upload　开始上传

0x1E – Upload　上传

0x1F - End upload　结束上传

0x28 - PLC Control　PLC 控制

0x29 - PLC Stop　PLC 停机

②第二类：

首段

字段	协议编号	远程操作服务控制	冗余识别	参考码	参数段长度	数据段长度
位数	1	2	3-4	5-6	7-8	9-10

参数段

字段	参数段首部	参数长度	请求/应答	类型	函数簇	子函数	序列号	是否为最后一个数据单元	错误代码
位数	11-13	14	15	16		17	18	19	20-21

报文结构如图所示，根据第 13 位的数值可以判断控制字（功能码）和子功能码，不同的数值代表了不同的功能码和子功能码，功能码罗列如下：

0x1	Programer commands function group	编程者命令函数簇
0x2	Cyclic data function group	循环（周期）函数簇
0x3	Block function group	块函数簇
0x4	CPU function group	CPU 函数簇
0x5	Userdata security function group	安全函数簇
0x6	Time function group	时钟函数簇

子功能码罗列如下：

Programer commands function group	Cyclic data function group	Block function group	CPU function group	Security function group	Time function group
0x01 - Request diag data (Type 1)	0x01 - Memory	0x01 - List blocks	0x01 - Read SZL	0x01 - PLC password	0x01 - Read clock
0x02 - VarTab	0x04 - Unsubscribe	0x02 - List blocks of type	0x02 - Message service		0x02 - Set clock
0x0c - Erase		0x03 - Get block info	0x03 - Transition to stop		0x03 - Read clock (following)
0x0e - Read diag data			0x0b - Alarm was acknowledged in HMI/SCADA 1		0x04 - Set clock
0x0f - Remove diag data			0x0c - Alarm was acknowledged in HMI/SCADA 2		
0x10 - Forces			0x11 - PLC is indicating a ALARM message		
0x13 -			0x13 - HMI/SCADA		

Request diag data (Type2)			initiating ALARM subscription		

3. 协议告警

3.1 告警等级的划分

协议告警等级共有 4 级，分别是 1 级、2 级、3 级、4 级，其中 4 级级别最高，1 级级别最低。等级划分的原则是：

1 级：建立连接等

2 级：读取数据、读取寄存器等

3 级：读取控制程序、读取设备标示和状态、写入数据等

4 级：写入控制程序、重启设备、停机等

依据上述原则，将 s7comm 的控制字做如下分级：

1 级

Setup communication

2 级

0x04 - Read Variable

S7COMM_UD_FUNCGROUP_TIME-> S7COMM_UD_SUBF_TIME_READ 0x01

S7COMM_UD_FUNCGROUP_TIME-> S7COMM_UD_SUBF_TIME_READF

S7COMM_UD_FUNCGROUP_TIME-> S7COMM_UD_SUBF_TIME_SET

S7COMM_UD_FUNCGROUP_CYCLIC->S7COMM_UD_SUBF_CYCLIC_MEM 0x01

S7COMM_UD_FUNCGROUP_CYCLIC->S7COMM_UD_SUBF_CYCLIC_UNSUBSCRIBE

S7COMM_UD_FUNCGROUP_PROG->S7COMM_UD_SUBF_PROG_REQDIAGDATA

S7COMM_UD_FUNCGROUP_PROG->S7COMM_UD_SUBF_PROG_VARTAB1

S7COMM_UD_FUNCGROUP_PROG->S7COMM_UD_SUBF_PROG_ERASE

S7COMM_UD_FUNCGROUP_PROG->S7COMM_UD_SUBF_PROG_READDIAGDATA

S7COMM_UD_FUNCGROUP_PROG->S7COMM_UD_SUBF_PROG_REMOVEDIAGDATA

S7COMM_UD_FUNCGROUP_PROG->S7COMM_UD_SUBF_PROG_FORCE

S7COMM_UD_FUNCGROUP_PROG->S7COMM_UD_SUBF_PROG_REQDIAGDATA2

3 级

CPU services

Write Variable

Start upload

Upload

End upload

S7COMM_UD_FUNCGROUP_SZL-> S7COMM_UD_SUBF_SZL_READ

S7COMM_UD_FUNCGROUP_SZL-> S7COMM_UD_SUBF_SZL_ASMESS

S7COMM_UD_FUNCGROUP_BLOCK-> S7COMM_UD_SUBF_BLOCK_LIST

S7COMM_UD_FUNCGROUP_BLOCK -> S7COMM_UD_SUBF_BLOCK_LISTTYPE

S7COMM_UD_SUBF_BLOCK_BLOCKINFO

4 级

Request download

Download block

Download ended

PLC Control PLC

PLC Stop PLC

S7COMM_UD_FUNCGROUP_SEC-> S7COMM_UD_SUBF_SEC_PASSWD

3.2 告警函数编写

3.2.1 json 结构体的遍历

协议的告警，是指依据 3.1 节中规定的控制字与告警等级的关系，遍历 commandlist 中的所有 command，以确定每个 command 的告警等级。其中的所有 command 中最高的告警等级，就确定为这条流的告警等级。

s7comm 协议告警函数名称为 alert_s7comm，它的输入量是 flow->commandlist 结构体，它是由协议控制字识别函数中断地用 json 添加内容

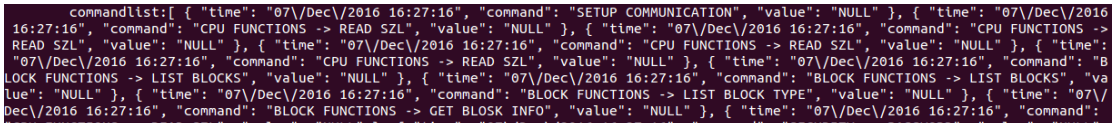
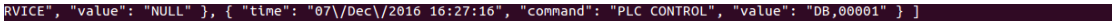


图 4 结构体 commandlist 打印输出结果



得到的，这个结构体的打印结果如图所示。

它的元素用表格表示如表 1 所示。

“time”	“command”	“value”
"07\/Dec\/2016 16:27:16"	"SETUP COMMUNICATION"	"NULL"
"07\/Dec\/2016 16:27:16"	"CPU FUNCTIONS -> READ SZL"	"NULL"
"07\/Dec\/2016 16:27:16"	"CPU FUNCTIONS -> READ SZL"	"NULL"
……	……	……

由 json 的定义可以知道整个 commandlist 是一个 array，第一行中的“time”、“command”、“value”是三个 key，commandlist 中除第一行外的每一行是一个 object，每个 object 都有自己的 index（从 0 开始编号）。

通过查阅 json_object 的函数说明手册（http://json-c.github.io/json-c/json-c-0.10/doc/html/json__object_8h.html），查到 json_object_array_length、json_object_array_get_idx、json_object_to_json_string 三个函数，这些函数的说明如下。

int **json_object_array_length** (struct json_object * obj)

功能：读取 json_object * obj 结构体的长度

Get the length of a json_object of type json_type_array

Parameters:

obj the json_object instance

Returns:

an int

struct **json_object* json_object_object_get**(struct json_object * obj,const char * key)

功能读取 json_object * obj 结构体中 key 对应的子结构体

Get the json_object associate with a given object field

Parameters:

obj the json_object instance

key the object field name

Returns:

the json_object associated with the given field name

struct json_object* **json_object_array_get_idx** (struct json_object * obj,int idx)

读取 json_object * obj 结构体中第 idx 个子结构体

Get the element at specificed index of the array (a json_object of type json_type_array)

Parameters:

obj the json_object instance

idx the index to get the element at

Returns:

the json_object at the specified index (or NULL)

const char* **json_object_to_json_string**(struct json_object * obj)

将 json_object * obj 转换成字符串格式

Stringify object to json format. Equivalent to json_object_to_json_string_ext(obj, JSON_C_TO_STRING_SPACED)

Parameters:

obj the json_object instance

Returns:

a string in JSON format

用这三个 json 函数，遍历结构体 commandlist 中 command 的算法框图如图所示。

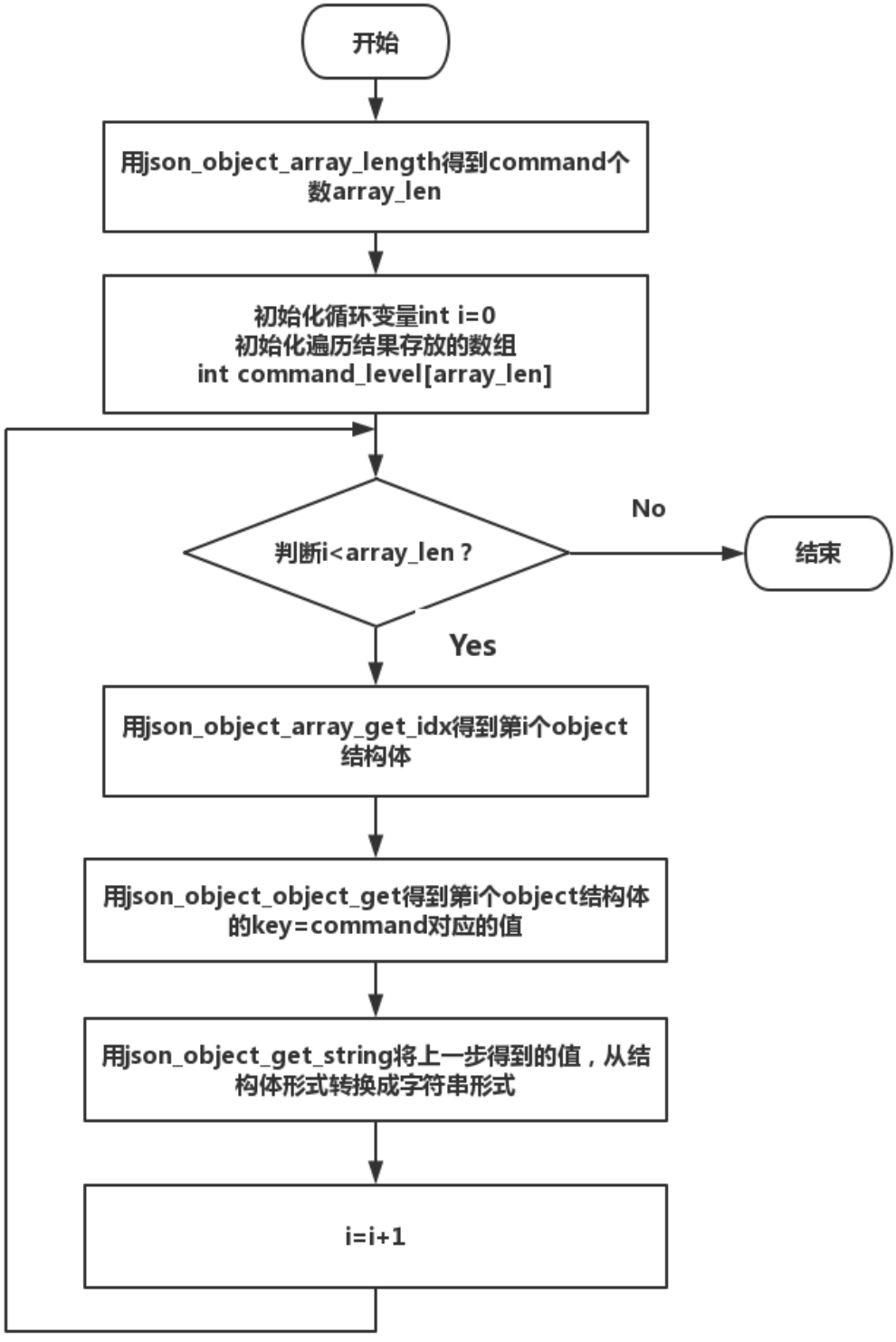


图 5　遍历结构体 commandlist 中 command 的算法框图

3.2.2 字符串匹配的判断

在 3.2.1 节中得到存储 command 的字符串 char *command_char，然后用字

符串匹配函数 strcmp 将它和所有控制字进行比较，依据比较结果赋予告警等级（1234），存放在 command_level 数组中。最后将判断 command_level 数组中，最大的元素，这个最大的元素就是整条流的告警等级。

附录 s7comm 告警函数

```
void alert_s7comm(flow->commandlist)
    {
        int    array_len = json_object_array_length(flow->commandlist);
        //printf("length======================%d\n",array_len );
        int i = 0;
        int command_level[array_len];

        for(i=0;i<array_len;i++)
        {
          struct json_object* instance_obj =
json_object_array_get_idx(flow->commandlist,i);
          char *instance_obj_rsp = json_object_to_json_string(instance_obj);
          //printf("==%s\n",instance_obj_rsp);
          struct json_object* instance_obj_command =
json_object_object_get(instance_obj,"command");
          //char *instance_obj_command_rsp =
json_object_to_json_string(instance_obj_command);
          //printf("command:%s\n",instance_obj_command_rsp);
          char*command_char=json_object_get_string(instance_obj_command);
          //printf("%s\n",command_char);

          if(!(strcmp(command_char,"REQUEST DOWNLOAD")
    &strcmp(command_char,"DOWNLOAD BLOCK")
    &strcmp(command_char,"DOWNLOAD ENDED")
    &strcmp(command_char,"PLC CONTROL")
    &strcmp(command_char,"PLC STOP")))
          {
              command_level[i] = 4;
```

```
        }
            else
            {
                if(!(strcmp(command_char, "CPU SERVICE")
    &strcmp(command_char,"WRITE VARIABLE")
    &strcmp(command_char,"START UPLOAD")
    &strcmp(command_char,"UPLOAD")&strcmp(command_char,"END
UPLOAD")
    &strcmp(command_char,"CPU FUNCTIONS -> READ SZL")
    &strcmp(command_char,"BLOCK FUNCTIONS -> LIST BLOCKS")
    &strcmp(command_char,"BLOCK FUNCTIONS -> LIST BLOCK TYPE")
    &strcmp(command_char,"BLOCK FUNCTIONS -> GET BLOSK INFO")))
                {
                   command_level[i]=3;
                }
                else
                {
                   if(!(strcmp(command_char, "SETUP COMMUNICATION")))
                   {
                      command_level[i] = 1;
                   }
                   else
                   {
                      command_level[i] = 2;
                   }
                }
            }
            //printf("%d\n",command_level[i]);
            }
```

```
            //command_level[i] sorting begin
            int command_level_high = command_level[0];
            if(array_len!=0)
            {
               for(i=1;i<array_len;i++)
               {

if(command_level[i]>command_level_high)command_level_high=command_level[i]
;
               }
            }
            else
            {
               command_level_high=0;
            }
            //command_level[i] sorting end
            printf("%d\n",command_level_high);
            return command_level_high;
        }
    }
```