INSTITUTO POLITÉCNICO NACIONAL
ESCUELA SUPERIOR DE CÓMPUTO

Redes de Computadoras

"Mi Trama"

Abstract

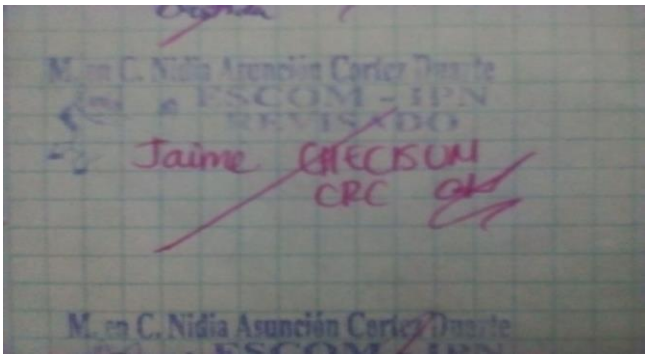
This is the report of a practice in which we build a completely new protocol in class named "Mi Trama", here we analyse and explain how we construct it.

By:

Bastida Prado Jaime Armando

Professor:
MSc. NIDIA ASUNCIÓN CORTEZ DUARTE

June 2017



Index

Contenido

Introduction:

This practice is about developing a protocol from scratch, defining its fields, codes to be used, in which language the frames can be recognized, in which way the information will be send, which mechanisms to use in error control, which in flow control, type of routing and line codes. This is a highly pedagogical manner in which students can learn how a protocol works.

Literature review:

First, we define the **Language** (2 bits) supported by the protocol these are:

Code	Language
00	French
01	German
10	English
11	Spanish

Table 1: Language codes

The **ID's** (16 bits) of the groups of users that will be using the protocol:

Code	Group	Number of Students
00	2CM8	41
01	3CM5	34
10	4CV7	17

Table 2: ID's codes

Types of routing (2 bits):

Code	Routing
00	Static
01	Static
10	Dijkstra
11	Bellman Ford

Table 3: Routing codes

Flow Control (2 bits):

Code	Control
00	Stop/Wait
01	Stop/Wait
10	Go Back N
11	Selective Reject

Table 4: Flow Control codes

Error Control (2-4 Bits):

Code	Control
000?	Parity Bit Even
001?	Parity Bit Odd
010	CRC (1 byte)
011	CRC (2 bytes)
10	Checksum (2 bytes)

Table 5: Error Control codes

Communication Form (3 bits)**:**

Code	Form
100	Coaxial
101	Twisted Pair
110	FO
111	Telephonic
011	Bluetooth
010	Infrared
001	Wi-Fi

Table 6: Communication Form codes

Line Code (8 bits), this is a special case because the analysis will be done checking which bits are On in the sequence of the byte:

0/1	RZ
0/1	NRZ
0/1	Single-Pole
0/1	Polar
0/1	AMI
0/1	Pseudo ternary
0/1	Manchester
0/1	M. differential

Table 7: Line Code codes

The **Length** (6 bits) of our protocol is the total length of only the message. It will be given in words of 16 bits for example, if the value of this field is 6 then the length of the message will be: 6 x 2 = 12 bytes.

The header of our protocol looks like this:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Language		**Length**						**Source ID**							
Destination ID								**Message (Var)**							
Routing		**Flow Control**		**Error Control (Var)**				**Communication F.**			**0**	**0**	**0**	**0**	**0**
Line Code															

Figure 1: Mi Trama Header

Here is an example of how a frame could look like, with the following characteristics:

Source ID: Student no. 34, Group 2CM8

Destination ID: Student no. 5, Group 3CV5

Error Control: Parity Bit Even

Communication Form: Wi-Fi

Flow Control: Selective Reject

Routing: Dijkstra

Line Code: Polar, RZ

Message: Hello!

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	0	0	0	0	0	1	1	0	0	1	0	0	0	1	0
0	1	0	0	0	1	0	1	0	1	0	0	1	0	0	0
0	1	0	0	0	1	0	1	0	1	0	0	1	1	0	0
0	1	0	0	1	1	0	0	0	1	0	0	1	1	1	1
0	0	1	0	0	0	0	1	1	0	1	1	0	0	0	1
0	0	0	0	1	0	0	1	0	0	0	0				

Language
Length
Source ID
Destination ID
Message
Routing
Flow Control
Error Control
Line Code

Figure 2: Example of a header using Protocol "Mi Trama"

Although, more conventionally the message can be seen in hex form:

0x83224548454C4C421B12090

Notes:

- Error Control is variable going from 2 to 4 bits, padding with zeros when necessary.
- Communication Form always occupies 3 bits and the next 5 bits are always padded with zero in order to complete a byte.
- Length value cannot be less than 0.

Software (libraries, packages, tools):

The equipment used:

-Laptop

-Pencil

-Sheets

-The program was codified in C language using text editor Sublime Text 3

-Compiler: GNU (GCC) compiler.

Procedure:

We are going to use the same example as in [*Figure 2],* to explain how the implementation of the protocol “Mi Trama” works:

```
10    BYTE t[] = {0x83, 0x22, 0x45, 0x48, 0x45, 0x4C, 0x4C, 0x4F, 0x21, 0xB8, 0x20, 0x90}; //Mayúsculas
```

Figure 3: Example of frame using protocol “Mi Trama”

We will use the type “BYTE” which is just an unsigned char, this is because we need to save as much memory as possible and to make the calculus more efficient and easy.

Now define an array of bytes that will contain the languages in string form, the same with Group, routing, and flow control.

```
12    BYTE idioma[][4] = {"Fr", "Ale", "Ing", "Esp"};
13    BYTE grupo[][5] = {"2CM8", "3CM5", "4CV7"};
14    BYTE enrutamiento[][13] = {"Estatico", "Estatico", "Dijstra", "Bellmau Ford"};
15    BYTE control_flujo[][18] = {"Parar/Esperar", "Parar/Esperar", "Retroceder N", "Rechazo Selectivo"};
```

Figure 4: Arrays to be used in the program

As I said before, we try to save as much memory as possible, to do so we try to use as few variables as we can but sometimes the revenue of using one, worth it. The example applies with the variable "tam" that holds the length of the protocol, since we need to check for that value several times along the program.

```
16      BYTE tam = (t[0] & 63) * 2;
```

Figure 5: Variable "tam"

The mechanism to obtain the length is:

- t[0] holds the value of the length but only in the 6 less significative bits of that byte.
- To obtain only the value of those 6 bits, do an AND operation with t[0] and 63 which looks like this: 00**111111,** in binary form. That operation will return only the value of those 6 bits not the entire byte.
- Multiply that value by 2 since field **Length** is given in words of 16 bits as mentioned before.

The next 4 variables are just of support to use them in iterations:

```
17      BYTE i = 0, j = 0, k = 0, l = 0;
```

Figure 6: Iteration variables

The last variables are used to calculate checksum, parity bit and CRC and are going to be explained later.

```
18      BYTE contador_unos;
19      BYTE px = 0x07;
20      BYTE crc = 0;
21      BYTE nat_begin = 1;
22      unsigned short checksum = 0, temp = 0;
```

Figure 7: Support variables

Language code is into the first 2 bits (more significative) of t[0] so, to obtain that value right shift t[0] six positions, for example:

t[0] = 0x83 = **10**000011 >> 6 = 00000**10**

00000**10** is equal to 2 in decimal value so, when we write "idioma[t[0] >> 6]" it is equal to "t[2]" which is the string "Ing" that means English language.

```
24      printf("Idioma: %s\n", idioma[t[0] >> 6]);
```

Figure 8: Printing language

Now, printing the values of the **Length, Source ID and Destination ID** is easy

```
25      printf("Tamano: %d bytes\n", tam);
26      printf("IDo: %s No.%d\n", grupo[t[1] >> 6], t[1] & 63);
27      printf("IDd: %s No.%d\n", grupo[t[2] >> 6], t[2] & 63);
```

Figure 9: Printing Length, and ID's

To print **Length** just print the variable "tam". To print the group we need to obtain the code of the group which is held into the first two most significative bits of t[1] (Source) and t[2] (Destination), to do so just right shift said bytes six positions, for example:

t[1] = 0x22 = **00**100010 >> 6 = 00000**00**

The code is **00** what is equal to just 0 in decimal value, so when we write "grupo[t[1] >> 6]" this is equal to "grupo[0]" which is the string "2CM8". The same process applies to the Destination ID.

Message is always going to be printed from t[3] to t[tam + 3], so write an iteration as the following one:

```
29      for(i = 3; i < tam + 3; i++)
30          printf("%c", t[i]);
```

Figure 10: Printing the Message

Note that we write "%c" not "%s" since we are writing byte by byte, not a single string.

From now on the position of the remaining fields is going to be conditioned by the length of the message, for example **Routing** will held the "3 + tam" position with **Flow and Error Control**. **Communication Form** the "4 + tam" position and **Line Code** the "5 + tam position". The same principle of printing from the previous fields will apply with the next ones.

```
31      printf("\nEnrutamiento: %s\n", enrutamiento[t[tam + 3] >> 6]);
32      printf("Control de Flujo: %s\n", control_flujo[(t[tam + 3] >> 4) & 3]);
33      printf("Control de Error: ");
```

Figure 11: Printing Routing and Flow Control

To check which of the mechanisms of Error Control we need to check for the values given in [Table 5], here we check for the fifth and sixth bits (from most significative to less) of t[3 + tam], if the value is 0 then Parity Bit, if 1 then CRC or if 2 Checksum. In this case t[3 + tam] = 0xB8 = 1011**10**00 which after the next operation:

```
34      switch((t[tam + 3] >> 2) & 3)
35      {
36          case 0:
```

Figure 12: Checking for Error Control

Will throw the value 2 in decimal notation, which corresponds with the case of Checksum according to what is given in [Table 5].

Checksum value is a 16 bit one's complement checksum of the "Mi Trama" header from **Language to Routing, Flow Control and Error Control** fields, this means from i = 0 to i = 3 + tam. The implementation of Checksum, CRC and Parity Bit goes beyond the limits of the explanation of this practice.

```
124         case 2:
125             printf("CHECKSUM\n");
126             checksum = 0;
127             temp = 0;
```

Figure 13: Case Checksum

```
140             printf("Checksum: %4X\n", checksum);
141             for(i = 2; i < 4 + tam; i++)
142             {
143                 for(j = 128; j > 0; j >>= 1)
```

Figure 14: Checksum For

Note that we use "%4X", the "X" to print the checksum in its hex form and "4" to pad with spaces when necessary.

Results

The result of running our program with the frame given in [Figure 3] are shown in console:

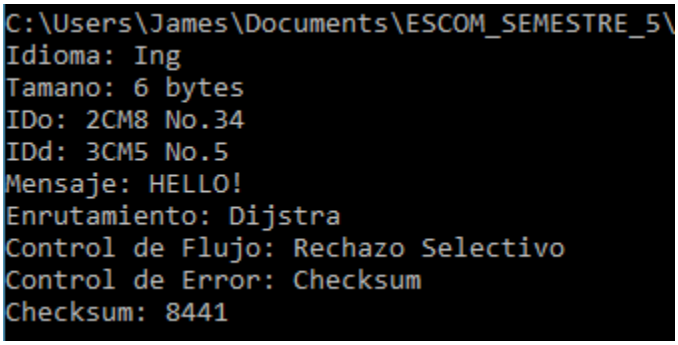


Figure 15: Result of example given in [Figure 3]

If we change the value of the byte t[3 + tam] to 0xB1, **Error Control** will be Parity Bit Even:

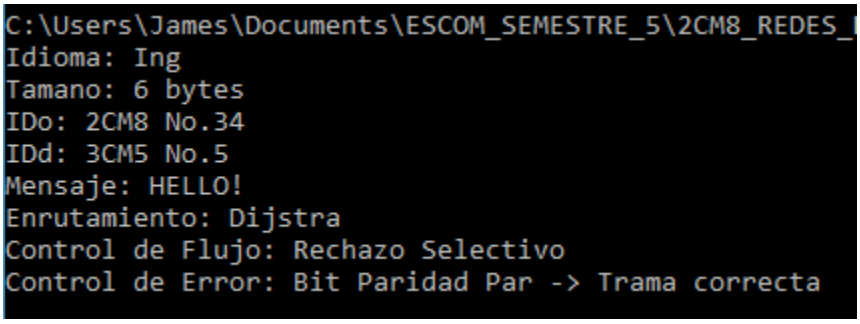


Figure 16: Result when Error Control is Parity Bit Even

If we change t[3 + tam] to 0xB2, **Error Control** will be Parity Bit Odd:

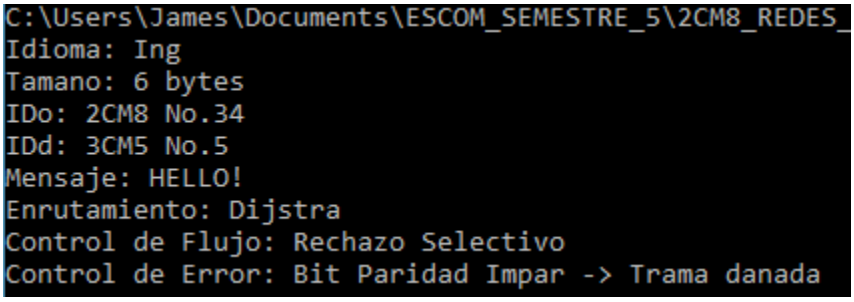


Figure 17: Result when Error Control is Parity Bit Odd

And finally If we change t[3 + tam] to 0xB4, we have CRC-8. In this case only CRC-8 was implemented:

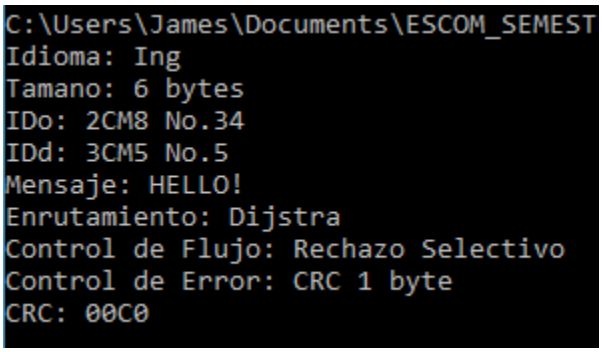


Figure 18: Result when Error Control is CRC-8

Discussion:

The results of the practice are quite considerable, maybe some people can find it easy to implement and some others difficult depending on the level of expertise managing C binary operators, especially with the algorithms of **Error Control.** Despite that facts the process of developing this program is of highly importance since many or at least me, find it very appropriate to really understand how a protocol works and even more important that any person can implement its own.

Conclusions:

I learned through this practice the importance of protocols in the Web, why they are implemented in the way they are, how to implement them, how to understand any of them just by analyzing its headers and how to use binary operators in C language.

Any or all the previous things I mentioned can be applied to any kind of application, program, software that needs a connection with other devices or the Internet.

References:

K.N. King, C Programming A Modern Approach Second Edition. W.W. Norton, 2008.

Code

```c
#include <stdio.h>

typedef unsigned char BYTE;

BYTE ones_counter(BYTE t[], BYTE size);

int main(void)
{
    //BYTE t[] = {0x83, 0x22, 0x45, 0x48, 0x65, 0x6C, 0x6C, 0x6F, 0x21, 0xB8, 0x20, 0x90}; //Minúsculas
    BYTE t[] = {0x83, 0x22, 0x45, 0x48, 0x45, 0x4C, 0x4C, 0x4F, 0x21, 0xB4, 0x20, 0x90}; //Mayúsculas
    //BYTE t[] = {0x81, 0x22, 0x45, 0x48, 0x45, 0xB4, 0x20, 0x90}; //Prueba mensaje "HE"
    BYTE idioma[][4] = {"Fr", "Ale", "Ing", "Esp"};
    BYTE grupo[][5] = {"2CM8", "3CM5", "4CV7"};
    BYTE enrutamiento[][13] = {"Estatico", "Estatico", "Dijstra", "Bellmau Ford"};
    BYTE control_flujo[][18] = {"Parar/Esperar", "Parar/Esperar", "Retroceder N", "Rechazo Selectivo"};
    BYTE tam = (t[0] & 63) * 2;
    BYTE i = 0, j = 0, k = 0, l = 0;
    BYTE contador_unos;
    BYTE px = 0x07;
    BYTE crc = 0;
    BYTE nat_begin = 1;
    unsigned short checksum = 0, temp = 0;

    printf("Idioma: %s\n", idioma[t[0] >> 6]);
    printf("Tamano: %d bytes\n", tam);
    printf("IDo: %s No.%d\n", grupo[t[1] >> 6], t[1] & 63);
    printf("IDd: %s No.%d\n", grupo[t[2] >> 6], t[2] & 63);
    printf("Mensaje: ");
    for(i = 3; i < tam + 3; i++)
        printf("%c", t[i]);
    printf("\nEnrutamiento: %s\n", enrutamiento[t[tam + 3] >> 6]);
    printf("Control de Flujo: %s\n", control_flujo[(t[tam + 3] >> 4) & 3]);
    printf("Control de Error: ");
    switch((t[tam + 3] >> 2) & 3)
    {
        case 0:
            printf("Bit Paridad ");
            if(t[tam + 3] & 2)
            {
                printf("Impar -> ");
                contador_unos = ones_counter(t, 4 + tam);
                if(!(contador_unos & 1))
                    printf("Trama danada\n");
                else
                    printf("Trama correcta\n");
            }
            else
            {
                printf("Par -> ");
                contador_unos = ones_counter(t, 4 + tam);
                if(contador_unos & 1)
                    printf("Trama danada\n");
                else
                    printf("Trama correcta\n");
```

```
            }
        break;
        case 1:
            printf("CRC ");
            if(t[tam + 3] & 2)
            {
                printf("2 bytes\n");
            }
            else
            {
                printf("1 byte ");
                crc = 0;
                l = 128;
                for(i = 0; i < 4 + tam; i++)
                {
                    for(j = l, k = 0; j > 0; j = j >> 1)
                    {
                        if(crc & 128)
                        {
                            if(t[i] & j)
                                crc = (crc << 1) | 1;
                            else
                                crc = crc << 1;
                            k++;

                            crc = crc ^ px;
                            //printf("CRC: %d\n", crc);
                        }
                        else
                        {
                            if(t[i] & j)
                                crc = (crc << 1) | 1;
                            else
                                crc = crc << 1;
                            k++;
                            l = 128;

                            if((i == (3 + tam)) && k == 8)
                                break;

                            if((crc & 128) && ((j >> 1) == 0))
                            {
                                if(t[i + 1] & 128)
                                    crc = (crc << 1) | 1;
                                else
                                            crc = crc << 1;

                                        crc = crc ^ px;
                                        //printf("CRC: %d\n", crc);
                                        l = 64;
                                    }
                                    else if((crc & 128) && ((j >> 1) != 0))
                                    {
                                        j = j >> 1;
                                        if(t[i] & j)
                                            crc = (crc << 1) | 1;
                                        else
                                            crc = crc << 1;
                                        k++;

                                        crc = crc ^ px;
```

```
                                        //printf("CRC: %d\n", crc);
                                    }
                                }
                            }
                        }
                        printf("\nCRC: %.4X\n", crc);
                    }
                break;
                case 2:
                    printf("Checksum\n");
                    checksum = 0;
                    temp = 0;

                    for(j = 128; j > 0; j >>= 1)
                        if(t[0] & j)
                            checksum = (checksum << 1) | 1;
                        else
                            checksum <<= 1;
                    for(j = 128; j > 0; j >>= 1)
                        if(t[1] & j)
                            checksum = (checksum << 1) | 1;
                        else
                            checksum <<= 1;

                    //printf("Checksum: %4X\n", checksum);
                    for(i = 2; i < 4 + tam; i++)
                    {
                        for(j = 128; j > 0; j >>= 1)
                            if(t[i] & j)
                                temp = (temp << 1) | 1;
                            else
                                temp <<= 1;
                        i++;
                        for(j = 128; j > 0; j >>= 1)
                            if(t[i] & j)
                                temp = (temp << 1) | 1;
                            else
                                temp <<= 1;

                        if(((checksum >> 12) + (temp >> 12)) > 15)
                            checksum += 1;
                        //printf("Temp: %4X\n", temp);
                        checksum += temp;
                        //printf("Checksum: %4X\n", checksum);
                        //printf("\n");
                    }
                    checksum = ~checksum;
                    printf("Checksum: %4X\n", checksum);
                break;
                case 3:
                    printf("~\n");
                break;
            }
            return 0;
        }

        BYTE ones_counter(BYTE t[], BYTE size)
        {
            BYTE contador_unos = 0;

            for(BYTE i = 0; i < size; i++)
```

```
177.                for(BYTE j = 128; j > 0; j = j >> 1)
178.                    if(t[i] & j)
179.                        contador_unos++;
180.
181.            return contador_unos;
182.        }
```