|  |  |  |
| --- | --- | --- |
| **ipn** | **INSTITUTO POLITÉCNICO NACIONAL**  **ESCUELA SUPERIOR DE CÓMPUTO** |  |

**Redes de Computadoras**

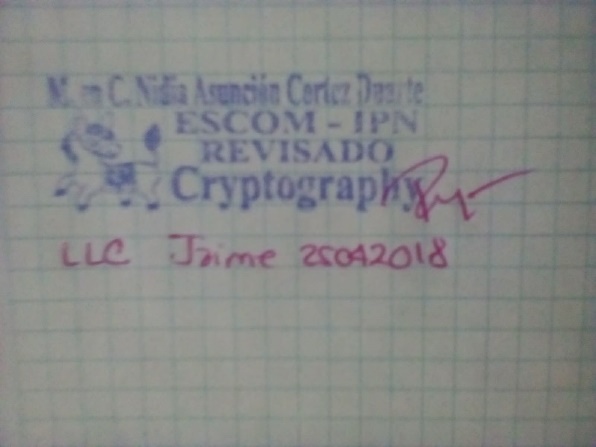
**“LLC FRAME ANALYZER”**

**By:**

**Bastida Prado Jaime Armando**

Professor:

M. en C. NIDIA ASUNCIÓN CORTEZ DUARTE

****April 2018

**Index**

Contenido

[Problem: 3](#_Toc381082154)

[Hypothesis: 3](#_Toc381082155)

[Software (libraries, packages, tools): 3](#_Toc381082156)

[Procedure: 3](#_Toc381082157)

[Results (Data): 3](#_Toc381082158)

[Conclusions: 3](#_Toc381082159)

[References: 4](#_Toc381082160)

[Code 4](#_Toc381082161)

# Problem:

What we need to do in this practice is to develop a C program to analyze LLC frames.

The Logical Link Control (LLC) data communication protocol layer is the upper sublayer of the data link layer (layer 2) of the seven-layer OSI model. The LLC sublayer provides multiplexing mechanisms that make it possible for several network protocols (e.g. IP, IPX, Decnet and Appletalk) to coexist within a multipoint network and to be transported over the same network medium. It can also provide flow control and automatic repeat request (ARQ) error management mechanisms.

First of all, we need to understand how an Ethernet frame is composed:

Destination

MAC

(6 bytes)

EtherType

(2 bytes)

Source

MAC

(6 bytes)

What we need to do is to check whether the EtherType is

* <= 1500 (in which case we are talking about a LLC frame),
* <= 2048 (in which case we are talking about an IP),
* <= 2056 (in which case it is an ARP).

As we have discussed earlier we are only going to analyze LLC frames, so next step is to understand how the bytes that follow the EtherType are composed and what they mean.

Destination

SAP

(1 bytes)

Control

(1-2 bytes)

Source

SAP

(1 bytes)

Information

(Variable bytes)

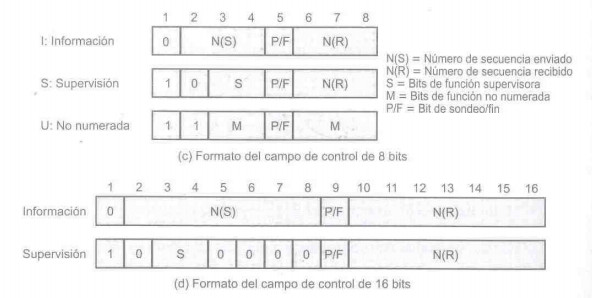
The Destination and Source SAPs tell us the addresses. Control is the most important field in this problem since it will tell us a lot of information about the frame.

A Control field can be 1 to 2 bytes long, it depends on which transference communication mode the stations are communicating, there are three types:

* NRM (Normal Response Mode)
* ABM (Asynchronous Balanced Mode)
* ARM (Asynchronous Response Mode)

All of the three previous modes are 1-byte long. The extended versions (NRME, ABME, ARME) are 2 bytes long.

The control field is as follows:

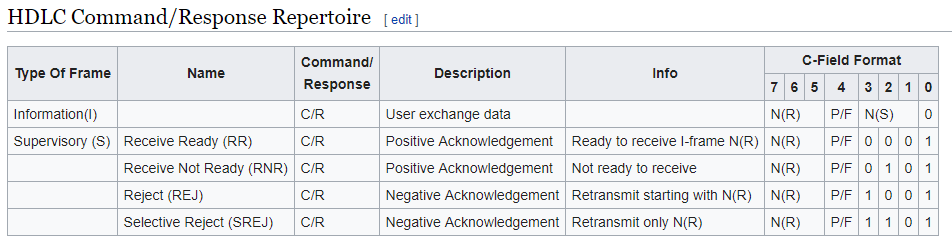


A Control field defines the three types of frames that exist in the LLC protocol; Unnumbered, Supervision or Information. In the image above, we can see how each one of them can be recognized, for example a Information frame has its first (LSB) Control bit off, a Supervision frame has its first bit on and the second off, and a Unnumbered frame has both on.

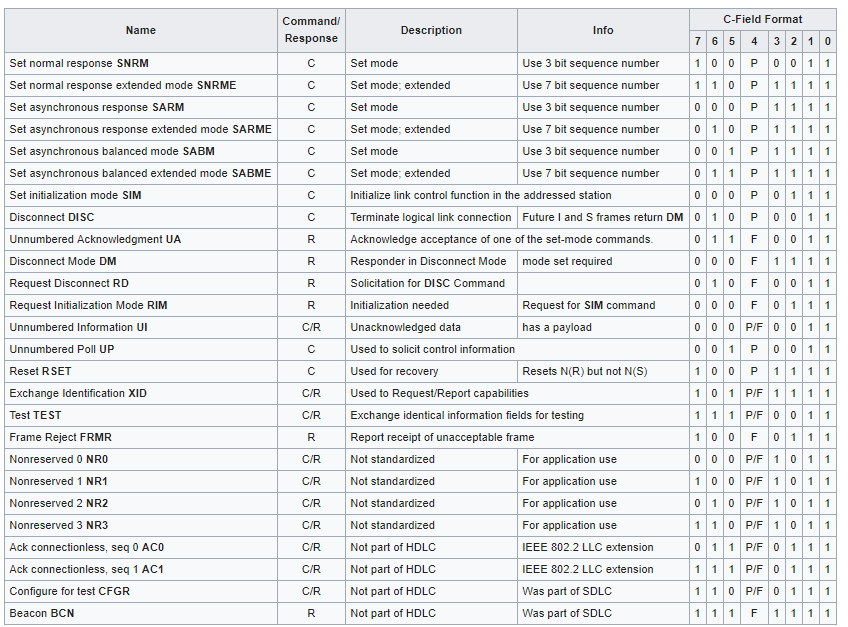
N(s) and N(r) are just numbers of sequence that help with the order in which the frames are being sent and received.

If we detect that a frame is of the Supervision type, then the bits 3 and 4 (from the LSB to the MSB) are the code of Supervision we are handling with.

The next table will help us determine what that codes mean:



# If we are treating with an Unnumbered frame, then we will have to check the “M” bits that will give us a code, the following table gives us the meaning of this codes:



The Unnumbered frames can be a Response or a Command, to be able to know what type is, we have to check the P/F bit, if it is on then we have to check the LSB of the Source SAP, if it is on then the Unnumbered frame is a Response, if it is off then the frame is a Unnumbered command.

Our program will receive 33 frames which will be saved in a bi-dimensional static array, then it will call the function “analizaTrama()” inside a for to analyze the 33 frames.

The function must analyze the frame with the following specifications:

* EtherType <= 1500 LLC [Show the analysis]
* EtherType <= 2048 IP [Just print IP]
* EtherType <= 2056 ARP [Just print ARP]

If the frame checks with any of the three cases defined, it will fill the string “resultado” with the requested information and return 1. If it does not check with any, it will return 0.

The function “analizaTrama()” will receive the frame and the string “resultado”.

# Hypothesis:

If we base the solution of the problem into a complete analysis of what we know so far about LLC protocol, then we will be able to develop the correct program to solve the task.

# Software (libraries, packages, tools):

The equipment used in the lab:

-Laptop

-Pencil

-Sheets

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

-Compiler: GNU (GCC) compiler.

# Procedure:

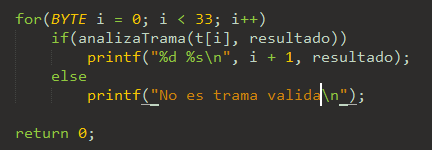
First, to optimize the code define a type named BYTE that as its name tells, will stand as a type to save bytes since the C standard does not define such a type.



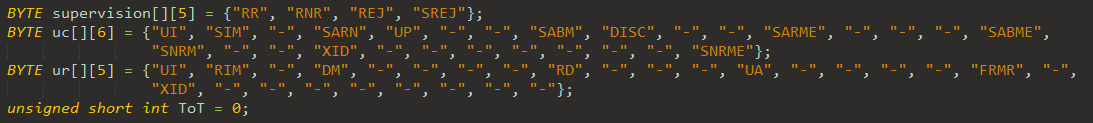
Next, declare and fill the bi-dimensional array that will contain all the 33 frames



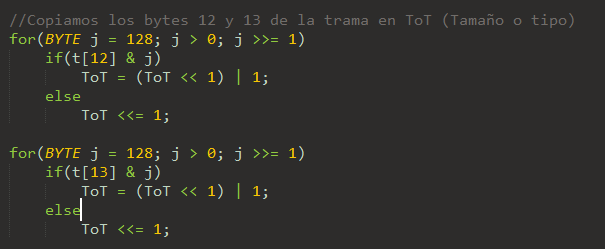
Next, a for to analyze the 33 frames, if the function called returns 1, print the result else, print “No es trama valida”.



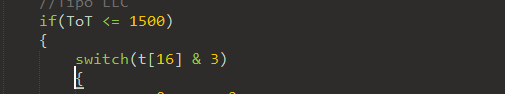
Inside the function, declare an array for each of the Supervision, Unnumbered Responses and Unnumbered Commands, and fill them with the respective data.



Also declare a short integer to save the EtherType value and fill it with the respective value contained in the bytes 12 and 13 of the frame.



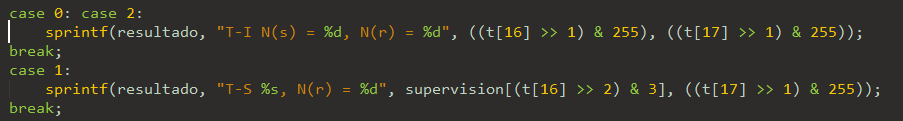
Now, an If-ElseIf-Else structure to handle LLC, IP and ARP cases. Inside the LLC cases is where the core of our program lays. First to decide which type of frame if Supervision, Information or Unnumbered is, check the first two bits of the Control field i.e. the 16 byte of the frame.



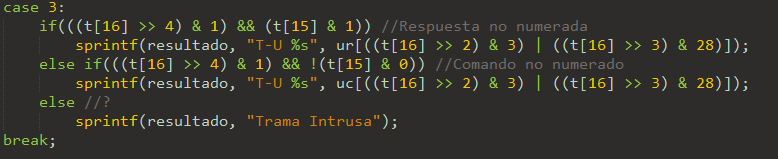
Following the structure of the Control field discussed in the Problem section determine the type with a switch.

For case 0 and 2 we are treating with a Information frame so the only thing needed to do is to print the N(r) and N(s) values.

For case 1 print the N(r) and the description of the code given, printing the respective string saved into the “supervision[]” array, done by checking the S bits.

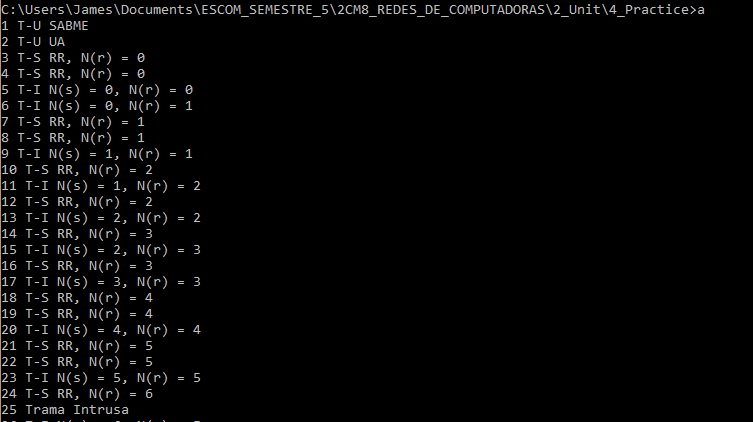


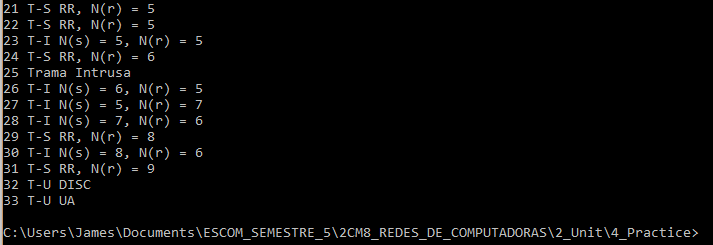
For case 3 we have to check the P/F bit as discussed in the Problem section, once we know which type is: Command or Response, fill resultado with the respective description of the code given in the M bits, and in case none of it is, fill resultado with “Trama Intrusa” message.



# Results (Data):

Making a test for the 33 frames given by the teacher:





# Conclusions:

I can accept my hypothesis as true because the analysis of all the data given in the class about the LLC was sufficient to solve the problem.

The results were concise and according to the results calculated in paper, there is nothing until this point that can be optimized or modified since the only thing the program needed to do was to print the requested results described into the Problem section. Except for errors that could appear at the moment of fill the data of the frames into the bi-dimensional array in case there is a modification requested by the teacher.

I believe I have not learned something useful enough to apply to a real-life situation, for now. But I found it very interesting to know how a protocol like the LLC works.

# References:

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

Wikipedia. (2018, April 11). High-Level Data Link Control. Retrieved from https://en.wikipedia.org/wiki/High-Level\_Data\_Link\_Control

# Code

1. #include <stdio.h>
3. **typedef** unsigned **char** **BYTE**;
5. **int** analizaTrama(**BYTE** t[], **BYTE** resultado[]);
7. **int** main(**void**)
8. {
9. **BYTE** t[ ][32] = {   {0x00,0x02,0xb3,0x9c,0xae,0xba,0x00,0x02,0xb3,0x9c,0xdf,0x1b,0x00,0x03,0xf0,0xf0, 0x7f,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00}, //Trama 1
11. {0x00,0x02,0xb3,0x9c,0xdf,0x1b,0x00,0x02,0xb3,0x9c,0xae,0xba,0x00,0x03,0xf0,0xf1, 0x73,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00}, //Trama 2
13. {0x00,0x02,0xb3,0x9c,0xae,0xba,0x00,0x02,0xb3,0x9c,0xdf,0x1b,0x00,0x04,0xf0,0xf0, 0x01,0x01,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00},//Trama 3
15. {0x00,0x02,0xb3,0x9c,0xdf,0x1b,0x00,0x02,0xb3,0x9c,0xae,0xba,0x00,0x04,0xf0,0xf1, 0x01,0x01,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00}, //Trama 4
17. {0x00,0x02,0xb3,0x9c,0xae,0xba,0x00,0x02,0xb3,0x9c,0xdf,0x1b,0x00,0x12,0xf0,0xf0, 0x00,0x01,0x0e,0x00,0xff,0xef,0x19,0x8f,0xbc,0x05,0x7f,0x00,0x23,0x00,0x7f,0x23}, //Trama 5
19. {0x00,0x02,0xb3,0x9c,0xdf,0x1b,0x00,0x02,0xb3,0x9c,0xae,0xba,0x00,0x12,0xf0,0xf0, 0x00,0x03,0x0e,0x00,0xff,0xef,0x19,0x8f,0xbc,0x05,0x7f,0x00,0x23,0x00,0x7f,0x23}, //Trama 6
21. {0x00,0x02,0xb3,0x9c,0xdf,0x1b,0x00,0x02,0xb3,0x9c,0xae,0xba,0x00,0x04,0xf0,0xf1, 0x01,0x03,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00}, //Trama 7
23. {0x00,0x02,0xb3,0x9c,0xdf,0x1b,0x00,0x02,0xb3,0x9c,0xae,0xba,0x00,0x04,0xf0,0xf1, 0x01,0x03,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00}, //Trama 8
25. {0x00,0x02,0xb3,0x9c,0xae,0xba,0x00,0x02,0xb3,0x9c,0xdf,0x1b,0x00,0xac,0xf0,0xf0, 0x02,0x02,0x0e,0x00,0xff,0xef,0x16,0x04,0x00,0x00,0x00,0x00,0x28,0x00,0x7f,0x23}, //Trama 9
27. {0x00,0x02,0xb3,0x9c,0xdf,0x1b,0x00,0x02,0xb3,0x9c,0xae,0xba,0x00,0x04,0xf0,0xf1, 0x01,0x04,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00}, //Trama 10
29. {0x00,0x02,0xb3,0x9c,0xdf,0x1b,0x00,0x02,0xb3,0x9c,0xae,0xba,0x00,0x5f,0xf0,0xf0, 0x02,0x04,0x0e,0x00,0xff,0xef,0x16,0x0c,0x00,0x00,0x28,0x00,0x28,0x00,0x7f,0x23}, //Trama 11
31. {0x00,0x02,0xb3,0x9c,0xae,0xba,0x00,0x02,0xb3,0x9c,0xdf,0x1b,0x00,0x04,0xf0,0xf1, 0x01,0x04,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00}, //Trama 12
33. {0x00,0x02,0xb3,0x9c,0xae,0xba,0x00,0x02,0xb3,0x9c,0xdf,0x1b,0x00,0x91,0xf0,0xf0, 0x04,0x04,0x0e,0x00,0xff,0xef,0x16,0x0c,0x00,0x00,0x28,0x00,0x28,0x00,0x7f,0x23}, //Trama 13
35. {0x00,0x02,0xb3,0x9c,0xdf,0x1b,0x00,0x02,0xb3,0x9c,0xae,0xba,0x00,0x04,0xf0,0xf1, 0x01,0x06,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00}, //Trama 14
37. {0x00,0x02,0xb3,0x9c,0xdf,0x1b,0x00,0x02,0xb3,0x9c,0xae,0xba,0x00,0x46,0xf0,0xf0, 0x04,0x06,0x0e,0x00,0xff,0xef,0x16,0x0c,0x00,0x00,0x28,0x00,0x28,0x00,0x7f,0x23}, //Trama 15
39. {0x00,0x02,0xb3,0x9c,0xae,0xba,0x00,0x02,0xb3,0x9c,0xdf,0x1b,0x00,0x04,0xf0,0xf1, 0x01,0x06,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00}, //Trama 16
41. {0x00,0x02,0xb3,0x9c,0xae,0xba,0x00,0x02,0xb3,0x9c,0xdf,0x1b,0x00,0x7e,0xf0,0xf0, 0x06,0x06,0x0e,0x00,0xff,0xef,0x16,0x0c,0x00,0x00,0x28,0x00,0x28,0x00,0x7f,0x23}, //Trama 17
43. {0x00,0x02,0xb3,0x9c,0xdf,0x1b,0x00,0x02,0xb3,0x9c,0xae,0xba,0x00,0x04,0xf0,0xf1, 0x01,0x08,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00}, //Trama 18
45. {0x00,0x02,0xb3,0x9c,0xae,0xba,0x00,0x02,0xb3,0x9c,0xdf,0x1b,0x00,0x04,0xf0,0xf1, 0x01,0x08,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00}, //Trama 19
47. {0x00,0x02,0xb3,0x9c,0xae,0xba,0x00,0x02,0xb3,0x9c,0xdf,0x1b,0x00,0x7e,0xf0,0xf0, 0x08,0x08,0x0e,0x00,0xff,0xef,0x16,0x0c,0x00,0x00,0x28,0x00,0x28,0x00,0x7f,0x23}, //Trama 20
49. {0x00,0x02,0xb3,0x9c,0xdf,0x1b,0x00,0x02,0xb3,0x9c,0xae,0xba,0x00,0x04,0xf0,0xf1, 0x01,0x0a,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00}, //Trama 21
51. {0x00,0x02,0xb3,0x9c,0xae,0xba,0x00,0x02,0xb3,0x9c,0xdf,0x1b,0x00,0x04,0xf0,0xf1, 0x01,0x0a,0x0e,0x00,0xff,0xef,0x19,0x8f,0xbc,0x05,0x7f,0x00,0x23,0x00,0x7f,0x23}, //Trama 22
53. {0x00,0x02,0xb3,0x9c,0xae,0xba,0x00,0x02,0xb3,0x9c,0xdf,0x1b,0x00,0x12,0xf0,0xf0, 0x0a,0x0b,0x0e,0x00,0xff,0xef,0x14,0x00,0x00,0x00,0x28,0x00,0x00,0x00,0x7f,0x23}, //Trama 23
55. {0x00,0x02,0xb3,0x9c,0xdf,0x1b,0x00,0x02,0xb3,0x9c,0xae,0xba,0x00,0x04,0xf0,0xf1, 0x01,0x0d,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00}, //Trama 24
57. {0x03,0x00,0x00,0x00,0x00,0x01,0x00,0x04,0xac,0x44,0x4d,0x02,0x00,0x8b,0xf0,0xf0, 0x03,0x2c,0x00,0xff,0xef,0x08,0x00,0x00,0x00,0x00,0x00,0x28,0x00,0x42,0x34,0x20}, //Trama 25
59. {0x00,0x02,0xb3,0x9c,0xae,0xba,0x00,0x02,0xb3,0x9c,0xdf,0x1b,0x00,0x35,0xf0,0xf0, 0x0c,0x0a,0x0e,0x00,0xff,0xef,0x16,0x04,0x00,0x00,0x00,0x00,0x28,0x00,0x7f,0x23}, //Trama 26
61. {0x00,0x02,0xb3,0x9c,0xae,0xba,0x00,0x02,0xb3,0x9c,0xdf,0x1b,0x00,0x35,0xf0,0xf0, 0x0a,0x0e,0x0e,0x00,0xff,0xef,0x16,0x04,0x00,0x00,0x00,0x00,0x28,0x00,0x7f,0x23}, //Trama 27
63. {0x00,0x02,0xb3,0x9c,0xae,0xba,0x00,0x02,0xb3,0x9c,0xdf,0x1b,0x00,0x12,0xf0,0xf0, 0x0e,0x0d,0x0e,0x00,0xff,0xef,0x14,0x00,0x00,0x00,0x28,0x00,0x00,0x00,0x7f,0x23}, //Trama 28
65. {0x00,0x02,0xb3,0x9c,0xdf,0x1b,0x00,0x02,0xb3,0x9c,0xae,0xba,0x00,0x04,0xf0,0xf1, 0x01,0x11,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00}, //Trama 29
67. {0x00,0x02,0xb3,0x9c,0xae,0xba,0x00,0x02,0xb3,0x9c,0xdf,0x1b,0x00,0x12,0xf0,0xf0, 0x10,0x0d,0x0e,0x00,0xff,0xef,0x18,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x7f,0x23}, //Trama 30
69. {0x00,0x02,0xb3,0x9c,0xdf,0x1b,0x00,0x02,0xb3,0x9c,0xae,0xba,0x00,0x04,0xf0,0xf1, 0x01,0x13,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00}, //Trama 31
71. {0x00,0x02,0xb3,0x9c,0xae,0xba,0x00,0x02,0xb3,0x9c,0xdf,0x1b,0x00,0x03,0xf0,0xf0, 0x53,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00}, //Trama 32
73. {0x00,0x02,0xb3,0x9c,0xdf,0x1b,0x00,0x02,0xb3,0x9c,0xae,0xba,0x00,0x03,0xf0,0xf1, 0x73,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00}}; //Trama 33
74. **BYTE** resultado[100];
76. **for**(**BYTE** i = 0; i < 33; i++)
77. **if**(analizaTrama(t[i], resultado))
78. printf("%d %s\n", i + 1, resultado);
79. **else**
80. printf("No es trama valida\n");
82. **return** 0;
83. }
85. **int** analizaTrama(**BYTE** t[], **BYTE** resultado[])
86. {
87. **BYTE** supervision[][5] = {"RR", "RNR", "REJ", "SREJ"};
88. **BYTE** uc[][6] = {"UI", "SIM", "-", "SARN", "UP", "-", "-", "SABM", "DISC", "-", "-", "SARME", "-", "-", "-", "SABME",
89. "SNRM", "-", "-", "XID", "-", "-", "-", "-", "-", "-", "-", "SNRME"};
90. **BYTE** ur[][5] = {"UI", "RIM", "-", "DM", "-", "-", "-", "-", "RD", "-", "-", "-", "UA", "-", "-", "-", "-", "FRMR", "-",
91. "XID", "-", "-", "-", "-", "-", "-", "-", "-"};
92. unsigned **short** **int** ToT = 0;
94. //Copiamos los bytes 12 y 13 de la trama en ToT (Tamaño o tipo)
95. **for**(**BYTE** j = 128; j > 0; j >>= 1)
96. **if**(t[12] & j)
97. ToT = (ToT << 1) | 1;
98. **else**
99. ToT <<= 1;
101. **for**(**BYTE** j = 128; j > 0; j >>= 1)
102. **if**(t[13] & j)
103. ToT = (ToT << 1) | 1;
104. **else**
105. ToT <<= 1;
107. //Tipo LLC
108. **if**(ToT <= 1500)
109. {
110. **switch**(t[16] & 3)
111. {
112. **case** 0: **case** 2:
113. sprintf(resultado, "T-I N(s) = %d, N(r) = %d", ((t[16] >> 1) & 255), ((t[17] >> 1) & 255));
114. **break**;
115. **case** 1:
116. sprintf(resultado, "T-S %s, N(r) = %d", supervision[(t[16] >> 2) & 3], ((t[17] >> 1) & 255));
117. **break**;
118. **case** 3:
119. **if**(((t[16] >> 4) & 1) && (t[15] & 1)) //Respuesta no numerada
120. sprintf(resultado, "T-U %s", ur[((t[16] >> 2) & 3) | ((t[16] >> 3) & 28)]);
121. **else** **if**(((t[16] >> 4) & 1) && !(t[15] & 0)) //Comando no numerado
122. sprintf(resultado, "T-U %s", uc[((t[16] >> 2) & 3) | ((t[16] >> 3) & 28)]);
123. **else** //?
124. sprintf(resultado, "Trama Intrusa");
125. **break**;
126. }
128. **return** 1;
129. }
130. **else** **if**(ToT <= 2048) //Tipo IP
131. {
132. sprintf(resultado, "IP\n");
133. **return** 1;
134. }
135. **else** **if**(ToT <= 2056) //Tipo ARP
136. {
137. sprintf(resultado, "ARP\n");
138. **return** 1;
139. }
141. **return** 0;
142. }