ITESO

Universidad Jesuita de Guadalajara

Title: Thread

Subject: Networks for embedded systems

Teacher: Estephania Martinez García

Student(s): Sergio Eduardo Borrayo Anzaldo (ie726673)

Isaac Segovia Hernández (ie721714)

Date: 22/11/2023

Theoretical framework

Router: Is a node that provide joining and security services for devices trying to join the network. Routers are not designed to sleep. Routers can downgrade their functionality and become REEDs (Router-eligible End Devices).

Leader: The Leader is an additional role of one Router in a Thread network. The Leader is an elected role of one Router, which takes certain decisions in the Thread network such as allowing REEDs to upgrade to Routers. If the Leader of a Thread Network fails, another Router will be dynamically selected to resume the role. All Routers have the required Thread Network Data to seamlessly assume this role.

End Device: Is a node that communicates primarily with a single Router, does not forward packets for other network devices and can disable its transceiver to reduce power.

IPv6 addresses: An IPv6 address is a 128-bit alphanumeric value that identifies an endpoint device in an Internet Protocol Version 6 (IPv6) network. The Internet Protocol (IP) is a method in which data is sent to different computers over the internet. Each network interface, or computer, on the internet will have at least one IP address that is used to uniquely identify that computer. Every device that connects to the internet is assigned an IP address. Which is why there was a concern with the number of IP addresses in IPv4, and why the Internet Engineering Task Force (IETF) defined the new IPv6 standard.

PanID: PAN ID is the identifier of a wireless network. Thread networks are identified by three unique identifiers:

- 2-byte Personal Area Network ID (PAN ID)
- 8-byte Extended Personal Area Network ID (XPAN ID)
- A human-readable Network Name

Development of modifications made to code.

The git hub link: https://github.com/SergioB17/Practica2_Equipo5

Network setup:

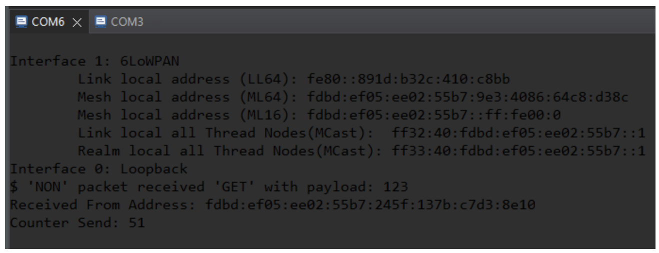
Devices and types:

-Router 1: Leader

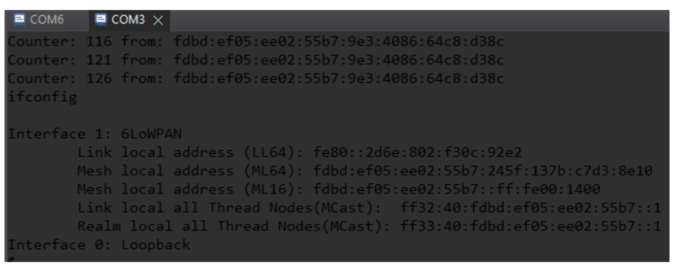
-Router 2: End device

IPv6 addresses (This were randomly assigned each time the board was reprogramed):

-Router 1:



-Router 2:



PanID (On app_thread_config.h):

-0x0505

```c
/*! The PAN identifier.
   If this value is 0xFFFF a random PAN ID will be generated on network creation */
#ifndef THR_PAN_ID
    #define THR_PAN_ID                                  0x0505//THR_ALL_FFs16
#endif
```

Channel (On app_thread_config.h):

-20

```
/*************************************/
/* 802.15.4 default configuration */
/*************************************/
/*! The channel mask used when a network scanning is performed (energy scan, active scan or both);
   0x07FFF800 sets all 2.4GHz 16 channels are used (from channel id 11 to 26).
   This channel mask is used during network creation to select the best channel or during network discovery
   to find the available networks on that channels.
   To set the scan mask to a single channel, set the mask to 0x00000001 shifted with the channel ID
   e.g.: to set mask for channel 25, set THR_SCANCHANNEL_MASK to: (0x00000001 << 25) */
#ifndef THR_SCANCHANNEL_MASK
    #define THR_SCANCHANNEL_MASK                        (0x00000001 << 20)
#endif
```

Network name:

-Sergio_Isaac

```
/*! The default network name */
#ifndef THR_NETWORK_NAME
    #define THR_NETWORK_NAME                            {14, "Sergio_Isaac"}
#endif
```

According to the next instructions, the code was integrated sequentially. Starting with:

Práctica Thread

PART 1

Leader/Router 1:

1. Create your own CoAP URI with your team's name. ie "/team1"

This step was already done with a LAB work in class, so we just use that code.

On router_elegible_device_app.c we need to define the URI path names that will be used in the shell to access the resources on "router_elegible_device_app.c".

```
//PARTE 1.1.1
#define APP_RESOURCE1_URI_PATH                "/team5"
#define APP_RESOURCE2_URI_PATH                "/resource2"
```

Then, we declare the URI resources with coapUriPath_t. When using this struct the user must enter the length of the URI path and the path created in the last step on "router_elegible_device_app.c".

```
const coapUriPath_t gAPP_RESOURCE1_URI_PATH = {SizeOfString(APP_RESOURCE1_URI_PATH), APP_RESOURCE1_URI_PATH};

const coapUriPath_t gAPP_RESOURCE2_URI_PATH = {SizeOfString(APP_RESOURCE2_URI_PATH), APP_RESOURCE2_URI_PATH};
```

The next step consists in create the callbacks for the resources. These callbacks will handle the packet received and perform the desired action depending on the type of COAP method received on “router_elegible_device_app.c”.

```c
static void APP_CoapResource1Cb(coapSessionStatus_t sessionStatus, void *pData, coapSession_t *pSession, uint32_t dataLen);
static void APP_CoapResource2Cb(coapSessionStatus_t sessionStatus, void *pData, coapSession_t *pSession, uint32_t dataLen);
```

Now, we add the callback handler for the packet received, in this function it will be defined which action will be performed depending on the type of COAP method received on “router_elegible_device_app.c”.

```c
//PARTE 1.1.2
static void APP_CoapResource1Cb
(
coapSessionStatus_t sessionStatus,
void *pData,
coapSession_t *pSession,
uint32_t dataLen
)

{
  static uint8_t pMySessionPayload[1];
  pMySessionPayload[0] = g_Counter_Timer;
  static uint32_t pMyPayloadSize=1;
  coapSession_t *pMySession = NULL;
  pMySession = COAP_OpenSession(mAppCoapInstId);
  COAP_AddOptionToList(pMySession,COAP_URI_PATH_OPTION, APP_RESOURCE2_URI_PATH,SizeOfString(APP_RESOURCE2_URI_PATH));

  char IPAddress[INET6_ADDRSTRLEN];

  //PARTE 1.3 de la practica
    if (gCoapConfirmable_c == pSession->msgType)
  {
    if (gCoapGET_c == pSession->code)
    {
      shell_write("'CON' packet received 'GET' with payload: ");
    }
    if (gCoapPOST_c == pSession->code)
    {
      shell_write("'CON' packet received 'POST' with payload: ");
```

After creating the callbacks, those must be registered in the CoAP callback array in the function *APP_InitCoapDemo(void)* on “router_elegible_device_app.c”.

```c
static void APP_InitCoapDemo
(
    void
)
{
    coapRegCbParams_t cbParams[] =  {{APP_CoapLedCb,  (coapUriPath_t *)&gAPP_LED_URI_PATH},
                                     {APP_CoapTempCb, (coapUriPath_t *)&gAPP_TEMP_URI_PATH},

                                     {APP_CoapResource1Cb, (coapUriPath_t*)&gAPP_RESOURCE1_URI_PATH},
                                     {APP_CoapResource2Cb, (coapUriPath_t*)&gAPP_RESOURCE2_URI_PATH},
```

Now we need to change a setting related with the session, we need to close the session every time we open one, so we change this configuration on “coap.h”:

```c
/*! Do not close CoAP session */
#define COAP_KeepSessionOpen(pSession) pSession->autoClose = TRUE
```

And with that, we can consider finished the 1.1 part. The next statement says:

2. Start a timer (1 sec) that will have a counter from 1 to 200. This counter will be accessible to the network through your URI "/team1". Initiate the timer when the Router 2 joins the network.

The connection is made on “router_elegible_device_app.c”, specifically on “APP_Commissioning_Handler” on the case where the joiner has been accepted:

```c
        case gThrEv_MeshCop_CommissionerJoinerAccepted_c:

            //PARTE 1.2 en esta parte es cuando se une el dispositivo a la red
            shell_write("El dispositivo se unio correctamente\r\n");
            shell_write("Se iniciara el Timer\r\n");
            //Start Timer
            TMR_StartIntervalTimer(PlusCounter_ID, 1000, PlusCounterRouter1, NULL);

            break;
        case gThrEv_MeshCop_CommissionerNwkDataSynced_c:
            break;
    }

    /* Free event buffer */
    MEM_BufferFree(pEventParams);
}
```

This is where the timer jumps one the counter has expired on “router_elegible_device_app.c”:

```c
static void PlusCounterRouter1(void *param)
{
    g_Counter_Timer++;

    if(g_Counter_Timer > 200)
    {
        g_Counter_Timer = 0;
    }
    //shell_write("Si estoy jalando\r\n");
}
```

The counter is sent at the “APP_CoapResource1Cb” function that correspond to /team5 URI. First, we assigned the counter to the Payload of the package on “router_elegible_device_app.c”:

```c
//PARTE 1.1.2
static void APP_CoapResource1Cb
(
coapSessionStatus_t sessionStatus,
void *pData,
coapSession_t *pSession,
uint32_t dataLen
)

{
  static uint8_t pMySessionPayload[1];
  pMySessionPayload[0] = g_Counter_Timer;
  static uint32_t pMyPayloadSize=1;
  coapSession_t *pMySession = NULL;
  pMySession = COAP_OpenSession(mAppCoapInstId);
  COAP_AddOptionToList(pMySession,COAP_URI_PATH_OPTION, APP_RESOURCE2_URI_PATH,SizeOfString(APP_RESOURCE2_URI_PATH));
```

Now we send the data (we are still in the same file and function):

```c
  pMySession -> msgType=gCoapNonConfirmable_c;
  pMySession -> code= gCoapPOST_c;
  pMySession -> pCallback =NULL;
  FLib_MemCpy(&pMySession->remoteAddrStorage.ss_addr,&gCoapDestAddress,sizeof(ipAddr_t));
  COAP_Send(pMySession, gCoapMsgTypeNonPost_c,pMySessionPayload, pMyPayloadSize);
  shell_write("Counter Send: ");
  shell_writeDec(*pMySessionPayload);
  shell_write("\r\n");
  shell_write("\r\n");

}
```

The next statement says:

3. When a CoAP instruction is received, print in shell the IP address of the requester and indicate if it was a CON or NON request ie. CON instruction received from fd01:abcd::1234

This was coded in the "APP_CoapResource1Cb" function on "router_elegible_device_app.c:

```c
 //PARTE 1.3 de la practica
   if (gCoapConfirmable_c == pSession->msgType)
 {
   if (gCoapGET_c == pSession->code)
   {
     shell_write("'CON' packet received 'GET' with payload: ");
   }
   if (gCoapPOST_c == pSession->code)
   {
     shell_write("'CON' packet received 'POST' with payload: ");
   }
   if (gCoapPUT_c == pSession->code)
   {
     shell_write("'CON' packet received 'PUT' with payload: ");
   }
   if (gCoapFailure_c!=sessionStatus)
   {
     //PARTE 1.4 Responder con Ack por ser de tipo CON
     COAP_Send(pSession, gCoapMsgTypeAckSuccessChanged_c, pMySessionPayload, pMyPayloadSize);
   }
 }

 else if(gCoapNonConfirmable_c == pSession->msgType)
 {
   if (gCoapGET_c == pSession->code)
   {
     shell_write("'NON' packet received 'GET' with payload: ");
   }
   if (gCoapPOST_c == pSession->code)
   {
     shell_write("'NON' packet received 'POST' with payload: ");
   }
   if (gCoapPUT_c == pSession->code)
   {
     shell_write("'NON' packet received 'PUT' with payload: ");
   }
 }
```

Also, the address is extracted and show from here (we are still in the same file and function)

```
shell_writeN(pData, dataLen);
shell_write("\r\n");
ntop(AF_INET6,(ipAddr_t*)&pSession->remoteAddrStorage.ss_addr,IPAddress,INET6_ADDRSTRLEN);

shell_write("Received From Address: ");
shell_writeN(IPAddress,INET6_ADDRSTRLEN);
shell_write("\r\n");
```

The last statement of the first part says:

4. If it was a CON request, reply with a CoAP Ack, if it was a NON request, don't reply with a CoAP Ack.

This is also coded between the last part:

```
//PARTE 1.3 de la practica
  if (gCoapConfirmable_c == pSession->msgType)
{
  if (gCoapGET_c == pSession->code)
  {
    shell_write("'CON' packet received 'GET' with payload: ");
  }
  if (gCoapPOST_c == pSession->code)
  {
    shell_write("'CON' packet received 'POST' with payload: ");
  }
  if (gCoapPUT_c == pSession->code)
  {
    shell_write("'CON' packet received 'PUT' with payload: ");
  }
  if (gCoapFailure_c!=sessionStatus)
  {
    //PARTE 1.4 Responder con Ack por ser de tipo CON
    COAP_Send(pSession, gCoapMsgTypeAckSuccessChanged_c, pMySessionPayload, pMyPayloadSize);
  }
}

else if(gCoapNonConfirmable_c == pSession->msgType)
{
  if (gCoapGET_c == pSession->code)
  {
    shell_write("'NON' packet received 'GET' with payload: ");
  }
  if (gCoapPOST_c == pSession->code)
  {
    shell_write("'NON' packet received 'POST' with payload: ");
  }
  if (gCoapPUT_c == pSession->code)
  {
    shell_write("'NON' packet received 'PUT' with payload: ");
  }
}
```

Now we can start the Router 2 part, the statement says:

Router 2:

1. Once you are connected to the network, start an Interval Timer that will request every 5 seconds the current value stored in your team's URI from the Leader.

For this, we initialize the timer just after the Router 2 is accepted in the network, on "router_elegible_device_app.c

```
        case gThrEv_MeshCop_JoinerAccepted_c:
            PlusCounter_ID = TMR_AllocateTimer();
            //PARTE 2.1 Una vez se conecta al Leader, se empieza con un contador de 5 segundos
            TMR_StartIntervalTimer(PlusCounter_ID, 5000, PlusCounterRouter2,NULL);
            break;
```

Once the timer expires, it jumps here:

```
static void PlusCounterRouter2(void *param)
{
    //shell_write("Holaaaaaaa\n");
    APP_GetCounter();
}
```

And finally we get to "APP_GetCounter()" where the package with the command "/team5" is send it:

```
static void APP_GetCounter(void)
{
    static uint8_t pMySessionPayload[3]={0x31,0x32,0x33};
    static uint32_t pMyPayloadSize=3;
    coapSession_t *pMySession = NULL;
    pMySession = COAP_OpenSession(mAppCoapInstId);
    COAP_AddOptionToList(pMySession,COAP_URI_PATH_OPTION, APP_RESOURCE1_URI_PATH,SizeOfString(APP_RESOURCE1_URI_PATH));

    pMySession -> msgType = gCoapConfirmable_c;
    pMySession -> code = gCoapGET_c;
    pMySession -> pCallback =NULL;

    FLib_MemCpy(&pMySession->remoteAddrStorage.ss_addr,&gCoapDestAddress,sizeof(ipAddr_t));
    COAP_Send(pMySession, gCoapMsgTypeConGet_c,pMySessionPayload, pMyPayloadSize);

}
```

The next statement says:

2. Print in shell the value that was requested and the source IP address of the packet
   ie. Counter = 50 from fd01:abcd::1111

In the previous step, we send a package with the command “/team5”, in Router 1, once is received and get into the handler of that resource, we send the counter with the command “/resource2” in Router 1 to Router 2:

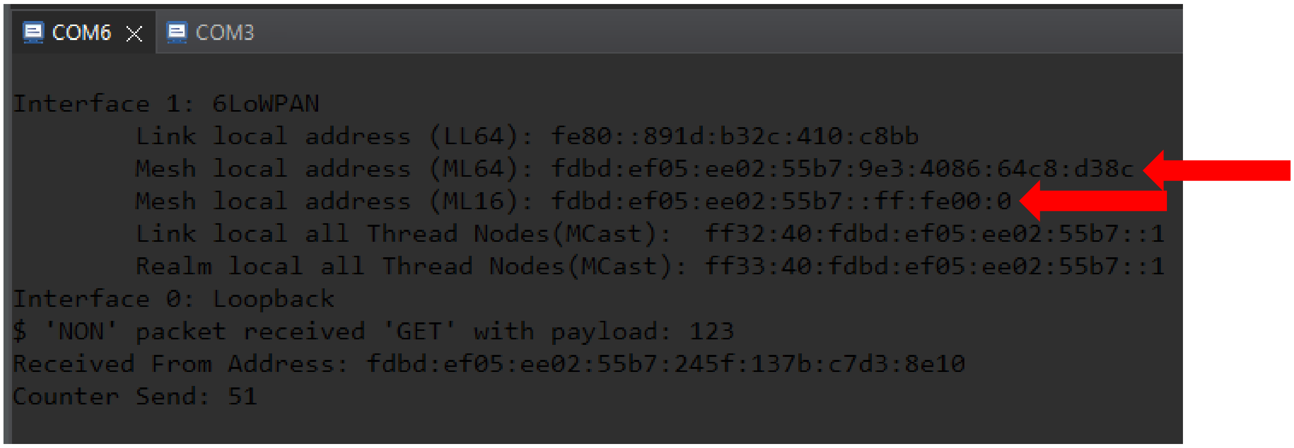
Code of Router 1

```
//PARTE 1.1.2
static void APP_CoapResource1Cb
(
coapSessionStatus_t sessionStatus,
void *pData,
coapSession_t *pSession,
uint32_t dataLen
)

{
  static uint8_t pMySessionPayload[1];
  pMySessionPayload[0] = g_Counter_Timer;
  static uint32_t pMyPayloadSize=1;
  coapSession_t *pMySession = NULL;
  pMySession = COAP_OpenSession(mAppCoapInstId);
  COAP_AddOptionToList(pMySession,COAP_URI_PATH_OPTION, APP_RESOURCE2_URI_PATH,SizeOfString(APP_RESOURCE2_URI_PATH));
```

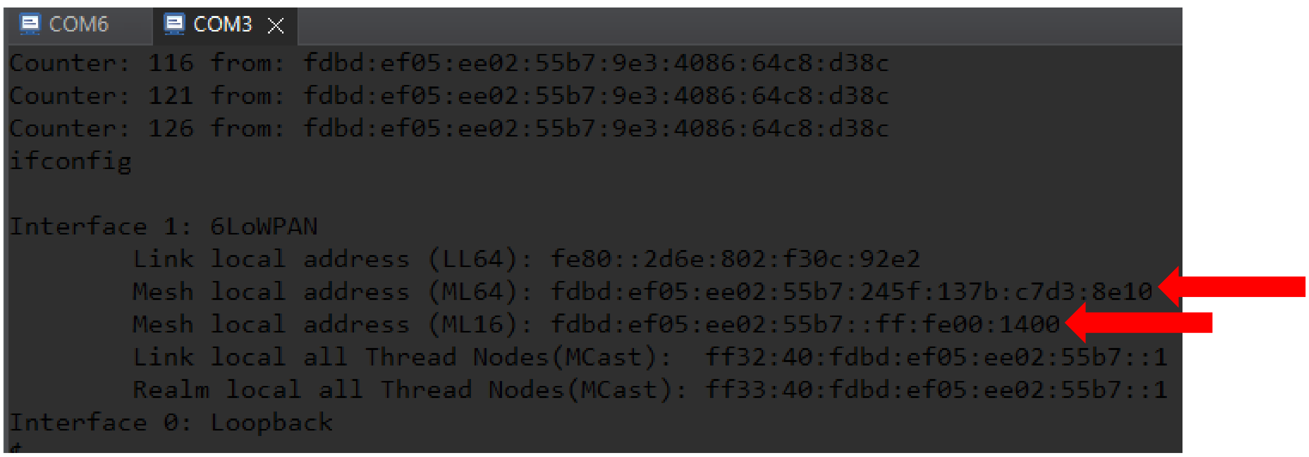
So, this means that in Router 2, will get to the handler of the other resource added “/resource2”, and there is where we print in shell the content and the address of the device that send the counter (Router 1), the function was named “APP_CoapResource2cb()” due to the LAB work made in class:

```
static void APP_CoapResource2Cb
(
coapSessionStatus_t sessionStatus,
void *pData,
coapSession_t *pSession,
uint32_t dataLen
)

{
    uint32_t Data = *(uint8_t*)pData;
    char IP_address[INET6_ADDRSTRLEN];
    ntop(AF_INET6,(ipAddr_t*)&pSession->remoteAddrStorage.ss_addr,IP_address,INET6_ADDRSTRLEN);
    shell_write("Counter: ");
    shell_writeDec(Data);
    shell_write(" from: ");
    shell_writeN(IP_address,INET6_ADDRSTRLEN);
    shell_write("\r\n");

}
```

Now, the next images are the terminals with the Router 1 and Router 2 working. First, Router 1 (we can know this because the last digit of the Mesh Local Address is 0):



Router 2:



Now again Router 1 but sending the counter. Pls notice that the first two lines were for testing and validation. The “ ’NON’ packet received ‘GET’ with payload: 123 ” and “ Received From Address:” are executed right before the counter is send. So the Adress print is from Router 2, the one that finish with 8e10:

COM6 COM3

```
'NON' packet received 'GET' with payload: 123
Received From Address: fdbd:ef05:ee02:55b7:245f:137b:c7d3:8e10
Counter Send: 134

'NON' packet received 'GET' with payload: 123
Received From Address: fdbd:ef05:ee02:55b7:245f:137b:c7d3:8e10
Counter Send: 139

'NON' packet received 'GET' with payload: 123
Received From Address: fdbd:ef05:ee02:55b7:245f:137b:c7d3:8e10
Counter Send: 144
```

```
    shell_write("Received From Address: ");
    shell_writeN(IPAddress,INET6_ADDRSTRLEN);
    shell_write("\r\n");

    pMySession -> msgType=gCoapConfirmable_c;
    pMySession -> code= gCoapPOST_c;
    pMySession -> pCallback =NULL;
    FLib_MemCpy(&pMySession->remoteAddrStorage.ss_addr,&gCoapDestAddress,sizeof(ipAddr_t));
    COAP_Send(pMySession, gCoapMsgTypeNonPost_c,pMySessionPayload, pMyPayloadSize);
    shell_write("Counter Send: ");
    shell_writeDec(*pMySessionPayload);
    shell_write("\r\n");
    shell_write("\r\n");

}
```

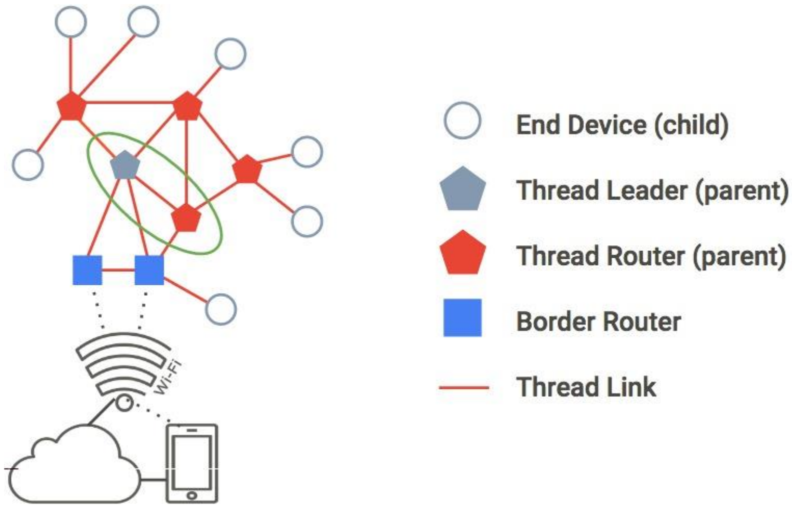
Then, In Router 2 is received the counter (we should see the address of the Router 1 that finish with d38c):

COM6 COM3

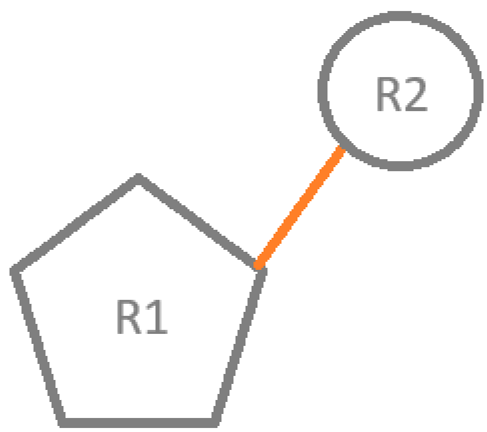
```
Counter: 94 from: fdbd:ef05:ee02:55b7:9e3:4086:64c8:d38c
Counter: 99 from: fdbd:ef05:ee02:55b7:9e3:4086:64c8:d38c
Counter: 104 from: fdbd:ef05:ee02:55b7:9e3:4086:64c8:d38c
Counter: 109 from: fdbd:ef05:ee02:55b7:9e3:4086:64c8:d38c
Counter: 114 from: fdbd:ef05:ee02:55b7:9e3:4086:64c8:d38c
Counter: 119 from: fdbd:ef05:ee02:55b7:9e3:4086:64c8:d38c
Counter: 124 from: fdbd:ef05:ee02:55b7:9e3:4086:64c8:d38c
Counter: 129 from: fdbd:ef05:ee02:55b7:9e3:4086:64c8:d38c
Counter: 134 from: fdbd:ef05:ee02:55b7:9e3:4086:64c8:d38c
Counter: 139 from: fdbd:ef05:ee02:55b7:9e3:4086:64c8:d38c
Counter: 144 from: fdbd:ef05:ee02:55b7:9e3:4086:64c8:d38c
Counter: 149 from: fdbd:ef05:ee02:55b7:9e3:4086:64c8:d38c
Counter: 154 from: fdbd:ef05:ee02:55b7:9e3:4086:64c8:d38c
```

Network Diagram

Given the following image (obtained from the second reference [2]):

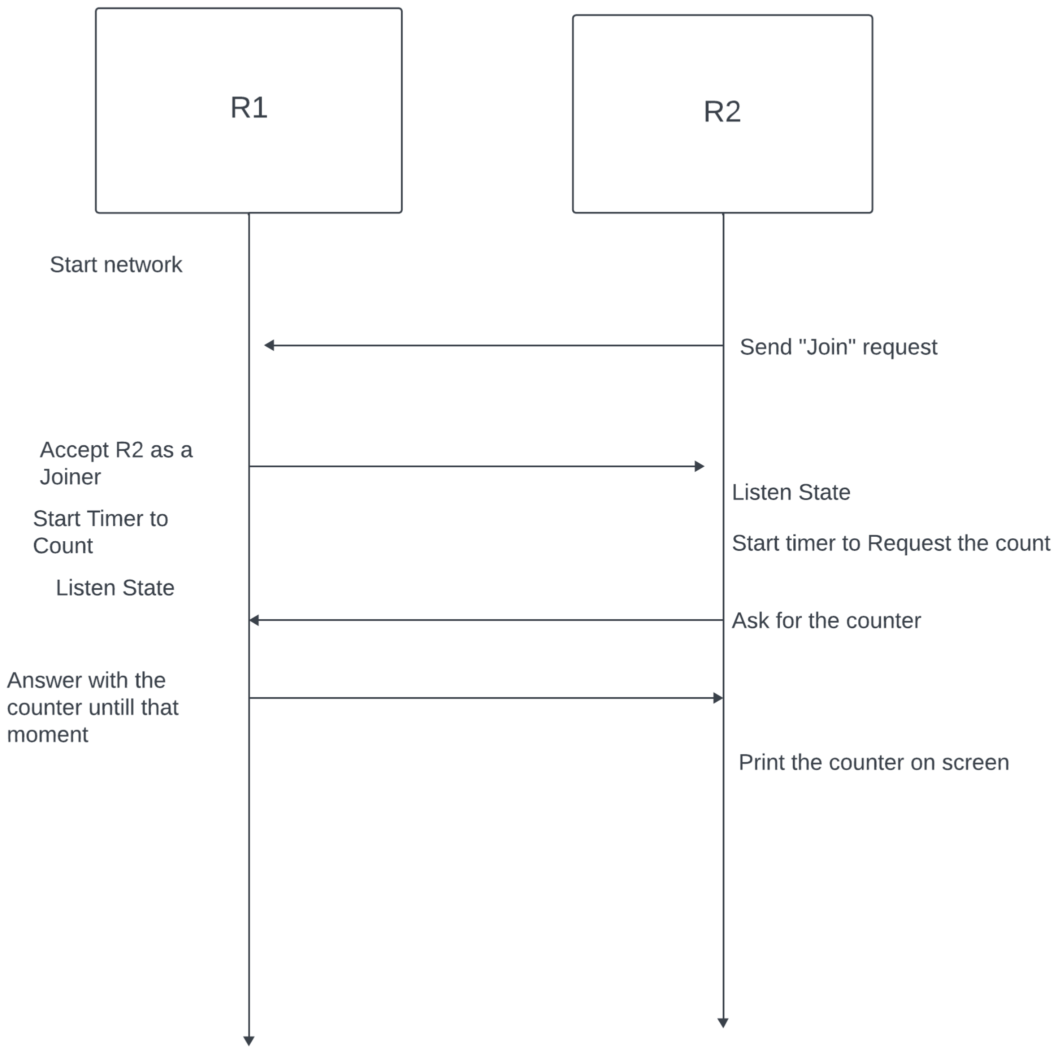


We can assume that our network diagram of two board looks like this:



Where R1 is Router 1 and R2 Router2. As we already mention in the Theoretical framework, the Leader is an additional role of one Router in a Thread network. The Leader is an elected role of one Router, which takes certain decisions in the Thread network such as allowing REEDs to upgrade to Routers. That's why in a gray pentagon.

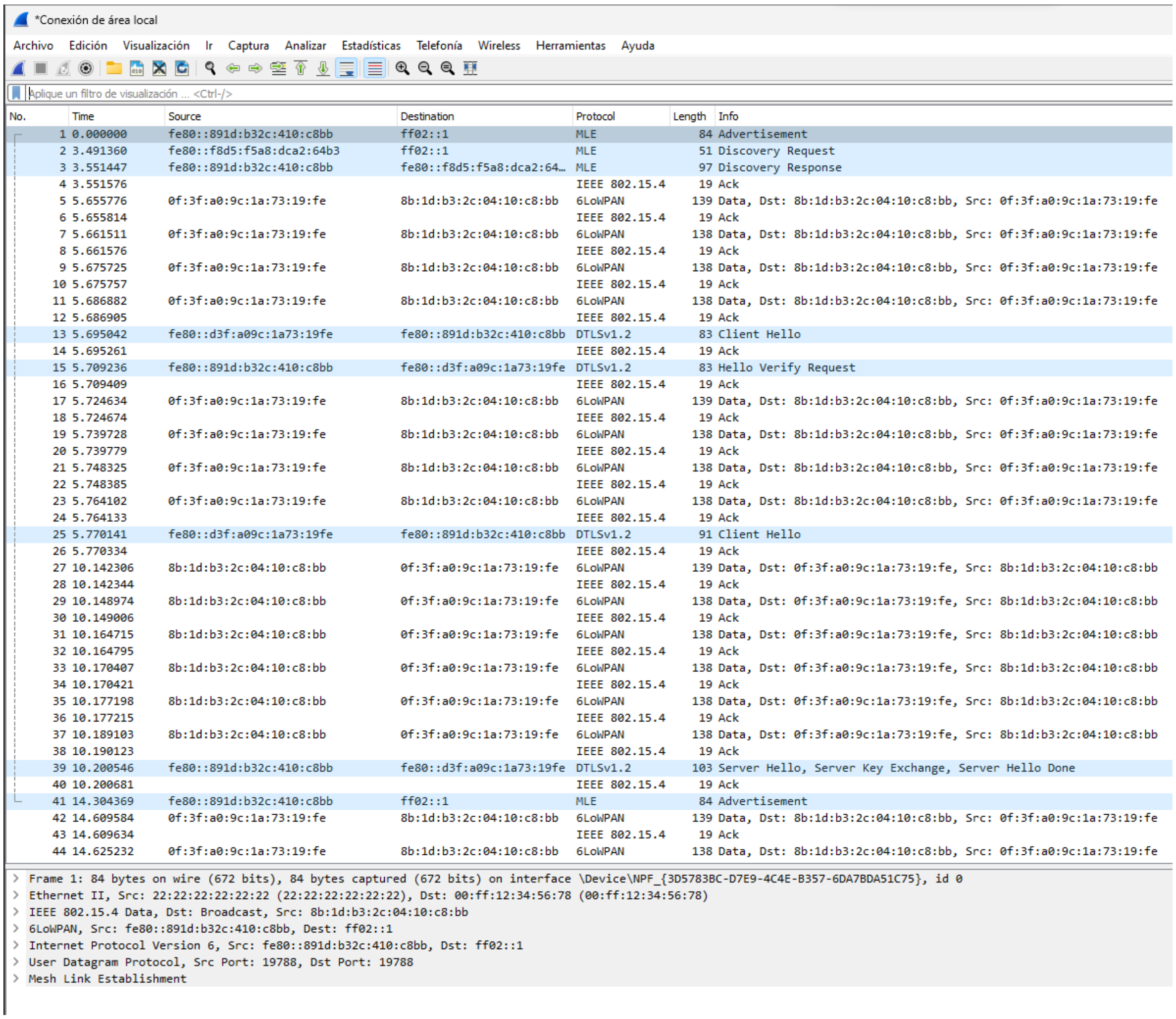
Sequence diagram



After the “Print the counter on screen”, the sequence from “Ask for the counter” form R2 to R1 start again.

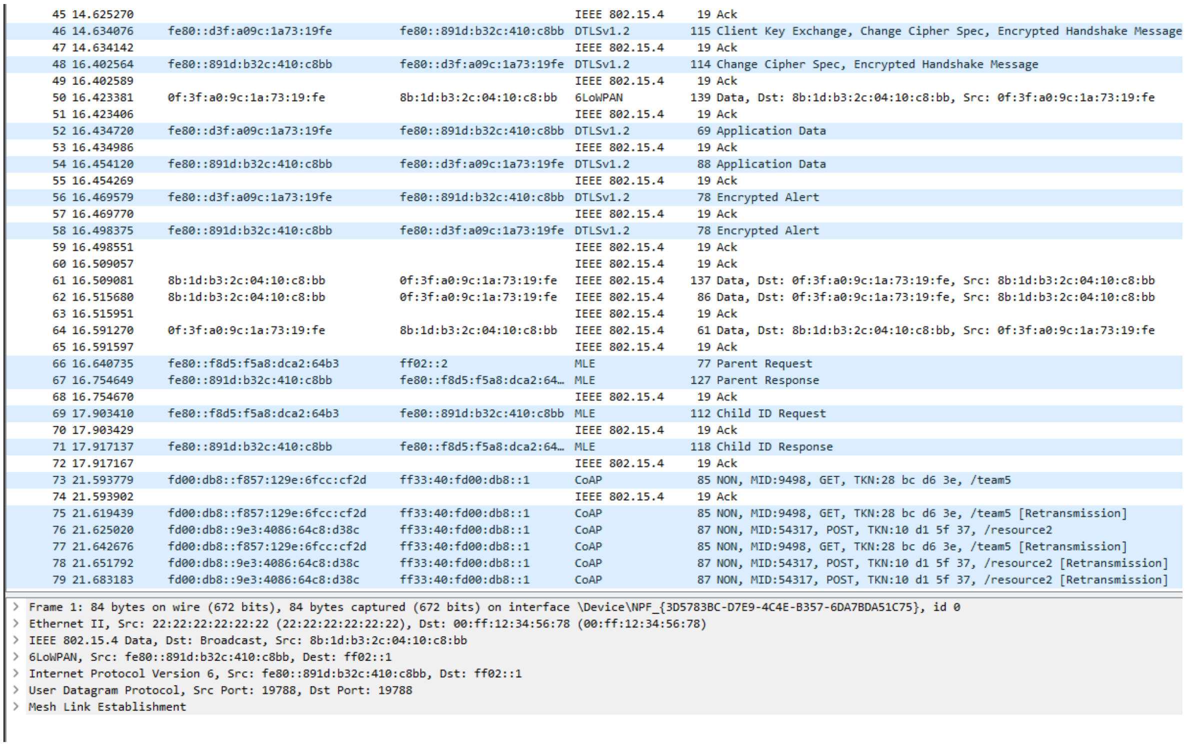
Sniffer Captures

The next image is wireshark capturing the packages that the boards send to be connected. It starts with the discovery request (message to know that someone wants to join to the network, the they start to verify with “Client Hello” and the respective answer that is the “Server Hello”, then they send the Key change message (Server and Client respectively), after that they make an Encrypted Handshake Message, then they do a Application Data protocol exchange, also an Encrypted Alter exchange. AND finally they do the Paren and child request-response):



No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	fe80::891d:b32c:410:c8bb	ff02::1	MLE	84	Advertisement
2	3.491360	fe80::f8d5:f5a8:dca2:64b3	ff02::1	MLE	51	Discovery Request
3	3.551447	fe80::891d:b32c:410:c8bb	fe80::f8d5:f5a8:dca2:64…	MLE	97	Discovery Response
4	3.551576			IEEE 802.15.4	19	Ack
5	5.655776	0f:3f:a0:9c:1a:73:19:fe	8b:1d:b3:2c:04:10:c8:bb	6LoWPAN	139	Data, Dst: 8b:1d:b3:2c:04:10:c8:bb, Src: 0f:3f:a0:9c:1a:73:19:fe
6	5.655814			IEEE 802.15.4	19	Ack
7	5.661511	0f:3f:a0:9c:1a:73:19:fe	8b:1d:b3:2c:04:10:c8:bb	6LoWPAN	138	Data, Dst: 8b:1d:b3:2c:04:10:c8:bb, Src: 0f:3f:a0:9c:1a:73:19:fe
8	5.661576			IEEE 802.15.4	19	Ack
9	5.675725	0f:3f:a0:9c:1a:73:19:fe	8b:1d:b3:2c:04:10:c8:bb	6LoWPAN	138	Data, Dst: 8b:1d:b3:2c:04:10:c8:bb, Src: 0f:3f:a0:9c:1a:73:19:fe
10	5.675757			IEEE 802.15.4	19	Ack
11	5.686882	0f:3f:a0:9c:1a:73:19:fe	8b:1d:b3:2c:04:10:c8:bb	6LoWPAN	138	Data, Dst: 8b:1d:b3:2c:04:10:c8:bb, Src: 0f:3f:a0:9c:1a:73:19:fe
12	5.686905			IEEE 802.15.4	19	Ack
13	5.695042	fe80::d3f:a09c:1a73:19fe	fe80::891d:b32c:410:c8bb	DTLSv1.2	83	Client Hello
14	5.695261			IEEE 802.15.4	19	Ack
15	5.709236	fe80::891d:b32c:410:c8bb	fe80::d3f:a09c:1a73:19fe	DTLSv1.2	83	Hello Verify Request
16	5.709409			IEEE 802.15.4	19	Ack
17	5.724634	0f:3f:a0:9c:1a:73:19:fe	8b:1d:b3:2c:04:10:c8:bb	6LoWPAN	139	Data, Dst: 8b:1d:b3:2c:04:10:c8:bb, Src: 0f:3f:a0:9c:1a:73:19:fe
18	5.724674			IEEE 802.15.4	19	Ack
19	5.739728	0f:3f:a0:9c:1a:73:19:fe	8b:1d:b3:2c:04:10:c8:bb	6LoWPAN	138	Data, Dst: 8b:1d:b3:2c:04:10:c8:bb, Src: 0f:3f:a0:9c:1a:73:19:fe
20	5.739779			IEEE 802.15.4	19	Ack
21	5.748325	0f:3f:a0:9c:1a:73:19:fe	8b:1d:b3:2c:04:10:c8:bb	6LoWPAN	138	Data, Dst: 8b:1d:b3:2c:04:10:c8:bb, Src: 0f:3f:a0:9c:1a:73:19:fe
22	5.748385			IEEE 802.15.4	19	Ack
23	5.764102	0f:3f:a0:9c:1a:73:19:fe	8b:1d:b3:2c:04:10:c8:bb	6LoWPAN	138	Data, Dst: 8b:1d:b3:2c:04:10:c8:bb, Src: 0f:3f:a0:9c:1a:73:19:fe
24	5.764133			IEEE 802.15.4	19	Ack
25	5.770141	fe80::d3f:a09c:1a73:19fe	fe80::891d:b32c:410:c8bb	DTLSv1.2	91	Client Hello
26	5.770334			IEEE 802.15.4	19	Ack
27	10.142306	8b:1d:b3:2c:04:10:c8:bb	0f:3f:a0:9c:1a:73:19:fe	6LoWPAN	139	Data, Dst: 0f:3f:a0:9c:1a:73:19:fe, Src: 8b:1d:b3:2c:04:10:c8:bb
28	10.142344			IEEE 802.15.4	19	Ack
29	10.148974	8b:1d:b3:2c:04:10:c8:bb	0f:3f:a0:9c:1a:73:19:fe	6LoWPAN	138	Data, Dst: 0f:3f:a0:9c:1a:73:19:fe, Src: 8b:1d:b3:2c:04:10:c8:bb
30	10.149006			IEEE 802.15.4	19	Ack
31	10.164715	8b:1d:b3:2c:04:10:c8:bb	0f:3f:a0:9c:1a:73:19:fe	6LoWPAN	138	Data, Dst: 0f:3f:a0:9c:1a:73:19:fe, Src: 8b:1d:b3:2c:04:10:c8:bb
32	10.164795			IEEE 802.15.4	19	Ack
33	10.170407	8b:1d:b3:2c:04:10:c8:bb	0f:3f:a0:9c:1a:73:19:fe	6LoWPAN	138	Data, Dst: 0f:3f:a0:9c:1a:73:19:fe, Src: 8b:1d:b3:2c:04:10:c8:bb
34	10.170421			IEEE 802.15.4	19	Ack
35	10.177198	8b:1d:b3:2c:04:10:c8:bb	0f:3f:a0:9c:1a:73:19:fe	6LoWPAN	138	Data, Dst: 0f:3f:a0:9c:1a:73:19:fe, Src: 8b:1d:b3:2c:04:10:c8:bb
36	10.177215			IEEE 802.15.4	19	Ack
37	10.189103	8b:1d:b3:2c:04:10:c8:bb	0f:3f:a0:9c:1a:73:19:fe	6LoWPAN	138	Data, Dst: 0f:3f:a0:9c:1a:73:19:fe, Src: 8b:1d:b3:2c:04:10:c8:bb
38	10.190123			IEEE 802.15.4	19	Ack
39	10.200546	fe80::891d:b32c:410:c8bb	fe80::d3f:a09c:1a73:19fe	DTLSv1.2	103	Server Hello, Server Key Exchange, Server Hello Done
40	10.200681			IEEE 802.15.4	19	Ack
41	14.304369	fe80::891d:b32c:410:c8bb	ff02::1	MLE	84	Advertisement
42	14.609584	0f:3f:a0:9c:1a:73:19:fe	8b:1d:b3:2c:04:10:c8:bb	6LoWPAN	139	Data, Dst: 8b:1d:b3:2c:04:10:c8:bb, Src: 0f:3f:a0:9c:1a:73:19:fe
43	14.609634			IEEE 802.15.4	19	Ack
44	14.625232	0f:3f:a0:9c:1a:73:19:fe	8b:1d:b3:2c:04:10:c8:bb	6LoWPAN	138	Data, Dst: 8b:1d:b3:2c:04:10:c8:bb, Src: 0f:3f:a0:9c:1a:73:19:fe

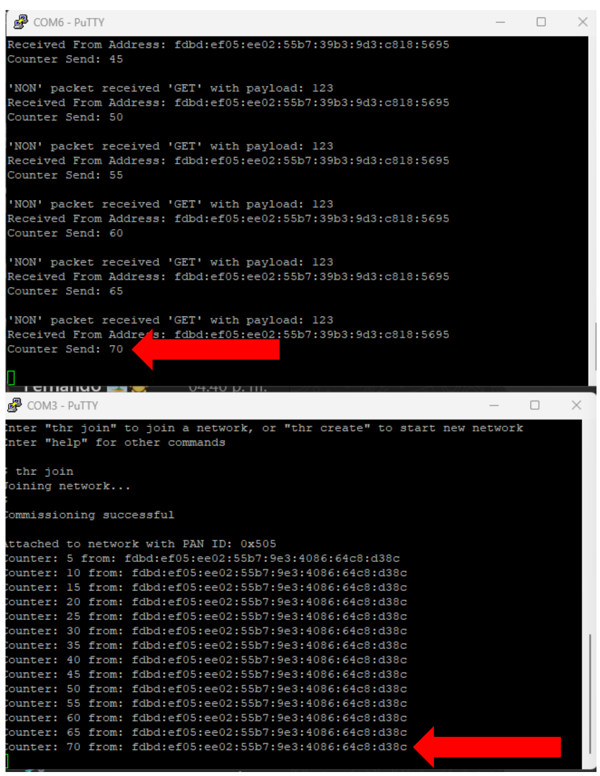
```
> Frame 1: 84 bytes on wire (672 bits), 84 bytes captured (672 bits) on interface \Device\NPF_{3D5783BC-D7E9-4C4E-B357-6DA7BDA51C75}, id 0
> Ethernet II, Src: 22:22:22:22:22:22 (22:22:22:22:22:22), Dst: 00:ff:12:34:56:78 (00:ff:12:34:56:78)
> IEEE 802.15.4 Data, Dst: Broadcast, Src: 8b:1d:b3:2c:04:10:c8:bb
> 6LoWPAN, Src: fe80::891d:b32c:410:c8bb, Dest: ff02::1
> Internet Protocol Version 6, Src: fe80::891d:b32c:410:c8bb, Dst: ff02::1
> User Datagram Protocol, Src Port: 19788, Dst Port: 19788
> Mesh Link Establishment
```



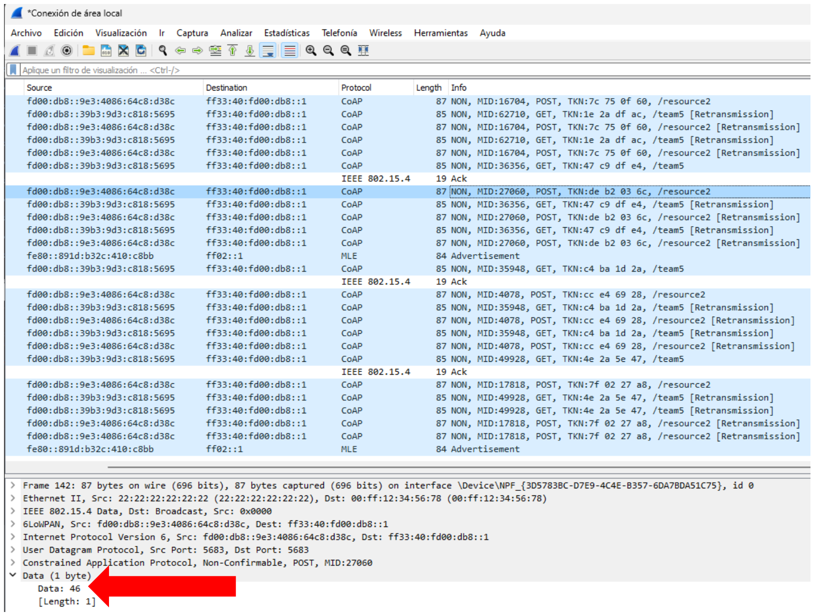
45	14.625270			IEEE 802.15.4	19	Ack
46	14.634076	fe80::d3f:a09c:1a73:19fe	fe80::891d:b32c:410:c8bb	DTLSv1.2	115	Client Key Exchange, Change Cipher Spec, Encrypted Handshake Message
47	14.634142			IEEE 802.15.4	19	Ack
48	16.402564	fe80::891d:b32c:410:c8bb	fe80::d3f:a09c:1a73:19fe	DTLSv1.2	114	Change Cipher Spec, Encrypted Handshake Message
49	16.402589			IEEE 802.15.4	19	Ack
50	16.423381	0f:3f:a0:9c:1a:73:19:fe	8b:1d:b3:2c:04:10:c8:bb	6LoWPAN	139	Data, Dst: 8b:1d:b3:2c:04:10:c8:bb, Src: 0f:3f:a0:9c:1a:73:19:fe
51	16.423406			IEEE 802.15.4	19	Ack
52	16.434720	fe80::d3f:a09c:1a73:19fe	fe80::891d:b32c:410:c8bb	DTLSv1.2	69	Application Data
53	16.434986			IEEE 802.15.4	19	Ack
54	16.454120	fe80::891d:b32c:410:c8bb	fe80::d3f:a09c:1a73:19fe	DTLSv1.2	88	Application Data
55	16.454269			IEEE 802.15.4	19	Ack
56	16.469579	fe80::d3f:a09c:1a73:19fe	fe80::891d:b32c:410:c8bb	DTLSv1.2	78	Encrypted Alert
57	16.469770			IEEE 802.15.4	19	Ack
58	16.498375	fe80::891d:b32c:410:c8bb	fe80::d3f:a09c:1a73:19fe	DTLSv1.2	78	Encrypted Alert
59	16.498551			IEEE 802.15.4	19	Ack
60	16.509057			IEEE 802.15.4	19	Ack
61	16.509081	8b:1d:b3:2c:04:10:c8:bb	0f:3f:a0:9c:1a:73:19:fe	IEEE 802.15.4	137	Data, Dst: 0f:3f:a0:9c:1a:73:19:fe, Src: 8b:1d:b3:2c:04:10:c8:bb
62	16.515680	8b:1d:b3:2c:04:10:c8:bb	0f:3f:a0:9c:1a:73:19:fe	IEEE 802.15.4	86	Data, Dst: 0f:3f:a0:9c:1a:73:19:fe, Src: 8b:1d:b3:2c:04:10:c8:bb
63	16.515951			IEEE 802.15.4	19	Ack
64	16.591270	0f:3f:a0:9c:1a:73:19:fe	8b:1d:b3:2c:04:10:c8:bb	IEEE 802.15.4	61	Data, Dst: 8b:1d:b3:2c:04:10:c8:bb, Src: 0f:3f:a0:9c:1a:73:19:fe
65	16.591597			IEEE 802.15.4	19	Ack
66	16.640735	fe80::f8d5:f5a8:dca2:64b3	ff02::2	MLE	77	Parent Request
67	16.754649	fe80::891d:b32c:410:c8bb	fe80::f8d5:f5a8:dca2:64…	MLE	127	Parent Response
68	16.754670			IEEE 802.15.4	19	Ack
69	17.903410	fe80::f8d5:f5a8:dca2:64b3	fe80::891d:b32c:410:c8bb	MLE	112	Child ID Request
70	17.903429			IEEE 802.15.4	19	Ack
71	17.917137	fe80::891d:b32c:410:c8bb	fe80::f8d5:f5a8:dca2:64…	MLE	118	Child ID Response
72	17.917167			IEEE 802.15.4	19	Ack
73	21.593779	fd00:db8::f857:129e:6fcc:cf2d	ff33:40:fd00:db8::1	CoAP	85	NON, MID:9498, GET, TKN:28 bc d6 3e, /team5
74	21.593902			IEEE 802.15.4	19	Ack
75	21.619439	fd00:db8::f857:129e:6fcc:cf2d	ff33:40:fd00:db8::1	CoAP	85	NON, MID:9498, GET, TKN:28 bc d6 3e, /team5 [Retransmission]
76	21.625020	fd00:db8::9e3:4086:64c8:d38c	ff33:40:fd00:db8::1	CoAP	87	NON, MID:54317, POST, TKN:10 d1 5f 37, /resource2
77	21.642676	fd00:db8::f857:129e:6fcc:cf2d	ff33:40:fd00:db8::1	CoAP	85	NON, MID:9498, GET, TKN:28 bc d6 3e, /team5 [Retransmission]
78	21.651792	fd00:db8::9e3:4086:64c8:d38c	ff33:40:fd00:db8::1	CoAP	87	NON, MID:54317, POST, TKN:10 d1 5f 37, /resource2 [Retransmission]
79	21.683183	fd00:db8::9e3:4086:64c8:d38c	ff33:40:fd00:db8::1	CoAP	87	NON, MID:54317, POST, TKN:10 d1 5f 37, /resource2 [Retransmission]

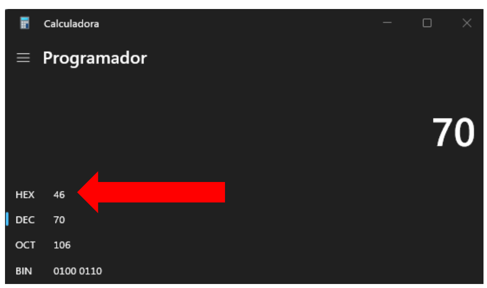
Frame 1: 84 bytes on wire (672 bits), 84 bytes captured (672 bits) on interface \Device\NPF_{3D5783BC-D7E9-4C4E-B357-6DA7BDA51C75}, id 0
Ethernet II, Src: 22:22:22:22:22:22 (22:22:22:22:22:22), Dst: 00:ff:12:34:56:78 (00:ff:12:34:56:78)
IEEE 802.15.4 Data, Dst: Broadcast, Src: 8b:1d:b3:2c:04:10:c8:bb
6LoWPAN, Src: fe80::891d:b32c:410:c8bb, Dest: ff02::1
Internet Protocol Version 6, Src: fe80::891d:b32c:410:c8bb, Dst: ff02::1
User Datagram Protocol, Src Port: 19788, Dst Port: 19788
Mesh Link Establishment

For the package with the counter 70, we can see it reflected on wireshark:

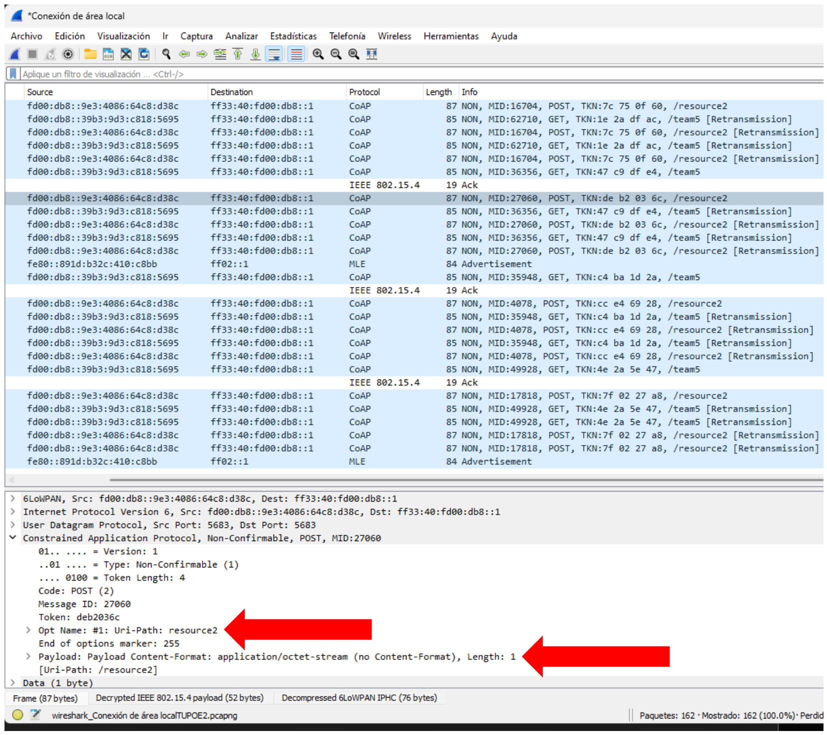


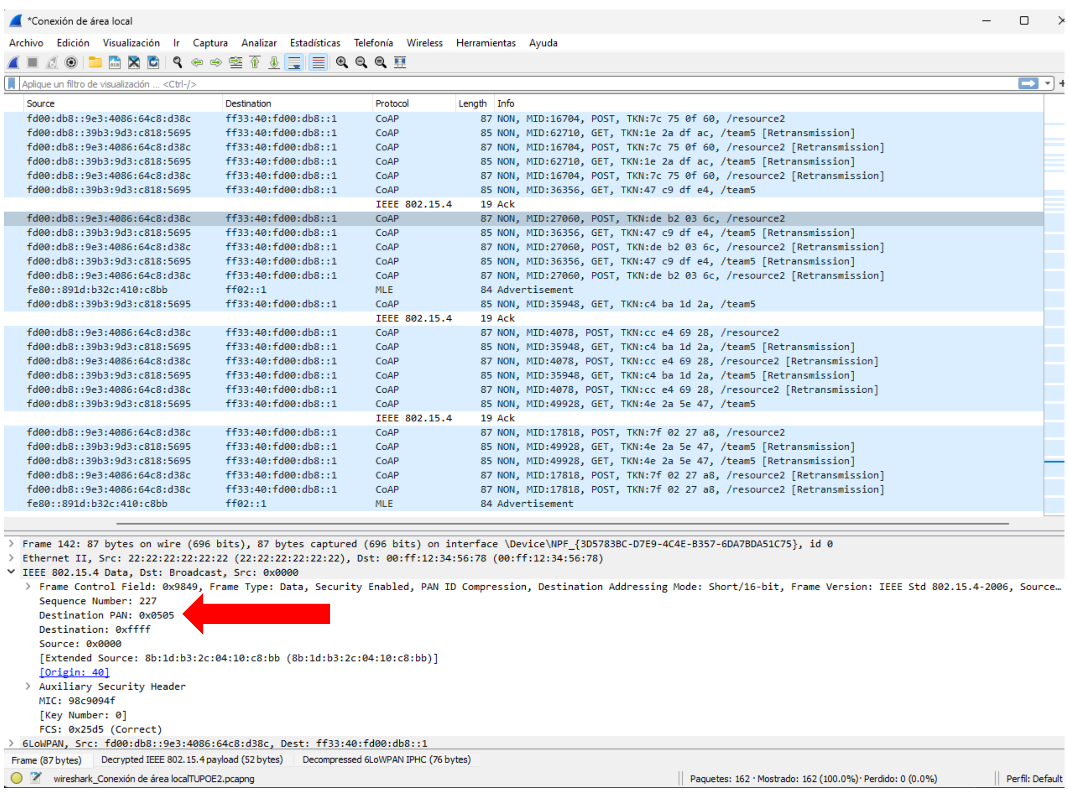
We can see the data 46 (hexadecimal) in wireshark

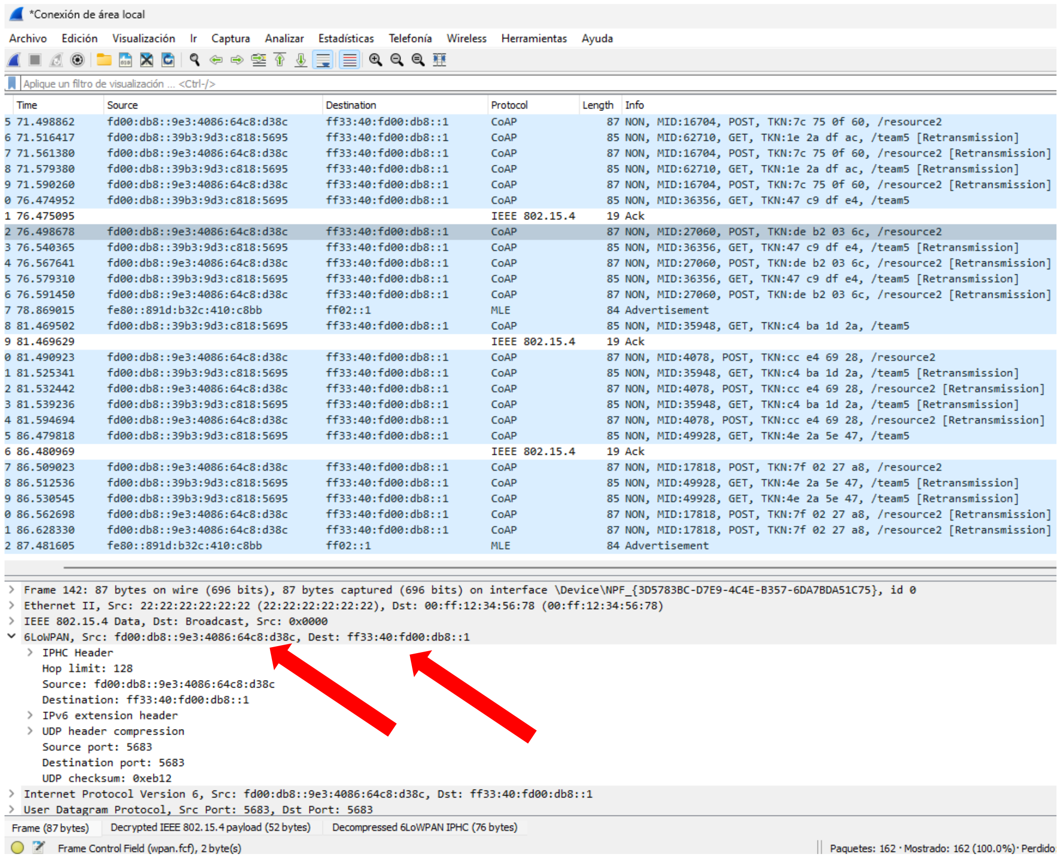




Also we can see other details of the package like the URI, data length specified, PanID, Source and Destination Address:






*Conexión de área local
Archivo Edición Visualización Ir Captura Analizar Estadísticas Telefonía Wireless Herramientas Ayuda
Aplique un filtro de visualización ... <Ctrl-/>
Time Source Destination Protocol Length Info
5 71.498862 fd00:db8::9e3:4086:64c8:d38c ff33:40:fd00:db8::1 CoAP 87 NON, MID:16704, POST, TKN:7c 75 0f 60, /resource2
6 71.516417 fd00:db8::39b3:9d3:c818:5695 ff33:40:fd00:db8::1 CoAP 85 NON, MID:62710, GET, TKN:1e 2a df ac, /team5 [Retransmission]
7 71.561380 fd00:db8::9e3:4086:64c8:d38c ff33:40:fd00:db8::1 CoAP 87 NON, MID:16704, POST, TKN:7c 75 0f 60, /resource2 [Retransmission]
8 71.579380 fd00:db8::39b3:9d3:c818:5695 ff33:40:fd00:db8::1 CoAP 85 NON, MID:62710, GET, TKN:1e 2a df ac, /team5 [Retransmission]
9 71.590260 fd00:db8::9e3:4086:64c8:d38c ff33:40:fd00:db8::1 CoAP 87 NON, MID:16704, POST, TKN:7c 75 0f 60, /resource2 [Retransmission]
0 76.474952 fd00:db8::39b3:9d3:c818:5695 ff33:40:fd00:db8::1 CoAP 85 NON, MID:36356, GET, TKN:47 c9 df e4, /team5
1 76.475095 IEEE 802.15.4 19 Ack
2 76.498678 fd00:db8::9e3:4086:64c8:d38c ff33:40:fd00:db8::1 CoAP 87 NON, MID:27060, POST, TKN:de b2 03 6c, /resource2
3 76.540365 fd00:db8::39b3:9d3:c818:5695 ff33:40:fd00:db8::1 CoAP 85 NON, MID:36356, GET, TKN:47 c9 df e4, /team5 [Retransmission]
4 76.567641 fd00:db8::9e3:4086:64c8:d38c ff33:40:fd00:db8::1 CoAP 87 NON, MID:27060, POST, TKN:de b2 03 6c, /resource2 [Retransmission]
5 76.579310 fd00:db8::39b3:9d3:c818:5695 ff33:40:fd00:db8::1 CoAP 85 NON, MID:36356, GET, TKN:47 c9 df e4, /team5 [Retransmission]
6 76.591450 fd00:db8::9e3:4086:64c8:d38c ff33:40:fd00:db8::1 CoAP 87 NON, MID:27060, POST, TKN:de b2 03 6c, /resource2 [Retransmission]
7 78.869015 fe80::891d:b32c:410:c8bb ff02::1 MLE 84 Advertisement
8 81.469502 fd00:db8::39b3:9d3:c818:5695 ff33:40:fd00:db8::1 CoAP 85 NON, MID:35948, GET, TKN:c4 ba 1d 2a, /team5
9 81.469629 IEEE 802.15.4 19 Ack
0 81.490923 fd00:db8::9e3:4086:64c8:d38c ff33:40:fd00:db8::1 CoAP 87 NON, MID:4078, POST, TKN:cc e4 69 28, /resource2
1 81.525341 fd00:db8::39b3:9d3:c818:5695 ff33:40:fd00:db8::1 CoAP 85 NON, MID:35948, GET, TKN:c4 ba 1d 2a, /team5 [Retransmission]
2 81.532442 fd00:db8::9e3:4086:64c8:d38c ff33:40:fd00:db8::1 CoAP 87 NON, MID:4078, POST, TKN:cc e4 69 28, /resource2 [Retransmission]
3 81.539236 fd00:db8::39b3:9d3:c818:5695 ff33:40:fd00:db8::1 CoAP 85 NON, MID:35948, GET, TKN:c4 ba 1d 2a, /team5 [Retransmission]
4 81.594694 fd00:db8::9e3:4086:64c8:d38c ff33:40:fd00:db8::1 CoAP 87 NON, MID:4078, POST, TKN:cc e4 69 28, /resource2 [Retransmission]
5 86.479818 fd00:db8::39b3:9d3:c818:5695 ff33:40:fd00:db8::1 CoAP 85 NON, MID:49928, GET, TKN:4e 2a 5e 47, /team5
6 86.480969 IEEE 802.15.4 19 Ack
7 86.509023 fd00:db8::9e3:4086:64c8:d38c ff33:40:fd00:db8::1 CoAP 87 NON, MID:17818, POST, TKN:7f 02 27 a8, /resource2
8 86.512536 fd00:db8::39b3:9d3:c818:5695 ff33:40:fd00:db8::1 CoAP 85 NON, MID:49928, GET, TKN:4e 2a 5e 47, /team5 [Retransmission]
9 86.530545 fd00:db8::39b3:9d3:c818:5695 ff33:40:fd00:db8::1 CoAP 85 NON, MID:49928, GET, TKN:4e 2a 5e 47, /team5 [Retransmission]
0 86.562698 fd00:db8::9e3:4086:64c8:d38c ff33:40:fd00:db8::1 CoAP 87 NON, MID:17818, POST, TKN:7f 02 27 a8, /resource2 [Retransmission]
1 86.628330 fd00:db8::9e3:4086:64c8:d38c ff33:40:fd00:db8::1 CoAP 87 NON, MID:17818, POST, TKN:7f 02 27 a8, /resource2 [Retransmission]
2 87.481605 fe80::891d:b32c:410:c8bb ff02::1 MLE 84 Advertisement
Frame 142: 87 bytes on wire (696 bits), 87 bytes captured (696 bits) on interface \Device\NPF_{3D5783BC-D7E9-4C4E-B357-6DA7BDA51C75}, id 0
Ethernet II, Src: 22:22:22:22:22:22 (22:22:22:22:22:22), Dst: 00:ff:12:34:56:78 (00:ff:12:34:56:78)
IEEE 802.15.4 Data, Dst: Broadcast, Src: 0x0000
6LoWPAN, Src: fd00:db8::9e3:4086:64c8:d38c, Dest: ff33:40:fd00:db8::1
IPHC Header
Hop limit: 128
Source: fd00:db8::9e3:4086:64c8:d38c
Destination: ff33:40:fd00:db8::1
IPv6 extension header
UDP header compression
Source port: 5683
Destination port: 5683
UDP checksum: 0xeb12
Internet Protocol Version 6, Src: fd00:db8::9e3:4086:64c8:d38c, Dst: ff33:40:fd00:db8::1
User Datagram Protocol, Src Port: 5683, Dst Port: 5683
Frame (87 bytes) Decrypted IEEE 802.15.4 payload (52 bytes) Decompressed 6LoWPAN IPHC (76 bytes)
Frame Control Field (wpan.fcf), 2 byte(s)
Paquetes: 162 · Mostrado: 162 (100.0%) · Perdido

Conclusions

Sergio Eduardo Borrayo: In conclusion this homework was more confusing than the previous one, it took us a lot of time to comprehend the flow of the code principally, the timers were easy to implement, but we needed to discover were to start them. After that we had some problems trying to put in the shell the counter that the R2 received in decimals, for the IP addresses another team helped us with the function ntop for the address to make it string.

After we solve how to paste everything in the shell we wasted a lot of time trying to discover why if we put a Confirmable message in the R2 to the leader it always printed a Nonconfirmable, it was frustrating, at the end we couldn't do that and with the time wasted we tried to implement at least the initialization of the accelerometer, but we couldn't do it neither, so yeah, we think the part 1 is well implemented but part 2 is not done.

Isaac Segovia Hernández: Thread is a protocol that is easy to use and with great flexibility that (it seems) is starting to have great reach. Regarding the practice, it was easy to implement the functions, but I consider that the complicated part was to finish understanding how it was originally coded and be able to make use of the functions. Finding the desired parts of the code to manipulate was the most time consuming, as well as validation. The validation was time consuming due to the IP that changes every time the board was reprogrammed (or reset to factory settings). The timers were very easy to understand and implement, there were no difficulties there.

Some real issues where more related to syntax and general knowledge of C, specifically to print in screen the address, it was eventually solved, but we get stuck a little bit there. Initially it was printing garbage or other characters in ascii, but after a good review of some files we found a function that makes proper conversions to be able to display the desired result on the screen.

References

[1]*How does Thread elect a Leader device?* (s/f). Internet of Things Stack Exchange. Recuperado el 22 de noviembre de 2023, de https://iot.stackexchange.com/questions/439/how-does-thread-elect-a-leader-device

[2](S/f). Threadgroup.org. Recuperado el 22 de noviembre de 2023, de https://www.threadgroup.org/Portals/0/documents/support/Thread%20Network%20Fundamentals_v3.pdf