

# Taller de Proyecto II

**Informe Final**

**Galileo - MQTT**

**Proyecto N° 16**

**Integrantes**

- Monti Agustín Pablo – 123/9

## 1- Propuesta del Proyecto

La propuesta del proyecto es poder controlar un motor servo desde una página web. Las placas de desarrollo a utilizar son dos Intel Galileos, una encargada de controlar el motor y la otra de servir la página web que permite al usuario enviar las señales para realizar la actuación, propiamente dicha. La interacción del usuario con dicha web se realiza mediante una PC. Tanto las Galileos como la PC están conectadas a un switch vía Ethernet. El protocolo de comunicación usado entre las placas es MQTT – Mosquitto 3.1.

## 2- Dispositivos

Se dispone de la siguiente lista de materiales aportadas por la catedra en su totalidad:

- 2 Placas Intel Galileo primer generación.
- 4 cables utp
- 1 servo-motor
- 2 memorias microSD, una de 2 Gb marca Sandisk y una de 16 Gb Kingston
- 1 switch Dlink DSS-8
- 3 fuentes de alimentación (2 corresponden a las placas y 1 del switch)

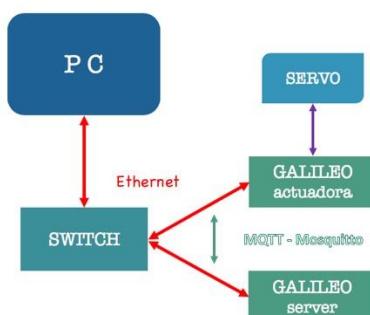
Presupuesto:

- (2 unidades) \$1.600 – [Intel Galileo \(incluyen fuentes\)](#) – 10/10/2018
- (4 unidades) \$26 – [Cable UTP](#) – 10/10/2018
- (1 unidades) \$84 – [Servomotor](#) – 10/10/2018
- (1 unidad) \$150– [microSD 2 Gb](#) – 10/10/2018
- (1 unidad) \$309– [microSD 16 Gb](#) – 10/10/2018
- (1 unidad) \$523 – [router Dlink \(con fuente\)](#) – 10/10/2018

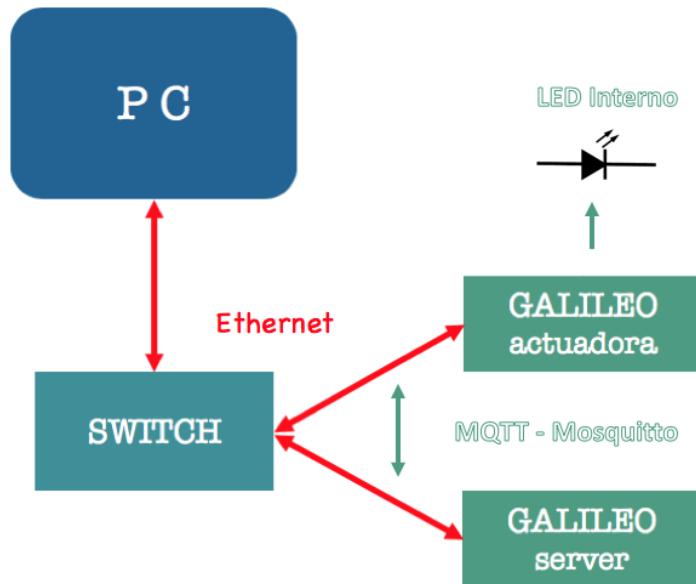
El servo-motor fue utilizado en pruebas independientes, pero no se logró el correcto funcionamiento con el resto del sistema (los errores son de compilación y se detallan al final del informe en la sección Anexo). En lo que respecta al actual informe y la presentación en clase se utiliza el led interno de la Intel Galileo en su lugar.

## 3- Descripción y Documentación General del Proyecto

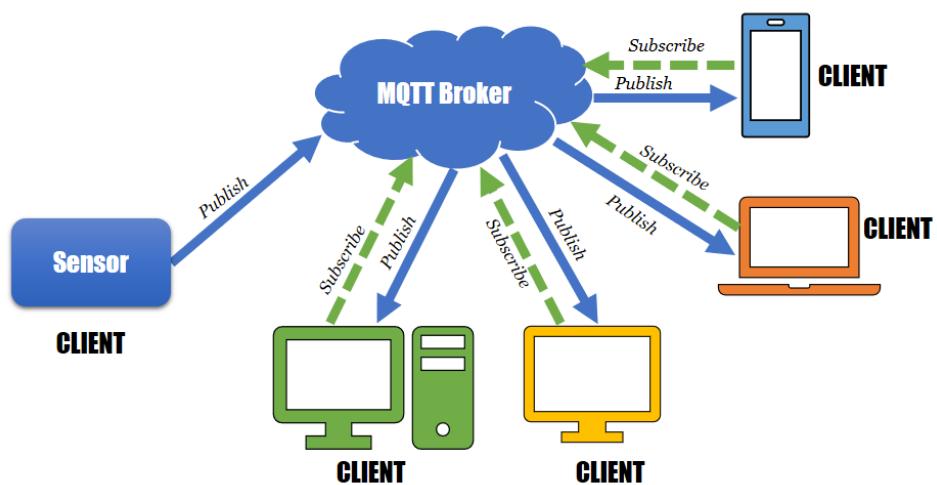
El proyecto completo contaba con los componentes enunciados en el punto anterior conectados de la siguiente manera:



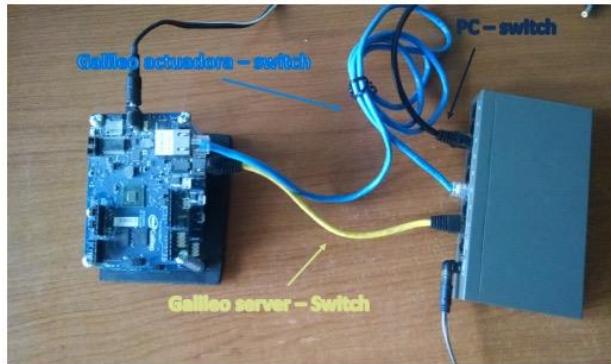
Debido a problemas de implementación finales con el servo-motor (en el anexo se encuentran las pruebas), se acordó con la cátedra utilizar el LED interno de la Galileo actuadora en su lugar, resultando el siguiente esquema general:



De manera general podemos describir los siguientes procesos. La PC sólo sirve para ver la web y permite dos tipos de interacciones básicas: un link para encender el LED y otro para apagarlo. La Galileo Server está constantemente a la espera de una interacción por parte del usuario. De haber detectado un click en algun link, publica el mensaje correspondiente al bróker (ella misma). La otra Galileo, la actuadora, recibe dicho mensaje por estar suscrita al mismo tópico y actúa en consecuencia encendiendo o apagando su LED interno. La PC se comunica por HTTP a la Galileo Server mientras que entre las Galileos la comunicación es usando el protocolo MQTT – Mosquitto 3.1. Las características principales son su pequeño footprint, su uso acotado de ancho de banda y su bajo consumo. Por estas razones es tan usado hoy dia en IoT.



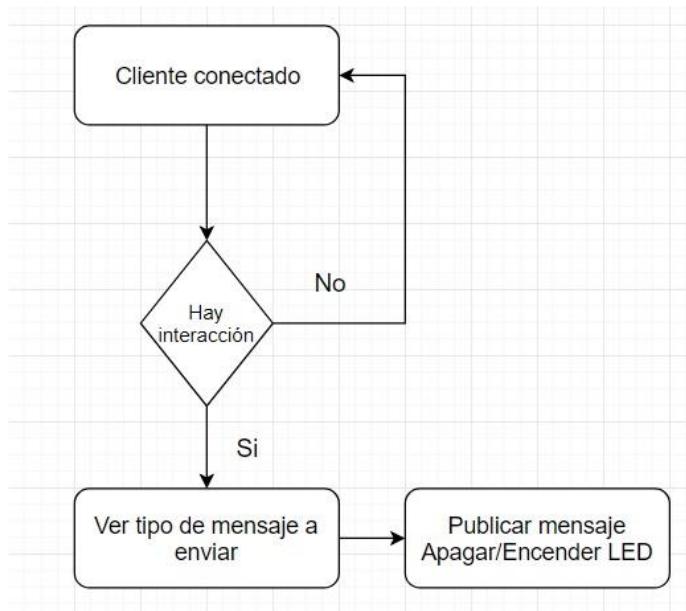
En la siguiente imagen visualizamos la interconexión física entre las placas, el switch y la PC:



La Galileo de la parte de superior es la actuadora y la inferior el server. La conexión con la PC mediante cable utp y usando Ethernet como se aprecia en la foto. Ante una interacción del usuario en la pagina web, la Galileo server toma el parámetro y publica el mensaje (ella misma es el broker de MQTT Mosquitto) y la Galileo actuadora lo recibe (mediante el switch, usando el protocolo MQTT Mosquitto). En función de que mensaje recibió enciende o apaga el led. El proceso general se repite con cada nueva interacción.

### Sección de Galileo Server

La placa server ejecuta un sketch hecho en mediante el IDE de Arduino. Luego de la inicialización de la misma podemos descomponer el siguiente diagrama de flujo:



El código que implementa el diagrama anterior es el siguiente:

```
void loop() {
    // listen for incoming clients
    EthernetClient client = server.available();
    if (client)
    {
        // make a String to hold incoming data from the client
        String currentLine = "";

        Serial.println("Nuevo Cliente");

        String selection = "No se ha seleccionado ninguna opcion aun";
        while (client.connected())
        {
            if (client.available())
            {
                char c = client.read();
                Serial.write(c);

                // Some checking/work is needed at the end of line
                if (c == '\n')
                {
                    // if the current line is blank, you got two newline characters in a
row.
                    // that's the end of the client HTTP request, so send a response:
                    if (currentLine.length() == 0)
                    {
                        sendPagetoClient(client, selection);
                        break;
                    }
                    currentLine = "";
                }
                else if (c != '\r') {
                    // if you got anything else but a carriage return character,
                    currentLine += c;    // add it to the end of the currentLine
                }
            }

            // Check to see the client request:
            if (currentLine.endsWith("GET /LED_on"))
            {
                selection = "Sel1";
                // envio orden de encender el Led
                system("mosquitto_pub --host 169.254.4.103 --topic led --message
\"LED_ON\"");
            }
        }
    }
}
```

```

        delay(2000);

    }

    if (currentLine.endsWith("GET /LED_off")) {
        selection = "Sel2";
        // envio orden de apagar el LED
        system("mosquitto_pub --host 169.254.4.103 --topic led --message
\"LED_OFF\"");
        delay(2000);
    }
}

// give the web browser time to receive the data
delay(1);
// close the connection:
client.stop();
Serial.println("Cliente Desconectado");
}

else {
    Serial.println("cliente no esta listo");
}
}

/****************************************/
void sendPagetoClient(EthernetClient client, String selected)
{
    client.println("HTTP/1.1 200 OK");
    client.println("Content-type:text/html");
    client.println();

    // the content of the HTTP response follows the header:
    client.print("Para encender el Led <a href=\"/LED_on\"> haga click
aqui</a><br>");
    client.print("Para apagarlo <a href=\"/LED_off\"> haga click</a><br>");

    if (selected == "LED_on") {
        client.print("Se ha encendido el LED ");
    }
    if (selected == "LED_off") {
        client.print("Se ha apagado el LED ");
    }
    // The HTTP response ends with another blank line:
    client.println();
}

```

Todos los retrasos que se hacen de 2 segundos son para evitar errores aleatorios propios de la placa. Del código anterior vale aclarar que la totalidad sirven para que la Galileo Server interactúe con la PC verificando que intención tiene el usuario salvo las líneas que

publican los mensajes. Explícitamente las líneas que hacen dicha publicación son las siguientes:

```
// envio orden de encender el Led
    system("mosquitto_pub --host 169.254.4.103 --topic led --message
\"LED_ON\"");
    delay(2000);
```

y

```
// envio orden de apagar el LED
    system("mosquitto_pub --host 169.254.4.103 --topic led --message
\"LED_OFF\"");
    delay(2000);
```

En la Intel Galileo se puede ejecutar comandos de bash de la siguiente forma:

```
system("comando_a_ejecutar");
```

El comando mosquitto\_pub permite publicar el mensaje. Los parámetros que usamos es el host (bróker de la red, que es la misma Galileo Server en nuestro caso), el tópico del mensaje (podrían ser varios, como por ejemplo manejar varios leds) y finalmente el mensaje a transmitir.

El restante código que permite la interacción con el usuario hace lo siguiente:

-Levanta el server:

```
// listen for incoming clients
EthernetClient client = server.available();
```

-Verifica que el cliente esté conectado. De resultar positivo, reconstruye el string y verifica si seleccionó la primera opción o la segunda:

```
while (client.connected())
{
    if (client.available())
    {
        char c = client.read();
        Serial.write(c);

        // Some checking/work is needed at the end of line
        if (c == '\n')
        {
            // if the current line is blank, you got two newline characters in a
row.
            // that's the end of the client HTTP request, so send a response:
            if (currentLine.length() == 0)
            {
                sendPageToClient(client, selection);
            }
        }
    }
}
```

```

        break;
    }
    currentLine = "";
}
else if (c != '\r') {
    // if you got anything else but a carriage return character,
    currentLine += c;    // add it to the end of the currentLine
}
// Check to see the client request:
if (currentLine.endsWith("GET /LED_on")) {
    selection = "Sel1";
    // envio orden de encender el Led
    system("mosquitto_pub --host 169.254.4.103 --topic led --message
\"LED_ON\"");
    delay(2000);

}
if (currentLine.endsWith("GET /LED_off")) {
    selection = "Sel2";
    // envio orden de apagar el LED
    system("mosquitto_pub --host 169.254.4.103 --topic led --message
\"LED_OFF\"");
    delay(2000);
}
}

```

La función que le envía el contenido al cliente para que pueda verlo en la web es la siguiente:

```

void sendPageToClient(EthernetClient client, String selected)
{
    client.println("HTTP/1.1 200 OK");
    client.println("Content-type:text/html");
    client.println();

    // the content of the HTTP response follows the header:
    client.print("Para encender el Led <a href=\"/LED_on\"> haga click
aqui</a><br>");
    client.print("Para apagarlo <a href=\"/LED_off\"> haga click</a><br>");

    if (selected == "LED_on") {
        client.print("Se ha encendido el LED ");
    }
    if (selected == "LED_off") {
        client.print("Se ha apagado el LED ");
    }
    // The HTTP response ends with another blank line:
    client.println();
}

```

Para poder inicializar la placa y posteriormente hacer lo que se describió, primero debe hacer los siguientes pasos:

- Inicializamos la interfaz Ethernet
- Inicializamos el demonio mosquitto que entre otras tareas abre el puerto 1883 y queda a la escucha de posibles conexiones
- Levantamos el server en el puerto 80

El código que permite lo anterior es:

```
#include <Ethernet.h>

EthernetServer server(80);

void setup() {
    // detengo el server si estaba corriendo
    system("killall lighttpd");
    delay(3000);

    system("telnetd -l /bin/sh");
    delay(2000);
    // bajo la interfaz enp0s20f6
    system("ip link set enp0s20f6 down");
    delay(2000);
    // la renombro a eth0 por convencion
    system("ip link set enp0s20f6 name eth0");
    delay(2000);
    //levanto la interfaz
    system("ip link set eth0 up");
    delay(2000);

    // Le asigno la ip dentro de la subred que se encuentra el resto del sistema
    //PC-Galileos
    system("ip addr add 169.254.4.103/8 dev eth0");
    delay(2000);
    // me posiciono en el directorio raiz
    system("cd /");
    // inicializamos el demonio mosquitto
    system("mosquitto -d");
    delay(2000);

    // start the server
    server.begin();
    delay(2000);
}
```

El código completo que corre la Galileo Server es el siguiente:

```
#include <Ethernet.h>
```

```

EthernetServer server(80);

void setup() {
    // detengo el server si estaba corriendo
    system("killall lighttpd");
    delay(3000);

    system("telnetd -l /bin/sh");
    delay(2000);
    // bajo la interfaz enp0s20f6
    system("ip link set enp0s20f6 down");
    delay(2000);
    // la renombro a eth0 por convencion
    system("ip link set enp0s20f6 name eth0");
    delay(2000);
    //levanto la interfaz
    system("ip link set eth0 up");
    delay(2000);

    // Le asigno la ip dentro de la subred que se encuentra el resto del sistema
    //PC-Galileos
    system("ip addr add 169.254.4.103/8 dev eth0");
    delay(2000);
    // me posiciono en el directorio raiz
    system("cd /");
    // inicializamos el demonio mosquitto
    system("mosquitto -d");
    delay(2000);

    // start the server
    server.begin();
    delay(2000);
}

void loop() {
    // listen for incoming clients
    EthernetClient client = server.available();
    if (client)
    {
        // make a String to hold incoming data from the client
        String currentLine = "";

        Serial.println("Nuevo Cliente");

        String selection = "No se ha seleccionado ninguna opcion aun";
        while (client.connected())
        {
            if (client.available())
            {

```

```

char c = client.read();
Serial.write(c);

// Some checking/work is needed at the end of line
if (c == '\n')
{
    // if the current line is blank, you got two newline characters in a
row.
    // that's the end of the client HTTP request, so send a response:
    if (currentLine.length() == 0)
    {
        sendPagetoClient(client, selection);
        break;
    }
    currentLine = "";
}
else if (c != '\r') {
    // if you got anything else but a carriage return character,
    currentLine += c;    // add it to the end of the currentLine
}
// Check to see the client request:
if (currentLine.endsWith("GET /LED_on")) {
    selection = "Sel1";
    // envio orden de encender el Led
    system("mosquitto_pub --host 169.254.4.103 --topic led --message
\"LED_ON\"");
    delay(2000);

}
if (currentLine.endsWith("GET /LED_off")) {
    selection = "Sel2";
    // envio orden de apagar el LED
    system("mosquitto_pub --host 169.254.4.103 --topic led --message
\"LED_OFF\"");
    delay(2000);
}
// give the web browser time to receive the data
delay(1);
// close the connection:
client.stop();
Serial.println("Cliente Desconectado");
}
else {
    Serial.println("cliente no esta listo");
}
}

```

```

/*****************/
void sendPageToClient(EthernetClient client, String selected)
{
    client.println("HTTP/1.1 200 OK");
    client.println("Content-type:text/html");
    client.println();

    // the content of the HTTP response follows the header:
    client.print("Para encender el Led <a href=\"/LED_on\"> haga click  
aqui</a><br>");
    client.print("Para apagarlo <a href=\"/LED_off\"> haga click</a><br>");

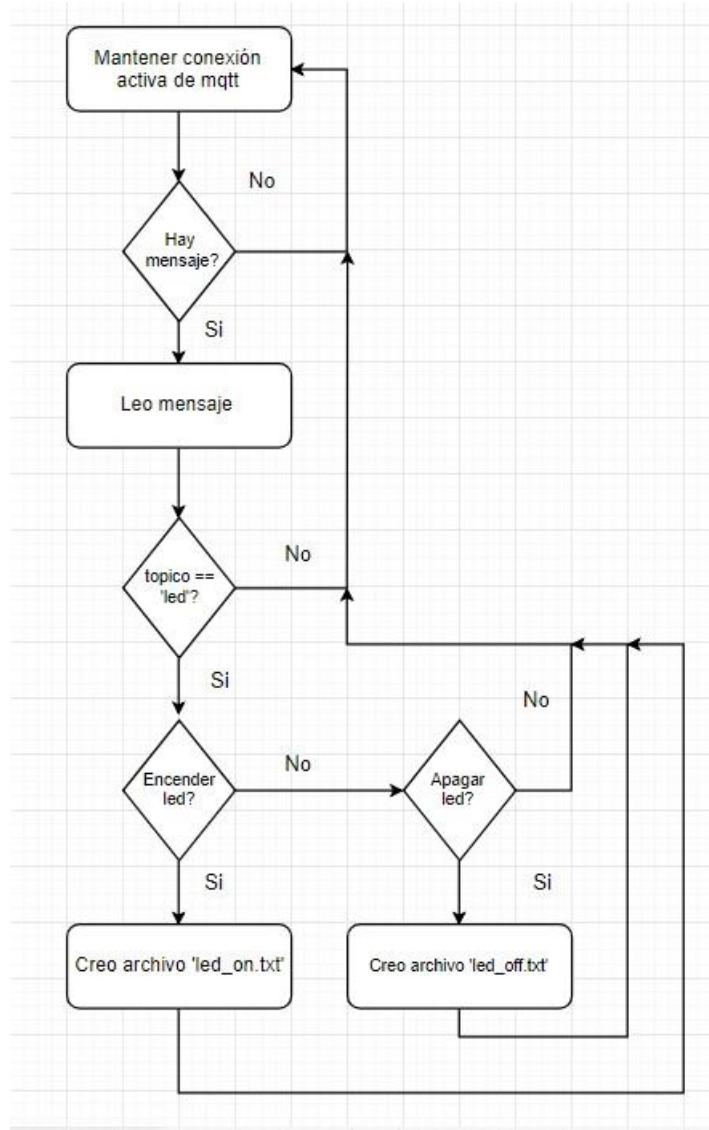
    if (selected == "LED_on") {
        client.print("Se ha encendido el LED ");
    }
    if (selected == "LED_off") {
        client.print("Se ha apagado el LED ");
    }
    // The HTTP response ends with another blank line:
    client.println();
}

```

## Sección de Galileo Actuadora

Pasando a la placa actuadora podemos describir dos partes funcionalmente separadas. Una encargada de recibir el mensaje y verificar la acción a realizar. La segunda de actuar en consecuencia. Por problemas a la hora del uso de las librerías que se consiguen para el IDE de Arduino (librería [PubSubClient](#)) se optó por que la recepción de mensajes sea en Python usando [Paho-mqtt](#) (las pruebas del mal funcionamiento de PubSubClient en la Intel Galileo se encuentran en el Anexo) mientras que la interacción con el LED (o servo originalmente) es en C mediante el IDE de Arduino. Para que el código en C se “entere” que hubo una comunicación (recibida en python) y qué tipo de comunicación fue, se hace uso de archivos. Cuando se detecta un nuevo mensaje, se crea un nuevo archivo con un nombre clave (led\_on.txt y led\_off.txt, según corresponda) desde Python. Desde el sketch de Arduino se está consultando constantemente por la existencia de alguno de los dos archivos. De existir alguno, se actúa encendiéndo o apagando el led y borrando el archivo para no interferir en futuros mensajes. El ciclo se repite constantemente.

El siguiente diagrama de flujo ayuda a comprender el trabajo de la placa actuadora a la hora de recibir el mensaje (script en Python):



La implementación del flujo anterior es la siguiente:

```

#!/usr/bin/python
import paho.mqtt.client as mqtt

# The callback for when the client receives a CONNACK response from the
server.
def on_connect(client, userdata, flags, rc):
    client.subscribe("led")
    # Subscribing in on_connect() means that if we lose the connection and
    # reconnect then subscriptions will be renewed.

# The callback for when a PUBLISH message is received from the server.
def on_message(client, userdata, msg):
    mensaje_recibido = str(msg.payload)
    topico_recibido = str(msg.topic)
    if topico_recibido == 'led':
        
```

```

if mensaje_recibido == 'LED_ON':
    f = open('led_on.txt', 'w')
    # Creo un archivo llamado led_on.txt
    # y desde arduino enciendo dicho led
    f.write('led_on')
    f.close()
if mensaje_recibido == 'LED_OFF':
    f = open('led_off.txt', 'w')
    # Creo un archivo llamado led_off.txt
    # y desde arduino apago dicho led
    f.write('led_off')
    f.close()

mensaje_recibido= ''
topico_recibido = ''
client = mqtt.Client()
client.on_connect = on_connect
client.on_message = on_message

client.connect("169.254.4.103", 1883, 60)

# Blocking call that processes network traffic, dispatches callbacks and
# handles reconnecting.
# Other loop*() functions are available that give a threaded interface and a
# manual interface.
client.loop_forever()

```

La explicación del script es la siguiente:

Se importa la clase necesaria para inicializar un objeto cliente dentro de la librería mqtt de paho:

```
import paho.mqtt.client as mqtt
```

Luego se crea la conexión a la ip del bróker (galileo server, usando el puerto destinado para ello y explícitamente asignando un *keep alive* de 60):

```
client.connect("169.254.4.103", 1883, 60)
```

Al realizar la conexión satisfactoriamente se ejecuta esta función que se suscribe al tópico led y ante una desconexión y posterior reconexión renueva la suscripción:

```

# The callback for when the client receives a CONNACK response from the
server.
def on_connect(client, userdata, flags, rc):
    client.subscribe("led")
    # Subscribing in on_connect() means that if we lose the connection and
    # reconnect then subscriptions will be renewed.

```

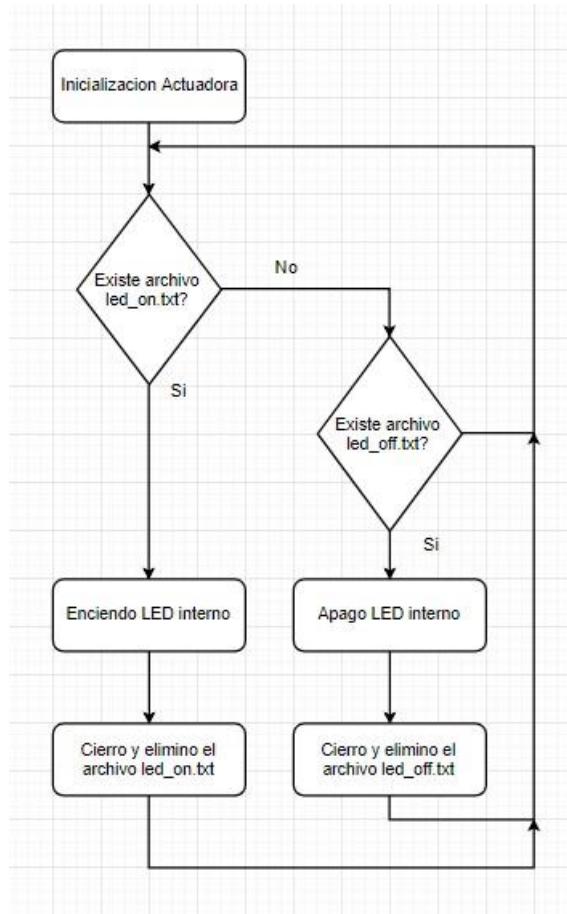
Si se recibe un mensaje consultamos si es el tópico buscado (led) y en el caso de serlo verificamos si se quiere encender el led o apagarlo. Se crea el archivo correspondiente según sea el caso:

```
# The callback for when a PUBLISH message is received from the server.
def on_message(client, userdata, msg):
    mensaje_recibido = str(msg.payload)
    topico_recibido = str(msg.topic)
    if topico_recibido == 'led':
        if mensaje_recibido == 'LED_ON':
            f = open('led_on.txt', 'w')
            # Creo un archivo llamado led_on.txt
            # y desde arduino enciendo dicho led
            f.write('led_on')
            f.close()
        if mensaje_recibido == 'LED_OFF':
            f = open('led_off.txt', 'w')
            # Creo un archivo llamado led_off.txt
            # y desde arduino apago dicho led
            f.write('led_off')
            f.close()
```

Continuamente se itera con una función que se encarga entre otras cosas, de mantener la conexión viva, que si se desconecta lo reconecte, etc:

```
# Blocking call that processes network traffic, dispatches callbacks and
# handles reconnecting.
# Other loop*() functions are available that give a threaded interface and a
# manual interface.
client.loop_forever()
```

La otra funcionalidad de la Galileo Actuadora se puede visualizar en el siguiente diagrama de flujo:



No se detalla en el diagrama la inicialización de la Galileo ya que es sencillo y esta explicado en los comentarios propios y explicación del código.

El código del sketch de Arduino que implementa dicho diagrama es:

```

void setup() {
  // bajo la interfaz enp0s20f6
  system("ip link set enp0s20f6 down");
  // la renombro a eth0 por convencion
  system("ip link set enp0s20f6 name eth0");
  delay(2000);
  //levanto la interfaz
  system("ip link set eth0 up");
  delay(2000);

  //Le asigno la ip dentro de la subred que se encuentra el resto del sistema
  //PC-Galileos
  system("ip addr add 169.254.4.105/8 dev eth0");
  delay(2000);
  // pin 13 como salida -> LED interno
  pinMode(13, OUTPUT);
  delay(2000);
  // Apago LED interno por si estaba encendido
  
```

```

digitalWrite(13, LOW);
delay(2000);
// inicializamos el demonio mosquitto
system("mosquitto -d");
delay(2000);
//ejecutamos el script de python que se comunica mediante mqtt mosquitto
system("cd /");
system("chmod a+x cliente_suscriptor.py");
delay(2000);
system("./cliente_suscriptor.py");
delay(2000);
}
void loop() {
FILE *f_led_on;
FILE *f_led_off;
do {
    delay(1000);
    // intenta abrir led_on.txt
    f_led_on = fopen("led_on.txt", "r");
    delay(1000);
    // intenta abrir led_off.txt
    f_led_off = fopen("led_off.txt", "r");
} while (f_led_on == NULL && f_led_off == NULL);
// verifico cual archivo abrio exitosamente
if (f_led_on) {
    //se enciende el led
    digitalWrite(13, HIGH);
    fclose(f_led_on);
    system("cd /");
    system("rm led_on.txt");
}
if (f_led_off) {
    //se apaga el led
    digitalWrite(13, LOW);
    fclose(f_led_off);
    system("rm led_off.txt");
}
}

```

Analizando por partes vemos que al principio se inicializa la Galileo al igual que se hizo con la placa Server asignándole lógicamente una ip distinta pero dentro de la misma subred. En este caso se asegura que el led esté apagado al iniciar el sistema y se dispara el script de Python ya descripto.

Dentro del bucle infinito se hace lo siguiente:

Verificamos si la existencia de algun archivo 'led\_off.txt' o 'led\_on.txt':

```
FILE *f_led_on;
```

```

FILE *f_led_off;
do {
    delay(1000);
    // intenta abrir led_on.txt
    f_led_on = fopen("led_on.txt", "r");
    delay(1000);
    // intenta abrir led_off.txt
    f_led_off = fopen("led_off.txt", "r");
} while (f_led_on == NULL && f_led_off == NULL);

```

De ser exitosa alguna de las lecturas, verificamos si el archivo abierto es led\_on.txt o led\_off.txt y según sea el caso se enciende o apaga el led, se cierra y se elimina el archivo para evitar posibles futuras fallas en lecturas:

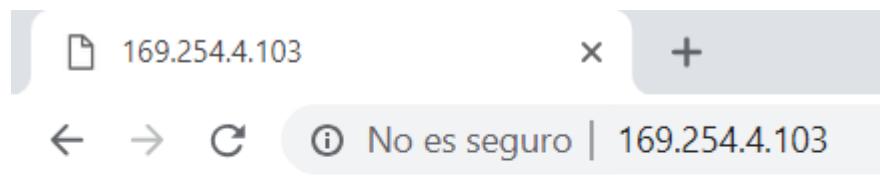
```

// verifico cual archivo abrio exitosamente
if (f_led_on) {
    //se enciende el led
    digitalWrite(13, HIGH);
    fclose(f_led_on);
    system("cd /");
    system("rm led_on.txt");
}
if (f_led_off) {
    //se apaga el led
    digitalWrite(13, LOW);
    fclose(f_led_off);
    system("rm led_off.txt");
}

```

## Sección PC - página web

La página web presenta sólo dos links, uno para encender el led y otro para apagarlo:

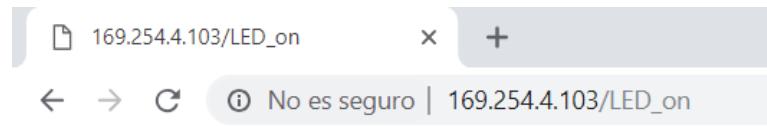


Para encender el Led [haga click aqui](#)

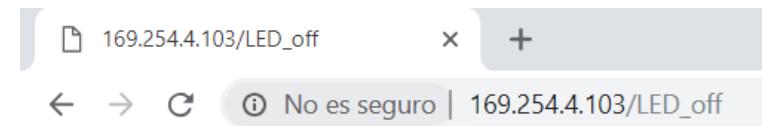
Para apagarlo [haga click](#)

Cada uno de ellos refresca la página mostrando el endpoint: ip\_asignado + '/LED\_on' si se clickeó encender (en nuestro caso [http://169.254.4.103/LED\\_on](http://169.254.4.103/LED_on)) y ip\_asignado + '/LED\_off' si fue apagar(en nuestro caso [http://169.254.4.103/LED\\_off](http://169.254.4.103/LED_off)).

Las vistas con las rutas anteriores las podemos ver en las siguientes imágenes:



Para encender el Led [haga click aqui](#)  
Para apagarlo [haga click](#)



Para encender el Led [haga click aqui](#)  
Para apagarlo [haga click](#)

Aquí dejamos el enlace a la [bitácora](#) utilizada.

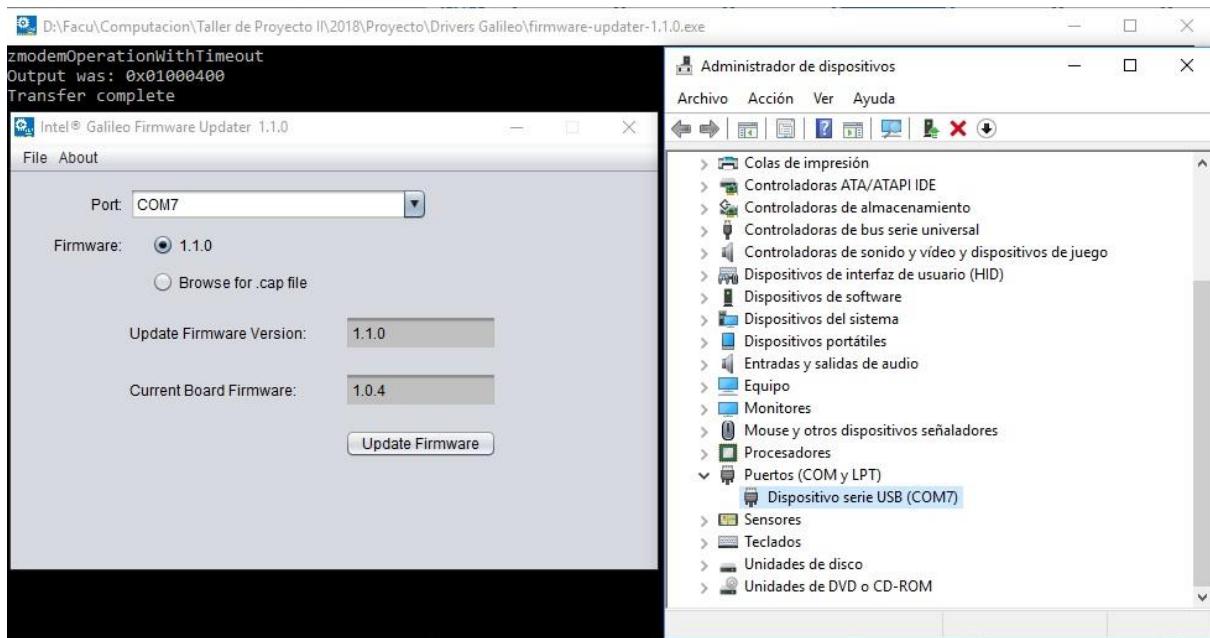
## 4- Guía de Instalación: Proyecto y Ambiente de Desarrollo

### Sección desarrollo

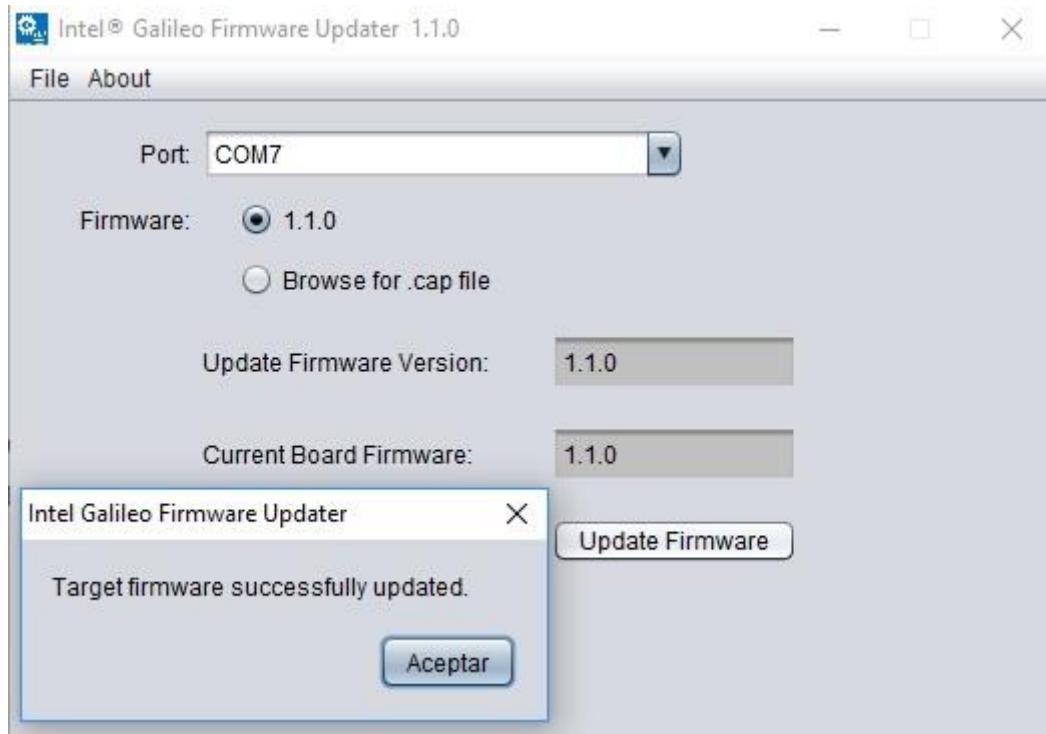
#### Actualizar drivers de la Galileo:

Para actualizar los Drivers de la Galileo los descargamos [aquí](#).

En nuestro caso usamos los que son para Windows 10. Conectamos mediante el puerto USB cliente la placa a la PC sin tener la memoria microSD colocada. Abrimos el programa y vemos en la siguiente imagen que los drivers de nuestra Galileo están desactualizados. Los actualizamos haciendo click en update firmware:



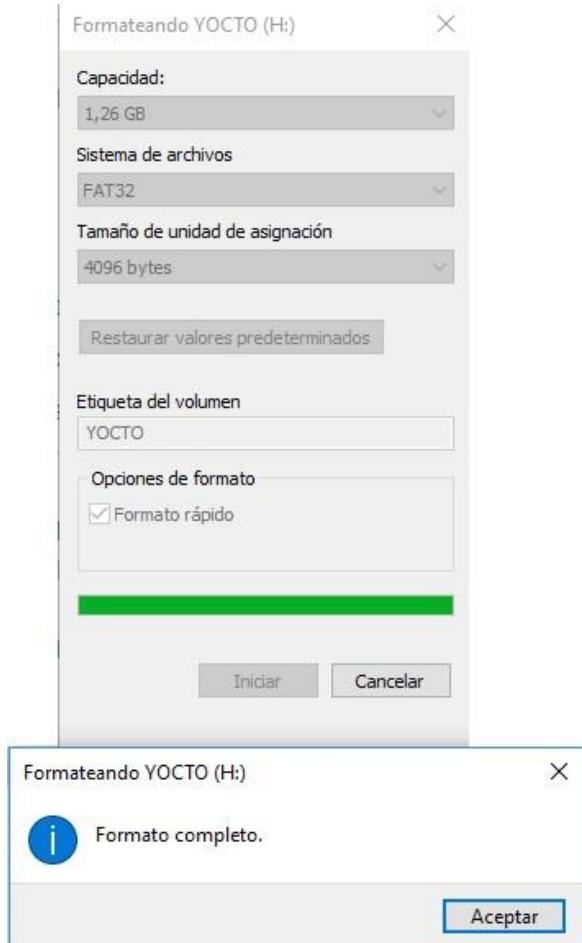
Aceptamos los siguientes modals que nos consultan si la placa está conectada a una fuente de tensión externa y si estamos seguros de subir de versión. Esperando 5 minutos aproximadamente si finaliza exitosamente, podremos ver el siguiente mensaje:



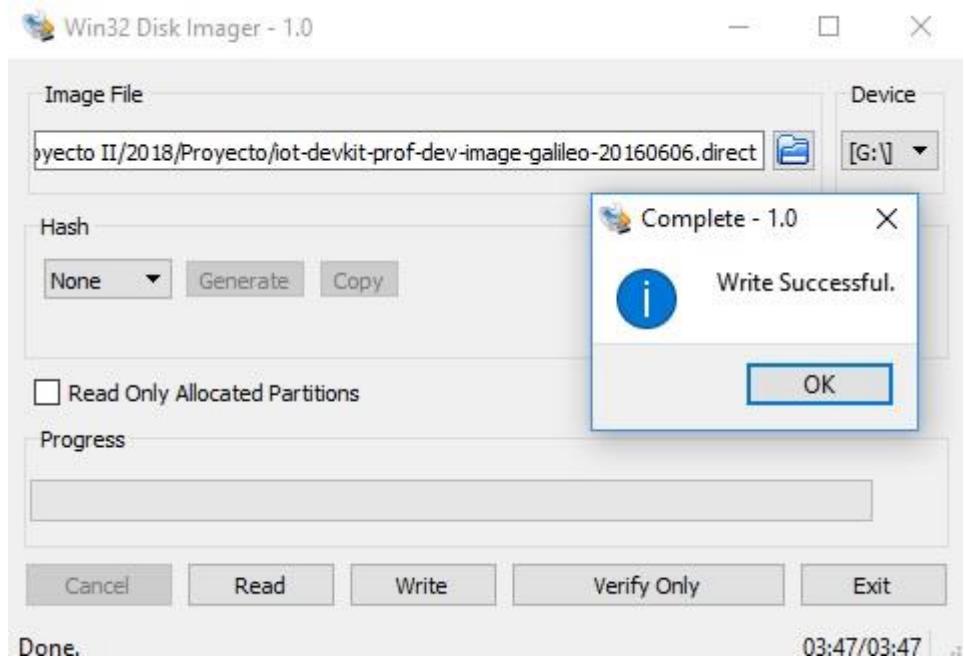
## Instalación de linux en la Galileo:

Descargamos la imagen de Linux que usaremos con la memoria microSD [aquí](#). Al 30/09/2018 el archivo se llama “Galileo\_Poky\_SW\_image\_20160606” y se encuentra en formato zip. Para grabar la imagen linux Yocto en la SD desde el explorador de archivos de

windows con la tarjeta colocada hacemos click derecho sobre la unidad y clickeamos formatear. Tener en cuenta que el formato por default que esta seleccionado suele ser FAT (predeterminado) que no sirve para nuestro caso ya que el File System es FAT16. Por lo tanto, seleccionamos la opción FAT32 y le asignamos un nombre para darle formato como se ve a continuación:



Luego usamos “Win32DiskImager” para copiar la imagen a la SD (abrir con permisos de administrador o puede fallar al intentar escribir). Elegimos el archivo con extensión “direct” que obtenemos al descomprimir el zip descargado y si finalizó exitosamente deberíamos ver el siguiente resultado:

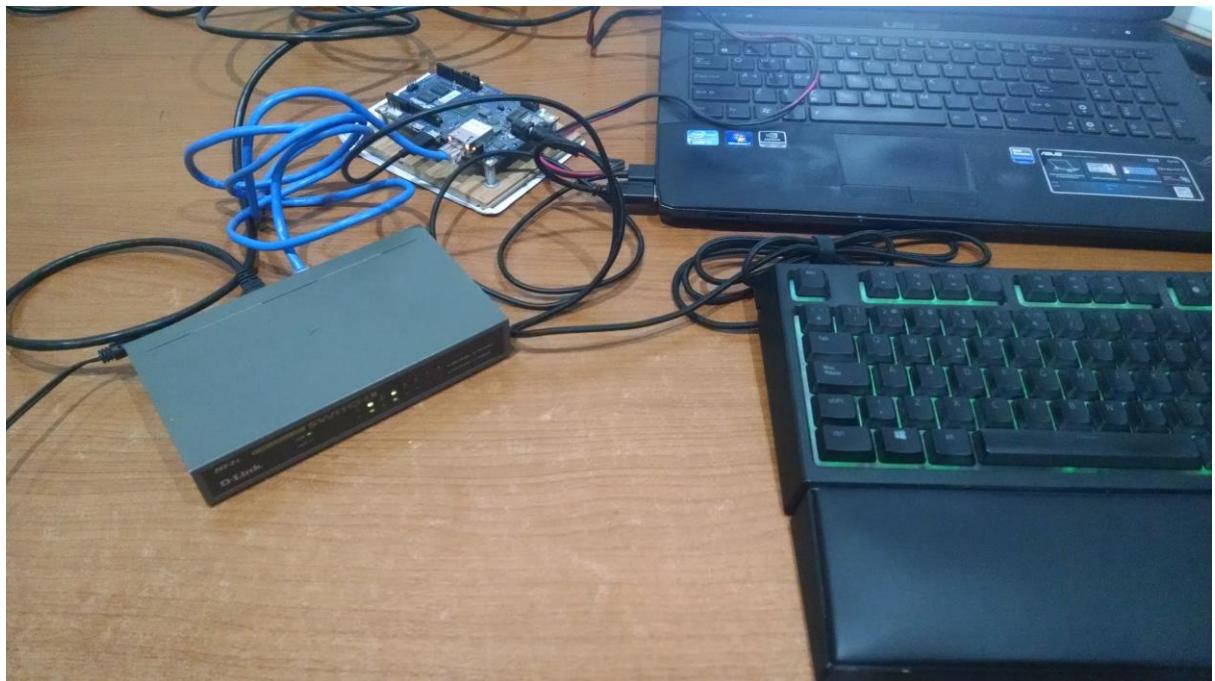


## Instalación del IDE Arduino para Galileo:

Descargamos el IDE para el SO que usemos [aquí](#) (Windows en nuestro caso). Lo abrimos y obtenemos las librerías que necesitamos para la Galileo buscando Intel desde Herramientas -> Gestionar Librerías.

## Inicializando Galileo para conectarse via ethernet con la PC:

Conectamos la PC y la Galileo a un switch de la siguiente manera:



Usando de cabecera [ésta guia](#) de comandos básicos de linux definimos la interfaz, ip, mascara y Gateway. Usando la consola powershell desde Windows averiguamos que ip tiene nuestra PC de manera de asignarle una ip dentro del dominio a la placa Galileo:

```

prueba_telnet Arduino 1.8.7
Archivo Editar Programa Herramientas Ayuda
prueba_telnet$ 

void setup() {
    // put your setup code here, to run once:
    system("telnetd -l /bin/sh"); //Start the telnet server on Galileo
    // configuro eth0 de manera estatica y le asigno ip, submascara y gateway
    system("auto eth0");
    system("ifconfig eth0 inet static");
    system("address 169.254.7.224");
    system("netmask 255.255.0.0");
    system("gateway 169.254.0.1");
    // la activo y muestro las interfaces activas asi como las rutas que conoce la Galileo
    system("ip link set eth0 up");
    system("ip addr show > /dev/ttyS0");
    system("ip route show > /dev/ttyS0");
}

void loop() {
    // put your main code here, to run repeatedly:
}

```

Luego abrimos Putty y nos conectamos mediante SSH usando la ip que le asignamos en el sketch anteponiendo el nombre de usuario con el que iniciaremos session (root@169.254.7.224):

PuTTY Configuration

Session

Host Name (or IP address): root  
Port: 22

Serial

De resultar exitosa la conexión veremos lo siguiente:

```

l: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue
    01:48:11.431 ->      link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    01:48:11.431 ->      inet 127.0.0.1/8 brd 127.0.0.1 scope host lo
    01:48:11.431 ->          inet6 ::1/128 brd :: scope host lo
    01:48:11.431 ->              valid_lft forever preferred_lft forever
    01:48:11.431 -> 2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast qlen 1000
    01:48:11.431 ->      link/ether 98:4f:ee:01:0e:05 brd ff:ff:ff:ff:ff:ff
    01:48:11.431 ->      inet 169.254.7.224/16 brd 169.254.255.255 scope link eth0
    01:48:11.431 ->          inet6 fe80::9a4f:eff:fe01:ee5/64 scope link
    01:48:11.431 ->              valid_lft forever preferred_lft forever
    default dev eth0
    01:48:11.464 -> 169.254.0.0/16 dev eth0  src 169.254.7.224

```

169.254.7.224 - PuTTY  
Using username "root".  
root@galileo:~#

Transfer complete

Ahora debemos transferir los archivos que nos permitan utilizar el protocolo MQTT Mosquitto entre nuestras placas. Debido a que en el presente proyecto ninguna de ellas dispone de acceso a internet directamente, optamos por utilizar un programa que nos permita realizar dicha transferencia mediante SSH desde nuestra PC. Para ello descargamos e instalamos [winSCP](#). Habiendo instalado el programa, descargamos el archivo fuente de mosquitto desde su [página](#). A la fecha se encuentra disponible la versión 1.5.3:

Source

- mosquito-1.5.3.tar.gz (319kB) (GPG signature)
- mosquito-1.5.3.tar.gz (via Eclipse)
- GitHub source code repository (github.com)

Older downloads are available at <http://mosquitto.org/files/>

Binary Installation

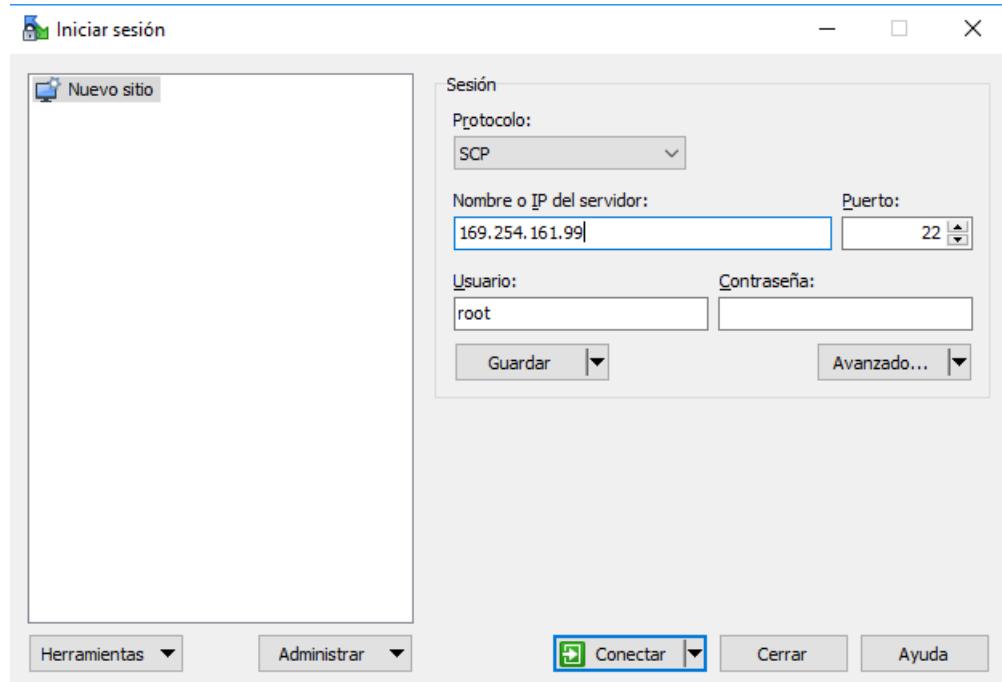
The binary packages listed below are supported by the Mosquitto project. In many cases Mosquitto is also available directly from official Linux/BSD distributions.

Windows

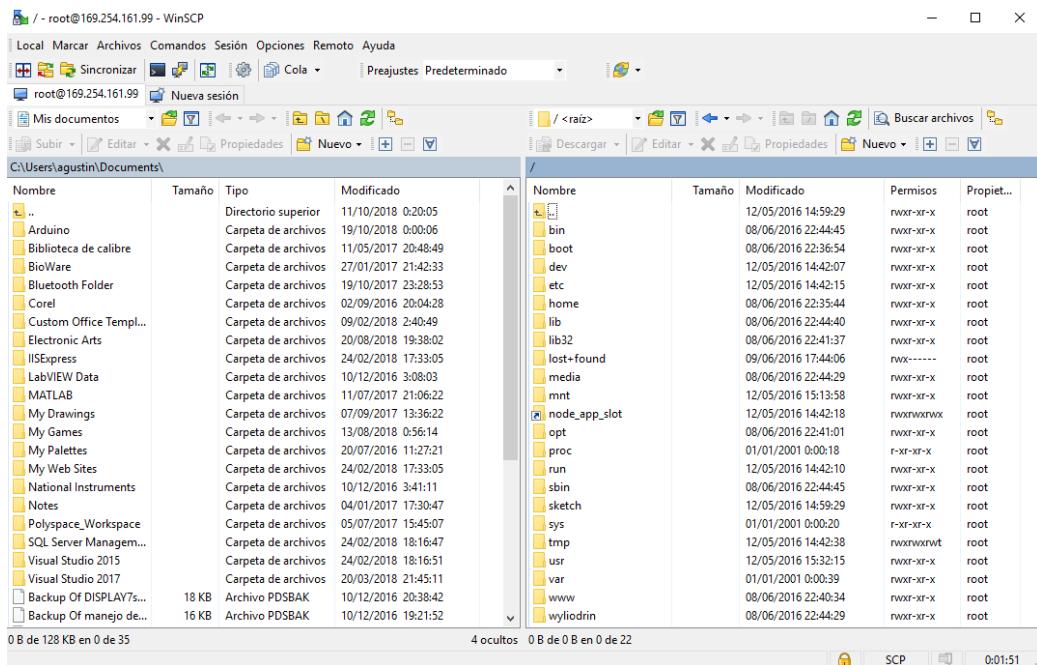
- mosquito-1.5.3-install-windows-x64.exe (~360 kB) (64-bit build, Windows Vista and up, built with Visual Studio Community 2017)
- mosquito-1.5.3-install-windows-x32.exe (~360 kB) (32-bit build, Windows Vista and up, built with Visual Studio Community 2017)

See also `readme-windows.txt` after installing.

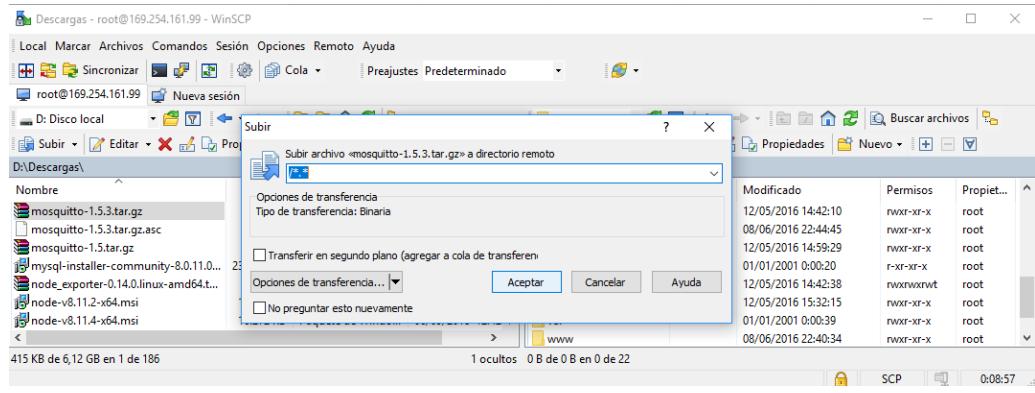
Una vez descargado el archivo con extensión .tar.gz abrimos winSCP y nos conectamos a la ip que le asignamos (169.254.161.99 en nuestro caso):



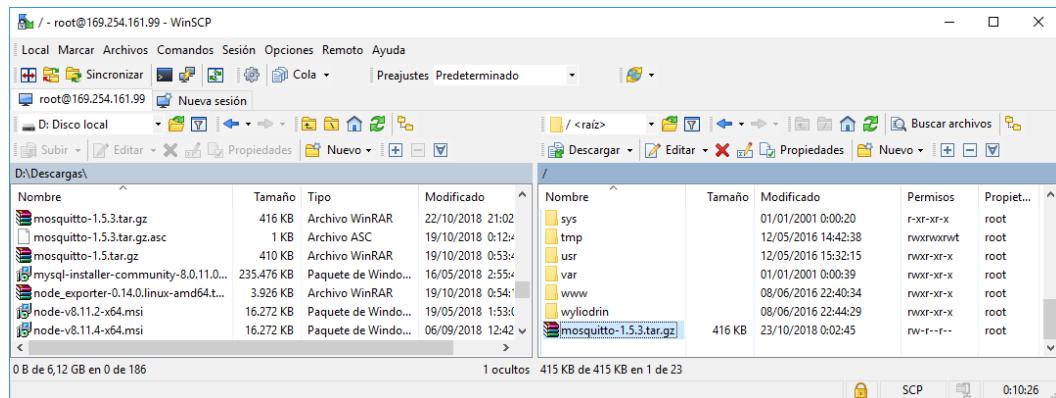
Al conectarnos de manera exitosa vemos la siguiente pantalla:



Haciendo click derecho en el archivo a subir y eligiendo subir vemos el siguiente cuadro de dialogo:



Aceptando vemos como el contenido se encuentra en la carpeta destino de nuestra Galileo:



Este archivo lo necesitamos en ambas Galileos, ya que las dos deben poder ejecutar el demonio mosquitto, poder publicar y subscribir. Por ello hay que repetir lo hecho recién con la otra Galileo (lógicamente con la ip que corresponda en su caso) y también se debe repetir el siguiente paso que hace el make de mosquitto.

Ahora nos conectamos a la placa mediante Putty y nos posicionamos en el directorio raíz:

```
169.254.161.99 - PuTTY
root@galileo:~# cd ..
root@galileo:/home# cd ..
root@galileo:# ls
bin etc lib32 mnt opt sbin tmp www
boot home lost+found mosquitto-1.5.3.tar.gz proc sketch usr wyliodrin
dev lib media node_app_slot run sys var
root@galileo:#
```

Verificamos la fecha que nos muestra el sistema usando el comando date:

```
169.254.161.99 - PuTTY
root@galileo:# date
Thu May 12 14:54:16 UTC 2016
root@galileo:#
```

Actualizamos la fecha del sistema a la correcta (día de la fecha 29/10/2018):

```
root@galileo:/# date -s "29 OCT 2018 22:00:00"
Mon Oct 29 22:00:00 UTC 2018
```

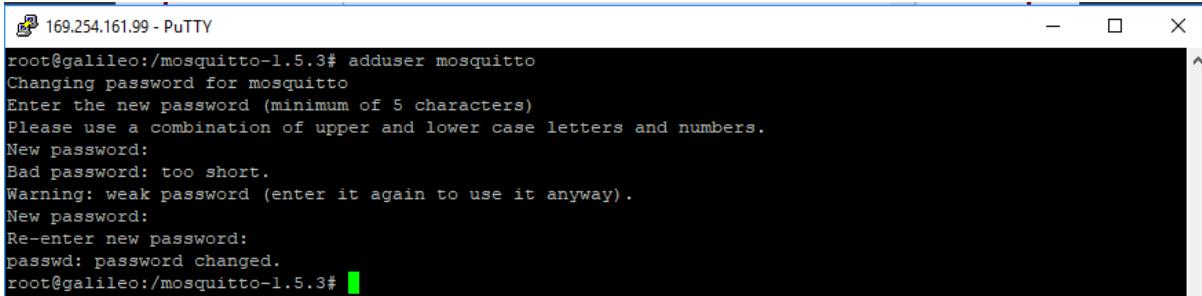
También actualizamos el RTC a la fecha que acabamos de configurar:

```
root@galileo:/# hwclock --systohc
root@galileo:/# hwclock
Mon Oct 29 22:07:31 2018  0.000000 seconds
root@galileo:/#
```

Realizamos la descompresión del archivo, nos posicionamos dentro de la carpeta creada y hacemos “make WITH\_SRV=no” (demora aproximadamente 20 minutos):

```
root@galileo:/mosquitto-1.5.3# make WITH_SRV=no
set -e; for d in lib client src; do make -C ${d}; done
make[1]: Entering directory '/mosquitto-1.5.3/lib'
cc -Wall -ggdb -O2 -I. -I.. -I../lib -fPIC -DWITH_TLS -DWITH_TLS_PSK -DWITH_THREADING -DWITH_SOCKETS -c mosquitto.c -o mosquitto.o
cc -Wall -ggdb -O2 -I. -I.. -I../lib -fPIC -DWITH_TLS -DWITH_TLS_PSK -DWITH_THREADING -DWITH_SOCKETS -c actions.c -o actions.o
cc -Wall -ggdb -O2 -I. -I.. -I../lib -fPIC -DWITH_TLS -DWITH_TLS_PSK -DWITH_THREADING -DWITH_SOCKETS -c callbacks.c -o callbacks.o
cc -Wall -ggdb -O2 -I. -I.. -I../lib -fPIC -DWITH_TLS -DWITH_TLS_PSK -DWITH_THREADING -DWITH_SOCKETS -c connect.c -o connect.o
cc -Wall -ggdb -O2 -I. -I.. -I../lib -fPIC -DWITH_TLS -DWITH_TLS_PSK -DWITH_THREADING -DWITH_SOCKETS -c handle_connack.c -o handle_connack.o
cc -Wall -ggdb -O2 -I. -I.. -I../lib -fPIC -DWITH_TLS -DWITH_TLS_PSK -DWITH_THREADING -DWITH_SOCKETS -c handle_ping.c -o handle_ping.o
cc -Wall -ggdb -O2 -I. -I.. -I../lib -fPIC -DWITH_TLS -DWITH_TLS_PSK -DWITH_THREADING -DWITH_SOCKETS -c handle_pubackcomp.c -o handle_pubackcomp.o
cc -Wall -ggdb -O2 -I. -I.. -I../lib -fPIC -DWITH_TLS -DWITH_TLS_PSK -DWITH_THREADING -DWITH_SOCKETS -c handle_publish.c -o handle_publish.o
cc -Wall -ggdb -O2 -I. -I.. -I../lib -fPIC -DWITH_TLS -DWITH_TLS_PSK -DWITH_THREADING -DWITH_SOCKETS -c handle_pubrec.c -o handle_pubrec.o
cc -Wall -ggdb -O2 -I. -I.. -I../lib -fPIC -DWITH_TLS -DWITH_TLS_PSK -DWITH_THREADING -DWITH_SOCKETS -c handle_pubrel.c -o handle_pubrel.o
cc -Wall -ggdb -O2 -I. -I.. -I../lib -fPIC -DWITH_TLS -DWITH_TLS_PSK -DWITH_THREADING -DWITH_SOCKETS -c handle_suback.c -o handle_suback.o
cc -Wall -ggdb -O2 -I. -I.. -I../lib -fPIC -DWITH_TLS -DWITH_TLS_PSK -DWITH_THREADING -DWITH_SOCKETS -c handle_unsuback.c -o handle_unsuback.o
cc -Wall -ggdb -O2 -I. -I.. -I../lib -fPIC -DWITH_TLS -DWITH_TLS_PSK -DWITH_THREADING -DWITH_SOCKETS -c helpers.c -o helpers.o
cc -Wall -ggdb -O2 -I. -I.. -I../lib -fPIC -DWITH_TLS -DWITH_TLS_PSK -DWITH_THREADING -DWITH_SOCKETS -c logging_mosq.c -o logging_mosq.o
cc -Wall -ggdb -O2 -I. -I.. -I../lib -fPIC -DWITH_TLS -DWITH_TLS_PSK -DWITH_THREADING -DWITH_SOCKETS -c loop.c -o loop.o
cc -Wall -ggdb -O2 -I. -I.. -I../lib -fPIC -DWITH_TLS -DWITH_TLS_PSK -DWITH_THREADING -DWITH_SOCKETS -c memory_mosq.c -o memory_mosq.o
cc -Wall -ggdb -O2 -I. -I.. -I../lib -fPIC -DWITH_TLS -DWITH_TLS_PSK -DWITH_THREADING -DWITH_SOCKETS -c messages_mosq.c -o messages_mosq.o
cc -Wall -ggdb -O2 -I. -I.. -I../lib -fPIC -DWITH_TLS -DWITH_TLS_PSK -DWITH_THREADING -DWITH_SOCKETS -c net_mosq.c -o net_mosq.o
cc -Wall -ggdb -O2 -I. -I.. -I../lib -fPIC -DWITH_TLS -DWITH_TLS_PSK -DWITH_THREADING -DWITH_SOCKETS -c options.c -o options.o
cc -Wall -ggdb -O2 -I. -I.. -I../lib -fPIC -DWITH_TLS -DWITH_TLS_PSK -DWITH_THREADING -DWITH_SOCKETS -c packet_mosq.c -o packet_mosq.o
cc -Wall -ggdb -O2 -I. -I.. -I../lib -fPIC -DWITH_TLS -DWITH_TLS_PSK -DWITH_THREADING -DWITH_SOCKETS -c read_handle.c -o read_handle.o
cc -Wall -ggdb -O2 -I. -I.. -I../lib -fPIC -DWITH_TLS -DWITH_TLS_PSK -DWITH_THREADING -DWITH_SOCKETS -c send_connect.c -o send_connect.o
cc -Wall -ggdb -O2 -I. -I.. -I../lib -fPIC -DWITH_TLS -DWITH_TLS_PSK -DWITH_THREADING -DWITH_SOCKETS -c send_disconnect.c -o send_disconnect.o
cc -Wall -ggdb -O2 -I. -I.. -I../lib -fPIC -DWITH_TLS -DWITH_TLS_PSK -DWITH_THREADING -DWITH_SOCKETS -c send_subscribe.c -o send_subscribe.o
cc -Wall -ggdb -O2 -I. -I.. -I../lib -fPIC -DWITH_TLS -DWITH_TLS_PSK -DWITH_THREADING -DWITH_SOCKETS -c send_unsubscribe.c -o send_unsubscribe.o
```

Usamos el comando adduser mosquitto y ingresamos el password que deseemos (ninguna en nuestro caso, apretamos enter):



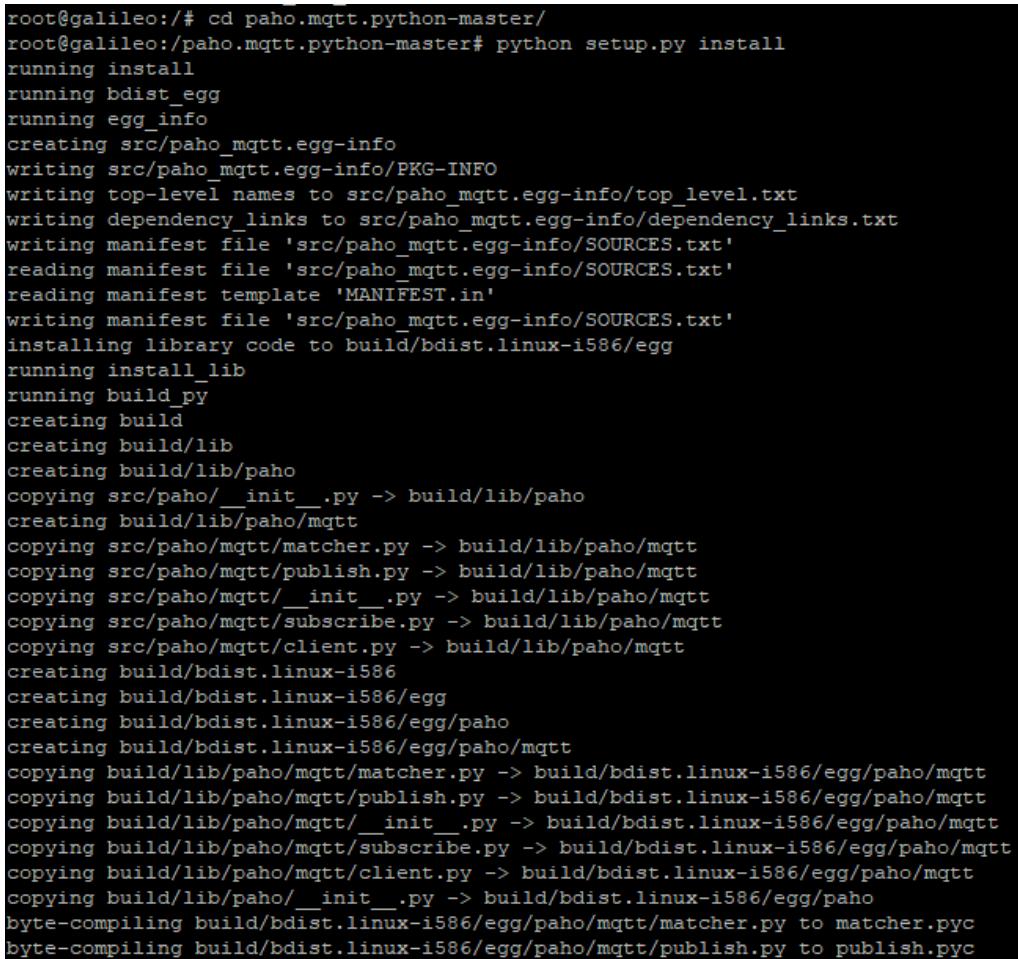
```
169.254.161.99 - PuTTY
root@galileo:/mosquitto-1.5.3# adduser mosquitto
Changing password for mosquitto
Enter the new password (minimum of 5 characters)
Please use a combination of upper and lower case letters and numbers.
New password:
Bad password: too short.
Warning: weak password (enter it again to use it anyway).
New password:
Re-enter new password:
passwd: password changed.
root@galileo:/mosquitto-1.5.3#
```

A continuación, realizamos la instalación en la Galileo actuadora de otra librería llamada Eclipse MQTT Paho que tiene varias implementaciones, optamos por la que usa Python.

Descargamos el repositorio desde Github desde [aquí](#).

Usamos WinSCP para pasar la carpeta (descomprimir el zip antes de pasarlo para facilitar su uso).

Luego de posicionarnos en el directorio raiz, entramos a la carpeta paho.mqtt.python y ejecutamos el comando `python setup.py install`:



```
root@galileo:/# cd paho.mqtt.python-master/
root@galileo:/paho.mqtt.python-master# python setup.py install
running install
running bdist_egg
running egg_info
creating src/paho_mqtt.egg-info
writing src/paho_mqtt.egg-info/PKG-INFO
writing top-level names to src/paho_mqtt.egg-info/top_level.txt
writing dependency_links to src/paho_mqtt.egg-info/dependency_links.txt
writing manifest file 'src/paho_mqtt.egg-info/SOURCES.txt'
reading manifest file 'src/paho_mqtt.egg-info/SOURCES.txt'
reading manifest template 'MANIFEST.in'
writing manifest file 'src/paho_mqtt.egg-info/SOURCES.txt'
installing library code to build/bdist.linux-i586/egg
running install_lib
running build_py
creating build
creating build/lib
creating build/lib/paho
copying src/paho/__init__.py -> build/lib/paho
creating build/lib/paho/mqtt
copying src/paho/mqtt/matcher.py -> build/lib/paho/mqtt
copying src/paho/mqtt/publish.py -> build/lib/paho/mqtt
copying src/paho/mqtt/__init__.py -> build/lib/paho/mqtt
copying src/paho/mqtt/subscribe.py -> build/lib/paho/mqtt
copying src/paho/mqtt/client.py -> build/lib/paho/mqtt
creating build/bdist.linux-i586
creating build/bdist.linux-i586/egg
creating build/bdist.linux-i586/egg/paho
creating build/bdist.linux-i586/egg/paho/mqtt
copying build/lib/paho/mqtt/matcher.py -> build/bdist.linux-i586/egg/paho/mqtt
copying build/lib/paho/mqtt/publish.py -> build/bdist.linux-i586/egg/paho/mqtt
copying build/lib/paho/mqtt/__init__.py -> build/bdist.linux-i586/egg/paho/mqtt
copying build/lib/paho/mqtt/subscribe.py -> build/bdist.linux-i586/egg/paho/mqtt
copying build/lib/paho/mqtt/client.py -> build/bdist.linux-i586/egg/paho/mqtt
copying build/lib/paho/__init__.py -> build/bdist.linux-i586/egg/paho
byte-compiling build/bdist.linux-i586/egg/paho/mqtt/matcher.py to matcher.pyc
byte-compiling build/bdist.linux-i586/egg/paho/mqtt/publish.py to publish.pyc
```

Para inicializar el cliente y abrir el puerto debemos usar la instrucción `mosquitto` con la opción `-d` para forzar que quede corriendo el “demonio” y poder así continuar usando dicha instancia como vemos a continuación:


 192.168.0.12 - PuTTY  

```

root@galileo:/# mosquitto -d
root@galileo:/# netstat -a
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address          Foreign Address        State
tcp      0      0 localhost.localdomain:58888 0.0.0.0:*          LISTEN
tcp      0      0 0.0.0.0:6379            0.0.0.0:*
tcp      0      0 0.0.0.0:http           0.0.0.0:*
tcp      0      0 localhost.localdomain:domain 0.0.0.0:*
tcp      0      0 0.0.0.0:1883            0.0.0.0:*
tcp      0      0 0.0.0.0:1534            0.0.0.0:*
tcp      0      0 localhost:domain        :::*
tcp      0      0 ::ssh                  :::*
tcp      0      0 :::1883                :::*
tcp      0      0 ::ffff:192.168.0.12:ssh ::ffff:192.168.0.8:50102 ESTABLISHED
tcp      0      0 ::ffff:192.168.0.12:ssh ::ffff:192.168.0.8:49990 ESTABLISHED
tcp      0      0 ::ffff:192.168.0.12:ssh ::ffff:192.168.0.8:50112 ESTABLISHED
tcp      0      0 ::ffff:192.168.0.12:ssh ::ffff:192.168.0.8:50101 ESTABLISHED
udp     704      0 localhost.localdomain:56492  localhost.localdomain:domain ESTABLISHED
udp      0      0 0.0.0.0:mdns             0.0.0.0:*
udp      0      0 0.0.0.0:60650            0.0.0.0:*
udp      0      0 galileo.local:55799       dnccache01-horl.fibertel.com.ar:domain ESTABLISHED
udp      0      0 0.0.0.0:1534            0.0.0.0:*
udp      0      0 galileo.local:33798       nrdns07.fibertel.com.ar:domain ESTABLISHED
udp     704      0 localhost.localdomain:49957  localhost.localdomain:domain ESTABLISHED
udp      0      0 0.0.0.0:55590            0.0.0.0:*
udp     704      0 localhost.localdomain:58664  localhost.localdomain:domain ESTABLISHED
udp      0      0 galileo.local:34869       192.168.0.1:domain    ESTABLISHED
udp      0      0 localhost.localdomain:domain 0.0.0.0:*
udp      0      0 0.0.0.0:56701            0.0.0.0:*
udp      0      0 galileo.local:58243       192.168.0.1:domain    ESTABLISHED
udp      0      0 localhost:domain         :::*
udp     704      0 localhost:53832          localhost:domain        ESTABLISHED
udp     704      0 localhost:40533          localhost:domain        ESTABLISHED
udp     704      0 localhost:49007          localhost:domain        ESTABLISHED
udp      0      0 ::ffff:51614             :::*

```

A continuación, debemos crear un nuevo archivo dentro del IDE de Arduino para la Galileo Actuadora. Compilarlo y bajarlo con el usb conectado (tener en cuenta que la dirección ip puede cambiar para cada caso, verificar que subred tiene disponible):

```

void setup() {
  // bajo la interfaz enp0s20f6
  system("ip link set enp0s20f6 down");
  // la renombro a eth0 por convencion
  system("ip link set enp0s20f6 name eth0");
  delay(2000);
  //levanto la interfaz
  system("ip link set eth0 up");
  delay(2000);

  //Le asigno la ip dentro de la subred que se encuentra el resto del sistema
  //PC-Galileos
  system("ip addr add 169.254.4.105/8 dev eth0");
  delay(2000);
  // pin 13 como salida -> LED interno
  pinMode(13, OUTPUT);
  delay(2000);
  // Apago LED interno por si estaba encendido
  digitalWrite(13, LOW);
}

```

```

delay(2000);
// inicializamos el demonio mosquitto
system("mosquitto -d");
delay(2000);
//ejecutamos el script de python que se comunica mediante mqtt mosquitto
system("cd /");
system("chmod a+x cliente_suscriptor.py");
delay(2000);
system("./cliente_suscriptor.py");
delay(2000);
}
void loop() {
FILE *f_led_on;
FILE *f_led_off;
do {
    delay(1000);
    // intenta abrir led_on.txt
    f_led_on = fopen("led_on.txt", "r");
    delay(1000);
    // intenta abrir led_off.txt
    f_led_off = fopen("led_off.txt", "r");
} while (f_led_on == NULL && f_led_off == NULL);
// verifico cual archivo abrio exitosamente
if (f_led_on) {
    //se enciende el led
    digitalWrite(13, HIGH);
    fclose(f_led_on);
    system("cd /");
    system("rm led_on.txt");
}
if (f_led_off) {
    //se apaga el led
    digitalWrite(13, LOW);
    fclose(f_led_off);
    system("rm led_off.txt");
}
}
}

```

Para Galileo Server debemos crear dos archivos, uno para bajar mediante el IDE de Arduino y otro un script de python que pasaremos usando winSCP de la misma manera que lo hicimos para pasar el comprimido con mosquitto.

Crear un nuevo archivo dentro del IDE de Arduino con lo siguiente y bajarlo a la Galileo Server:

```

#include <Ethernet.h>

EthernetServer server(80);

void setup() {

```

```

// detengo el server si estaba corriendo
system("killall lighttpd");
delay(3000);

system("telnetd -l /bin/sh");
delay(2000);
// bajo la interfaz enp0s20f6
system("ip link set enp0s20f6 down");
delay(2000);
// la renombro a eth0 por convencion
system("ip link set enp0s20f6 name eth0");
delay(2000);
//levanto la interfaz
system("ip link set eth0 up");
delay(2000);

// Le asigno la ip dentro de la subred que se encuentra el resto del sistema
//PC-Galileos
system("ip addr add 169.254.4.103/8 dev eth0");
delay(2000);
// me posiciono en el directorio raiz
system("cd /");
// inicializamos el demonio mosquitto
system("mosquitto -d");
delay(2000);

// start the server
server.begin();
delay(2000);
}

void loop() {
// listen for incoming clients
EthernetClient client = server.available();
if (client)
{
// make a String to hold incoming data from the client
String currentLine = "";

Serial.println("Nuevo Cliente");

String selection = "No se ha seleccionado ninguna opcion aun";
while (client.connected())
{
if (client.available())
{
char c = client.read();
Serial.write(c);
}
}
}
}

```

```

// Some checking/work is needed at the end of line
if (c == '\n')
{
    // if the current line is blank, you got two newline characters in a
row.
    // that's the end of the client HTTP request, so send a response:
    if (currentLine.length() == 0)
    {
        sendPagetoClient(client, selection);
        break;
    }
    currentLine = "";
}
else if (c != '\r') {
    // if you got anything else but a carriage return character,
    currentLine += c;    // add it to the end of the currentLine
}
// Check to see the client request:
if (currentLine.endsWith("GET /LED_on")) {
    selection = "Sel1";
    // envio orden de encender el Led
    system("mosquitto_pub --host 169.254.4.103 --topic led --message
\"LED_ON\"");
    delay(2000);

}
if (currentLine.endsWith("GET /LED_off")) {
    selection = "Sel2";
    // envio orden de apagar el LED
    system("mosquitto_pub --host 169.254.4.103 --topic led --message
\"LED_OFF\"");
    delay(2000);
}
// give the web browser time to receive the data
delay(1);
// close the connection:
client.stop();
Serial.println("Cliente Desconectado");
}
else {
    Serial.println("cliente no esta listo");
}
}

/***********************/
void sendPagetoClient(EthernetClient client, String selected)
{

```

```

client.println("HTTP/1.1 200 OK");
client.println("Content-type:text/html");
client.println();

// the content of the HTTP response follows the header:
client.print("Para encender el Led <a href=\"/LED_on\"> haga click  

aqui</a><br>");
client.print("Para apagarlo <a href=\"/LED_off\"> haga click</a><br>");

if (selected == "LED_on") {
    client.print("Se ha encendido el LED ");
}
if (selected == "LED_off") {
    client.print("Se ha apagado el LED ");
}
// The HTTP response ends with another blank line:
client.println();
}

```

Ahora crear un archivo llamado cliente\_suscriptor.py y dentro copiar el siguiente contenido:

```

#!/usr/bin/python
import paho.mqtt.client as mqtt

# The callback for when the client receives a CONNACK response from the
server.
def on_connect(client, userdata, flags, rc):
    client.subscribe("led")
    # Subscribing in on_connect() means that if we lose the connection and
    # reconnect then subscriptions will be renewed.

# The callback for when a PUBLISH message is received from the server.
def on_message(client, userdata, msg):
    mensaje_recibido = str(msg.payload)
    topico_recibido = str(msg.topic)
    if topico_recibido == 'led':
        if mensaje_recibido == 'LED_ON':
            f = open('led_on.txt', 'w')
            # Creo un archivo llamado led_on.txt
            # y desde arduino enciendo dicho led
            f.write('led_on')
            f.close()
        if mensaje_recibido == 'LED_OFF':
            f = open('led_off.txt', 'w')
            # Creo un archivo llamado led_off.txt
            # y desde arduino apago dicho led
            f.write('led_off')
            f.close()

```

```

mensaje_recibido= ''
topico_recibido = ''
client = mqtt.Client()
client.on_connect = on_connect
client.on_message = on_message

client.connect("169.254.4.103", 1883, 60)

# Blocking call that processes network traffic, dispatches callbacks and
# handles reconnecting.
# Other loop*() functions are available that give a threaded interface and a
# manual interface.
client.loop_forever()

```

Conectarse por winSCP a la ip de la Galileo Server y subir cliente\_suscripcion.py al directorio raiz.

Ya encuentra todo lo necesario para poder reproducir el proyecto completo, editar y modificar lo que necesite.

## **Obtención de los sketch, el script de python y subida a las respectivas Galileos**

Descargamos la siguiente [carpeta](#) que descomprimimos y abrimos desde el IDE de Arduino para bajar a la Galileo Server.

Repetimos lo anterior con este [link](#), descargando, descomprimiendo y abriendola desde el IDE de Arduino pero esta vez conectado a la Galileo actuadora. Finalmente, mediante winSCP pasamos el siguiente [script](#) descomprimido al directorio raíz de la placa actuadora.

Con los pasos seguidos debería poder levantar el sistema, tener la página web e interactuar con el LED interno.

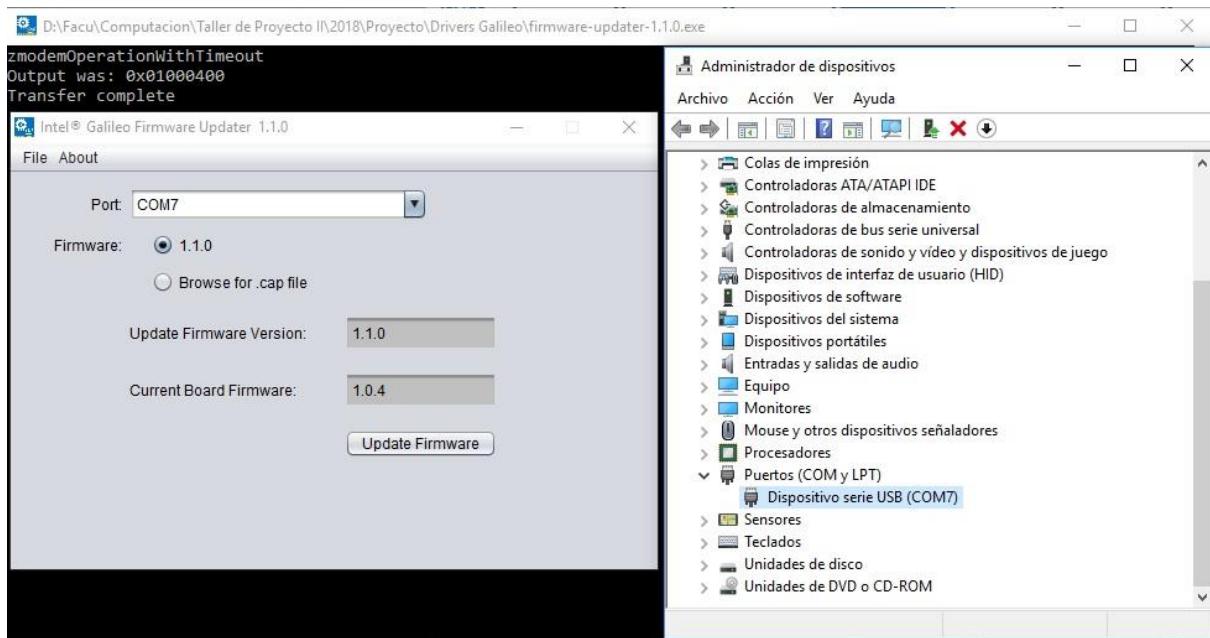
## **Sección proyecto**

A continuación, describiremos solo los pasos necesarios para poder ejecutar el proyecto asumiendo que se tienen los materiales, pero sin nada instalado aun:

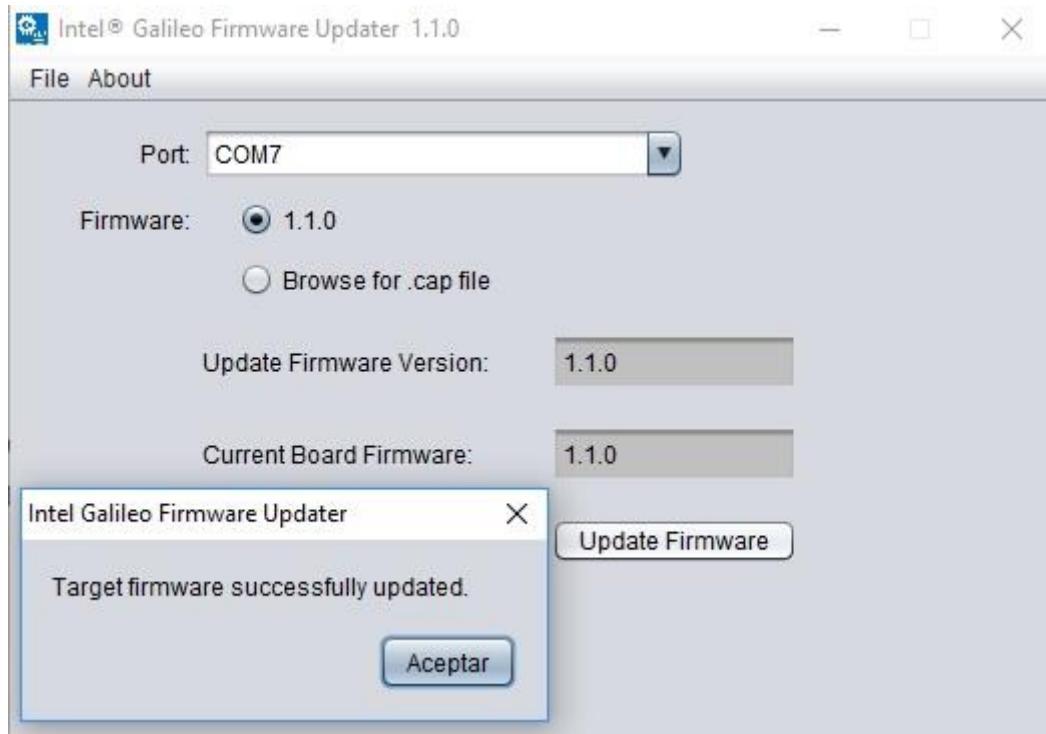
### **Actualizar drivers de la Galileo:**

Para actualizar los Drivers de la Galileo los descargamos [aquí](#).

En nuestro caso usamos los que son para Windows 10. Conectamos mediante el puerto USB cliente la placa a la PC sin tener la memoria microSD colocada. Abrimos el programa y vemos en la siguiente imagen que los drivers de nuestra Galileo están desactualizados. Los actualizamos haciendo click en update firmware:



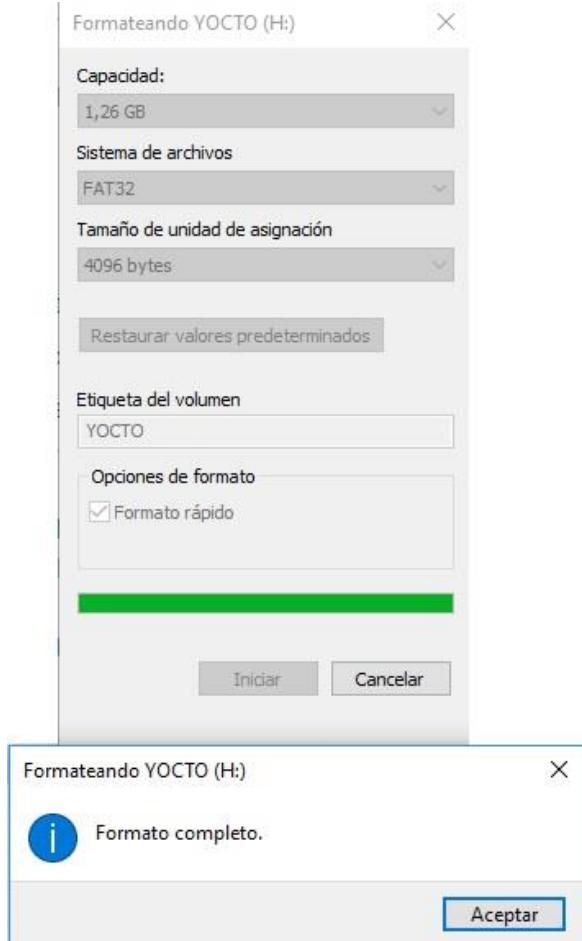
Aceptamos los siguientes modals que nos consultan si la placa está conectada a una fuente de tensión externa y si estamos seguros de subir de versión. Esperando 5 minutos aproximadamente si finaliza exitosamente, podremos ver el siguiente mensaje:



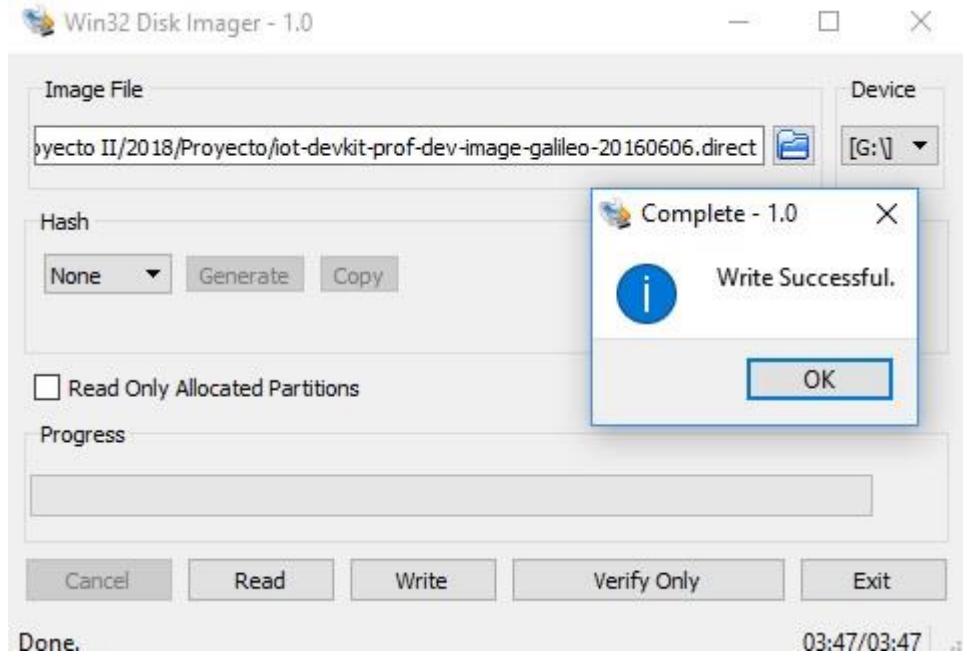
## Instalación de linux en la Galileo:

Descargamos la imagen de Linux que usaremos con la memoria microSD [aquí](#). Al 30/09/2018 el archivo se llama “Galileo\_Poky\_SW\_image\_20160606” y se encuentra en formato zip. Para grabar la imagen linux Yocto en la SD desde el explorador de archivos de

windows con la tarjeta colocada hacemos click derecho sobre la unidad y clickeamos formatear. Tener en cuenta que el formato por default que esta seleccionado suele ser FAT (predeterminado) que no sirve para nuestro caso ya que el File System es FAT16. Por lo tanto, seleccionamos la opción FAT32 y le asignamos un nombre para darle formato como se ve a continuación:



Luego usamos “Win32DiskImager” para copiar la imagen a la SD (abrir con permisos de administrador o puede fallar al intentar escribir). Elegimos el archivo con extensión “direct” que obtenemos al descomprimir el zip descargado y si finalizó exitosamente deberíamos ver el siguiente resultado:



## Instalación del IDE Arduino para Galileo:

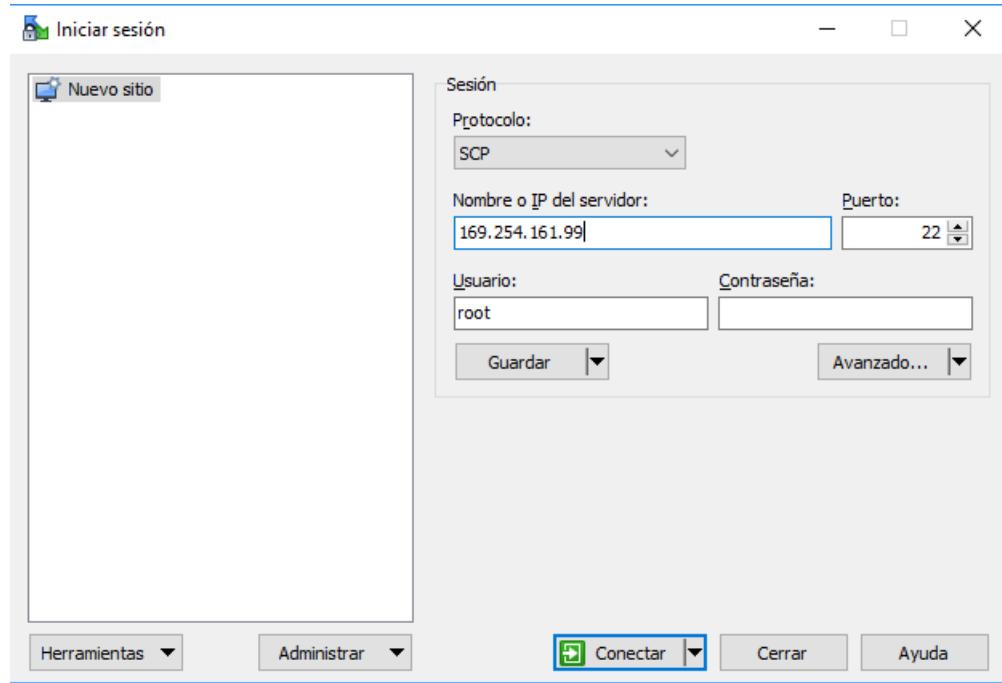
Descargamos el IDE para el SO que usemos [aquí](#) (Windows en nuestro caso). Lo abrimos y obtenemos las librerías que necesitamos para la Galileo buscando Intel desde Herramientas -> Gestión de Librerías.

## Instalación de Mosquitto en ambas Galileo:

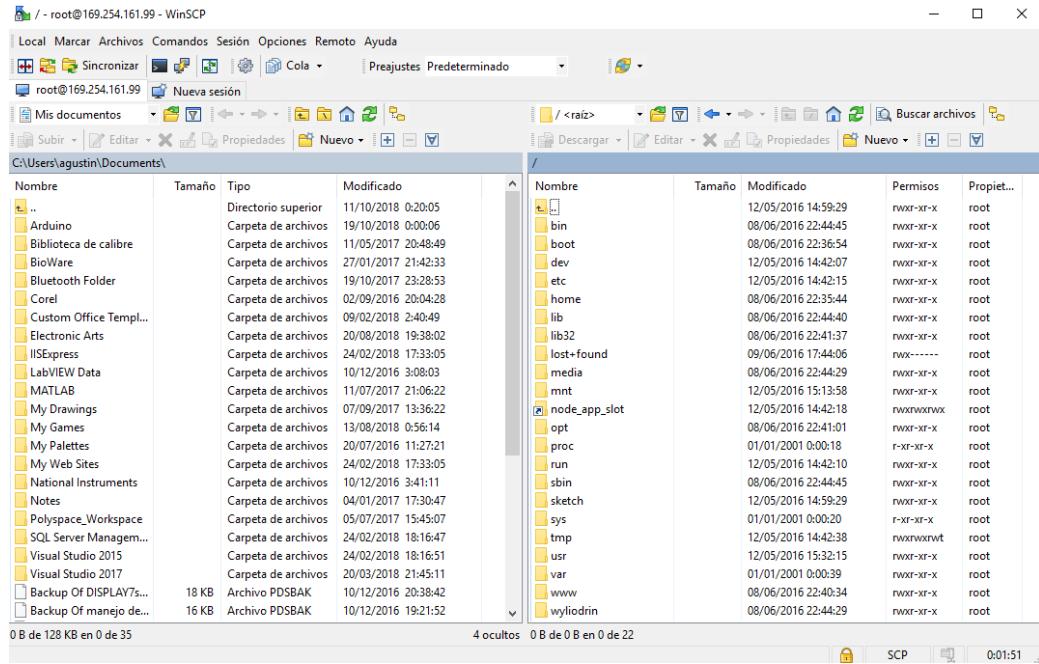
Ahora debemos transferir los archivos que nos permitan utilizar el protocolo MQTT Mosquitto entre nuestras placas. Debido a que en el presente proyecto ninguna de ellas dispone de acceso a internet directamente, optamos por utilizar un programa que nos permita realizar dicha transferencia mediante SSH desde nuestra PC. Para ello descargamos e instalamos [winscp](#). Habiendo instalado el programa, descargamos el archivo fuente de mosquitto desde su página. A la fecha se encuentra disponible la versión 1.5.3:

The screenshot shows the Mosquitto download page at <https://mosquitto.org/download/>. The page features the Mosquitto logo and links for Eclipse and Home. A prominent 'Download' button is visible. Below it, the 'Source' section lists 'mosquitto-1.5.3.tar.gz' (319kB) with GPG signature and a link to the Git source code repository. A note states 'Older downloads are available at <http://mosquitto.org/files/>'. The 'Binary Installation' section notes that binary packages are supported by the Mosquitto project. The 'Windows' section lists two executables: 'mosquitto-1.5.3-install-windows-x64.exe' and 'mosquitto-1.5.3-install-windows-x32.exe', both built with Visual Studio Community 2017. A note at the bottom suggests reading 'readme-windows.txt' after installing.

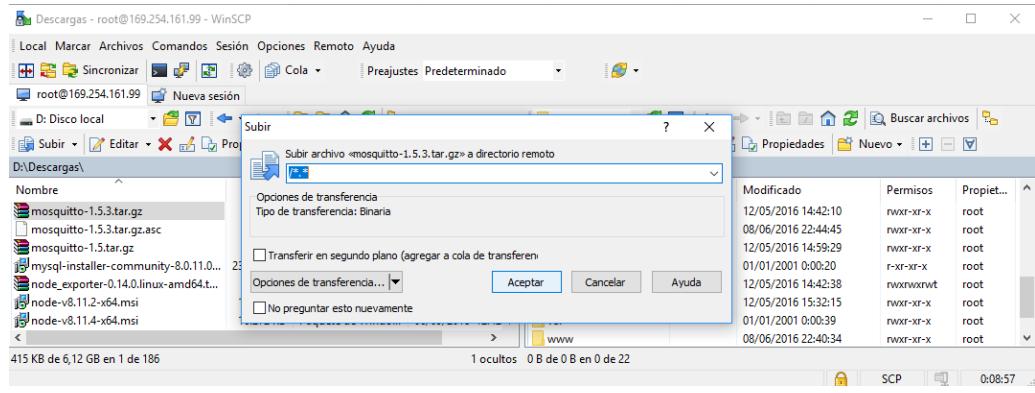
Una vez descargado el archivo con extensión .tar.gz abrimos winSCP y nos conectamos a la ip que le asignamos (169.254.161.99 en nuestro caso):



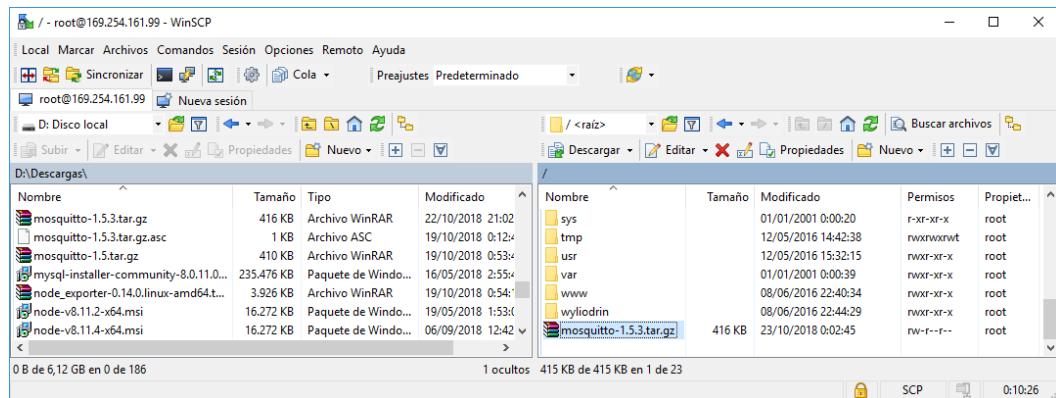
Al conectarnos de manera exitosa vemos la siguiente pantalla:



Haciendo click derecho en el archivo a subir y eligiendo subir vemos el siguiente cuadro de dialogo:



Aceptando vemos como el contenido se encuentra en la carpeta destino de nuestra Galileo:



Este archivo lo necesitamos en ambas Galileos, ya que las dos deben poder ejecutar el demonio mosquitto, poder publicar y subscribir. Por ello hay que repetir lo hecho recién con la otra Galileo (lógicamente con la ip que corresponda en su caso) y también se debe repetir el siguiente paso que hace el *make* de mosquitto.

Ahora nos conectamos a la placa mediante Putty y nos posicionamos en el directorio raíz:

```
169.254.161.99 - PuTTY
root@galileo:~# cd ..
root@galileo:/home# cd ..
root@galileo:# ls
bin  etc  lib32      mnt          opt  sbin  tmp  www
boot home lost+found  mosquitto-1.5.3.tar.gz  proc  sketch  usr  wylidrin
dev   lib   media     node_app_slot    run   sys   var
root@galileo:#
```

Verificamos la fecha que nos muestra el sistema usando el comando date:

```
169.254.161.99 - PuTTY
root@galileo:# date
Thu May 12 14:54:16 UTC 2016
root@galileo:#
```

Actualizamos la fecha del sistema a la correcta (día de la fecha 29/10/2018):

```
root@galileo:/# date -s "29 OCT 2018 22:00:00"
Mon Oct 29 22:00:00 UTC 2018
```

También actualizamos el RTC a la fecha que acabamos de configurar:

```
root@galileo:/# hwclock --systohc
root@galileo:/# hwclock
Mon Oct 29 22:07:31 2018  0.000000 seconds
root@galileo:/#
```

Realizamos la descompresión del archivo, nos posicionamos dentro de la carpeta creada y hacemos “make WITH\_SRV=no” (demora aproximadamente 20 minutos):

```
root@galileo:/mosquitto-1.5.3# make WITH_SRV=no
set -e; for d in lib client src; do make -C ${d}; done
make[1]: Entering directory '/mosquitto-1.5.3/lib'
cc -Wall -ggdb -O2 -I. -I.. -I../lib -fPIC -DWITH_TLS -DWITH_TLS_PSK -DWITH_THREADING -DWITH_SOCKETS -c mosquitto.c -o mosquitto.o
cc -Wall -ggdb -O2 -I. -I.. -I../lib -fPIC -DWITH_TLS -DWITH_TLS_PSK -DWITH_THREADING -DWITH_SOCKETS -c actions.c -o actions.o
cc -Wall -ggdb -O2 -I. -I.. -I../lib -fPIC -DWITH_TLS -DWITH_TLS_PSK -DWITH_THREADING -DWITH_SOCKETS -c callbacks.c -o callbacks.o
cc -Wall -ggdb -O2 -I. -I.. -I../lib -fPIC -DWITH_TLS -DWITH_TLS_PSK -DWITH_THREADING -DWITH_SOCKETS -c connect.c -o connect.o
cc -Wall -ggdb -O2 -I. -I.. -I../lib -fPIC -DWITH_TLS -DWITH_TLS_PSK -DWITH_THREADING -DWITH_SOCKETS -c handle_connack.c -o handle_connack.o
cc -Wall -ggdb -O2 -I. -I.. -I../lib -fPIC -DWITH_TLS -DWITH_TLS_PSK -DWITH_THREADING -DWITH_SOCKETS -c handle_ping.c -o handle_ping.o
cc -Wall -ggdb -O2 -I. -I.. -I../lib -fPIC -DWITH_TLS -DWITH_TLS_PSK -DWITH_THREADING -DWITH_SOCKETS -c handle_pubackcomp.c -o handle_pubackcomp.o
cc -Wall -ggdb -O2 -I. -I.. -I../lib -fPIC -DWITH_TLS -DWITH_TLS_PSK -DWITH_THREADING -DWITH_SOCKETS -c handle_publish.c -o handle_publish.o
cc -Wall -ggdb -O2 -I. -I.. -I../lib -fPIC -DWITH_TLS -DWITH_TLS_PSK -DWITH_THREADING -DWITH_SOCKETS -c handle_pubrec.c -o handle_pubrec.o
cc -Wall -ggdb -O2 -I. -I.. -I../lib -fPIC -DWITH_TLS -DWITH_TLS_PSK -DWITH_THREADING -DWITH_SOCKETS -c handle_pubrel.c -o handle_pubrel.o
cc -Wall -ggdb -O2 -I. -I.. -I../lib -fPIC -DWITH_TLS -DWITH_TLS_PSK -DWITH_THREADING -DWITH_SOCKETS -c handle_suback.c -o handle_suback.o
cc -Wall -ggdb -O2 -I. -I.. -I../lib -fPIC -DWITH_TLS -DWITH_TLS_PSK -DWITH_THREADING -DWITH_SOCKETS -c handle_unsuback.c -o handle_unsuback.o
cc -Wall -ggdb -O2 -I. -I.. -I../lib -fPIC -DWITH_TLS -DWITH_TLS_PSK -DWITH_THREADING -DWITH_SOCKETS -c helpers.c -o helpers.o
cc -Wall -ggdb -O2 -I. -I.. -I../lib -fPIC -DWITH_TLS -DWITH_TLS_PSK -DWITH_THREADING -DWITH_SOCKETS -c logging_mosq.c -o logging_mosq.o
cc -Wall -ggdb -O2 -I. -I.. -I../lib -fPIC -DWITH_TLS -DWITH_TLS_PSK -DWITH_THREADING -DWITH_SOCKETS -c loop.c -o loop.o
cc -Wall -ggdb -O2 -I. -I.. -I../lib -fPIC -DWITH_TLS -DWITH_TLS_PSK -DWITH_THREADING -DWITH_SOCKETS -c memory_mosq.c -o memory_mosq.o
cc -Wall -ggdb -O2 -I. -I.. -I../lib -fPIC -DWITH_TLS -DWITH_TLS_PSK -DWITH_THREADING -DWITH_SOCKETS -c messages_mosq.c -o messages_mosq.o
cc -Wall -ggdb -O2 -I. -I.. -I../lib -fPIC -DWITH_TLS -DWITH_TLS_PSK -DWITH_THREADING -DWITH_SOCKETS -c net_mosq.c -o net_mosq.o
cc -Wall -ggdb -O2 -I. -I.. -I../lib -fPIC -DWITH_TLS -DWITH_TLS_PSK -DWITH_THREADING -DWITH_SOCKETS -c options.c -o options.o
cc -Wall -ggdb -O2 -I. -I.. -I../lib -fPIC -DWITH_TLS -DWITH_TLS_PSK -DWITH_THREADING -DWITH_SOCKETS -c packet_mosq.c -o packet_mosq.o
cc -Wall -ggdb -O2 -I. -I.. -I../lib -fPIC -DWITH_TLS -DWITH_TLS_PSK -DWITH_THREADING -DWITH_SOCKETS -c read_handle.c -o read_handle.o
cc -Wall -ggdb -O2 -I. -I.. -I../lib -fPIC -DWITH_TLS -DWITH_TLS_PSK -DWITH_THREADING -DWITH_SOCKETS -c send_connect.c -o send_connect.o
cc -Wall -ggdb -O2 -I. -I.. -I../lib -fPIC -DWITH_TLS -DWITH_TLS_PSK -DWITH_THREADING -DWITH_SOCKETS -c send_disconnect.c -o send_disconnect.o
cc -Wall -ggdb -O2 -I. -I.. -I../lib -fPIC -DWITH_TLS -DWITH_TLS_PSK -DWITH_THREADING -DWITH_SOCKETS -c send_mosq.c -o send_mosq.o
cc -Wall -ggdb -O2 -I. -I.. -I../lib -fPIC -DWITH_TLS -DWITH_TLS_PSK -DWITH_THREADING -DWITH_SOCKETS -c send_publish.c -o send_publish.o
cc -Wall -ggdb -O2 -I. -I.. -I../lib -fPIC -DWITH_TLS -DWITH_TLS_PSK -DWITH_THREADING -DWITH_SOCKETS -c send_subscribe.c -o send_subscribe.o
be.o
```

## Instalación de Paho-mqtt solo en Galileo actuadora:

A continuación, realizamos la instalación en la Galileo actuadora de otra librería llamada Eclipse MQTT Paho que tiene varias implementaciones, optamos por la que usa Python.

Descargamos el repositorio desde Github desde [aquí](#).

Usamos WinSCP para pasar la carpeta (descomprimir el zip antes de pasarlo para facilitar su uso).

Luego de posicionarnos en el directorio raiz, entramos a la carpeta paho.mqtt.python y ejecutamos el comando `python setup.py install`:

```

root@galileo:/# cd paho.mqtt.python-master/
root@galileo:/paho.mqtt.python-master# python setup.py install
running install
running bdist_egg
running egg_info
creating src/paho_mqtt.egg-info
writing src/paho_mqtt.egg-info/PKG-INFO
writing top-level names to src/paho_mqtt.egg-info/top_level.txt
writing dependency_links to src/paho_mqtt.egg-info/dependency_links.txt
writing manifest file 'src/paho_mqtt.egg-info/SOURCES.txt'
reading manifest file 'src/paho_mqtt.egg-info/SOURCES.txt'
reading manifest template 'MANIFEST.in'
writing manifest file 'src/paho_mqtt.egg-info/SOURCES.txt'
installing library code to build/bdist.linux-i586/egg
running install_lib
running build_py
creating build
creating build/lib
creating build/lib/paho
copying src/paho/_init__.py -> build/lib/paho
creating build/lib/paho/mqtt
copying src/paho/mqtt/matcher.py -> build/lib/paho/mqtt
copying src/paho/mqtt/publish.py -> build/lib/paho/mqtt
copying src/paho/mqtt/_init__.py -> build/lib/paho/mqtt
copying src/paho/mqtt/subscribe.py -> build/lib/paho/mqtt
copying src/paho/mqtt/client.py -> build/lib/paho/mqtt
creating build/bdist.linux-i586
creating build/bdist.linux-i586/egg
creating build/bdist.linux-i586/egg/paho
creating build/bdist.linux-i586/egg/paho/mqtt
copying build/lib/paho/mqtt/matcher.py -> build/bdist.linux-i586/egg/paho/mqtt
copying build/lib/paho/mqtt/publish.py -> build/bdist.linux-i586/egg/paho/mqtt
copying build/lib/paho/mqtt/_init__.py -> build/bdist.linux-i586/egg/paho/mqtt
copying build/lib/paho/mqtt/subscribe.py -> build/bdist.linux-i586/egg/paho/mqtt
copying build/lib/paho/mqtt/client.py -> build/bdist.linux-i586/egg/paho/mqtt
copying build/lib/paho/_init__.py -> build/bdist.linux-i586/egg/paho
byte-compiling build/bdist.linux-i586/egg/paho/mqtt/matcher.py to matcher.pyc
byte-compiling build/bdist.linux-i586/egg/paho/mqtt/publish.py to publish.pyc

```

## Obtención de los sketch, el script de python y subida a las respectivas Galileos

Ahora descargamos la siguiente [carpeta](#) que descomprimimos y abrimos desde el IDE de Arduino para bajar a la Galileo Server.

Repetimos lo anterior con este [link](#), descargando, descomprimiendo y abriendola desde el IDE de Arduino pero esta vez conectado a la Galileo actuadora. Finalmente, mediante winSCP pasamos el siguiente [script](#) descomprimido al directorio raíz de la placa actuadora.

Con los pasos seguidos debería poder levantar el sistema, tener la página web e interactuar con el LED interno.

## 5- Documentación en Formato Gráfico y Video

Adjunto el siguiente [video](#) que condensa toda la información gráfica del sistema en funcionamiento.

Resumidamente, vemos la página web desde la PC, a la izquierda vemos ambas Galileo (La superior es la actuadora y la inferior la Server) y el switch. Se puede apreciar que luego de un buen rato (demora bastante la primera vez que se le hace click a algunos de los links, luego se puede ver que responde más rápidamente) se enciende el led en la placa superior. Lógicamente, en el switch se ven las 3 luces parpadeando constantemente, indicando la transmisión de paquetes. Los roles que corresponden a cada parte son: la página web y la interacción con el usuario es competencia de la Galileo Server (que se encuentra en la parte inferior); mientras que del encendido y apagado del led se encarga la Galileo

Actuadora (que se encuentra en la parte superior). Ambas comparten parte del mecanismo de comunicación de mensajes MQTT – Mosquitto. La Server publica los mensajes indicando la acción requerida por el usuario en la web y la Actuadora, suscrita al tópico en cuestión, recibe el mensaje y actúa según corresponda sobre el LED.

## Anexo

### **Errores encontrados con sus soluciones (algunas no se pudieron solucionar aun) y pruebas hechas.**

Al principio mostramos los dos errores fundamentales que tenemos. El primero me impide usar el servo y el segundo error es al intentar usar la librería del IDE de Arduino que permite el uso de MQTT.

#### **4/12/2018 - Sketch de Arduino para manejar el servo dentro del código final:**

El proyecto finalmente no actúa sobre el servo debido a que no compila el archivo de la actuadora en el IDE de Arduino. El algoritmo básico de control del servo que se quiso implementar es el siguiente:

El servo se puede iniciar y detener. La primera vez que se inicia inicia en la posición inicial y empieza a girar en sentido horario hasta lograr un ángulo de 180 momento en el que vuelve en sentido antihorario a la posición inicial. En cualquier momento se puede detener el barrido del motor y volver a iniciar. Siempre retoma el barrido desde la última posición. Como detalle enciende el led interno cuando está avanzando en sentido horario.

```
#include <Servo.h>
Servo myservo;
// variable que almacena la posicion del servo
int pos = 0;
int sentido_horario = 1;
void setup() {
    system("telnetd -l /bin/sh");
    delay(2000);

    system("ip link set enp0s20f6 down");

    Serial.println("enps... down > /dev/ttyGS0");

    system("ip link set enp0s20f6 name eth0 > /dev/ttyGS0");
    delay(2000);

    system("ip link set eth0 up");
    delay(2000);
```

```

// eglibc ethernet naming...
system("ip addr add 169.254.4.105/8 dev eth0 > /dev/ttyGS0");
delay(2000);

// attaches the servo on pin 9 to the servo object
myservo.attach(9);
system("ip addr show > /dev/ttyGS0");
delay(2000);
// pin 13 como salida -> LED interno
pinMode(13, OUTPUT);
delay(2000);
// Apago LED interno por si estaba encendido
digitalWrite(13, LOW);
delay(2000);

// inicializamos el demonio mosquitto
system("mosquitto -d");
delay(2000);
//ejecutamos el script de python que se comunica mediante mqtt mosquitto
system("cd /");
system("chmod a+x cliente_suscriptor.py");
delay(2000);
system("./cliente_suscriptor.py");
delay(2000);
}

void loop() {
FILE *f_led_on;
FILE *f_led_off;

//espero a que enciendan el servo
do {
    f_led_on = fopen("led_on.txt", "r");
    delay(2000);
} while (f_led_on == NULL);
fclose(f_led_on);
delay(2000);
system("rm led_on.txt");
delay(2000);
// se enciende el motor, comienza la rotacion horario
while (f_led_off == NULL) {
    f_led_off = fopen("led_off.txt", "r");
    delay(2000);
    if (pos < 180 && sentido_horario){
        //giro sentido horario
        pos = pos + 1;
        digitalWrite(13, HIGH);
        delay(2000);
    }
    else {

```

```

sentido_horario = 0;
if (pos == 0) {
    sentido_horario = 1;
}
//giro sentido antihorario
pos = pos - 1;
}
myservo.write(pos);
delay(15);
}
fclose(f_led_off);
system("rm led_off.txt");
}

```

El error de compilación que nos surje es:

```

redefinition of 'int pos'
ejemplo-servo-sencillo:3:5: error: redefinition of 'int pos'

ejemplo-servo:4:5: error: 'int pos' previously defined here

ejemplo-servo-sencillo:4:5: error: redefinition of 'int sentido_horario'

ejemplo-servo:5:5: error: 'int sentido_horario' previously defined here

D:\Facu\Computacion\Taller de Proyecto II\2018\Proyecto\ejemplo-servo\ejemplo-servo-sencillo.ino: In function 'void setup()':

ejemplo-servo-sencillo:5:6: error: redefinition of 'void setup()'

ejemplo-servo:6:6: error: 'void setup()' previously defined here

D:\Facu\Computacion\Taller de Proyecto II\2018\Proyecto\ejemplo-servo\ejemplo-servo-sencillo.ino: In function 'void loop()':

ejemplo-servo-sencillo:44:6: error: redefinition of 'void loop()'

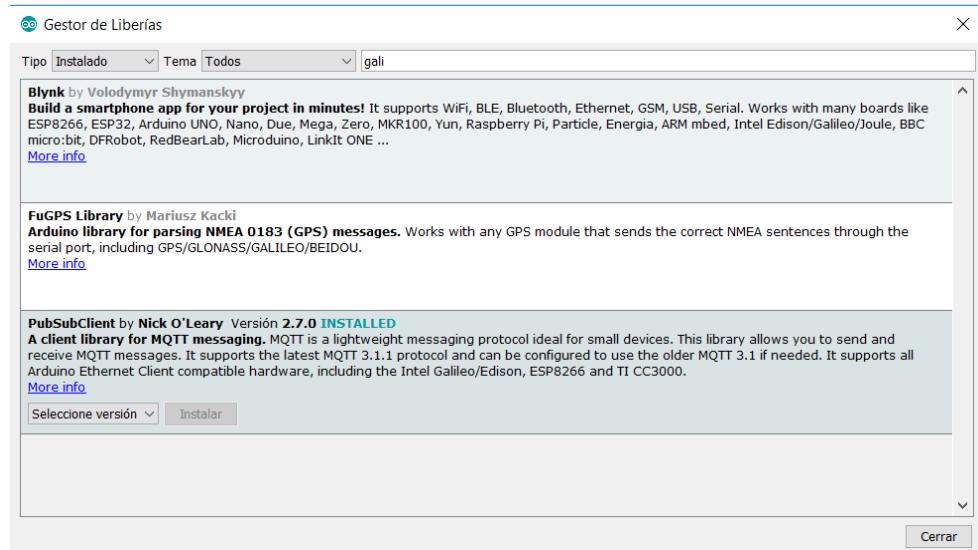
ejemplo-servo:45:6: error: 'void loop()' previously defined here

Se encontraron múltiples librerías para "Servo.h"
Usado: C:\Users\agustin\AppData\Local\Arduino15\packages\Intel\hardware\i586\1.6.7+1.0\libraries\Servo
No usado: C:\Users\agustin\Documents\Arduino\libraries\Servo
No usado: C:\Program Files (x86)\Arduino\libraries\Servo
exit status 1

```

### 30/10/2018 - Compilación de algunos de los ejemplos que trae la librería PubSubClient del IDE de Arduino:

Primero descargamos la librería desde el IDE de Arduino. Para ello con el programa abierto vamos a Herramientas -> Gestión de librerías. Escribimos galileo y descargamos la que diga PubSubClient. En mi caso ya está instalada:



Desde Archivos -> Ejemplos buscamos el correspondiente a la librería llamado basic y lo cargamos.

El código es el siguiente:

```
/*
Basic MQTT example

This sketch demonstrates the basic capabilities of the library.
It connects to an MQTT server then:
- publishes "hello world" to the topic "outTopic"
- subscribes to the topic "inTopic", printing out any messages
  it receives. NB - it assumes the received payloads are strings not binary

It will reconnect to the server if the connection is lost using a blocking
reconnect function. See the 'mqtt_reconnect_nonblocking' example for how to
achieve the same result without blocking the main loop.

*/
#include <SPI.h>
#include <Ethernet.h>
#include <PubSubClient.h>

// Update these with values suitable for your network.
byte mac[] = { 0xDE, 0xED, 0xBA, 0xFE, 0xFE, 0xED };
IPAddress ip(172, 16, 0, 100);
IPAddress server(172, 16, 0, 2);

void callback(char* topic, byte* payload, unsigned int length) {
  Serial.print("Message arrived [");
  Serial.print(topic);
  Serial.print("] ");
  for (int i=0;i<length;i++) {
    Serial.print((char)payload[i]);
  }
}
```

```
        }
        Serial.println();
    }

EthernetClient ethClient;
PubSubClient client(ethClient);

void reconnect() {
    // Loop until we're reconnected
    while (!client.connected()) {
        Serial.print("Attempting MQTT connection...");
        // Attempt to connect
        if (client.connect("arduinoClient")) {
            Serial.println("connected");
            // Once connected, publish an announcement...
            client.publish("outTopic","hello world");
            // ... and resubscribe
            client.subscribe("inTopic");
        } else {
            Serial.print("failed, rc=");
            Serial.print(client.state());
            Serial.println(" try again in 5 seconds");
            // Wait 5 seconds before retrying
            delay(5000);
        }
    }
}

void setup()
{
    Serial.begin(57600);

    client.setServer(server, 1883);
    client.setCallback(callback);

    Ethernet.begin(mac, ip);
    // Allow the hardware to sort itself out
    delay(1500);
}

void loop()
{
    if (!client.connected()) {
        reconnect();
    }
    client.loop();
}
```

Intentamos compilarlo y surge el siguiente error:

```
Error compilando para la tarjeta Intel® Galileo.
C:\Users\agustin\Documents\Arduino\libraries\PubSubClient\src\PubSubClient.cpp: In member function 'boolean PubSubClient::readByte(uint8_t*)':
C:\Users\agustin\Documents\Arduino\libraries\PubSubClient\src\PubSubClient.cpp:225:12: error: 'yield' was not declared in this scope
Se encontraron múltiples librerías para "Ethernet.h"
Usado: C:\Users\agustin\AppData\Local\Arduino15\packages\Intel\hardware\i586\1.6.7+1.0\libraries\Ethernet
  No usado: C:\Program Files (x86)\Arduino\libraries\Ethernet
exit status 1
Error compilando para la tarjeta Intel® Galileo.

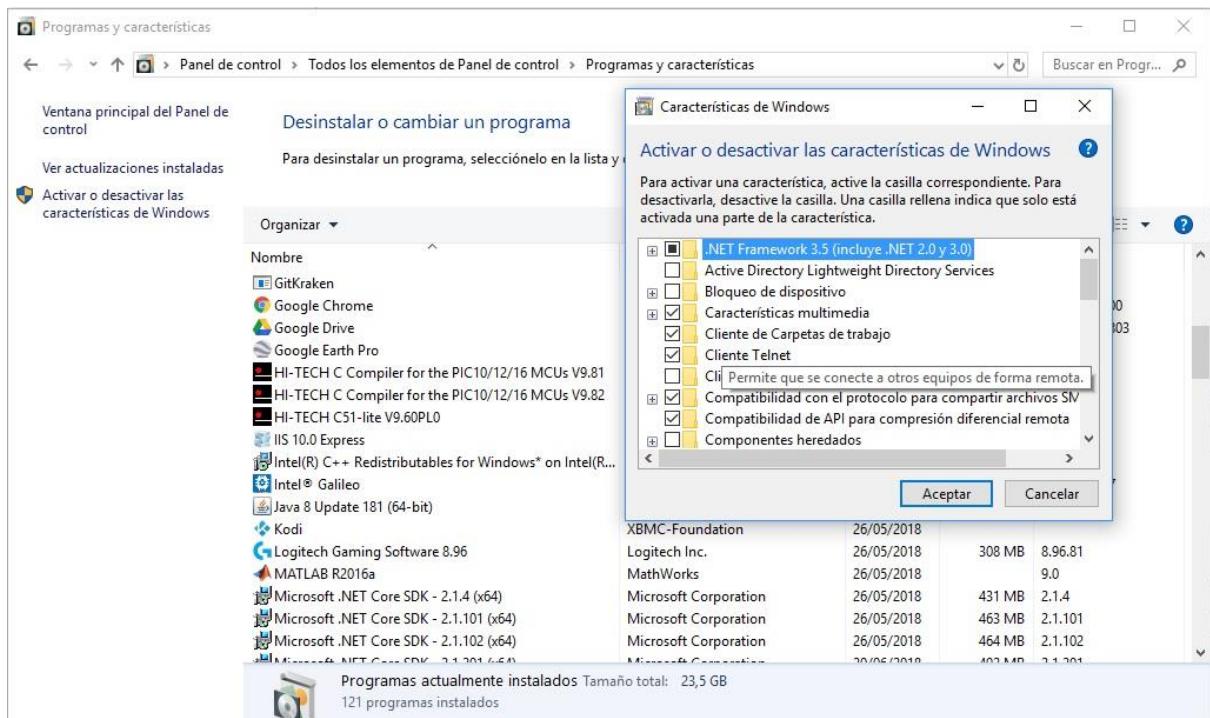
61
```

Si bien se encontró un [Issue](#) cerrado del [repositorio de Github del autor](#) de la librería, se desconoce la solución ya que el autor de dicha librería dice que anda para la línea Galileo/Edison y comprobé anteriormente que no es así. Adjunto el link a dicha [página](#). Por este motivo se optó finalmente usar la librería Paho-mqtt implementada en Python.

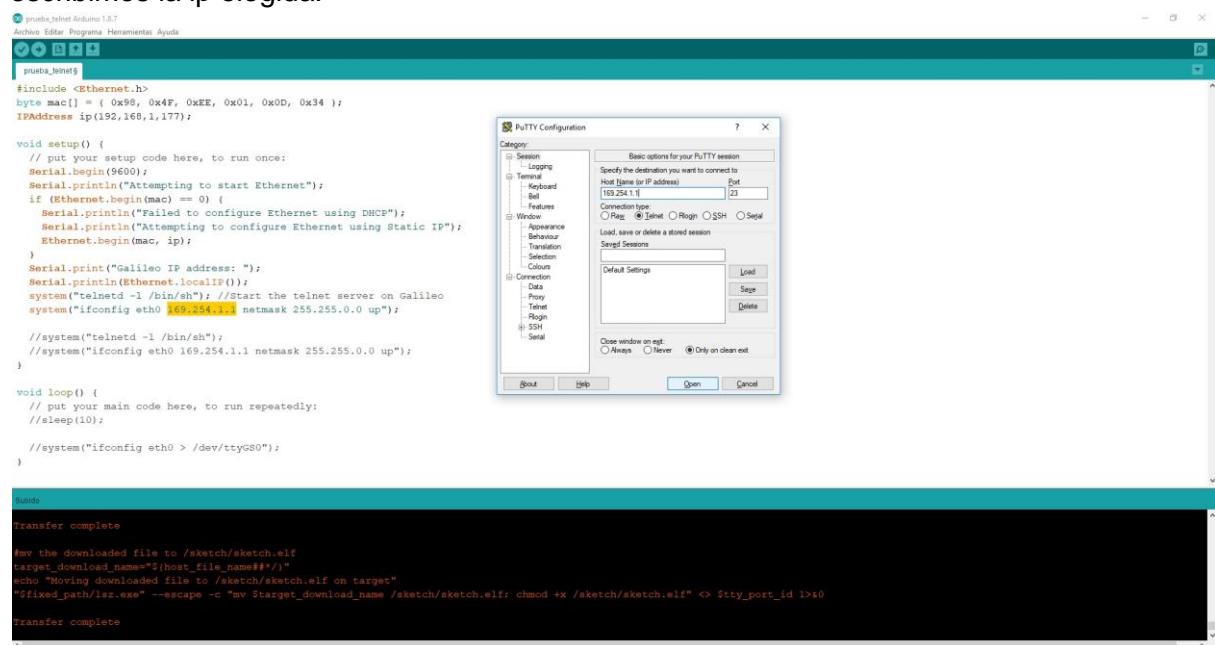
Continuamos con los errores encontramos en orden cronológico, sus soluciones y pruebas que ayudan a comprender el desarrollo final del presente proyecto.

## A1) Ejemplo de uso de la Galileo via USB con el IDE de Arduino.

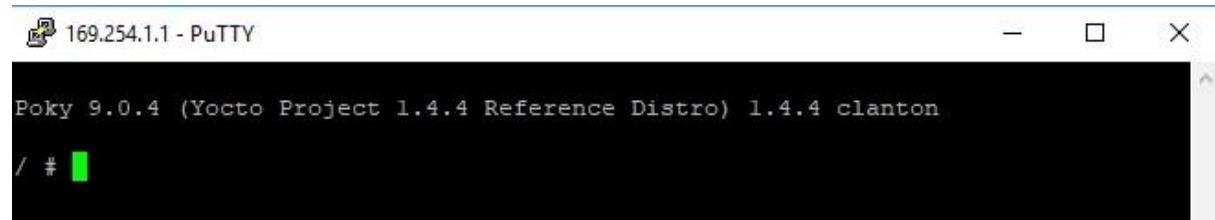
Para iniciar la conexión ethernet conectamos la Galileo y la PC al switch. Creamos un pequeño algoritmo que inicialice la MAC y una dirección ip que nos permita mediante telnet conectarnos usando [esta guia](#) de cabecera. Nosotros usamos Putty pero si se desea usar el comando telnet directamente desde la consola powershell en Windows 10 debemos habilitarlo ya que viene deshabilitado por default. Para ello desde “inicio->panel del control-> programas” haciendo click en “activar o desactivar las características de Windows” podemos tildar el checkbox Telnet como vemos en la siguiente captura:



Luego de haber subido el sketch a la Galileo, abrimos putty y tildando la opcion Telnet escribimos la ip elegida:

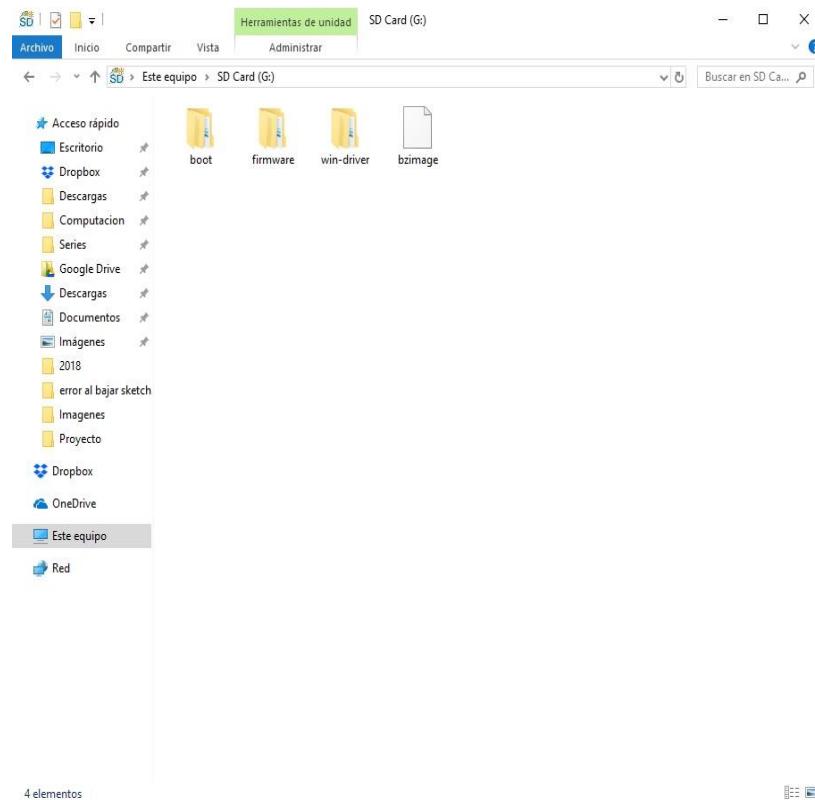


Si anduvo debemos ver el siguiente resultado:



## A2) Reproduciendo error al bajar sketch a Galileo con SD.

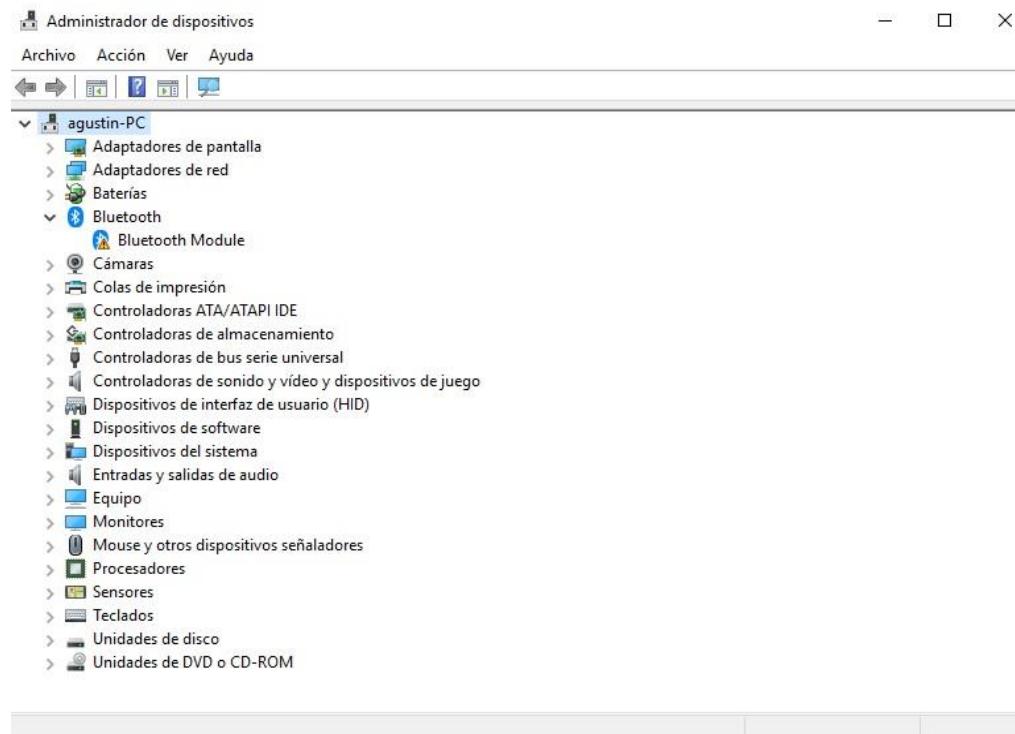
Contenido de la SD luego de grabar Yocto via [SD Card Formatter](#):



Con la placa alimentada y con la tarjeta microSD conectada y sin el usb conectado de la siguiente manera:



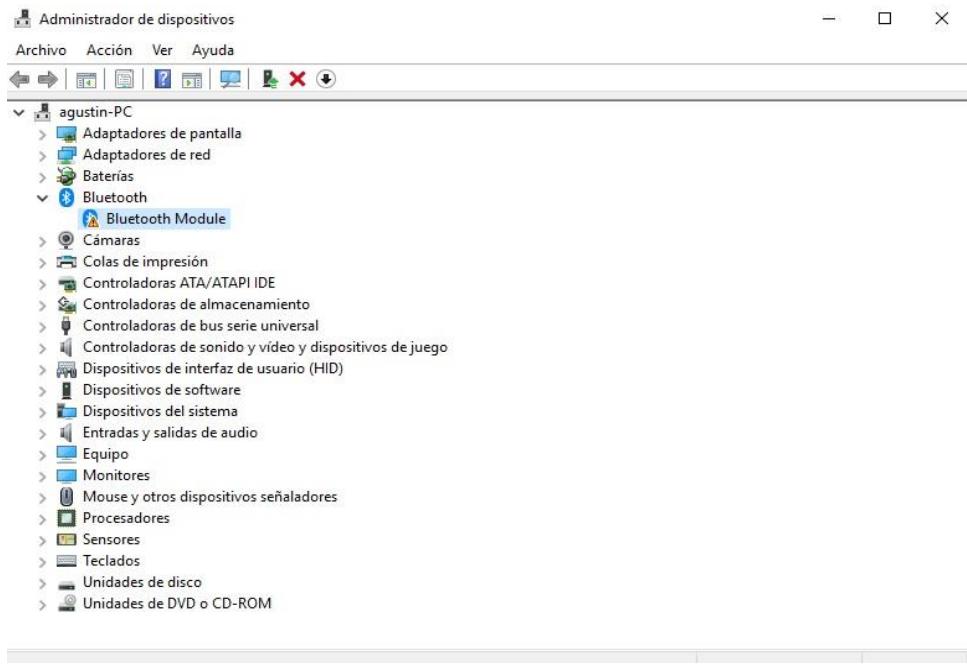
El Administrador de Dispositivos nos muestra lo siguiente:



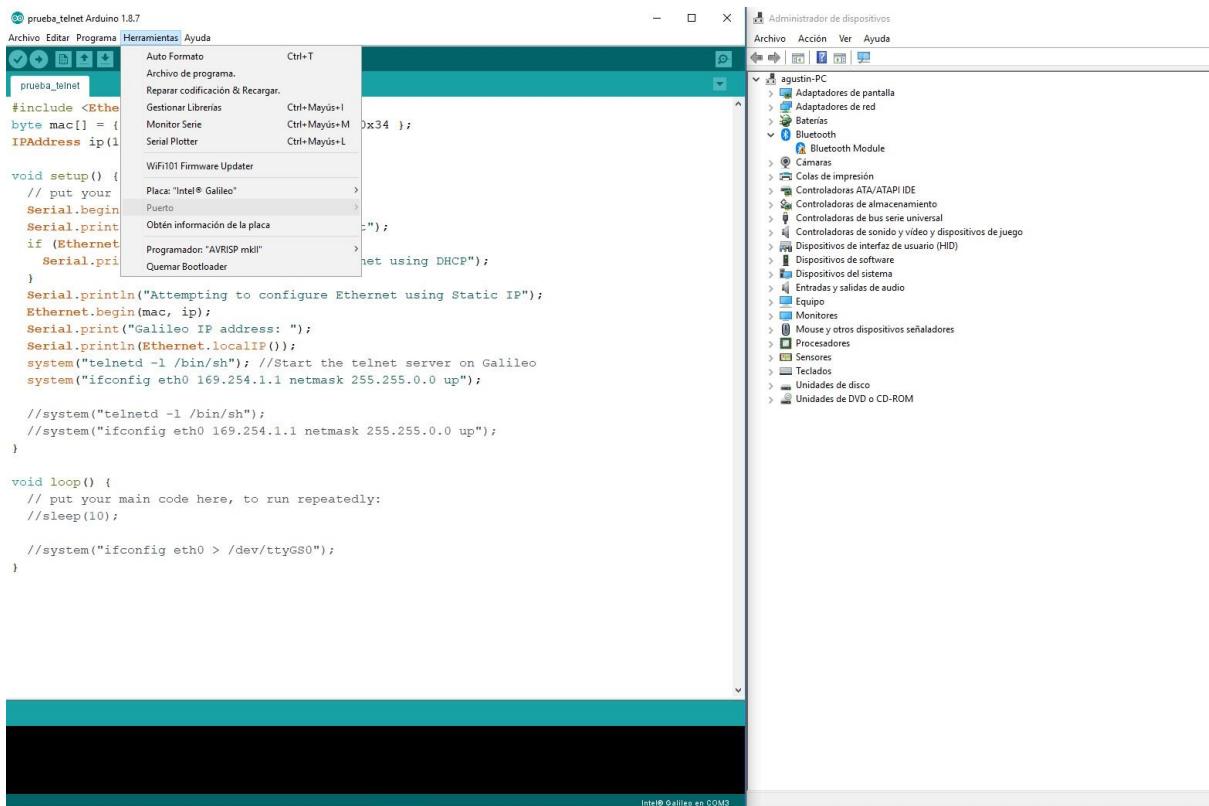
Con la placa alimentada y con la tarjeta microSD conectada y con el usb conectado de la siguiente manera:



El Administrador de Dispositivos nos muestra lo siguiente:



Abriendo el IDE de Arduino con la conexión anteriormente mencionada y el administrador también abierto, vemos que no permite asignar el puerto para conectar la Galileo:



Al intentar subir el sketch, nos muestra el siguiente error (copiado desde la consola de arduino):

"Arduino:1.8.7 (Windows 10), Tarjeta:"Intel® Galileo"

El Sketch usa 58946 bytes (0%) del espacio de almacenamiento de programa. El máximo es 10000000 bytes.

```

#!/bin/sh
starting download script

# clupload script to invoke lsz
# Copyright (C) 2014 Intel Corporation
#
Args           to          shell:
C:\Users\agustin\AppData\Local\Arduino15\packages\Intel\tools\sketchUploader\1.6.2+1.0/x8
6/bin      C:\Users\agustin\AppData\Local\Temp\arduino_build_920640/prueba_telnet.ino.elf
COM3
COM PORT 3
Converted COM Port COM3 to tty port /dev/ttyS2
# This library is free software; you can redistribute it and/or
Sending Command String to move to download if not already in download mode
# modify it under the terms of the GNU Lesser General Public
# License as published by the Free Software Foundation; either
# version 2.1 of the License, or (at your option) any later version.
#
# This library is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
# Lesser General Public License for more details.
#
# You should have received a copy of the GNU Lesser General Public
# License along with this library; if not, write to the Free Software
# Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA
#
echo "starting download script"
echo "Args to shell:" $*

# ARG 1: Path to lsz executable.
# ARG 2: Elf File to download
# ARG 3: COM port to use.
Deleting existing sketch on target

#path contains \ need to change all to /
path_to_exe=$1
fixed_path=${path_to_exe//\\W}

#COM ports are not always setup to be addressed via COM for redirect.
#/dev/ttysx are present. However, COMy -> /dev/ttysx where x = y - 1

com_port_arg=$3
com_port_id=${com_port_arg/COM/}
echo "COM PORT" $com_port_id

```

```

tty_port_id=/dev/ttyS$((com_port_id-1))
echo "Converted COM Port" $com_port_arg "to tty port" $tty_port_id

echo "Sending Command String to move to download if not already in download mode"
echo "~sketch downloadGalileo" > $tty_port_id
C:\Users\agustin\AppData\Local\Arduino15\packages\Intel\tools\sketchUploader\1.6.2+1.0/cl
upload/cluploadGalileo_win.sh: line 42: /dev/ttyS2: No such device or address

#Move the existing sketch on target.
echo "Deleting existing sketch on target"
"$fixed_path/lrz.exe" --escape -c "mv -f /sketch/sketch.elf /sketch/sketch.elf.old" <>
$tty_port_id 1>&0
C:\Users\agustin\AppData\Local\Arduino15\packages\Intel\tools\sketchUploader\1.6.2+1.0/cl
upload/cluploadGalileo_win.sh: line 46: /dev/ttyS2: No such device or address
# Execute the target download command

#Download the file.
host_file_name=$2
"$fixed_path/lrz.exe" --escape --binary --overwrite $host_file_name <> $tty_port_id 1>&0
C:\Users\agustin\AppData\Local\Arduino15\packages\Intel\tools\sketchUploader\1.6.2+1.0/cl
upload/cluploadGalileo_win.sh: line 51: /dev/ttyS2: No such device or address

#mv the downloaded file to /sketch/sketch.elf
Moving downloaded file to /sketch/sketch.elf on target
target_download_name="${host_file_name##*/}"
echo "Moving downloaded file to /sketch/sketch.elf on target"
"$fixed_path/lrz.exe" --escape -c "mv $target_download_name /sketch/sketch.elf; chmod +x
/sketch/sketch.elf" <> $tty_port_id 1>&0
C:\Users\agustin\AppData\Local\Arduino15\packages\Intel\tools\sketchUploader\1.6.2+1.0/cl
upload/cluploadGalileo_win.sh: line 56: /dev/ttyS2: No such device or address
Ha ocurrido un error mientras se enviaba el sketch

```

Se verificó que el error se originaba por estar formateando en FAT16 la tarjeta microSD. El programa SD Memory Card Formatter detecta automáticamente el tamaño de la tarjeta y elige el formato, sin dar opción al usuario y no aclara que tipo de FAT usa, dice 'FAT (predeterminado)'. Se recomienda evitar su uso.

### **A3) Reproduciendo error al intentar conectarse mediante telnet a la Galileo.**

Usando esta [guía](#), encendemos la placa con la sd colocada y luego de un minuto conectamos el cable ethernet y el cable usb. Vemos la configuración de ethernet de la pc:

← Configuración

## ⓘ Red no identificada

### Conexión de uso medido

Si tienes un plan de datos limitado y quieres tener más control sobre el uso de datos, convierte esta conexión en una red de uso medido. Puede que el funcionamiento de algunas aplicaciones cambie para reducir el uso de datos cuando estés conectado a esta red.

Establecer como conexión de uso medido



Si estableces un límite de datos, Windows establecerá el ajuste de conexión de uso medido para que te ayude a no alcanzar el límite.

[Establecer un límite de datos para ayudar a controlar el uso de datos en esta red](#)

### Propiedades

Servidores DNS IPv6:	fec0:0:0:ffff::1%1 fec0:0:0:ffff::2%1 fec0:0:0:ffff::3%1
Fabricante:	Realtek
Descripción:	Realtek PCIe GBE Family Controller
Versión del controlador:	9.1.406.2015
Dirección física (MAC):	F4-6D-04-0B-EE-F5

Copiar

Abriendo una consola PowerShell desde inicio y usando la instrucción ipconfig vemos la ip asignada al puerto ethernet de la PC: 169.254.161.94:

```

Windows PowerShell
Copyright (C) Microsoft Corporation. Todos los derechos reservados.

PS C:\Users\agustin> ipconfig

Configuración IP de Windows

Adaptador de LAN inalámbrica Conexión de área local* 2:
    Estado de los medios. . . . . : medios desconectados
    Sufijo DNS específico para la conexión. . .

Adaptador de LAN inalámbrica Conexión de área local* 3:
    Estado de los medios. . . . . : medios desconectados
    Sufijo DNS específico para la conexión. . .

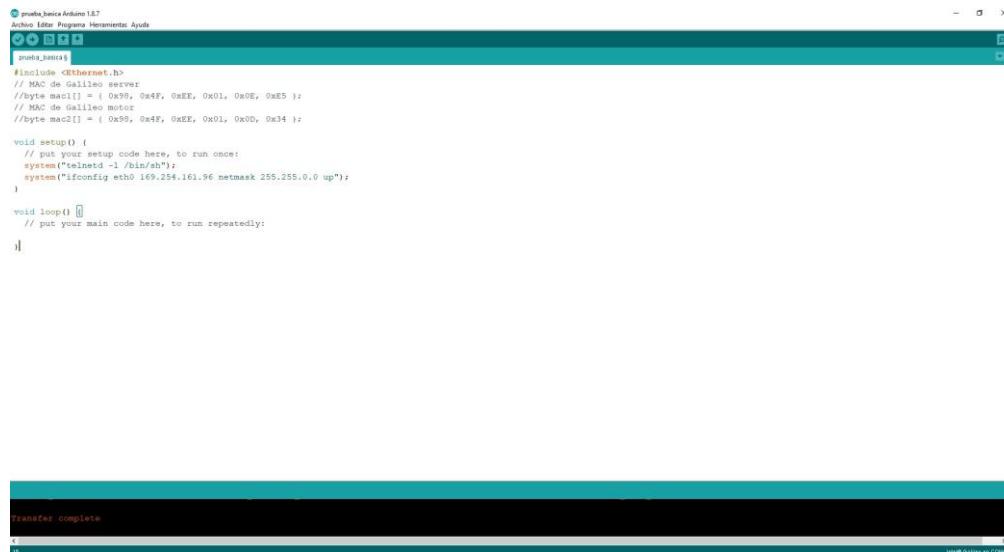
Adaptador de Ethernet Conexión de área local:
    Sufijo DNS específico para la conexión. . . :
        Vínculo: dirección IPv6 local. . . : fe80::d08b:8a8c:688c:a15e%5
        Dirección IPv4 de configuración automática: 169.254.161.94
        Máscara de subred . . . . . : 255.255.0.0
        Puerta de enlace predeterminada . . . . .

Adaptador de LAN inalámbrica Conexión de red inalámbrica:
    Sufijo DNS específico para la conexión. . . : fibertel.com.ar
        Vínculo: dirección IPv6 local. . . : fe80::f1d9:9f41:fb4e:ca12%23
        Dirección IPv4. . . . . : 192.168.0.8
        Máscara de subred . . . . . : 255.255.255.0
        Puerta de enlace predeterminada . . . . : 192.168.0.1

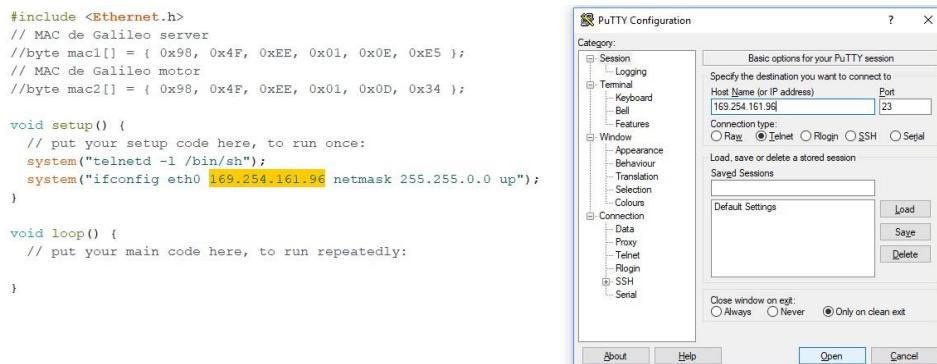
PS C:\Users\agustin>

```

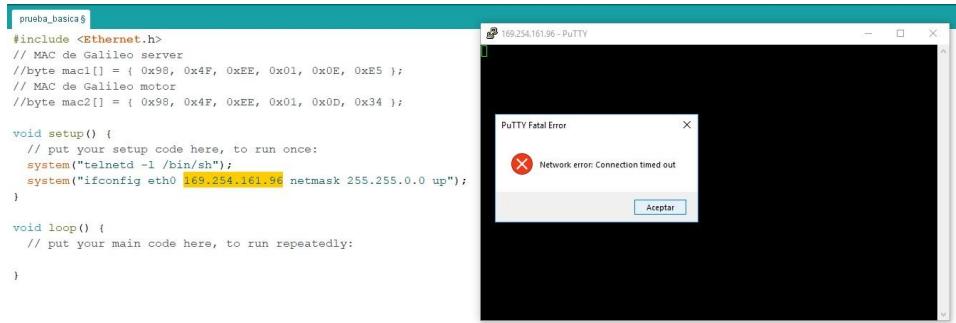
Le asignamos un ip dentro del mismo dominio a la placa Galileo de la siguiente manera (por ejemplo, 169.254.161.96):



Ahora nos conectamos usando Putty mediante Telnet a la ip que le asignamos:

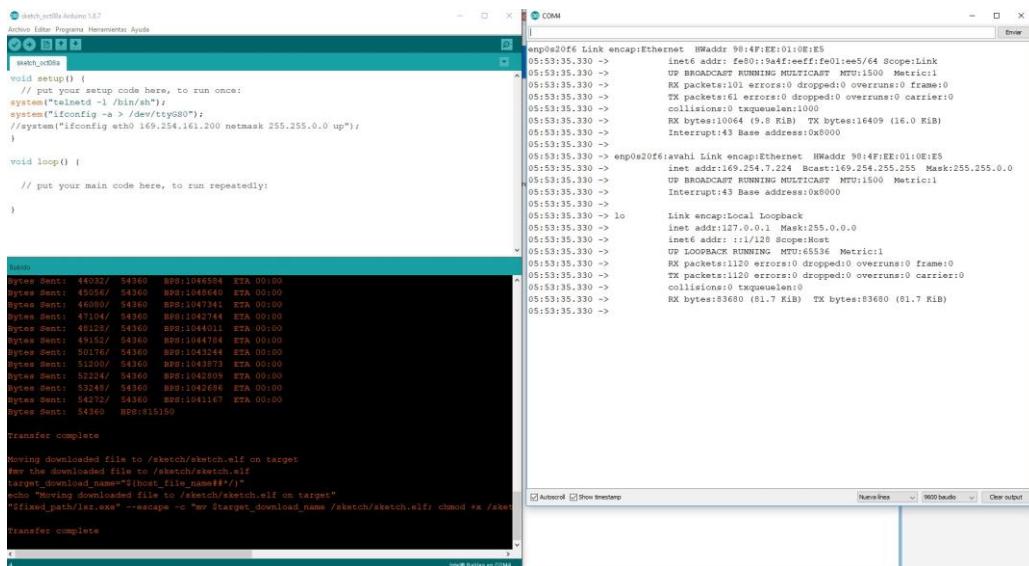


Obtenemos el siguiente error:



Repetimos el procedimiento con el usb desconectado, solamente conectamos el cable de red y obtuvimos el mismo error.

Investigando sobre la interfaz, verificamos que esté activa:



Vemos que no se denomina eth0, navegando encontramos [este post](#) que indica como renombrarlo a eth0:

```

void setup() {
    // put your setup code here, to run once:
    system("telnetd -l /bin/sh");
    system("ifconfig eth0 down");
    system("if link set eth0@0f6 name eth0");
    system("ifconfig eth0 up");
    system("ifconfig eth0 192.168.1.200 netmask 255.255.0.0 up");
}

void loop() {
    // put your main code here, to run repeatedly:
}

```

Bytes Sent: 54372/ 54520 BPS:903650 KTA: 00:00  
Bytes Sent: 54520 BPS:740657

Transfer complete

Moving downloaded file to /sketch/sketch.elf on target  
for the downloaded file to /sketch/sketch.elf  
target\_download\_name="\$host\_file\_name"/>  
echo "Moving downloaded file to /sketch/sketch.elf on target"  
"/fixed\_path/lzr.exe" --escape -c "\$host\_download\_name /sketch/sketch.elf; chmod +x /sketch/sketch.elf"

transfer complete

Probamos con la ip asignada en el paso anterior y por ser la primera vez nos consulta si deseamos confiar en la fuente a la que nos conectaremos, presionamos si:

```

void setup() {
    // put your setup code here, to run once:
    system("telnetd -l /bin/sh");
    system("ifconfig eth0 down");
    system("if link set eth0@0f6 name eth0");
    system("ifconfig eth0 up");
    system("ifconfig eth0 192.168.1.200 netmask 255.255.0.0 up");
}

void loop() {
    // put your main code here:
}

```

Bytes Sent: 54272/ 54520 BPS:903650 KTA: 00:00  
Bytes Sent: 54520 BPS:740657

Transfer complete

Moving downloaded file to /sketch/sketch.elf on target  
for the downloaded file to /sketch/sketch.elf  
target\_download\_name="\$host\_file\_name"/>  
echo "Moving downloaded file to /sketch/sketch.elf on target"  
"/fixed\_path/lzr.exe" --escape -c "\$host\_download\_name /sketch/sketch.elf; chmod +x /sketch/sketch.elf"

transfer complete

Desde la consola escribimos root y presionamos 2 veces enter. Luego listamos el contenido del directorio:

```

sketch_0001a Arduino 1.6.7
Archivo Editar Programa Herramientas Ayuda
sketch_0001a
void setup() {
    // put your setup code here, to run once:
    system("ifnetd -l /bin/sh");
    system("ifconfig enp0s0f6 down");
    system("ip link set enp0s0f6 name eth0");
    system("ifconfig eth0 up");
    system("ifconfig eth0 169.254.161.200 netmask 255.255.0.0 up");
    system("ifconfig > /dev/ttys0");
}

void loop() {
    // put your main code here, to run repeatedly:
}

169.254.161.200:~$ 
root@Galileo:~# ls
root@Galileo:~# 

eth0      Link encap:Ethernet HWaddr 00:0c:29:4f:11:00
          inet brd 169.254.161.200 Bcast:169.254.255.255 Mask:255.255.0.0
          inet6 addr: fe80::0c29:4fffe:11:64 Scope:Link
            UP BROADCAST MULTICAST MTU:1500 Metric:1
            RX packets:89 errors:0 dropped:0 overruns:0 frame:0
            TX packets:122 errors:0 dropped:0 overruns:0 carrier:0
            collisions:0 txqueuelen:1000
            RX bytes:8744 (8.5 Kib) TX bytes:26974 (26.3 Kib)
            Interrupt:43

lo       Link encap:Local Loopback
          inet addr:127.0.0.1 Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
            UP LOOPBACK RUNNING MTU:65536 Metric:1
            RX packets:801 errors:0 dropped:0 overruns:0 frame:0
            TX packets:801 errors:0 dropped:0 overruns:0 carrier:0
            collisions:0 txqueuelen:0
            RX bytes:60676 (59.2 Kib) TX bytes:60676 (59.2 Kib)

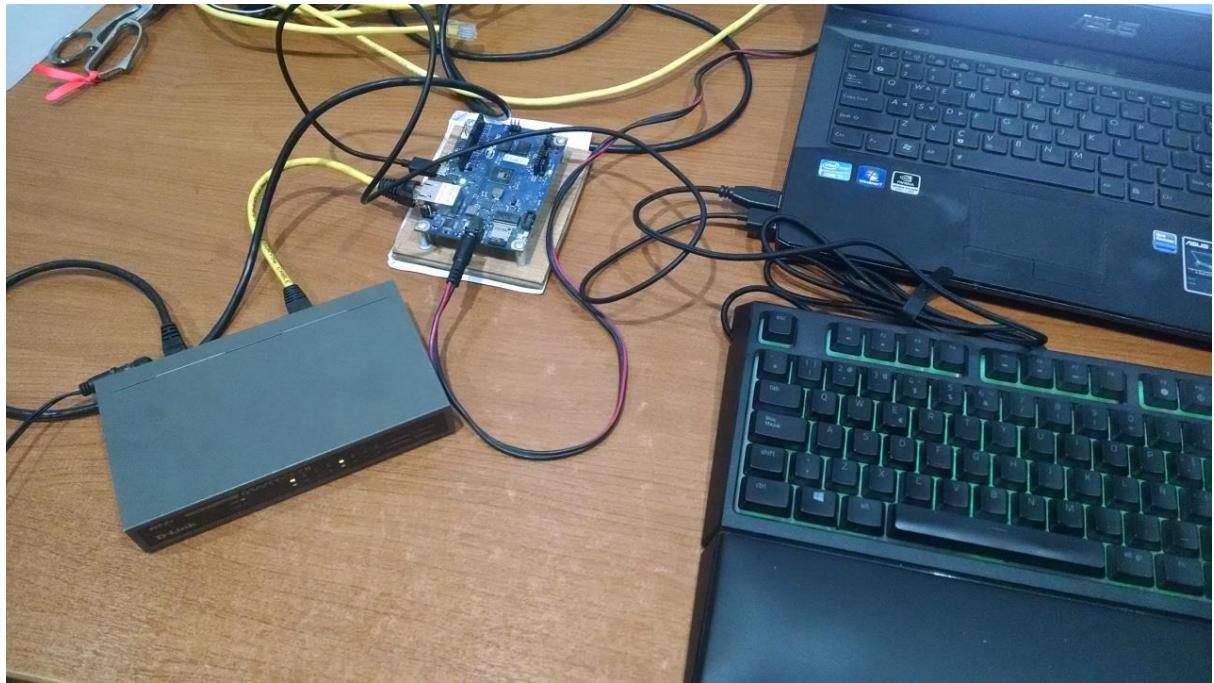
root@Galileo:~# 

```

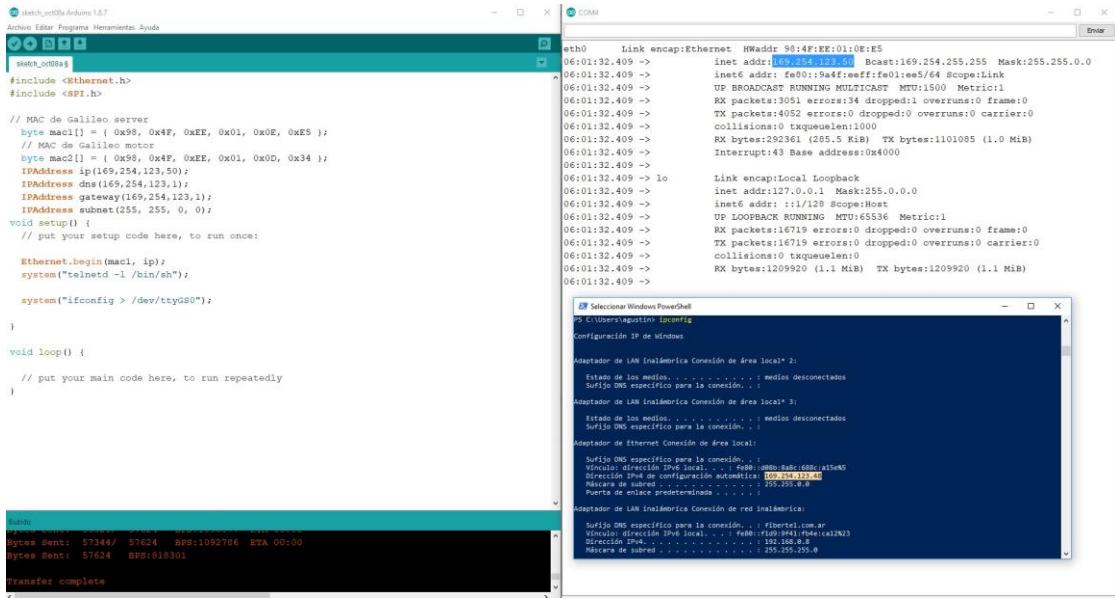
En este ejemplo si bien no se encontró solución al problema de conectarse vía Telnet a la Galileo, se evitó el problema usando SSH en su lugar (además, siempre se aconseja ya que permite una sesión segura).

#### A4) Reproduciendo error intentar conectarse mediante telnet a la Galileo estando conectado mediante el switch.

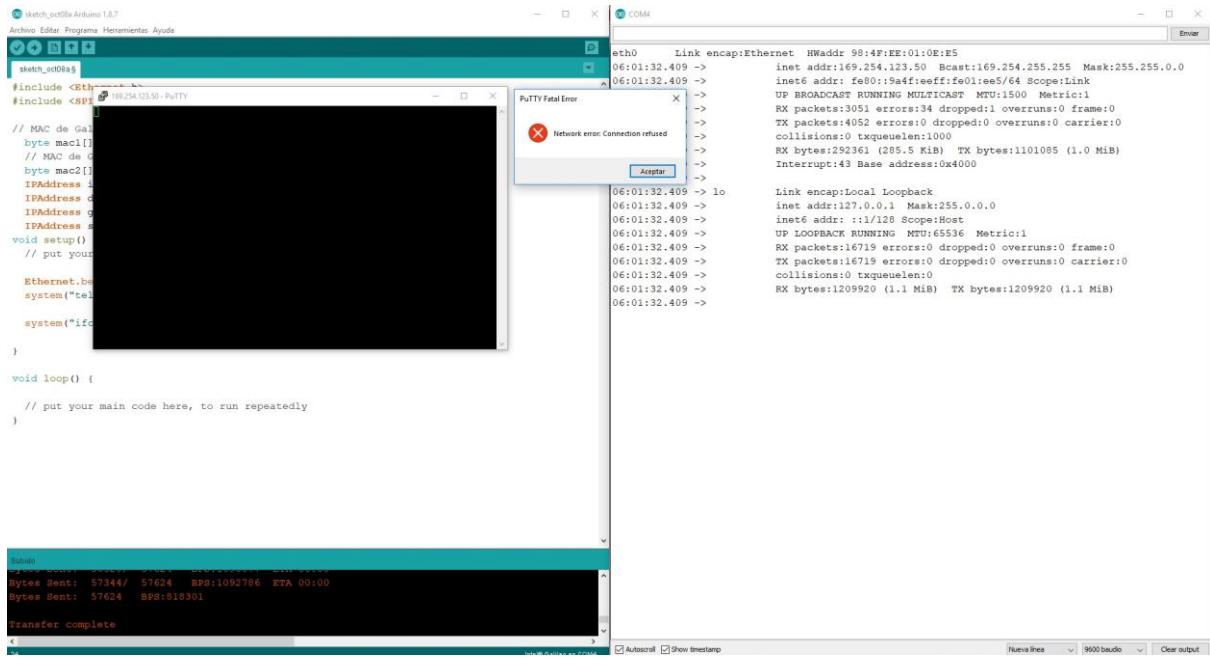
Ahora conectando una Galileo al switch y la PC al switch de la siguiente manera:



Probamos conectividad abriendo una consola powershell en Windows, hacemos ipconfig y vemos que el ip asignado al puerto ethernet es 169.254.123.48, con la máscara 255.255.0.0. Por lo tanto, a la placa le asignamos una ip dentro del dominio, por ejemplo 169.254.123.50 máscara 255.255.0.0 como vemos a continuación:



Probamos conectar por telnet mediante putty a dicha ip y falla como vemos a continuación:



Comprobamos que el uso de la librería Ethernet no es aconsejado ya que muchas funciones no están bien implementadas para la placa Galileo. Además, al igual que el apartado anterior 'A3', se intenta conectarse vía Telnet sin éxito.

## A5) Reproduciendo error de asignación de ip al conectarse mediante ssh a la Galileo estando conectado a ésta mediante el switch.

Teniendo conectadas las galileos y la PC al switch (una de las placas está conectada vía USB a la PC), cuando al bajar el sketch le asigno una ip pero no "queda guardada", toma otra dentro de la misma subred. Pareciera como si hubiera un router haciendo DHCP por lo

que reintente teniendo la placa de wifi deshabilitada, pero sucede lo mismo. Adjunto imágenes para clarificar situación.

Las placas conectadas al router:



Sketch utilizado:

```
init_galileo Arduino 1.8.7
Archivo Editar Programa Herramientas Ayuda
init_galileo

void setup() {
    // put your setup code here, to run once:
    system("telnetd -l /bin/sh"); //Start the telnet server on Galileo
    //renombro el dispositivo ethernet a uno standard
    system("ip link set enp0s20f6 down");
    system("ip link set enp0s20f6 name eth0");
    // configuro eth0 de manera estatica y le asigno ip, submascara y gateway
    system("auto eth0");
    system("iface eth0 inet static");
    system("address 169.254.161.98");
    system("netmask 255.255.0.0");
    system("gateway 169.254.0.1");
    // la activo y muestro las interfaces activas asi como las rutas que conoce la Galileo
    system("ip link set eth0 up");
    system("ip addr show > /dev/ttyGS0");
    system("ip route show > /dev/ttyGS0");
}

void loop() {
    // put your main code here, to run repeatedly:
}
```

El monitor serie muestra lo siguiente:

```

1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue
22:38:59.077 ->      link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
22:38:59.077 ->      inet 127.0.0.1/8 scope host lo
22:38:59.077 ->      inet6 ::1/128 scope host
22:38:59.077 ->          valid_lft forever preferred_lft forever
22:38:59.077 -> 2: eth0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc pfifo_fast qlen 1000
22:38:59.077 ->      link/ether 98:4f:ee:01:0d:34 brd ff:ff:ff:ff:ff:ff
22:38:59.077 ->      inet 169.254.4.103/16 brd 169.254.255.255 scope link eth0
169.254.0.0/16 dev eth0  src 169.254.4.103

```

Problema sin resolver hasta la fecha, se evita el mismo asignando posteriormente la ip que vemos que surge al correr el sketch.

Resuelto por Fernando (indicación del 10/10/2018). Básicamente hay que agregar retardos para que la Galileo asigne correctamente la IP. A continuación, vemos que le asignamos 169.254.161.99 con la submáscara 255.255.0.0 y en el monitor serie vemos efectivamente dicho resultado ("inet 169.254.161.99/16 scope global eth0"). Para confirmar accedemos mediante SSH usando PUTTY:

```

void setup() {
    // put your setup code here, to run once:
    system("telnetd -l /bin/sh");
    delay(1000);

    system("ip link set enp0s20f6 down");

    Serial.println("enps... down > /dev/ttyGS0");

    system("ip link set enp0s20f6 name eth0 > /dev/ttyGS0");
    delay(1000);

    system("ip link set eth0 up");
    delay(1000);

    // eglibc ethernet naming...
    system("ip addr add 169.254.161.99/16 dev eth0 > /dev/ttyGS0");
    delay(1000);

    system("ip addr show > /dev/ttyGS0");
}

void loop() {
    // put your main code here, to run repeatedly:
}


```

1: lo: <LOOPBACK,UP,LOWER\_UP> mtu 65536 qdisc noqueue  
19:05:37.976 -> link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00  
19:05:37.976 -> inet 127.0.0.1/8 scope host lo  
19:05:37.976 -> inet6 ::1/128 scope host  
19:05:37.976 -> valid\_lft forever preferred\_lft forever  
19:05:37.976 -> 2: eth0: <BROADCAST,MULTICAST,UP,LOWER\_UP> mtu 1500 qdisc pfifo\_fast qlen 1000  
19:05:37.976 -> link/ether 98:4f:ee:01:0d:34 brd ff:ff:ff:ff:ff:ff  
19:05:37.976 -> inet 169.254.161.99/16 scope global eth0  
19:05:37.976 -> inet6 fe80::9a4f:efffe01:d34/64 scope link  
19:05:37.976 -> valid\_lft forever preferred\_lft forever

## A5) Problema usando MQTT Mosquitto entre una placa galileo y la PC

Primero corremos mosquitto desde la galileo usando el parametro -d y luego inicializamos en windows desde la PC el servicio que corre mqtt.

Confundimos clientes y *brokers*. Lo resolvimos usando la galileo como *broker* (levantamos un server en ese puerto) y cliente a la vez.

## A6) Problema bajando Mosquitto MQTT a la galileo.

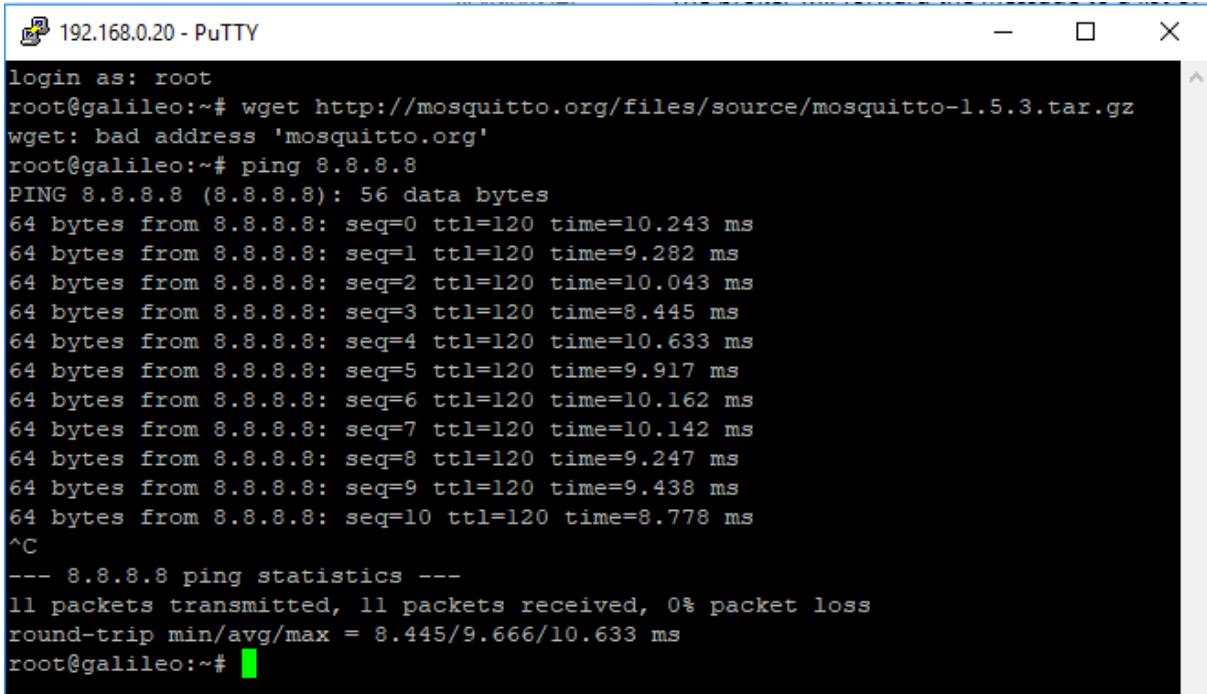
Siguiendo [ésta guia](#), nos encontramos con el primer problema al usar el comando wget:

 169.254.161.99 - PuTTY  
root@galileo:/# wget https://mosquitto.org/files/source/mosquitto-1.5.3.tar.gz  
wget: not an http or ftp url: https://mosquitto.org/files/source/mosquitto-1.5.3.  
.tar.gz  
root@galileo:/#

Buscando una solucion, encontramos el [siguiente post](#) que aclara el problema con los certificados ssl, volvemos a intentar explicitamente aclarando que no verifique los certificados y tampoco funciona:

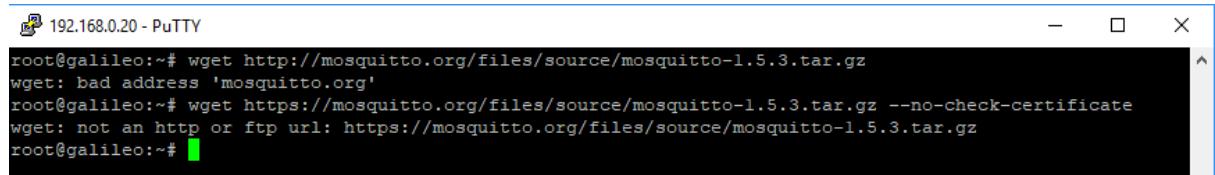
```
169.254.161.99 - PuTTY
root@galileo:/# wget http://mosquitto.org/files/source/mosquitto-1.5.3.tar.gz --no-check-certificate
wget: bad address 'mosquitto.org'
root@galileo:/#
```

Comprobado que no puede acceder mediante internet de la PC, conectamos otro cable al switch de un router propio con acceso a internet y volvemos a verificar (se volvió a asignar ip ya que cambio):



```
login as: root
root@galileo:~# wget http://mosquitto.org/files/source/mosquitto-1.5.3.tar.gz
wget: bad address 'mosquitto.org'
root@galileo:~# ping 8.8.8.8
PING 8.8.8.8 (8.8.8.8): 56 data bytes
64 bytes from 8.8.8.8: seq=0 ttl=120 time=10.243 ms
64 bytes from 8.8.8.8: seq=1 ttl=120 time=9.282 ms
64 bytes from 8.8.8.8: seq=2 ttl=120 time=10.043 ms
64 bytes from 8.8.8.8: seq=3 ttl=120 time=8.445 ms
64 bytes from 8.8.8.8: seq=4 ttl=120 time=10.633 ms
64 bytes from 8.8.8.8: seq=5 ttl=120 time=9.917 ms
64 bytes from 8.8.8.8: seq=6 ttl=120 time=10.162 ms
64 bytes from 8.8.8.8: seq=7 ttl=120 time=10.142 ms
64 bytes from 8.8.8.8: seq=8 ttl=120 time=9.247 ms
64 bytes from 8.8.8.8: seq=9 ttl=120 time=9.438 ms
64 bytes from 8.8.8.8: seq=10 ttl=120 time=8.778 ms
^C
--- 8.8.8.8 ping statistics ---
11 packets transmitted, 11 packets received, 0% packet loss
round-trip min/avg/max = 8.445/9.666/10.633 ms
root@galileo:~#
```

Vemos que pudo resolver las direcciones de manera exitosa por lo que repetimos la descarga usando wget sin exito. Tambien verificamos que no sea problema del certificado:

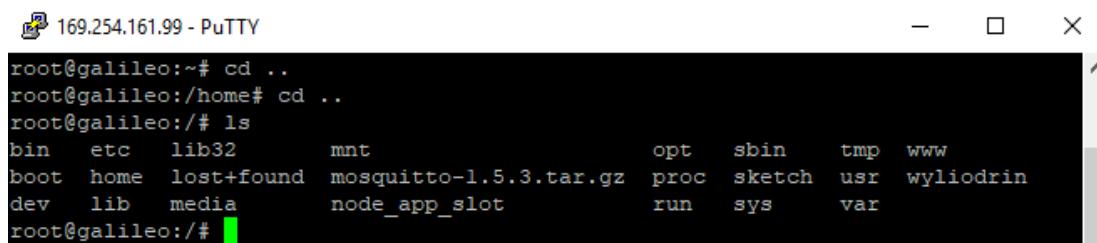


```
root@galileo:~# wget http://mosquitto.org/files/source/mosquitto-1.5.3.tar.gz
wget: bad address 'mosquitto.org'
root@galileo:~# wget https://mosquitto.org/files/source/mosquitto-1.5.3.tar.gz --no-check-certificate
wget: not an http or ftp url: https://mosquitto.org/files/source/mosquitto-1.5.3.tar.gz
root@galileo:~#
```

Evitamos el problema usando WinSCP para pasar los archivos.

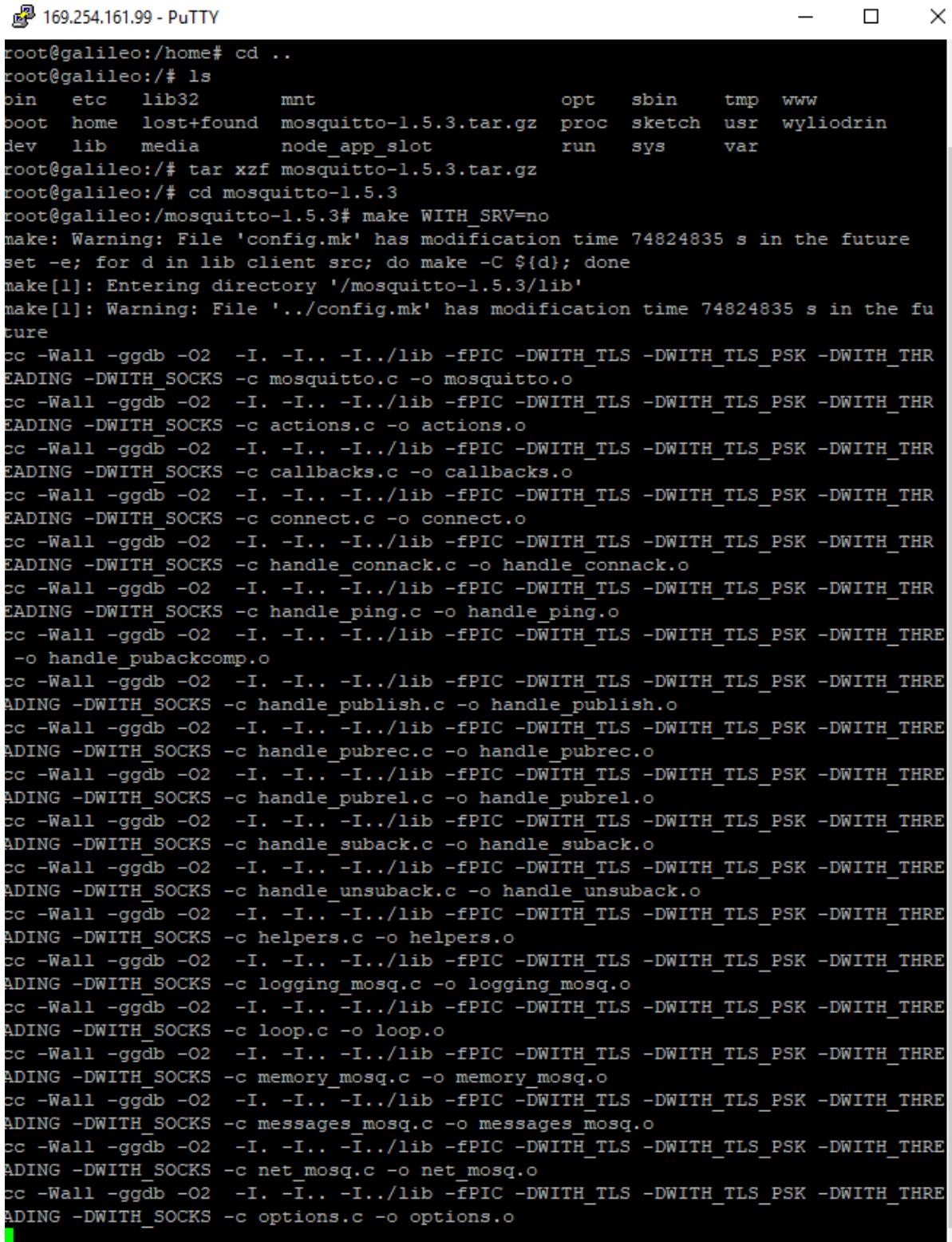
## A6) Problema haciendo el make de Mosquitto, por fechas incorrectas en la Galileo.

Nos conectamos a la placa mediante Putty y nos posicionamos en el directorio raiz:



```
root@galileo:~# cd ..
root@galileo:/home# cd ..
root@galileo:/# ls
bin   etc    lib32      mnt          opt    sbin    tmp    www
boot  home   lost+found  mosquitto-1.5.3.tar.gz  proc   sketch  usr  wylodrin
dev   lib     media      node_app_slot    run    sys     var
root@galileo:/#
```

realizamos la descompresión del archivo, nos posicionamos dentro de la carpeta creada y hacemos "*make WITH\_SRV=no*":



```
169.254.161.99 - PuTTY
root@galileo:/home# cd ..
root@galileo:/# ls
bin  etc  lib32      mnt          opt  sbin  tmp  www
boot  home  lost+found  mosquitto-1.5.3.tar.gz  proc  sketch  usr  wyliodrin
dev  lib   media      node_app_slot        run  sys   var
root@galileo:/# tar xzf mosquitto-1.5.3.tar.gz
root@galileo:/# cd mosquitto-1.5.3
root@galileo:/mosquitto-1.5.3# make WITH_SRV=no
make: Warning: File 'config.mk' has modification time 74824835 s in the future
set -e; for d in lib client src; do make -C ${d}; done
make[1]: Entering directory '/mosquitto-1.5.3/lib'
make[1]: Warning: File '../config.mk' has modification time 74824835 s in the future
cc -Wall -ggdb -O2 -I. -I.. -I../lib -fPIC -DWITH_TLS -DWITH_TLS_PSK -DWITH_THREADS -DWITH_SOCKETS -c mosquitto.c -o mosquitto.o
cc -Wall -ggdb -O2 -I. -I.. -I../lib -fPIC -DWITH_TLS -DWITH_TLS_PSK -DWITH_THREADS -DWITH_SOCKETS -c actions.c -o actions.o
cc -Wall -ggdb -O2 -I. -I.. -I../lib -fPIC -DWITH_TLS -DWITH_TLS_PSK -DWITH_THREADS -DWITH_SOCKETS -c callbacks.c -o callbacks.o
cc -Wall -ggdb -O2 -I. -I.. -I../lib -fPIC -DWITH_TLS -DWITH_TLS_PSK -DWITH_THREADS -DWITH_SOCKETS -c connect.c -o connect.o
cc -Wall -ggdb -O2 -I. -I.. -I../lib -fPIC -DWITH_TLS -DWITH_TLS_PSK -DWITH_THREADS -DWITH_SOCKETS -c handle_connack.c -o handle_connack.o
cc -Wall -ggdb -O2 -I. -I.. -I../lib -fPIC -DWITH_TLS -DWITH_TLS_PSK -DWITH_THREADS -DWITH_SOCKETS -c handle_ping.c -o handle_ping.o
cc -Wall -ggdb -O2 -I. -I.. -I../lib -fPIC -DWITH_TLS -DWITH_TLS_PSK -DWITH_THREADS -o handle_pubackcomp.o
cc -Wall -ggdb -O2 -I. -I.. -I../lib -fPIC -DWITH_TLS -DWITH_TLS_PSK -DWITH_THREADS -DWITH_SOCKETS -c handle_publish.c -o handle_publish.o
cc -Wall -ggdb -O2 -I. -I.. -I../lib -fPIC -DWITH_TLS -DWITH_TLS_PSK -DWITH_THREADS -DWITH_SOCKETS -c handle_pubrec.c -o handle_pubrec.o
cc -Wall -ggdb -O2 -I. -I.. -I../lib -fPIC -DWITH_TLS -DWITH_TLS_PSK -DWITH_THREADS -DWITH_SOCKETS -c handle_pubrel.c -o handle_pubrel.o
cc -Wall -ggdb -O2 -I. -I.. -I../lib -fPIC -DWITH_TLS -DWITH_TLS_PSK -DWITH_THREADS -DWITH_SOCKETS -c handle_suback.c -o handle_suback.o
cc -Wall -ggdb -O2 -I. -I.. -I../lib -fPIC -DWITH_TLS -DWITH_TLS_PSK -DWITH_THREADS -DWITH_SOCKETS -c handle_unsuback.c -o handle_unsuback.o
cc -Wall -ggdb -O2 -I. -I.. -I../lib -fPIC -DWITH_TLS -DWITH_TLS_PSK -DWITH_THREADS -DWITH_SOCKETS -c helpers.c -o helpers.o
cc -Wall -ggdb -O2 -I. -I.. -I../lib -fPIC -DWITH_TLS -DWITH_TLS_PSK -DWITH_THREADS -DWITH_SOCKETS -c logging_mosq.c -o logging_mosq.o
cc -Wall -ggdb -O2 -I. -I.. -I../lib -fPIC -DWITH_TLS -DWITH_TLS_PSK -DWITH_THREADS -DWITH_SOCKETS -c loop.c -o loop.o
cc -Wall -ggdb -O2 -I. -I.. -I../lib -fPIC -DWITH_TLS -DWITH_TLS_PSK -DWITH_THREADS -DWITH_SOCKETS -c memory_mosq.c -o memory_mosq.o
cc -Wall -ggdb -O2 -I. -I.. -I../lib -fPIC -DWITH_TLS -DWITH_TLS_PSK -DWITH_THREADS -DWITH_SOCKETS -c messages_mosq.c -o messages_mosq.o
cc -Wall -ggdb -O2 -I. -I.. -I../lib -fPIC -DWITH_TLS -DWITH_TLS_PSK -DWITH_THREADS -DWITH_SOCKETS -c net_mosq.c -o net_mosq.o
cc -Wall -ggdb -O2 -I. -I.. -I../lib -fPIC -DWITH_TLS -DWITH_TLS_PSK -DWITH_THREADS -DWITH_SOCKETS -c options.c -o options.o
```

Luego de esperar aproximadamente 25 minutos de build, vemos el siguiente resultado en la consola:

```

H_MEMORY_TRACKING -DWITH_SYS_TREE -DWITH_EC -DWITH_EPOLL -Ideps -c send_connack.c -c send_connack.o
cc -Wall -ggdb -O2 -I.. -I../lib -DVERSION=\"1.5.3\" -DWITH_BROKER -DWITH_TLS -DWITH_TLS_PSK -DWITH_UUID -DWITH_BRIDGE -DWITH_PERSISTENCE -DWITH_H_MEMORY_TRACKING -DWITH_SYS_TREE -DWITH_EC -DWITH_EPOLL -Ideps -c ../lib/send_connect.c -o send_connect.o
cc -Wall -ggdb -O2 -I.. -I../lib -DVERSION=\"1.5.3\" -DWITH_BROKER -DWITH_TLS -DWITH_TLS_PSK -DWITH_UUID -DWITH_BRIDGE -DWITH_PERSISTENCE -DWITH_H_MEMORY_TRACKING -DWITH_SYS_TREE -DWITH_EC -DWITH_EPOLL -Ideps -c ../lib/send_disconnect.c -o send_disconnect.o
cc -Wall -ggdb -O2 -I.. -I../lib -DVERSION=\"1.5.3\" -DWITH_BROKER -DWITH_TLS -DWITH_TLS_PSK -DWITH_UUID -DWITH_BRIDGE -DWITH_PERSISTENCE -DWITH_H_MEMORY_TRACKING -DWITH_SYS_TREE -DWITH_EC -DWITH_EPOLL -Ideps -c ../lib/send_message.c -o send_message.o
cc -Wall -ggdb -O2 -I.. -I../lib -DVERSION=\"1.5.3\" -DWITH_BROKER -DWITH_TLS -DWITH_TLS_PSK -DWITH_UUID -DWITH_BRIDGE -DWITH_PERSISTENCE -DWITH_H_MEMORY_TRACKING -DWITH_SYS_TREE -DWITH_EC -DWITH_EPOLL -Ideps -c ../lib/send_publish.c -o send_publish.o
cc -Wall -ggdb -O2 -I.. -I.. -I../lib -DVERSION=\"1.5.3\" -DWITH_BROKER -DWITH_TLS -DWITH_TLS_PSK -DWITH_UUID -DWITH_BRIDGE -DWITH_PERSISTENCE -DWITH_H_MEMORY_TRACKING -DWITH_SYS_TREE -DWITH_EC -DWITH_EPOLL -Ideps -c send_suback.c -o send_suback.o
cc -Wall -ggdb -O2 -I.. -I.. -I../lib -DVERSION=\"1.5.3\" -DWITH_BROKER -DWITH_TLS -DWITH_TLS_PSK -DWITH_UUID -DWITH_BRIDGE -DWITH_PERSISTENCE -DWITH_H_MEMORY_TRACKING -DWITH_SYS_TREE -DWITH_EC -DWITH_EPOLL -Ideps -c ../lib/send_subscribe.c -o send_subscribe.o
cc -Wall -ggdb -O2 -I.. -I.. -I../lib -DVERSION=\"1.5.3\" -DWITH_BROKER -DWITH_TLS -DWITH_TLS_PSK -DWITH_UUID -DWITH_BRIDGE -DWITH_PERSISTENCE -DWITH_H_MEMORY_TRACKING -DWITH_SYS_TREE -DWITH_EC -DWITH_EPOLL -Ideps -c ../lib/send_unsubscribe.c -o send_unsubscribe.o
cc -Wall -ggdb -O2 -I.. -I.. -I../lib -DVERSION=\"1.5.3\" -DWITH_BROKER -DWITH_TLS -DWITH_TLS_PSK -DWITH_UUID -DWITH_BRIDGE -DWITH_PERSISTENCE -DWITH_H_MEMORY_TRACKING -DWITH_SYS_TREE -DWITH_EC -DWITH_EPOLL -Ideps -c service.c -o service.o
cc -Wall -ggdb -O2 -I.. -I.. -I../lib -DVERSION=\"1.5.3\" -DWITH_BROKER -DWITH_TLS -DWITH_TLS_PSK -DWITH_UUID -DWITH_BRIDGE -DWITH_PERSISTENCE -DWITH_H_MEMORY_TRACKING -DWITH_SYS_TREE -DWITH_EC -DWITH_EPOLL -Ideps -c signals.c -o signals.o
cc -Wall -ggdb -O2 -I.. -I.. -I../lib -DVERSION=\"1.5.3\" -DWITH_BROKER -DWITH_TLS -DWITH_TLS_PSK -DWITH_UUID -DWITH_BRIDGE -DWITH_PERSISTENCE -DWITH_H_MEMORY_TRACKING -DWITH_SYS_TREE -DWITH_EC -DWITH_EPOLL -Ideps -c subs.c -o subs.o
cc -Wall -ggdb -O2 -I.. -I.. -I../lib -DVERSION=\"1.5.3\" -DWITH_BROKER -DWITH_TLS -DWITH_TLS_PSK -DWITH_UUID -DWITH_BRIDGE -DWITH_PERSISTENCE -DWITH_H_MEMORY_TRACKING -DWITH_SYS_TREE -DWITH_EC -DWITH_EPOLL -Ideps -c sys_tree.c -o sys_tree.o
cc -Wall -ggdb -O2 -I.. -I.. -I../lib -DVERSION=\"1.5.3\" -DWITH_BROKER -DWITH_TLS -DWITH_TLS_PSK -DWITH_UUID -DWITH_BRIDGE -DWITH_PERSISTENCE -DWITH_H_MEMORY_TRACKING -DWITH_SYS_TREE -DWITH_EC -DWITH_EPOLL -Ideps -c ../lib/time_message.c -o time_message.o
cc -Wall -ggdb -O2 -I.. -I.. -I../lib -DVERSION=\"1.5.3\" -DWITH_BROKER -DWITH_TLS -DWITH_TLS_PSK -DWITH_UUID -DWITH_BRIDGE -DWITH_PERSISTENCE -DWITH_H_MEMORY_TRACKING -DWITH_SYS_TREE -DWITH_EC -DWITH_EPOLL -Ideps -c ../lib/tls_message.c -o tls_message.o
cc -Wall -ggdb -O2 -I.. -I.. -I../lib -DVERSION=\"1.5.3\" -DWITH_BROKER -DWITH_TLS -DWITH_TLS_PSK -DWITH_UUID -DWITH_BRIDGE -DWITH_PERSISTENCE -DWITH_H_MEMORY_TRACKING -DWITH_SYS_TREE -DWITH_EC -DWITH_EPOLL -Ideps -c ../lib/utf8_message.c -o utf8_message.o
cc -Wall -ggdb -O2 -I.. -I.. -I../lib -DVERSION=\"1.5.3\" -DWITH_BROKER -DWITH_TLS -DWITH_TLS_PSK -DWITH_UUID -DWITH_BRIDGE -DWITH_PERSISTENCE -DWITH_H_MEMORY_TRACKING -DWITH_SYS_TREE -DWITH_EC -DWITH_EPOLL -Ideps -c ../lib/util_message.c -o util_message.o
cc -Wall -ggdb -O2 -I.. -I.. -I../lib -DVERSION=\"1.5.3\" -DWITH_BROKER -DWITH_TLS -DWITH_TLS_PSK -DWITH_UUID -DWITH_BRIDGE -DWITH_PERSISTENCE -DWITH_H_MEMORY_TRACKING -DWITH_SYS_TREE -DWITH_EC -DWITH_EPOLL -Ideps -c websockets.c -o websockets.o
cc -Wall -ggdb -O2 -I.. -I.. -I../lib -DVERSION=\"1.5.3\" -DWITH_BROKER -DWITH_TLS -DWITH_TLS_PSK -DWITH_UUID -DWITH_BRIDGE -DWITH_PERSISTENCE -DWITH_H_MEMORY_TRACKING -DWITH_SYS_TREE -DWITH_EC -DWITH_EPOLL -Ideps -c ../lib/will_message.c -o will_message.o
cc mosquito.o bridge.conf.o config_includedir.o context.o database.o handle_connack.o handle_connect.o handle_ping.o handle_pubackcomp.o handle_publis
h.o handle_pubrec.o handle_pubrel.o handle_suback.o handle_subscribe.o handle_unsubscribe.o handle_unsubscribe.o logging.o loop.o memory_message.o net.o net_
mosque.o packet_message.o persist.o plugin.o read_handle.o security.o security_default.o send_connack.o send_connect.o send_disconnect.o send_message.o send_pu
blish.o send_suback.o send_unsubscribe.o service.o signals.o subs.o sys_tree.o time_message.o tls_message.o utf8_message.o util_message.o websocket
ts.o will_message.o -ldl -lpthread -Wl,--dynamic-list linker.symbs -lssl -lcrypto -luuid
cc -I.. -IWall -ggdb -O2 -o mosquito_passwd.o -o mosquito_passwd -lcrypto
cc mosquito_passwd.o -o mosquito_passwd -lcrypto
make[1]: warning: Clock skew detected. Your build may be incomplete.
make[1]: Leaving directory '/mosquitto-1.5.3/src'
set -e; for d in man; do make -C $d; done
make[1]: Entering directory '/mosquitto-1.5.3/man'
make[1]: Warning: File '../config.mk' has modification time 74823525 s in the future
make[1]: Nothing to be done for 'all'.
make[1]: warning: Clock skew detected. Your build may be incomplete.
make[1]: Leaving directory '/mosquitto-1.5.3/man'
make[1]: warning: Clock skew detected. Your build may be incomplete.
make: warning: Clock skew detected. Your build may be incomplete.
root@galileo:/mosquitto-1.5.3#

```

Verificamos la fecha que nos muestra el sistema usando el comando date (problema que surge al ver las capturas anteriores):

```

root@galileo:/# date
Thu May 12 14:54:16 UTC 2016
root@galileo:/

```

Actualizamos la fecha del sistema a la correcta (dia de la fecha 29/10/2018):

```

root@galileo:/# date -s "29 OCT 2018 22:00:00"
Mon Oct 29 22:00:00 UTC 2018

```

También actualizamos el RTC a la fecha que acabamos de configurar:

```

root@galileo:/# hwclock --systohc
root@galileo:/# hwclock
Mon Oct 29 22:07:31 2018 0.000000 seconds
root@galileo:/

```

Repetimos el make y verificamos que no informa del error en la fecha:

```

root@galileo:/mosquitto-1.5.3# make WITH_SRV=no
set -e; for d in lib client src; do make -C ${d}; done
make[1]: Entering directory '/mosquitto-1.5.3/lib'
cc -Wall -ggdb -O2 -I. -I.. -I... -I.../lib -fPIC -DWITH_TLS -DWITH_TLS_PSK -DWITH_THREADING -DWITH_SOCKETS -c mosquitto.c -o mosquitto.o
cc -Wall -ggdb -O2 -I. -I.. -I... -I.../lib -fPIC -DWITH_TLS -DWITH_TLS_PSK -DWITH_THREADING -DWITH_SOCKETS -c actions.c -o actions.o
cc -Wall -ggdb -O2 -I. -I.. -I... -I.../lib -fPIC -DWITH_TLS -DWITH_TLS_PSK -DWITH_THREADING -DWITH_SOCKETS -c callbacks.c -o callbacks.o
cc -Wall -ggdb -O2 -I. -I.. -I... -I.../lib -fPIC -DWITH_TLS -DWITH_TLS_PSK -DWITH_THREADING -DWITH_SOCKETS -c connect.c -o connect.o
cc -Wall -ggdb -O2 -I. -I.. -I... -I.../lib -fPIC -DWITH_TLS -DWITH_TLS_PSK -DWITH_THREADING -DWITH_SOCKETS -c handle_connack.c -o handle_connack.o
cc -Wall -ggdb -O2 -I. -I.. -I... -I.../lib -fPIC -DWITH_TLS -DWITH_TLS_PSK -DWITH_THREADING -DWITH_SOCKETS -c handle_ping.c -o handle_ping.o
cc -Wall -ggdb -O2 -I. -I.. -I... -I.../lib -fPIC -DWITH_TLS -DWITH_TLS_PSK -DWITH_THREADING -DWITH_SOCKETS -c handle_pubackcomp.c -o handle_pubackcomp.o
cc -Wall -ggdb -O2 -I. -I.. -I... -I.../lib -fPIC -DWITH_TLS -DWITH_TLS_PSK -DWITH_THREADING -DWITH_SOCKETS -c handle_publish.c -o handle_publish.o
cc -Wall -ggdb -O2 -I. -I.. -I... -I.../lib -fPIC -DWITH_TLS -DWITH_TLS_PSK -DWITH_THREADING -DWITH_SOCKETS -c handle_pubrec.c -o handle_pubrec.o
cc -Wall -ggdb -O2 -I. -I.. -I... -I.../lib -fPIC -DWITH_TLS -DWITH_TLS_PSK -DWITH_THREADING -DWITH_SOCKETS -c handle_pubrel.c -o handle_pubrel.o
cc -Wall -ggdb -O2 -I. -I.. -I... -I.../lib -fPIC -DWITH_TLS -DWITH_TLS_PSK -DWITH_THREADING -DWITH_SOCKETS -c handle_suback.c -o handle_suback.o
cc -Wall -ggdb -O2 -I. -I.. -I... -I.../lib -fPIC -DWITH_TLS -DWITH_TLS_PSK -DWITH_THREADING -DWITH_SOCKETS -c handle_unsuback.c -o handle_unsuback.o
cc -Wall -ggdb -O2 -I. -I.. -I... -I.../lib -fPIC -DWITH_TLS -DWITH_TLS_PSK -DWITH_THREADING -DWITH_SOCKETS -c helpers.c -o helpers.o
cc -Wall -ggdb -O2 -I. -I.. -I... -I.../lib -fPIC -DWITH_TLS -DWITH_TLS_PSK -DWITH_THREADING -DWITH_SOCKETS -c logging_mosq.c -o logging_mosq.o
cc -Wall -ggdb -O2 -I. -I.. -I... -I.../lib -fPIC -DWITH_TLS -DWITH_TLS_PSK -DWITH_THREADING -DWITH_SOCKETS -c loop.c -o loop.o
cc -Wall -ggdb -O2 -I. -I.. -I... -I.../lib -fPIC -DWITH_TLS -DWITH_TLS_PSK -DWITH_THREADING -DWITH_SOCKETS -c memory_mosq.c -o memory_mosq.o
cc -Wall -ggdb -O2 -I. -I.. -I... -I.../lib -fPIC -DWITH_TLS -DWITH_TLS_PSK -DWITH_THREADING -DWITH_SOCKETS -c messages_mosq.c -o messages_mosq.o
cc -Wall -ggdb -O2 -I. -I.. -I... -I.../lib -fPIC -DWITH_TLS -DWITH_TLS_PSK -DWITH_THREADING -DWITH_SOCKETS -c net_mosq.c -o net_mosq.o
cc -Wall -ggdb -O2 -I. -I.. -I... -I.../lib -fPIC -DWITH_TLS -DWITH_TLS_PSK -DWITH_THREADING -DWITH_SOCKETS -c options.c -o options.o
cc -Wall -ggdb -O2 -I. -I.. -I... -I.../lib -fPIC -DWITH_TLS -DWITH_TLS_PSK -DWITH_THREADING -DWITH_SOCKETS -c packet_mosq.c -o packet_mosq.o
cc -Wall -ggdb -O2 -I. -I.. -I... -I.../lib -fPIC -DWITH_TLS -DWITH_TLS_PSK -DWITH_THREADING -DWITH_SOCKETS -c read_handle.c -o read_handle.o
cc -Wall -ggdb -O2 -I. -I.. -I... -I.../lib -fPIC -DWITH_TLS -DWITH_TLS_PSK -DWITH_THREADING -DWITH_SOCKETS -c send_connect.c -o send_connect.o
cc -Wall -ggdb -O2 -I. -I.. -I... -I.../lib -fPIC -DWITH_TLS -DWITH_TLS_PSK -DWITH_THREADING -DWITH_SOCKETS -c send_disconnect.c -o send_disconnect.o
cc -Wall -ggdb -O2 -I. -I.. -I... -I.../lib -fPIC -DWITH_TLS -DWITH_TLS_PSK -DWITH_THREADING -DWITH_SOCKETS -c send_mosq.c -o send_mosq.o
cc -Wall -ggdb -O2 -I. -I.. -I... -I.../lib -fPIC -DWITH_TLS -DWITH_TLS_PSK -DWITH_THREADING -DWITH_SOCKETS -c send_publish.c -o send_publish.o
cc -Wall -ggdb -O2 -I. -I.. -I... -I.../lib -fPIC -DWITH_TLS -DWITH_TLS_PSK -DWITH_THREADING -DWITH_SOCKETS -c send_subscribe.c -o send_subscribe.o

```

## A7) Instalando Eclipse Paho (cliente mqtt usando python) con la Galileo conectada directamente a un router con acceso a internet.

Nos volvemos a conectar via putty y realizamos la instalación usando pip install paho-mqtt como nos indica la [guía](#):

```

root@galileo:/# pip install paho-mqtt
Requirement already satisfied (use --upgrade to upgrade): paho-mqtt in /usr/lib/python2.7/site-packages/paho_mqtt-1.4.0.dev0-py2.7.egg
Cleaning up...

```

## A8) Prueba realizada entre la PC y una Galileo sobre el cliente Paho-mqtt:

Si bien no se necesita hacer este paso para el proyecto en cuestión, sirve para poder realizar pruebas sobre el protocolo más fácilmente (en el proyecto la pc sólo es usada para acceder a la página web).

Primero descargamos el instalador de windows de mosquitto [aquí](#).

Luego instalamos el correspondiente [cygwin](#)

Luego descargamos el programa openssl [aquí](#)

Luego descargamos pthreadVC2.dll desde [aquí](#)

Ejecutamos el instalador de windows de mosquitto, cuando finaliza instalamos openssl eligiendo la opcion de instalacion dentro del directorio '/bin'.

Al finalizar copiamos el contenido de dicha carpeta en el directorio donde se instaló mosquitto ('C:\program files' en nuestro caso).

Luego copiamos los archivos que faltan en la instalacion, ya que mosquitto para windows no instala las dependencias. Para ello usamos la siguiente [guia](#).

Los archivos a copiar dentro de la carpeta mosquito son:

```
cygcrypto-1.0.0.dll  
cyggcc_s-1.dll  
cygssl-1.0.0.dll  
cygwin1.dll  
cygz.dll  
libeay32.dll  
libssl32.dll  
msvcr100.dll  
ssleay32.dll
```

Finalmente validamos que el servicio broker de mosquito esté funcionando mediante el comando `netstat -a`. Vemos que está escuchando el puerto 1883.

```
:  
  
Windows PowerShell  
Copyright (C) Microsoft Corporation. Todos los derechos reservados.  
  
PS C:\Users\agustin> netstat -a  
  
Conexiones activas  
  
 Proto  Dirección local        Dirección remota      Estado  
 TCP    0.0.0.0:135           agustin-PC:0        LISTENING  
 TCP    0.0.0.0:445           agustin-PC:0        LISTENING  
 TCP    0.0.0.0:1801          agustin-PC:0        LISTENING  
 TCP    0.0.0.0:1883          agustin-PC:0        LISTENING  
 TCP    0.0.0.0:2103          agustin-PC:0        LISTENING  
 TCP    0.0.0.0:2105          agustin-PC:0        LISTENING  
 TCP    0.0.0.0:2107          agustin-PC:0        LISTENING  
 TCP    0.0.0.0:5040          agustin-PC:0        LISTENING  
 TCP    0.0.0.0:12000          agustin-PC:0        LISTENING
```

Para verificar el correcto funcionamiento realizamos un test usando un suscriptor y un publicador

Mosquitto que publica:

```
PS C:\Program Files\mosquitto> .\mosquitto_pub.exe -d -t /test -m prueba1  
Client mosqpub|9060-agustin-PC sending CONNECT  
Client mosqpub|9060-agustin-PC received CONNACK (0)  
Client mosqpub|9060-agustin-PC sending PUBLISH (d0, q0, r0, m1, '/test', ... (7 bytes))  
Client mosqpub|9060-agustin-PC sending DISCONNECT  
PS C:\Program Files\mosquitto>
```

Mosquitto que suscribe:

```

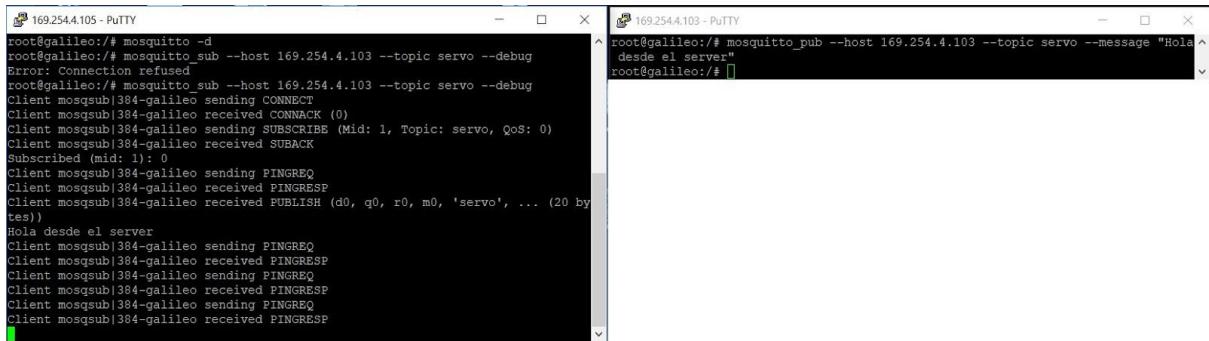
PS C:\Program Files\mosquitto> .\mosquitto_sub.exe -d -t /test
Client mosqsub|11516-agustin-P sending CONNECT
Client mosqsub|11516-agustin-P received CONNACK (0)
Client mosqsub|11516-agustin-P sending SUBSCRIBE (Mid: 1, Topic: /test, QoS: 0)
Client mosqsub|11516-agustin-P received SUBACK
Subscribed (mid: 1): 0
Client mosqsub|11516-agustin-P sending PINGREQ
Client mosqsub|11516-agustin-P received PINGRESP
Client mosqsub|11516-agustin-P received PUBLISH (d0, q0, r0, m0, '/test', ... (7 bytes))
prueba1
Client mosqsub|11516-agustin-P sending PINGREQ
Client mosqsub|11516-agustin-P received PINGRESP
Client mosqsub|11516-agustin-P sending PINGREQ
Client mosqsub|11516-agustin-P received PINGRESP
Client mosqsub|11516-agustin-P sending PINGREQ
Client mosqsub|11516-agustin-P received PINGRESP

```

Queda comprobado el correcto funcionamiento de MQTT Mosquitto en la PC.

### A9) Prueba de la comunicacion entre las Galileo usando MQTT via dos instancias de Putty:

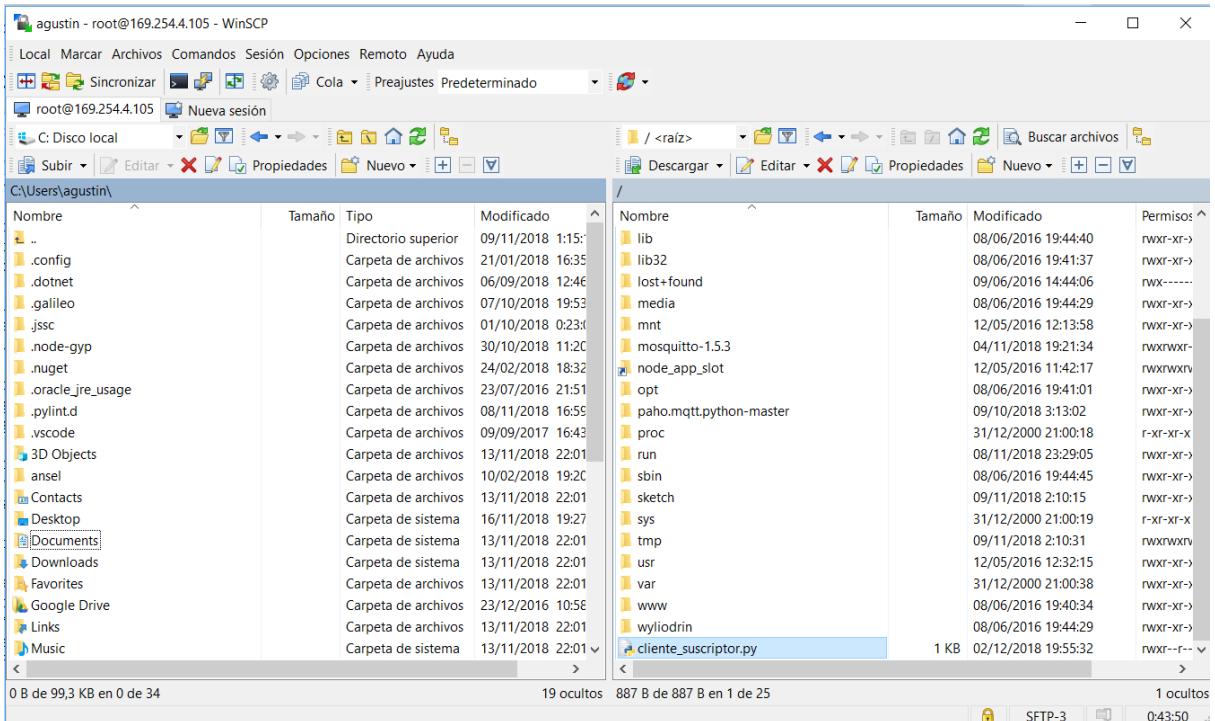
Para ello usamos dos instancias de Putty a cada una de la siguiente manera.  
 Primero levantamos mosquitto con el parámetro -d para que quede corriendo el demonio y luego desde cada una ejecutamos los comandos *mosquitto\_sub* y *mosquitto\_pub* (en la imagen se ve el primer intento de suscripción fallido debido a que no había hecho *mosquitto -d* previamente desde una de las Galileo):



The image shows two PuTTY windows side-by-side. The left window, titled '169.254.4.105 - PuTTY', shows a command-line session where a client (IP 169.254.4.103) is publishing a message to the 'servo' topic. The right window, titled '169.254.4.103 - PuTTY', shows another command-line session where a client (IP 169.254.4.105) is receiving the published message and printing it to the screen. Both clients are using the 'mosquitto\_pub' command with the --host parameter to connect to the other board's IP address.

### A10) Prueba de la comunicacion entre las Galileo usando Paho-mqtt via dos instancias de Putty corriendo un script:

Primero abrimos el winSCP y copiamos el script llamado cliente\_suscriptor.py para demostrar el funcionamiento de dicha librería:



El script básicamente se queda a la escucha de mensajes suscrito al canal “servo” (adjunto se encuentra al final en el anexo). Cuando recibe una publicación imprime el nombre del canal junto con el mensaje recibido.

Para esta primera prueba tenemos dos consolas putty, una a la Galileo actuadora (ip 169.254.4.105) y otra a en la Galileo server (ip 169.254.4.103). Para facilitar aún más la prueba, el script de Python lo lanzamos desde la misma terminal y facilitamos la verificación del mismo:

```

root@galileo:/# python cliente_suscriptor.py
Traceback (most recent call last):
  File "cliente_suscriptor.py", line 40, in <module>
    client.connect("169.254.4.103", 1883, 60)
  File "/usr/lib/python2.7/site-packages/paho_mqtt-1.4.0.dev0-py2.7.egg/paho/mqt
t/client.py", line 839, in connect
    return self.reconnect()
  File "/usr/lib/python2.7/site-packages/paho_mqtt-1.4.0.dev0-py2.7.egg/paho/mqt
t/client.py", line 962, in reconnect
    sock = socket.create_connection((self._host, self._port), source_address=(se
lf._bind_address, 0))
  File "/usr/lib/python2.7/socket.py", line 571, in create_connection
    raise err
socket.error: [Errno 111] Connection refused
root@galileo:/# python cliente_suscriptor.py
Conectado satisfactoriamente.
topico: servo - LED_ON
recibimos LED_ON
  
```

```

root@galileo:/# ls
bin          home      mnt          proc      tmp
boot        lib       mosquito-1.5.3   run      usr
cliente_suscriptor.py lib32     mosquito-1.5.3.tar.gz sbin      var
dev          lost+found node_app_slot sketch   www
etc          media      opt           sys      wylodrin
root@galileo:/# mosquitto -d
root@galileo:/# mosquitto_pub --host 169.254.4.103 --topic servo --message "LED_
ON"
root@galileo:/#
  
```

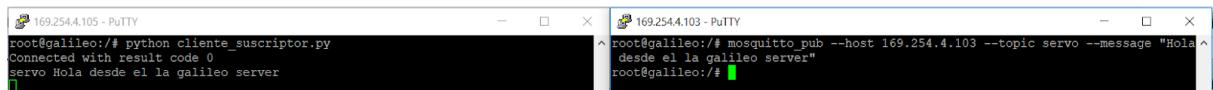
Aclaración: a la hora de correr el script, como estamos suscribiéndonos a un topico y una ip, dicha ip debe tener abierto el puerto 1883 en ese momento. El primer intento es fallido debido a que faltaba inicializar el demonio mosquitto en la Server. Por ello en el segundo intento se ve el correcto funcionamiento. Al recibir el mensaje se ejecuta un método avisando el éxito de la conexión y luego imprimo el tópico seguido del mensaje a recibir. A continuación, vemos el fragmento que se encarga de hacer dicha verificación del mensaje:

```

12  # The callback for when a PUBLISH message is received from the server.
13 def on_message(client, userdata, msg):
14     mensaje_recibido = str(msg.payload)
15     topico_recibido = str(msg.topic)
16     print("topico: " +topico_recibido +" - "+ mensaje_recibido)
17     if topico_recibido == 'servo':
18         if mensaje_recibido == 'LED_ON':
19             print("recibimos LED_ON")
20             # Creo un archivo llamado led_on.txt
21             # y desde arduino enciendo dicho led
22             f = open('led_on.txt', 'w')
23             f.write('led_on')
24             f.close()
25         if mensaje_recibido == 'LED_OFF':
26             print("recibimos LED_OFF")
27             # Creo un archivo llamado led_off.txt
28             # y desde arduino apago dicho led
29             f = open('led_off.txt', 'w')
30             f.write('led_off')
31             f.close()
32

```

A continuación, ejecutamos desde la consola dicho script y vemos el resultado al publicar desde la Galileo server:



Recordar que la Galileo con ip 169.254.4.103 es la webserver mientras que la 169.254.4.105 es la actuadora.

### A11) Falla intentando usar mosquitto en la Galileo Server (no había usado aun mosquitto, solo levantaba el server http):

Al intentar usar algunos de los comandos mosquitto\_sub y mosquitto\_pub en la placa que hace de server fallaba, no reconoce el comando mosquitto para inicializarlo.

Realizar lo siguiente:

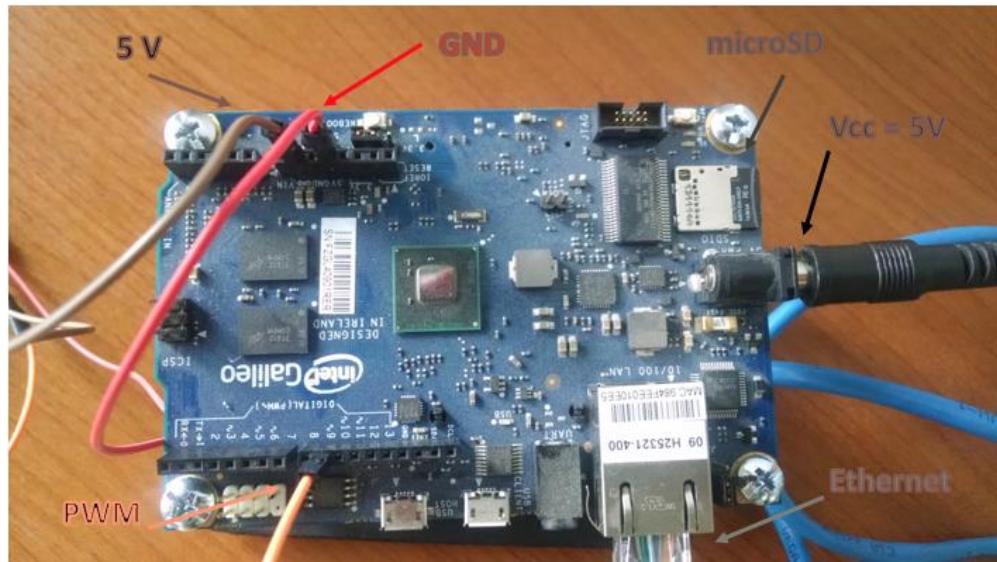
```

$> cp client/mosquitto_pub /usr/bin
$> cp client/mosquitto_sub /usr/bin
$> cp lib/libmosquitto.so.1 /usr/lib
$> cp src/mosquitto /usr/bin

```

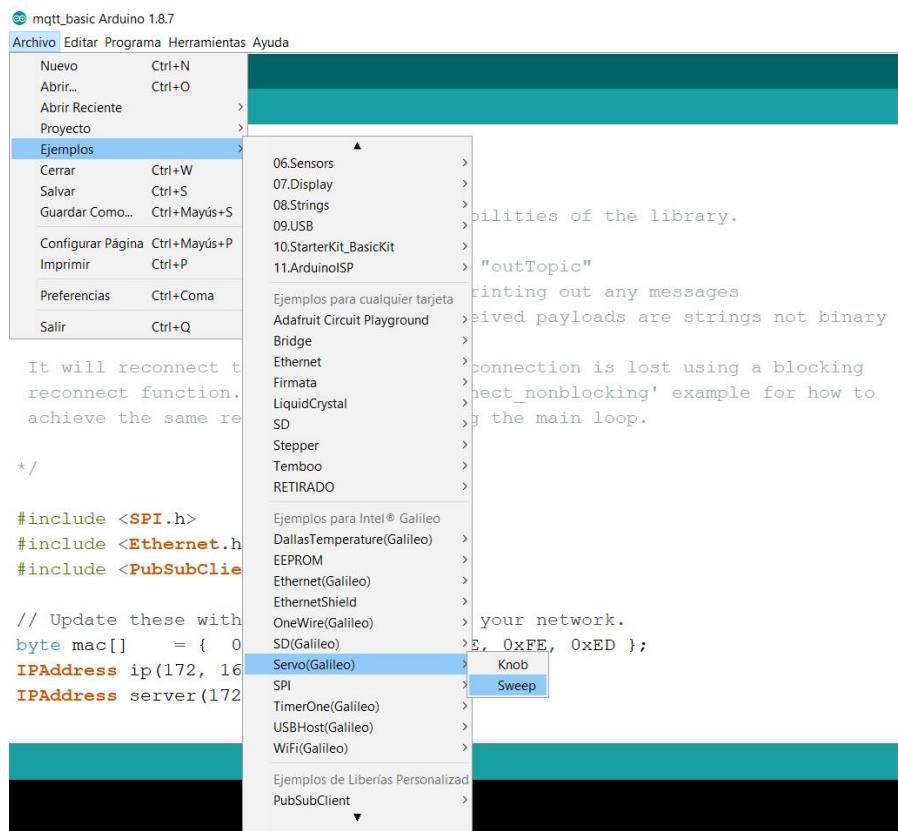
## A12) Prueba realizada con el servo-motor y una sola Galileo usando el ejemplo que trae el IDE de Arduino.

Primero observamos el conexionado siguiente del servo con la placa:



-Servo conectada al pin 9 para recibir la señal PWM, a Vin (5V) y GND de la Galileo

Teniendo todo conectado abrimos el IDE de Arduino y cargamos el ejemplo Sweep de la librería para Galileo:



Ejecutamos el ejemplo y vemos que el motor rota  $180^{\circ}$  aproximadamente y vuelve, constantemente.