Laboratorio-4

December 3, 2018

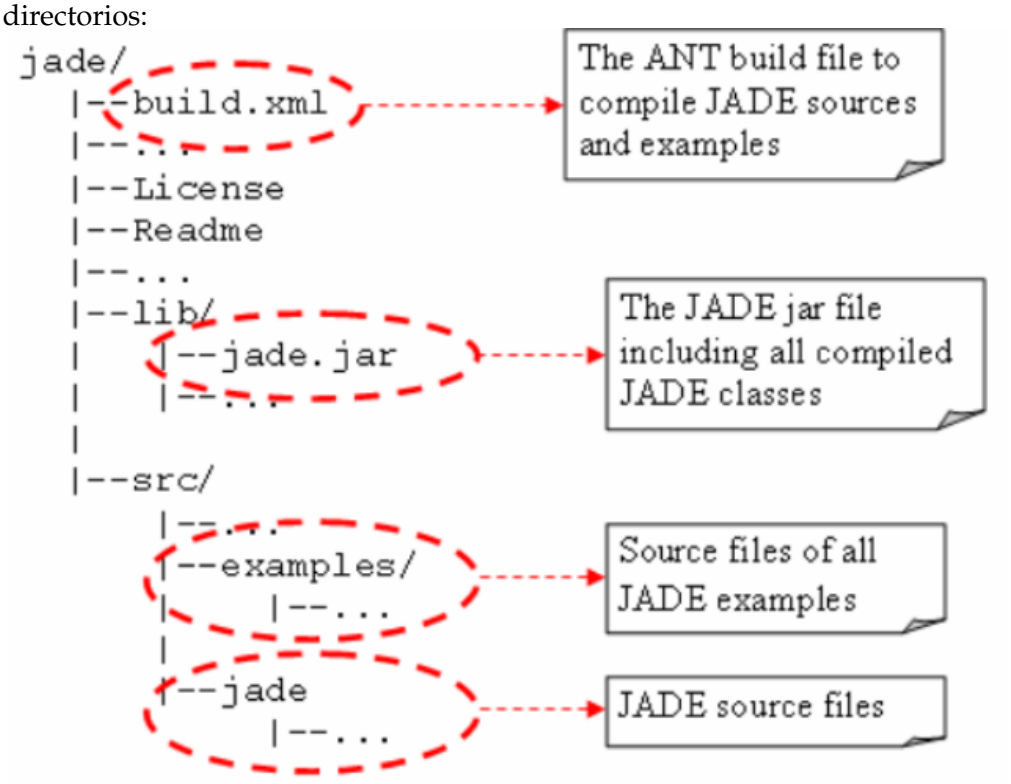
1 Código móvil

1.1 1. Código móvil:

1.1.1 a) Desarrollar, utilizando agentes móviles en JADE, un sistema rfs con las cuatro operaciones básicas: open, close, read y write.

1.1.2 b) Efectúe una instalación de JADE de modo tal de tener su contenedor principal corriendo en una maquina y un contenedor-1 corriendo en otra máquina de modo tal que este último contenga los archivos que serán objeto de la lectura/escritura.

Descargar la plataforma desde su sitio oficial: **http://jade.tilab.com/download/jade**, luego descomprimimos los diferentes ZIP en el mismo directorio, lo que nos dará la siguiente estructura de directorios:



Modo de ejecución Para lanzar la aplicación, se debe ejecutar el siguiente comando desde el directorio Jade:

```
java -cp lib/jade.jar jade.Boot -gui
```

Comandos útiles Para compilar un agente corremos el comando:

```
javac -cp lib/jade.jar -d <destino> <ruta fuente a compilar>
```

Para iniciar un nuevo contenedor el comando es:

```
java -cp lib/jade.jar:classes/ jade.Boot -container
```

Para iniciar un nuevo contenedor y asignarle un nombre el comando es:

```
java -cp lib/jade.jar:classes/ jade.Boot -container -container <nombre>
```

Para la creación de un contenedor remoto:

```
java -cp lib/jade.jar:classes/ jade.Boot -container -host <ip_dst> -local-host <ip_src>
```

1.1.3 c) Describa en qué cambia la arquitectura de la solución y cuáles son las ventajas y desventajas comparando esta solución con el estándar cliente/servidor.

La arquitectura de la solución cambia en que el programador se centra en implementar un agente, el cual se encarga de viajar hasta la máquina remota, realizar la tarea deseada, y retornar a destino (ya no se crean stubs como en cliente/servidor).

Mientras que en el estándar cliente/servidor, el programador se concentra en el conjunto de métodos o procedimientos a ejecutar en la máquina remota.

Ventajas

- No es necesario acordar los procedimientos entre cliente y servidor.
- No es necesario ejecutar una instalación del servidor.
- Se evitan los constantes pedidos y respuestas propias de cliente/servidor, disminuyendo con esto el tráfico de la red.

Desventajas

- Esta solución es solo compatible con la plataforma de JADE

1.1.4 d) Identifique las similitudes y diferencias de distribución/movilidad de código entre RMI y Jade.

1.1.5 Similitudes entre RMI y Jad

- Ambas tecnologías trabajan con objetos.
- Permiten la ejecución remota de código.
- Ambas están implementadas en Java.

Diferencias entre RMI y Jade

- RMI ejecuta código que se encuentra en una máquina remota (invoca métodos implementados en otra máquina).
- Mientras que Jade, tiene la posibilidad de migrar su código a otro host, ejecutarse y regresar al sitio de origen. (también permite la clonación de código)

1.2 2. Programar un agente móvil para que periódicamente recorra una secuencia de computadoras y conforme una tabla que contenga:

1.2.1 a) La Dirección IP o nombre de Host de cada una de ellas.

1.2.2 b) La Hora de llegada a cada una.

1.2.3 c) La carga de procesamiento de cada una de ellas.

1.2.4 d) Alguna otra información de interés que se le ocurra.

1.2.5 Al llegar a cada una de las computadoras el agente debe permanecer allí al menos 10 segundos, emitiendo una alerta en la pantalla local que indique su presencia.

1.2.6 Al retorno al punto de origen, se presentará la tabla, indicando además el tiempo total ocupado en el recorrido para recolectar la información.

Implementacion Se implementó un agente que contiene, en su método "action()", como estructura principal un switch con tres opciones. Cada una de ellas es ejecutada acorde al número de una variable estática numérica y global que indica el "estado" (**_state**) en el que se encuentra el agente. En nuestro caso, los estados son:

- **Opción 0:** El agente está en el contenedor en que se lo inició. En este punto se obtiene la lista de contenedores y se prepara el contexto para la migración.
- **Opción 1:** El agente está en alguno de los contenedores que deben ser escaneados para obtener información del host. Una vez escaneado esto, se mueve al siguiente contenedor.
- **Opción 2:** El agente volvió al contenedor inicial, esto quiere decir que ya circulo por todos los contenedores, entonces debe imprimir en la salida estándar la información recolectada y terminar.

Para el guardado de la información se implementó un arreglo de la clase "Informacion", la cual contiene toda la especificacion de lo requerido en cada lectura de cada contenedor. Entonces, el agente en cada contenedor que visita instancia un objeto "Informacion" que contiene toda la información escaneada y la agrega a dicho arreglo.

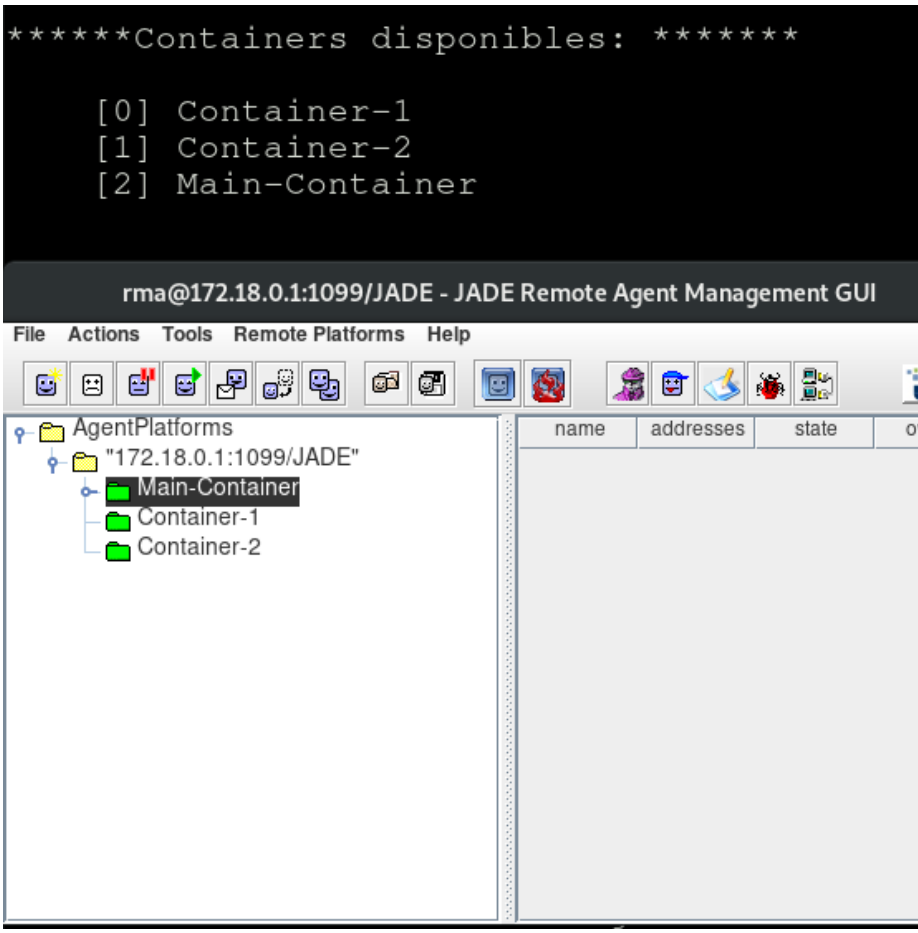
Luego, en el _State=2, donde el agente ya esta nuevamente en destino y debio haber escaneado todos los contenedores, recorre su arreglo imprimiendo la informacion de cada objeto de tipo "Informacion" perteneciente a cada contenedor escaneado.

En nuestro caso, la informacion escaneada es:

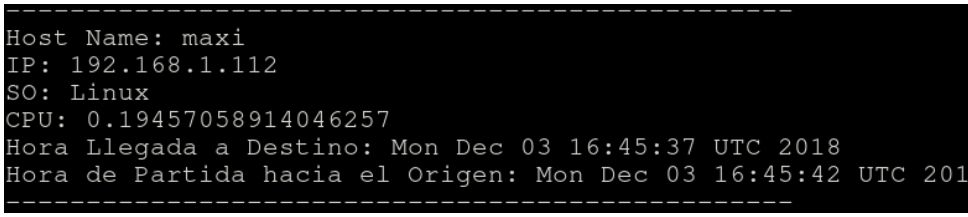
- Host Name
- IP Host
- SO
- Consumo de CPU

- Horario de llegada al Host Destino
- Horario de partida del Host

Ejemplo de Escaneo de Contenedores



Ejemplo de Imprecion de un Host



1.3 3. Realizar y documentar los siguientes experimentos:

1.3.1 Transferir un archivo de gran tamaño (GBs) completo las siguientes condiciones:

1.3.2 a) En la versión con Java Sockets del TP1.

1.3.3 b) En la versión Java RMI del TP2.

1.3.4 c) En la versión con Agentes Móviles del punto 1

1.3.5 Utilizar las primitivas de lectura o escritura con un tamaño de buffer igual y cronometrar (que el mismo cliente muestre cuanto demoró), en los tres casos, sacando las conclusiones de cada caso respecto a la performance de cada solución. Documentar lo realizado.

2 Comunicación Indirecta: MQTT e IoT

2.1 4. Realizar el siguiente desarrollo

2.1.1 a. Tomar dos kits de desarrollo (Galileo o Arduino) e instrumentar un protocolo de aplicación en modo Publicador/Suscriptor sobre MQTT, sobre la base de Tópicos y valores. Habrá una placa con sensores y actuadores y se simulará una aplicación de comando/monitor mediante MQTT-Spy (java).

2.1.2 En la placa publicadora instrumentar: Una medición de temperatura, dos botones (por touch y pulsado), como sensores y dos Leds (Rojo azul) como actuadores. La temperatura se publicará cada 10 segundos. El cambio de estado de los botones se publicará cuando ocurra y en modo eventual con estampa temporal.

Introducción

La simulación de este apartado del trabajo se utilizarán las siguientes tecnologías:

- Node-Red: para la simulación de la placa arduino y el despliegue de un panel de control
- Eclipse Mosquitto: para el despliegue del broker
- Docker: para la puesta en marcha de 3 contenedores:
  - El broker Mosquitto que tendrá la ip **172.16.240.10**
  - El nodo de Node-Red para la simulación de la placa que tendrá la ip **172.16.240.20**
  - El node de Node-Red para el panel de control que tendrá la ip **172.16.240.30**

Este ecosistema se encuentra especificado en el archivo ***docker-compose.yml***
Para reproducir el proyecto son necesarios los siguientes pasos

1) **Lanzar los contenedores**

Ejecutar el comando `docker-compose up --build` en el directorio raíz del mismo.
Luego:

- El nodo de simulación de la placa estará disponible en http://localhost:1880
- El nodo de planel de control estará disponible en http://localhost:1881

2) **Importar los nodos**

Los nodos están especificados en los archivos

- ***flows/publisher/flow.json*** (simulacion arduino, puerto 1880)
- ***flows/subscriber/flow.json*** (panel de control, puerto 1881)

Los nodos se importan con `Ctrl+i` y pegando el contenido del archivo json correspondiente en el modal. Luego presionar ***Done*** y luego ***Deploy***. Finalmente las interfaces web estarán disponibles en las rutas:

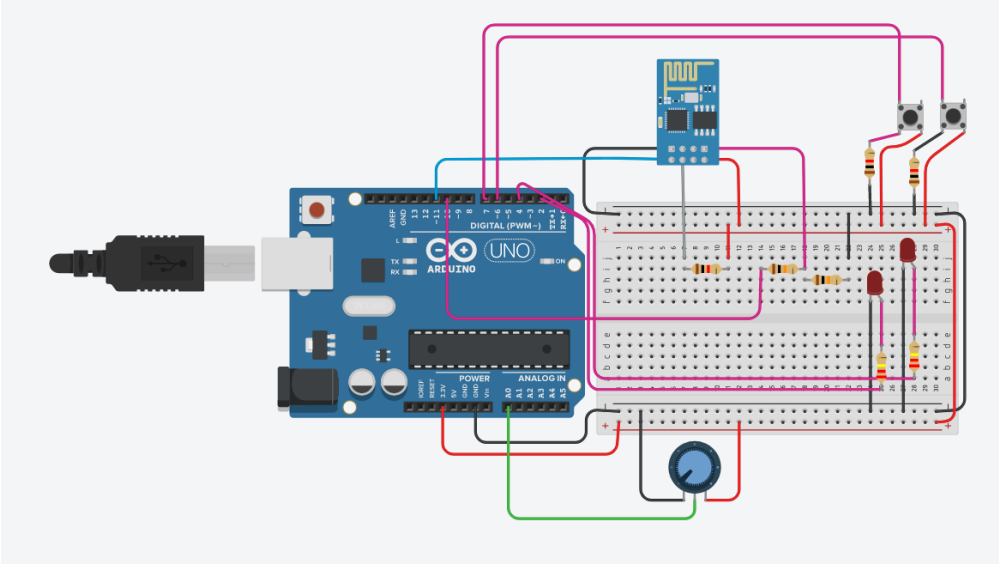
- http://localhost:1880/ui/ (simulación arduino)
- http://localhost:1881/ui/ (simulacion panel de control)

Diseño

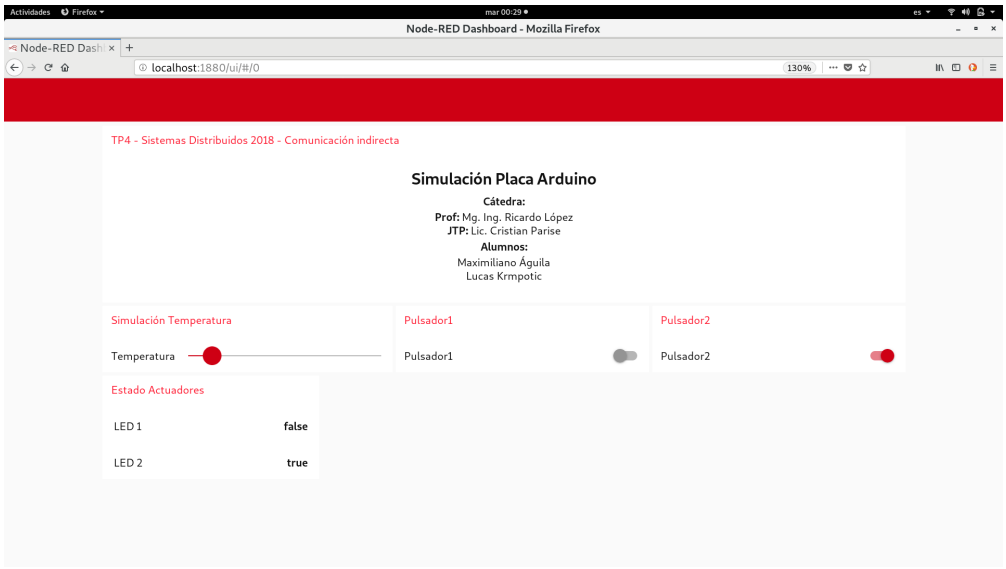
Según los requerimientos de la consigna la placa Arduino podría tener el siguiente esquema de conexiones:

- Un módulo ESP8266 para la conexión a la red
- Un potenciómetro para simular la toma de temperatura
- Dos pulsadores para la simulación de los sensores
- Dos led para la simulación de los actuadores

tal como se ve en la siguiente imagen



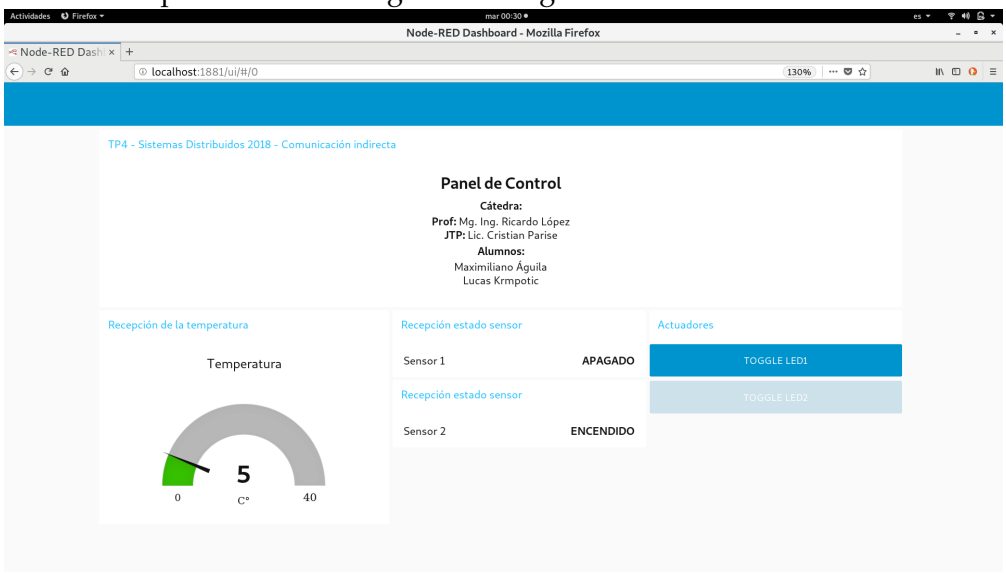
La correspondencia de este diseño arduino con la UI-web se puede ver en la siguiente imagen



Por su parte el panel de control cuenta con:

- Una aguja para mostrar el estado de la temperatura
- Dos cajas de texto para mostrar el estado de los sensores
- Dos botones para estimular los actuadores

Tal como se puede ver en la siguiente imagen:



Esquema de tópicos y programación de los nodos

En sistemas grandes puede resultar conveniente la utilización de un protocolo de aplicación (MARA por ejemplo) para publicar un conjunto amplio de variables sobre un conjunto limitado de tópicos, lo cierto es que en nuestro caso el problema puede resolverse con 5 tópicos que dimos en llamar:

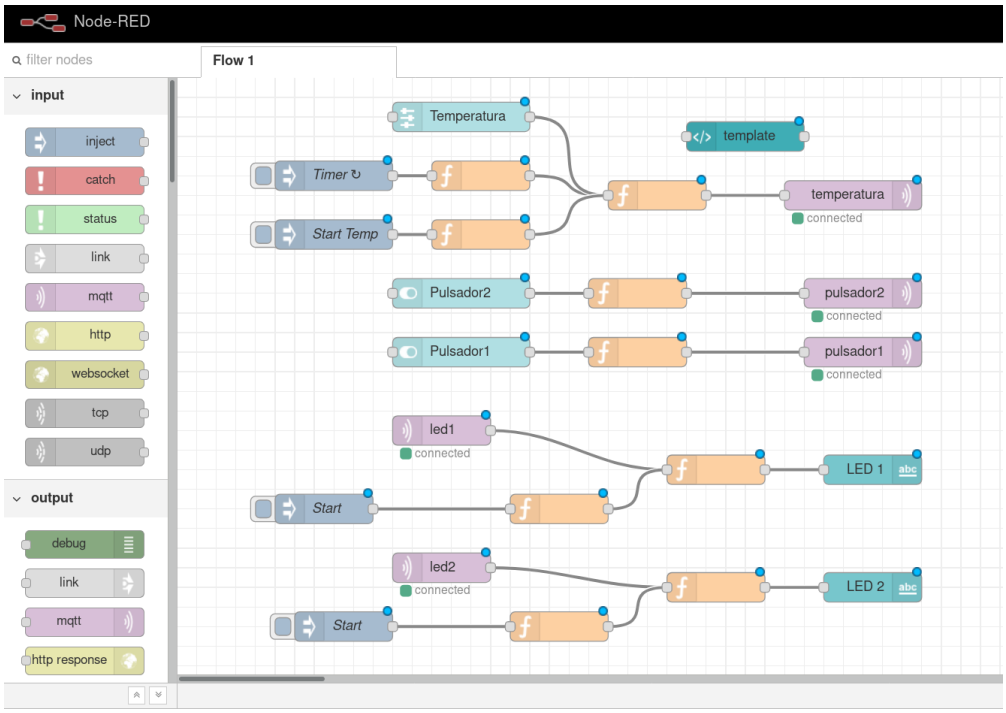
- **temperatura** - con arduino publicador y panel de control suscriptor

- **pulsador1** - con arduino publicador y panel de control suscriptor
- **pulsador2** - con arduino publicador y panel de control suscriptor
- **Led1** - con panel de control publicador y arduino suscriptor
- **Led2** - con panel de control publicador y arduino suscriptor

aprovechando las bondades de Node-Red y evitando trabajo adicional de parseo de strings

Aplicación de simulación de la placa Arduino En la imagen a continuación se presenta el flow correspondiente a la aplicación que simula la placa arduino, en ella se distinguen:

- **Nodos violetas:** son nodos MQTT de entrada (suscripción) o de salida (publicación).
- **Nodos celestes:** son nodos que definen componentes gráficos (html + javascript).
- **Nodos azules:** son nodos de inicio que sirven para inicializar variables javascript con algún valor de interés
- **Nodos función:** son nodos en los que se definen funciones js para manipular los mensajes entre los nodos.



Las conexiones entre los nodos representan flujos de eventos javascript en los que los nodos observadores reciben por parametro un objeto llamado ***msg*** que tiene un atributo llamado ***payload*** que transporta datos.

Entonces, por ejemplo, la simulación de los pulsadores hace uso del evento propio del componente nodo ***switch*** que envía *true* o *false* según corresponda. El evento de cambio de estado en el nodo switch es escuchado por el nodo función que agrega al *payload* el timesamp (como lo requería la consigna), y éste último emite el evento escuchado por el nodo de salida MQTT que publica el mensaje en el servidor. El código de dicho nodo función es el siguiente:

```
// recepción del mensaje con el payload (true/false) del switch
var estado = msg.payload;
// creación del nuevo payload
_payload = {}
_payload.estado = estado;
_payload.time = new Date().toLocaleString('es-AR', { timeZone: 'UTC' });

return {
    payload:_payload
}
```

Para la publicación de la temperatura, donde el requerimiento era que se publicase cada 10 seg, se utilizaron 2 variables con *scope* más profundo que el contexto de las funciones que intervienen en el flujo de eventos. No estamos seguros de si se trata de variables globales o definidas en el contexto del padre estático de las funciones del flujo, pero para el caso es lo mismo. Persisten la ejecución de la función.

El problema es que como en Node-Red toda la programación es dirigida por los eventos, es más natural "descartar un mensaje" que intentar tomar el valor actual de una propiedad html. Una de estas variables se utilizó para persistir el último valor de temperatura generado por el último evento del componente *slider*. La otra como flag para saber si se cayó el timer que indica que es momento de publicar la temperatura. De este modo cuando el splider que simula la toma de temperatura es estimulado, el nodo función que escucha el evento de cambio de valor hace dos cosas:

1) Guarda ese valor nuevo en la variable global *temperatura*
2) Consulta el estado de la variable global *state*, si es true publica ese valor y setea el flag en false, si es false no lo publica.

Cuando el evento se produce por la caida del timer, simplemente se publica el valor actual de la variable global temperatura y se setea el flag en false.

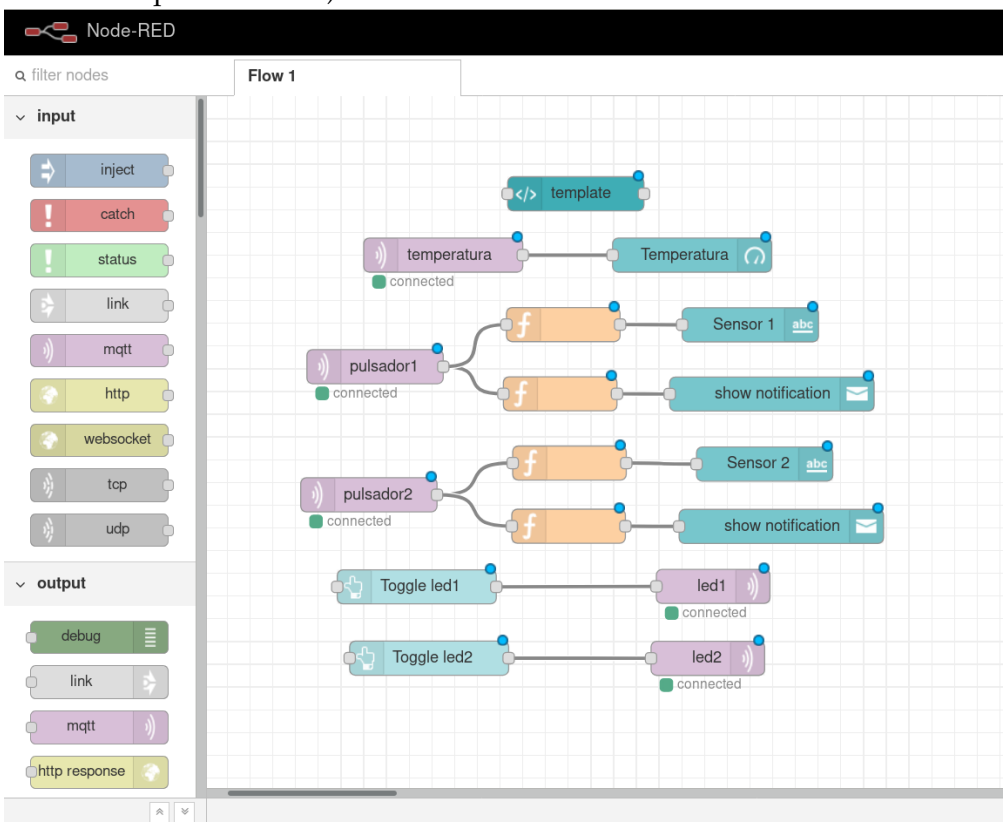
El fragmento de código de la función que escucha los dos eventos es el siguiente:

```
// si el payload es de tipo number es evento del sensor de temperatura
if(typeof msg.payload === 'number') {

    // guarda la temperatura en la variable global
    context.global.temperatura = msg.payload;

    // si el flag es true la publica y setea el flag en false
    if (context.global.state === true){
        context.global.state = false;
        return {
            payload: context.global.temperatura
        }
    }
}
// si el payload es true es evento del timer
if (msg.payload === true){
```

```
    // setea el flag en false
    context.global.state = false;

    // publica la temperatura actual
    return {
        payload: context.global.temperatura
    }
}
```

Aplicación Panel de Control Esta plicación es súmamente simple dado que no es necesario esfuerzo adicional de formateo de mensajes.

Las únicas funciones javascript utilizadas tienen por objetivo escuchar la recepción de mensajes en los tópicos correspondientes a los sonsores digitales y tomar del mensaje el valor del sensor o el timestamp para pasárselo al elemento gráfico pertinente (nodo caja de texto y nodo notificación respectivamente).



2.1.3 b. Probando con el servidor Mosquitto en localhost analizar mediante wireshark –en modo comando por consola o utilizando el MQTT-Spy en Java -, como es el comportamiento de los mensajes variando las distintas QOS (0, 1 y 2). Informe lo observado.

Introducción

Las calidades de servicio en MQTT determinan el modo en que se confirmarán, o no, las recepciones de mensajes de tipo *"Publish"*:

- **QoS 0:** no hay confirmación
- **QoS 1:** confirmación simple, receptor envía mensaje tipo ***"Publish Ack"***
- **QoS 2:** confirmación en 2 pasos:
  - receptor envía ***"Publish Received"***
  - emisor envía ***"Publish Release"***
  - receptor cierra la confirmación enviando ***"Publish Complete"***

Los términos *emisor* y *receptor* son adecuados dado que este comportemiento no varía por tratarse de clientes o servidores. En el caso de un cliente publicador, la QoS se establece directamente en el mensaje ***"Publish"***, mientras que los clientes suscriptores requieren una determinada calidad de servicio al momento de suscribirse a un tópico (en el campo ***"Requested QoS"*** del payload del mensaje ***"Subscribe"***).

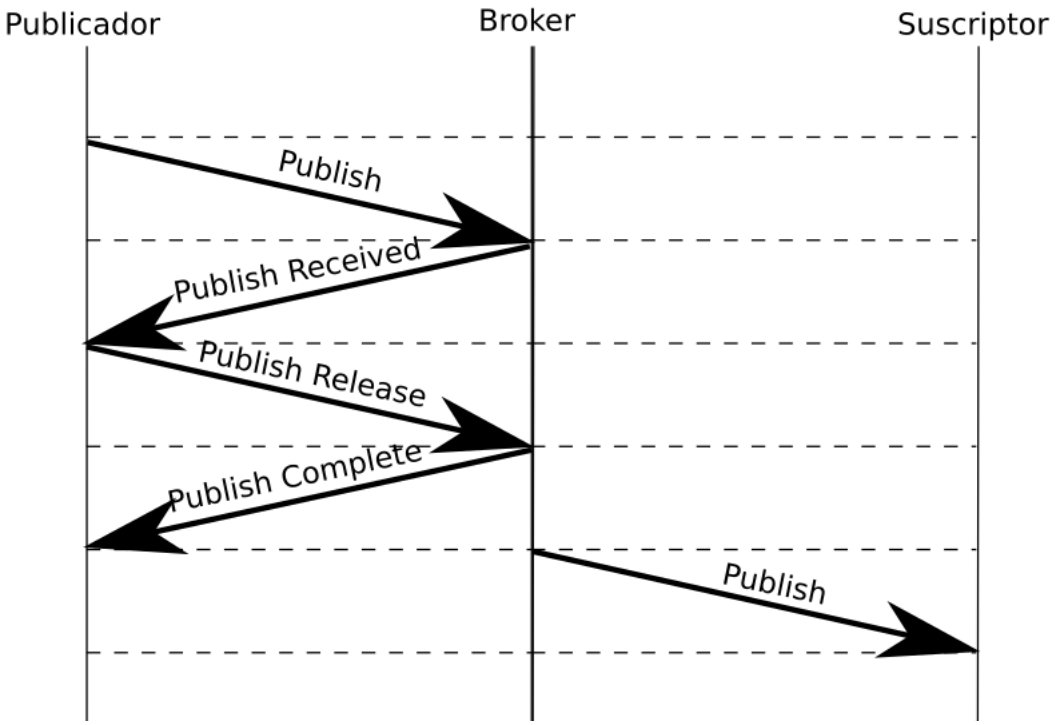
Para probar el concepto, se usará el escenario del ejercicio anterior, publicando mensajes al topico *"pulsador1"*. El publicador enviará siempre mensajes con QoS = 2, mientras que se variarán las calidades de servicio en los mensajes intercambiados entre el broker y el suscriptor.

Por una cuestión de visualización se utilizará la librería Scapy de python para sniffear la red, filtrar paquetes y disectarlos. No obstante los resultados se escribiran en archivos *pcap* que podrán ser leídos en Wireshark o herramientas similares.

Clase sniffer para filtrado y generación del pcap

```
In [1]: from scapy.all import *
        from scapy.contrib import mqtt

        # filtro paquetes tcp con origen o destino en el broker en el puerto 1883,
        FILTER="(dst 172.16.240.10 or scr 172.16.240.10) and tcp and dport mqtt"

        class MQTTSniffer():

            def __init__(self):
                self.packets = []

            def pkt_save(self, pkt):
                """ guarda el paquete en la lista controlando que sea MQTT """
                if pkt.haslayer(mqtt.MQTT):
                    self.packets.append(pkt)

            def to_pcap(self, file_name):
                """ genera un archivo pcap con la lista de paquetes """
                wrpcap(file_name, self.packets)
```

Prueba 1: QoS 2 en publicador, QoS 0 en suscriptor Escenario teórico esperado:
:



Ejecucion de la prueba

```
In [140]: # instanciamos nuestro sniffer
          sniffer = MQTTSniffer()

          # lo ponemos a sniffear 25 paquetes en la interfaz Docker que funciona como gateway
          sniff(iface="br-b1e46316c5ac", filter=FILTER, prn=sniffer.pkt_save, count=25)

          # escribimos el resultado en un archivo pcap
          sniffer.to_pcap("QoS-0.pcap")

          # mostramos el resultado en pantalla
          [pkt.summary() for pkt in sniffer.packets]

Out[140]: ['Ether / IP / TCP 172.16.240.20:59718 > 172.16.240.10:mqtt PA / MQTT / MQTTPublish',
           'Ether / IP / TCP 172.16.240.10:mqtt > 172.16.240.20:59718 PA / MQTT / MQTTPubrec',
           'Ether / IP / TCP 172.16.240.20:59718 > 172.16.240.10:mqtt PA / MQTT / MQTTPubrel',
           'Ether / IP / TCP 172.16.240.10:mqtt > 172.16.240.20:59718 PA / MQTT / MQTTPubcomp',
           'Ether / IP / TCP 172.16.240.10:mqtt > 172.16.240.30:45220 PA / MQTT / MQTTPublish']
```

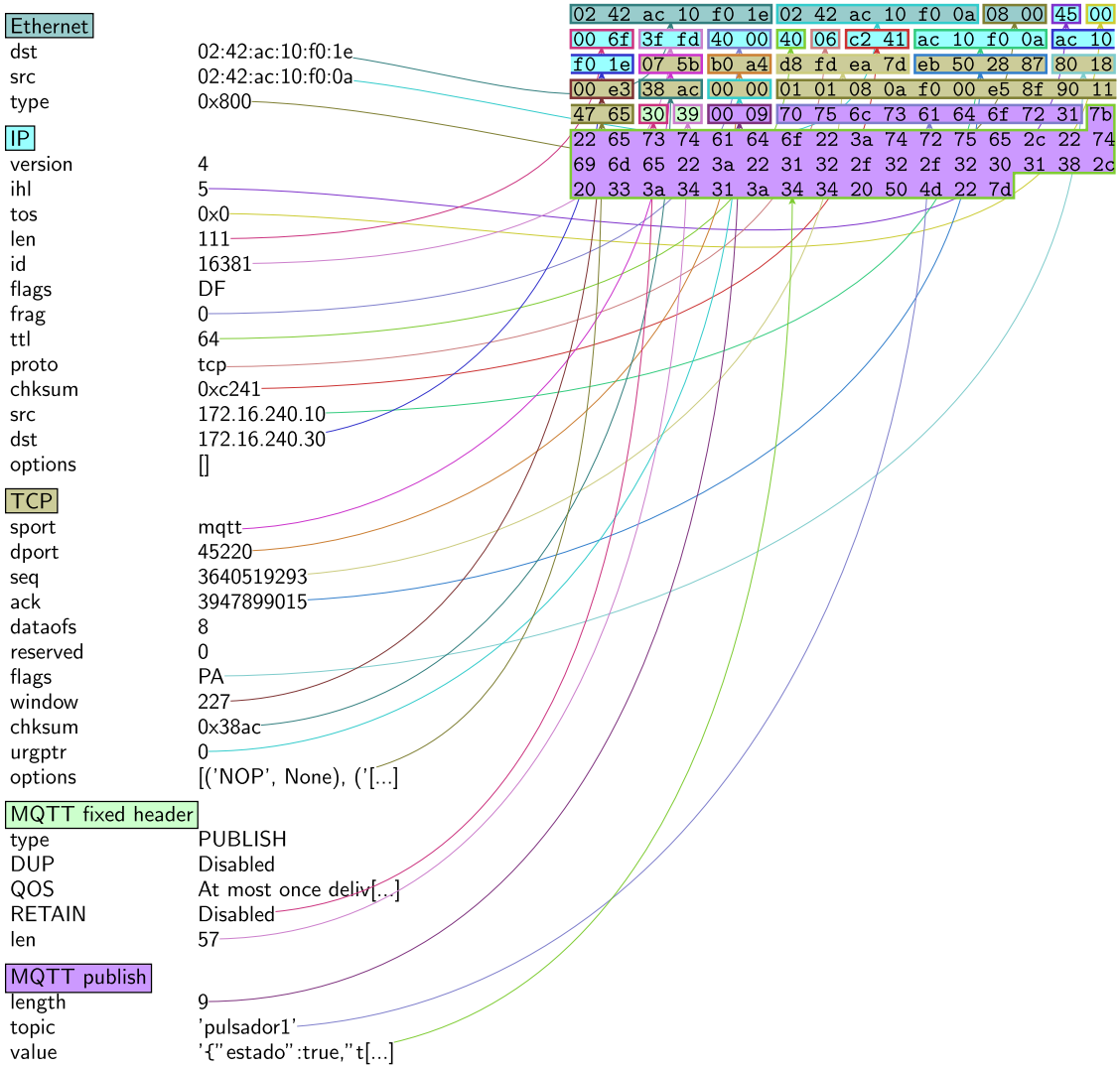
Vemos que efectivamente el sniffer capturó 5 paquetes correspondientes a los intercambios publicador/broker/suscriptor y que son consistentes con el Escenario teórico esperado:

.

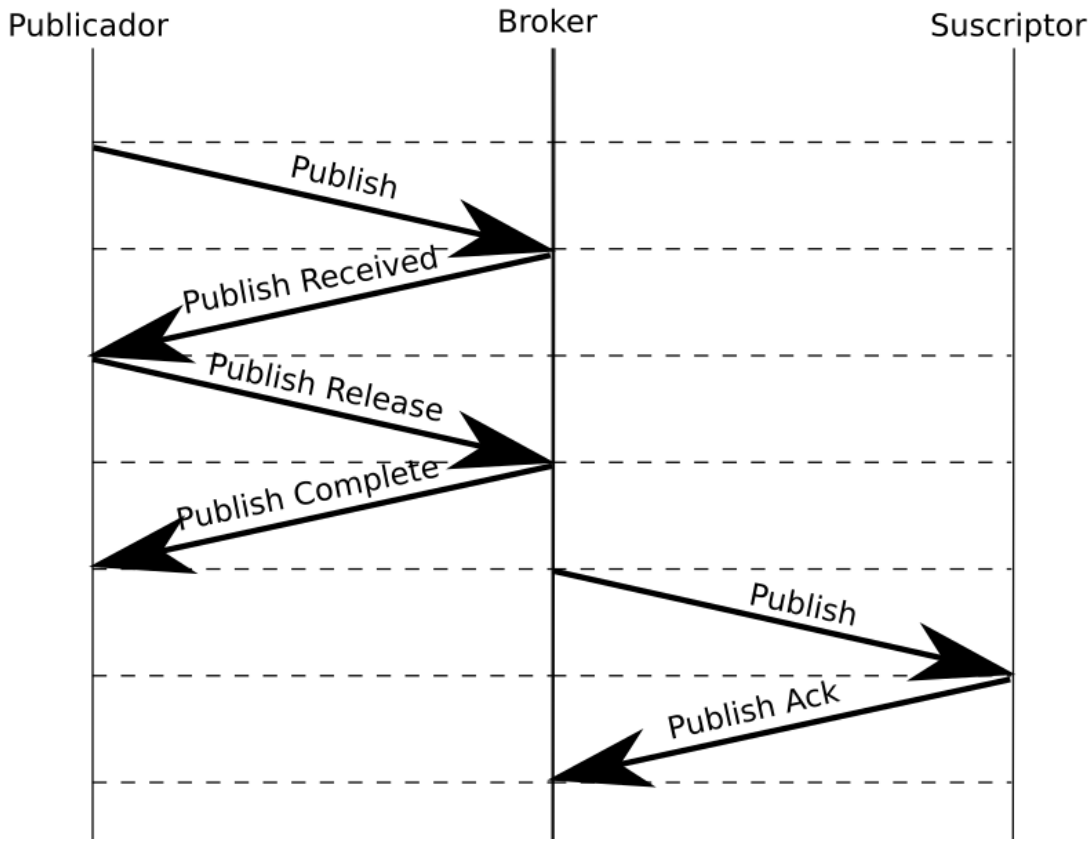
Finalmente ploteamos la disección del último paquete que corresponde al envío del mensaje desde el broker al suscriptor. En la imagen podemos ver que el campo QoS tiene el valor ***At most once delivery***, versión verbosa de QoS=0.

```
In [4]: cap=rdpcap("QoS-0.pcap")
        cap[4].canvas_dump()

Ignoring line 5841 in mapping file 'psfonts.map': Unknown token '<DSSerif-Bold'
Ignoring line 5843 in mapping file 'psfonts.map': Unknown token '<DSSerifUni-Bold'
Ignoring line 5841 in mapping file 'psfonts.map': Unknown token '<DSSerif-Bold'
Ignoring line 5843 in mapping file 'psfonts.map': Unknown token '<DSSerifUni-Bold'
```

Out[4]:



Prueba 2: QoS 2 en publicador QoS 1 en suscriptor Escenario teórico esperado:



Ejecución de la prueba

```
In [3]: sniffer = MQTTSniffer()
        sniff(iface="br-b1e46316c5ac", filter=FILTER, prn=sniffer.pkt_save, count=25)
        sniffer.to_pcap("QoS-1.pcap")
        [pkt.summary() for pkt in sniffer.packets]
```

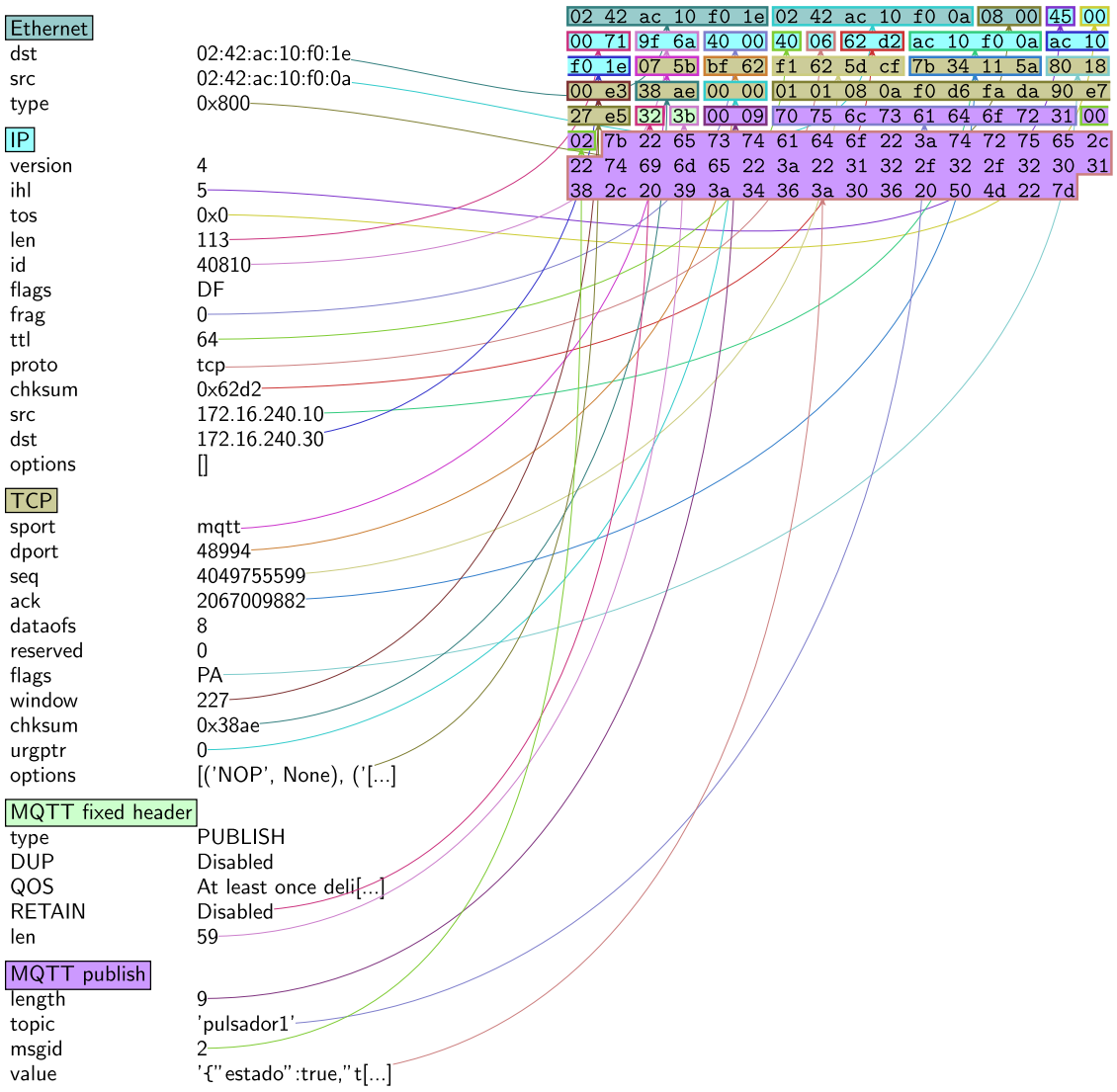
```
Out[3]: ['Ether / IP / TCP 172.16.240.20:59718 > 172.16.240.10:mqtt PA / MQTT / MQTTPublish',
         'Ether / IP / TCP 172.16.240.10:mqtt > 172.16.240.20:59718 PA / MQTT / MQTTPubrec',
         'Ether / IP / TCP 172.16.240.20:59718 > 172.16.240.10:mqtt PA / MQTT / MQTTPubrel',
         'Ether / IP / TCP 172.16.240.10:mqtt > 172.16.240.20:59718 PA / MQTT / MQTTPubcomp',
         'Ether / IP / TCP 172.16.240.10:mqtt > 172.16.240.30:48994 PA / MQTT / MQTTPublish',
         'Ether / IP / TCP 172.16.240.30:48994 > 172.16.240.10:mqtt PA / MQTT / MQTTPuback']
```

Nuevamente el escenario es el esperado y en la imagen del mensaje ***"Publish"*** desde el broker al suscriptor podemos comprobar que el campo QoS tiene el valor ***"At least once delivery"*** (QoS=1).

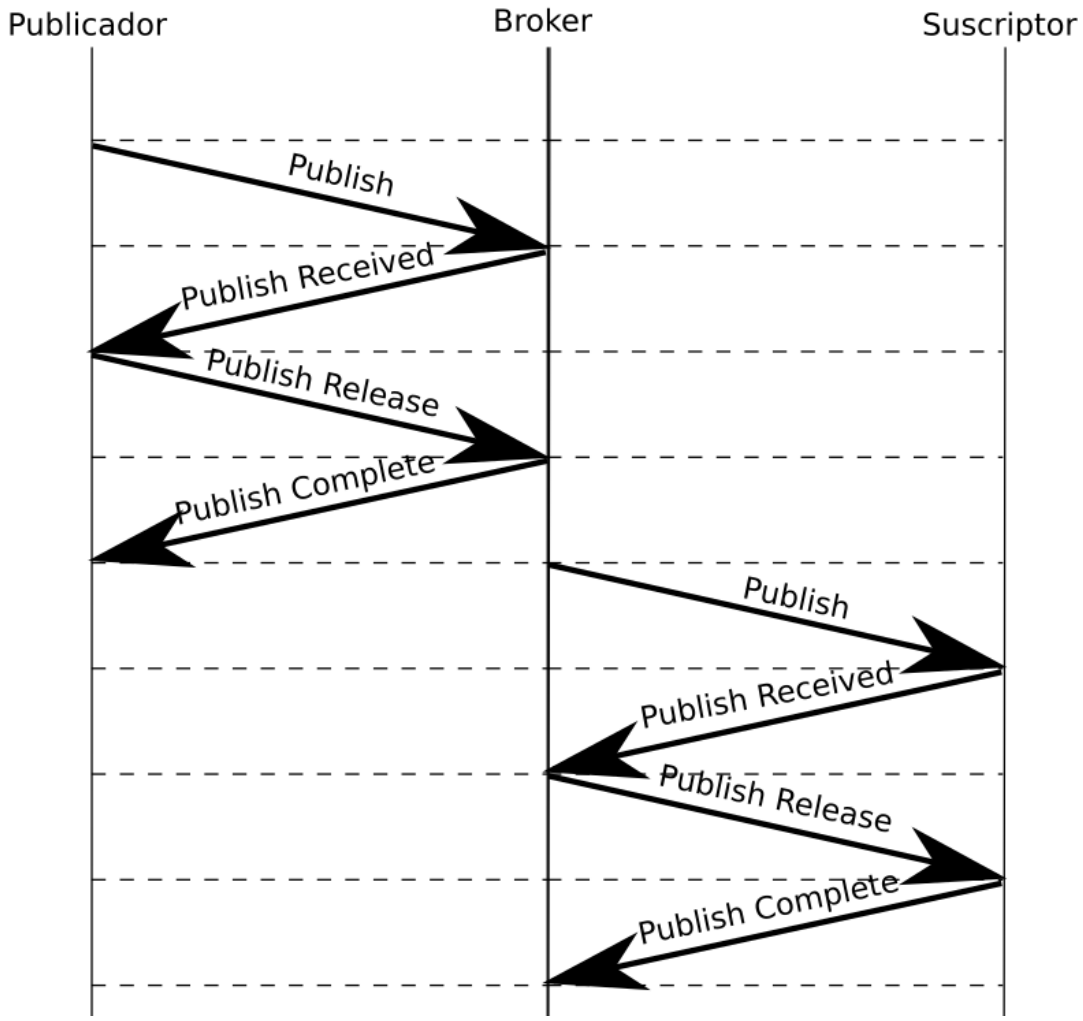
```
In [2]: cap=rdpcap("QoS-1.pcap")
        cap[4].canvas_dump()
```

```
Ignoring line 5841 in mapping file 'psfonts.map': Unknown token '<DSSerif-Bold'
Ignoring line 5843 in mapping file 'psfonts.map': Unknown token '<DSSerifUni-Bold'
Ignoring line 5841 in mapping file 'psfonts.map': Unknown token '<DSSerif-Bold'
Ignoring line 5843 in mapping file 'psfonts.map': Unknown token '<DSSerifUni-Bold'
```

Out[2]:



Prueba 3: QoS 2 en publicador, QoS 2 en suscriptor Escenario teórico esperado:



Ejecución de la prueba

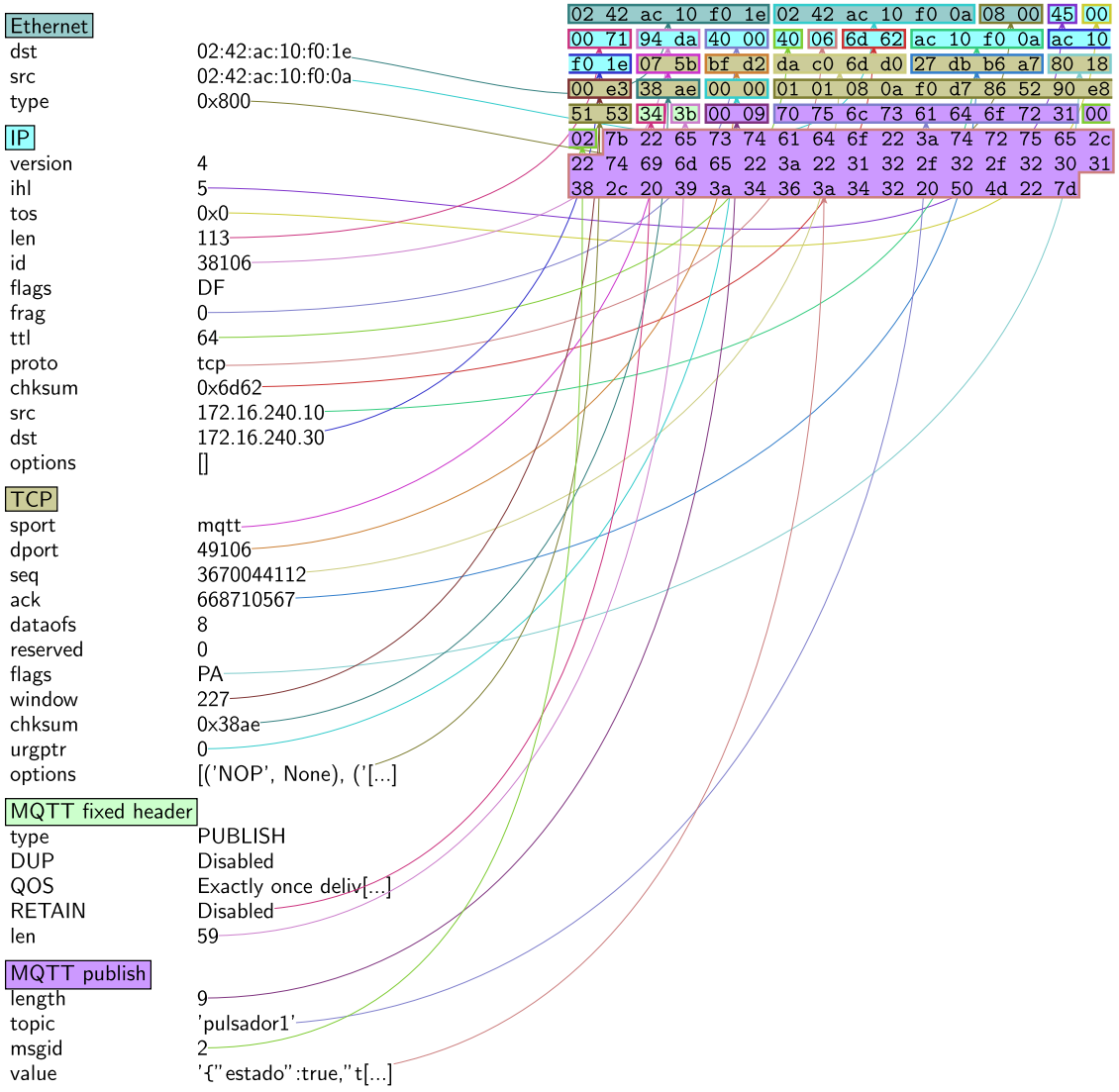
```
In [4]: sniffer = MQTTSniffer()
        sniff(iface="br-b1e46316c5ac", filter=FILTER, prn=sniffer.pkt_save, count=25)
        sniffer.to_pcap("QoS-2.pcap")
        [pkt.summary() for pkt in sniffer.packets]

Out[4]: ['Ether / IP / TCP 172.16.240.20:59718 > 172.16.240.10:mqtt PA / MQTT / MQTTPublish',
         'Ether / IP / TCP 172.16.240.10:mqtt > 172.16.240.20:59718 PA / MQTT / MQTTPubrec',
         'Ether / IP / TCP 172.16.240.20:59718 > 172.16.240.10:mqtt PA / MQTT / MQTTPubrel',
         'Ether / IP / TCP 172.16.240.10:mqtt > 172.16.240.20:59718 PA / MQTT / MQTTPubcomp',
         'Ether / IP / TCP 172.16.240.10:mqtt > 172.16.240.30:49106 PA / MQTT / MQTTPublish',
         'Ether / IP / TCP 172.16.240.30:49106 > 172.16.240.10:mqtt PA / MQTT / MQTTPubrec',
         'Ether / IP / TCP 172.16.240.10:mqtt > 172.16.240.30:49106 PA / MQTT / MQTTPubrel',
         'Ether / IP / TCP 172.16.240.30:49106 > 172.16.240.10:mqtt PA / MQTT / MQTTPubcomp']
```

Efectivamente 8 paquetes han sido capturados y el ploteo del mensaje muestra que el campo QoS tiene el valor ***"Exactly once delivery"*** (QoS=2).

```
In [3]: cap=rdpcap("QoS-2.pcap")
        cap[4].canvas_dump()
```

```
Ignoring line 5841 in mapping file 'psfonts.map': Unknown token '<DSSerif-Bold'
Ignoring line 5843 in mapping file 'psfonts.map': Unknown token '<DSSerifUni-Bold'
Ignoring line 5841 in mapping file 'psfonts.map': Unknown token '<DSSerif-Bold'
Ignoring line 5843 in mapping file 'psfonts.map': Unknown token '<DSSerifUni-Bold'
```

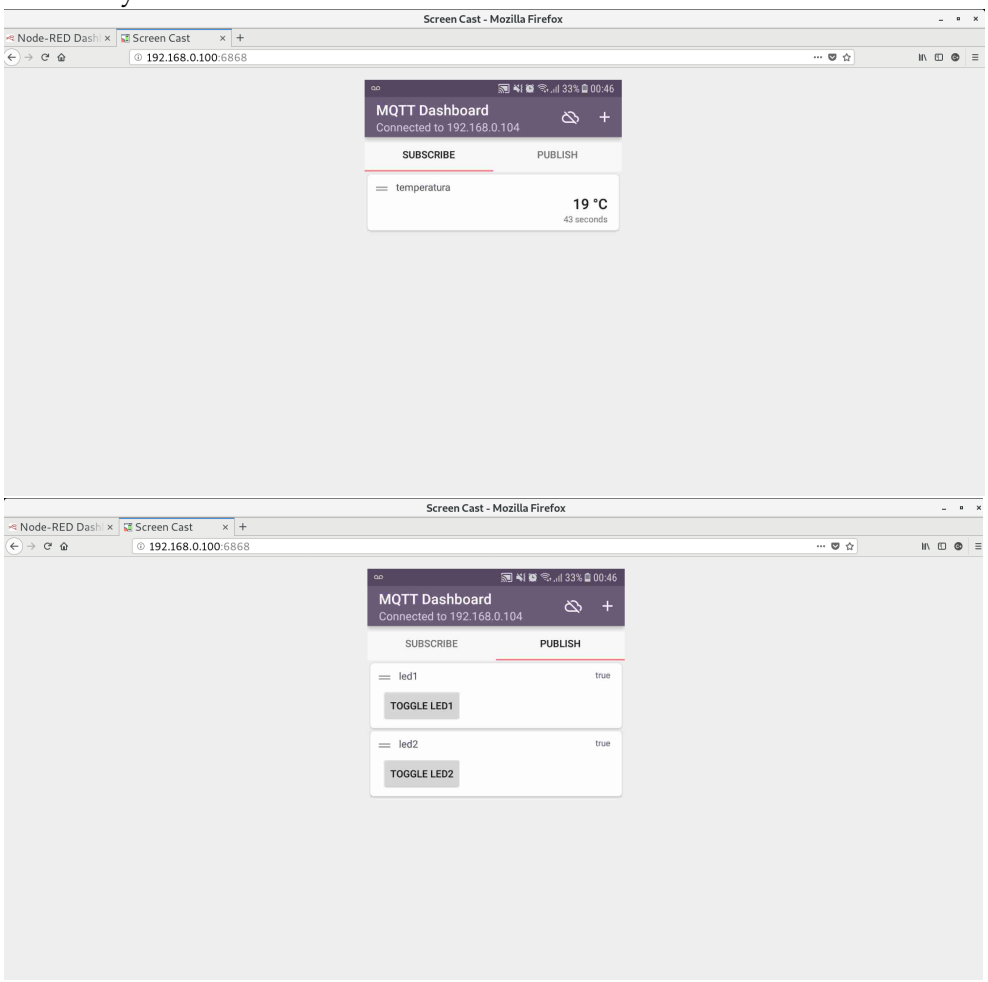
Out[3]:



2.1.4 c. Investigar sobre ANDROID utilizando una app de MQTT (ej: Mqtt DashBoard) y publicar mensajes del protocolo de aplicación desarrollado en el apartado a) para controlar y monitorear los sensores y actuadores de la placa.

A continuación se presentan 2 capturas de pantalla que documentan el uso de la aplicación MQTT Dashboard en el marco del escenario desarrollado en el incisa a).

La primera imagen muestra como es posible suscribirse al tópico temperatura y la segunda muestra la creación de 2 botones para hacer toggle sobre los actuadores, publicando sobre los tópicos led1 y led2.



Cabe mencionar que en el escenario planteado no se hizo huso de autenticación para la publicación/suscripcipón en los tópicos.