INSTITUTO TECNOLÓGICO Y DE ESTUDIOS SUPERIORES DE OCCIDENTE

Department of Engineering, Systems and Informatic

Practice 2

“Thread”

Members:

Diego Guart SI704548@iteso.mx

Pablo Alberto Avalos Chávez IE714518@iteso.mx

Fernanda Edith Galeana Torres IE718052@iteso.mx

Profesor:

Miguel Agustín Gonzales Rivera

Topic:

Networks for Embedded Systems

Tlaquepaque, Tuesday 2 of November of 2021

Introduction:

In this practice we will apply the knowledge of thread technology alongside with the implementations of Coap with the goal of programing two devices, one for a leader device and other for a router device in which both of them will have to meet the needed requirements.

Thread concepts:

6lowpan:

This stands for the ipv6 over low power wireless personal area network enabling the ipv6 protocol to work on IE 802.15.4 based networks achieving the necessary compression rates, fragmentation and forwarding capabilities to ensure the low power consumption.

Coap:

This stand for Constrained Application Protocol specialized for web transfer data in constrained nodes and networks enabling them to add new devices in a simple way with low bandwidth and availability, featuring simple over heading with Uniform resource identifier (URI) using API methodologies.

Bridge Router:

first point of contact between the thread network and the outside LAN/WAN with the capabilities of delivering WIFI connection to the area

RLOC:

Standing for Routing Locator this is the address that it is assigned to each device that connects to the thread network using 16 bits this allows a unique id to each device in the network

ML-EID:

Standing for Mesh local Endpoint Identifier as an independent topology IPv6 address that is routable on the thread network with the capacity to persist over power cycling

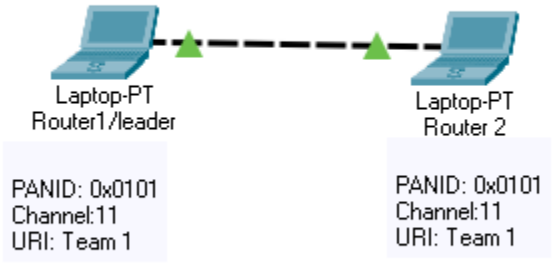
MPL:

It stands for Multicast Protocol for low power and lossy networks allowing them to communicate with no matter the topology used.

DTLS:

Datagram transport layer security providing privacy and other properties that protects the data transmitted

Topology:



Modifications made on code:

```
#ifndef THR_SCANCHANNEL_MASK
    #define THR_SCANCHANNEL_MASK                        (0x00000001 << 11)
#endif


/*! The scan duration time. This is an exponential scale, as seen in the 802.15.4 specification.
   Range:   1 - 14 */
#ifndef THR_SCAN_DURATION
    #define THR_SCAN_DURATION                           2
#endif

/*! Network creation channel. If different form 0, the router which creates a new
   network by calling THR_NwkCreate will OVERRIDE the SCAN channel and
   only use this channel. Range: 0, 11-26 */
#ifndef THR_NWK_CREATE_CHANNEL
    #define THR_NWK_CREATE_CHANNEL                      0
#endif

/*! 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                                  0x0101
#endif
```

in both projects the variables THR_SCANCHANNEL_MSK y THR_PAN_ID were

modified in order to match the one of our team number

```
87 #define APP_TIMER_URI_PATH                    "/team1"
88 #define APP_ACCEL_URI_PATH                    "/accel"
```

Also in both projects we defined the UIR Path

```
156 const coapUriPath_t gAPP_TIMER_URI_PATH = {SizeOfString(APP_TIMER_URI_PATH), (uint8_t *)APP_TIMER_URI_PATH};
157 const coapUriPath_t gAPP_ACCEL_URI_PATH = {SizeOfString(APP_ACCEL_URI_PATH), (uint8_t *)APP_ACCEL_URI_PATH};
```

We created the constants for those URI paths

```
513                                     {APP_CoapCounterRequestCb, (coapUriPath_t *)&gAPP_TIMER_URI_PATH},
514                                     {APP_CoapAccelRequestCb, (coapUriPath_t*)&gAPP_ACCEL_URI_PATH},
```

We assigned the callback to the Coap URI path

```
1592 static void APP_CoapCounterRequestCb(coapSessionStatus_t sessionStatus, uint8_t *pData, coapSession_t *pSession, uint32_t dataLen)
1593 {
1594     static uint8_t pMySessionPayload[3] = {0x31,0x32,0x33};
1595     static uint32_t pMyPayloadSize=3;
1596     char addrStr[INET6_ADDRSTRLEN];
1597     coapReqRespCodes_t requestCode = pSession->code;
1598     ntop(AF_INET6, (ipAddr_t*)&pSession->remoteAddrStorage.ss_addr, addrStr, INET6_ADDRSTRLEN);
1599     /* Send CoAP Ack if the message is CON */
1600     if (gCoapConfirmable_c == pSession->msgType)
1601     {
1602         /* Send the CoAP Ack */
1603         if (gCoapFailure_c!=sessionStatus)
1604         {
1605           COAP_Send(pSession, gCoapMsgTypeAckSuccessChanged_c, pMySessionPayload, pMyPayloadSize);
1606         }
1607     }
1608     if(gCoapPOST_c == pSession->code)
1609     {
1610         shell_printf("\tCounter = %u from: %s\n\r", *pData, addrStr);
1611     }
1612
1613 }
```

In the router 2 we added the callback of the request of the counter in order to print the value that receives for the /team1

```
1553 static void APP_SendAccelRequest(uint8_t *pParam)
1554 {
1555     uint8_t data = 50;
1556     coapMessageTypes_t requestType = gCoapConfirmable_c;
1557     coapReqRespCodes_t requestCode = gCoapGET_c;
1558
1559     if(!IP_IF_IsMyAddr(gIpIfSlp0_c, &gCoapDestAddress))
1560     {
1561         /* Start CoAP session */
1562         coapSession_t *pReqSession = COAP_OpenSession(mAppCoapInstId);
1563
1564         if(pReqSession)
1565         {
1566
1567             pReqSession->msgType = requestType;
1568             pReqSession->code = requestCode;
1569             pReqSession->pCallback = NULL;
1570             FLib_MemCpy(&pReqSession->remoteAddrStorage.ss_addr, &gCoapDestAddress, sizeof(ipAddr_t));
1571             pReqSession->pUriPath = (coapUriPath_t *)&gAPP_ACCEL_URI_PATH;
1572             if(!IP6_IsMulticastAddr(&gCoapDestAddress))
1573             {
1574                 COAP_SetCallback(pReqSession, APP_CoapGenericCallback);
1575             }
1576
1577             else
1578             {
1579                 COAP_SetCallback(pReqSession, APP_CoapAccelRequestCb);
1580             }
1581
1582             COAP_Send(pReqSession, gCoapMsgTypeUseSessionValues_c, &data, 1);
1583         }
1584     }
1585 }
```

Router2: we added the function to initiate the session of Coap in order to request the values of the counter value in to the router1/leader

```
1625 static void APP_CoapAccelRequestCb(coapSessionStatus_t sessionStatus, uint8_t *pData, coapSession_t *pSession, uint32_t dataLen)
1626 {
1627     static uint8_t pMySessionPayload[3] = {0x31,0x32,0x33};
1628     static uint32_t pMyPayloadSize=3;
1629     accel_data SentData;
1630     char addrStr[INET6_ADDRSTRLEN];
1631     coapReqRespCodes_t requestCode = pSession->code;
1632     ntop(AF_INET6, (ipAddr_t*)&pSession->remoteAddrStorage.ss_addr, addrStr, INET6_ADDRSTRLEN);
1633     /* Copy the received data to the variable */
1634     FLib_MemCpy(&SentData,pData,dataLen);
1635     /* Send CoAP Ack if the message is CON */
1636     if (gCoapConfirmable_c == pSession->msgType)
1637     {
1638         /* Send the CoAP Ack */
1639         if (gCoapFailure_c!=sessionStatus)
1640         {
1641           COAP_Send(pSession, gCoapMsgTypeAckSuccessChanged_c, pMySessionPayload, pMyPayloadSize);
1642         }
1643     }
1644     if(gCoapPOST_c == requestCode)
1645     {
1646         shell_printf("\tX = %d  Y = %d  Z = %d from IPv6 address: %s\n\r", SentData.xData, SentData.yData, SentData.zData, addrStr)
1647     }
1648
1649 }
```

Router2: callback of the accelerometer request into print the values that it receives for the /accel

```
1586 void APP_CounterConnection(void)
1587 {
1588     (void)NWKU_SendMsg(APP_SendCounterRequest, NULL, mpAppThreadMsgQueue);
1589     (void)NWKU_SendMsg(APP_SendAccelRequest, NULL, mpAppThreadMsgQueue);
1590 }
```

Router2: function to initiate the request of the counter and accelerometer, it is started at the timer callback

```
222     /* Initialize Timer task */
223     MyTask_Init(APP_CounterConnection);
224
```

Router2: we initialized the task for the timer (in the initialization of the application) passing the pointer to the function that will be called in the callback

```
1572 static void APP_CoapAccelCb(coapSessionStatus_t sessionStatus, uint8_t *pData, coapSession_t *pSession, uint32_t dataLen)
1573 {
1574     static uint8_t pMySessionPayload[3] = {0x31,0x32,0x33};
1575     static uint32_t pMyPayloadSize=3;
1576     coapSession_t *pMySession = NULL;
1577     accel_data pDataRead = {0};
1578     coapReqRespCodes_t sessionCode = pSession->code;
1579     char addrStr[INET6_ADDRSTRLEN];
1580     pMySession = COAP_OpenSession(mAppCoapInstId);
1581     /*Change the address to string */
1582     ntop(AF_INET6, (ipAddr_t*)&pSession->remoteAddrStorage.ss_addr, addrStr, INET6_ADDRSTRLEN);
1583     /* If it is a CON request */
1584     if (gCoapConfirmable_c == pSession->msgType)
1585     {
1586         /* Print the requester address */
1587         shell_printf("\tACCEL - CON instruction received from: %s\n\r", addrStr);
1588         /* Send the CoAP Ack */
1589         if (gCoapFailure_c!=sessionStatus)
1590         {
1591           COAP_Send(pSession, gCoapMsgTypeAckSuccessChanged_c, pMySessionPayload, pMyPayloadSize);
1592         }
1593     }
1594     /* If it is a NON request */
1595     else if(gCoapNonConfirmable_c == pSession->msgType)
1596     {
1597         /* Print the requester address */
1598         shell_printf("\tACCEL - NON instruction received from: %s\n\r", addrStr);
1599     }
1600
```

Router1/leader: callback of the accelerometer /accel

```
1518 static void APP_CoapTimerCb(coapSessionStatus_t sessionStatus, uint8_t *pData, coapSession_t *pSession, uint32_t dataLen)
1519 {
1520     static uint8_t pMySessionPayload[3] = {0x31,0x32,0x33};
1521     static uint32_t pMyPayloadSize=3;
1522     coapSession_t *pMySession = NULL;
1523     uint8_t data_counter;
1524     coapReqRespCodes_t sessionCode = pSession->code;
1525     char addrStr[INET6_ADDRSTRLEN];
1526     pMySession = COAP_OpenSession(mAppCoapInstId);
1527     //COAP_AddOptionToList(pMySession,COAP_URI_PATH_OPTION,(uint8_t*)APP_TIMER_URI_PATH,SizeOfString(APP_TIMER_URI_PATH));
1528     FLib_MemCpy(&gCoapDestAddress,&pSession->remoteAddrStorage.ss_addr,sizeof(ipAddr_t));
1529     /* Get counter value */
1530     data_counter = GetCounter();
1531     /*Change the address to string */
1532     ntop(AF_INET6, (ipAddr_t*)&pSession->remoteAddrStorage.ss_addr, addrStr, INET6_ADDRSTRLEN);
1533     /* If it is a CON request */
1534     if (gCoapConfirmable_c == pSession->msgType)
1535     {
1536         /* Print the requester address */
1537         shell_printf("\tTIMER - CON instruction received from: %s\n\r", addrStr);
1538         /* Send the CoAP Ack */
1539         if (gCoapFailure_c!=sessionStatus)
1540         {
1541           COAP_Send(pSession, gCoapMsgTypeAckSuccessChanged_c, pMySessionPayload, pMyPayloadSize);
1542         }
1543     }
1544     /* If it is a NON request */
1545     else if(gCoapNonConfirmable_c == pSession->msgType)
1546     {
1547         /* Print the requester address */
1548         shell_printf("\tTIMER - NON instruction received from: %s\n\r", addrStr);
1549     }
```

Router1/leader: callback for the counter in the timer

Conclusions:

Diego Guart:

In this practice I was able to understand and apply the knowledge of the thread's application in the real world, as the advantages that it offers of being low powered and admitting that much devices in the network made me understand the capabilities of the efficiency of today's protocols and how the need and persons pushes forward to it

Fernanda Galeana:

In this practice I realized the fact that the application of the Coap lab was really helpful as it allowed me to understand the principles of the Thread configuration alongside with the knowledge of the 6Plowpan that permitted to implement the advantages of the ipv6 protocol. the challenges were presented at the moment of handling the information of such addresses so being able to succeed on this improved my understanding of the subject

Pablo Avalos:

In this practice I learned about the thread protocol and its characteristics. I could realize about the complexity that thread can adopt in its network as mesh configuration, because I saw during the practice when I disconnected leader from the network the active router connected could become leader instead, so in this way we could paired more devices to the network. In addition, one of the most difficult parts on this practice, was the task to print the IP address with all the structure, such as mesh local address and the random address because took so much time to us, to find the function where we can print it, but at the end, the practice could be done with the specifications that teacher asked to us.