

UNPSJB

-LICENCIATURA EN SISTEMAS OGCPI-

-SISTEMAS DISTRIBUIDOS-

-Trabajo de Laboratorio 4-

-Código Móvil. Comunicación Indirecta. MQTT. IoT-

Cátedra

Mg. Ing Ricardo López
Lic. Cristian Parise

Integrantes:

Lucas Krmpotic
Maximiliano Águila

December 4, 2018



Contents

1	Código móvil	3
1.1	1. Código móvil:	3
1.1.1	a) Desarrollar, utilizando agentes móviles en JADE, un sistema rfs con las cuatro operaciones básicas: open, close, read y write.	3
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.	4
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.	5
1.1.4	d) Identifique las similitudes y diferencias de distribución/movilidad de código entre RMI y Jade.	6
1.1.5	Similitudes entre RMI y Jade	6
1.2	2. Programar un agente móvil para que periódicamente recorra una secuencia de computadoras y conforme una tabla que contenga:	6
1.2.1	a) La Dirección IP o nombre de Host de cada una de ellas.	6
1.2.2	b) La Hora de llegada a cada una.	6
1.2.3	c) La carga de procesamiento de cada una de ellas.	6
1.2.4	d) Alguna otra información de interés que se le ocurra.	6
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.	6
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.	6
1.3	3. Realizar y documentar los siguientes experimentos:	8
1.3.1	Transferir un archivo de gran tamaño (GBs) completo las siguientes condiciones:	8
1.3.2	a) En la versión con Java Sockets del TP1.	8
1.3.3	b) En la versión Java RMI del TP2.	8
1.3.4	c) En la versión con Agentes Móviles del punto 1	9
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.	9
2	Comunicación Indirecta: MQTT e IoT	9
2.1	4. Realizar el siguiente desarrollo	9
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).	9
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.	9

2.1.3	b. Probando con el servidor Mosquitto en localhost analizar mediante wire-shark –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.	15
2.1.4	c. Investigar sobre ANDROID utilizando una app de MQTT (ej: Mqtt Dash-Board) y publicar mensajes del protocolo de aplicación desarrollado en el apartado a) para controlar y monitorear los sensores y actuadores de la placa.	22

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.

Implementacion Para realizar un RFS con Agentes Moviles se opto por desarrollar una aplicacion lanzadora (**App.java**) la cual es en si un agente que puede ser lanzado por la plataforma JADE.

Esta aplicacion despliega un menu de opciones y dependiendo de la opcion **leer/escibir** se instancia un objeto de tipo **ReadAgent** o un objeto de tipo **WriteAgent**. Estos ultimos son los agentes que viajarian al destino y realizacar la escritura o lectura remota de archivos.

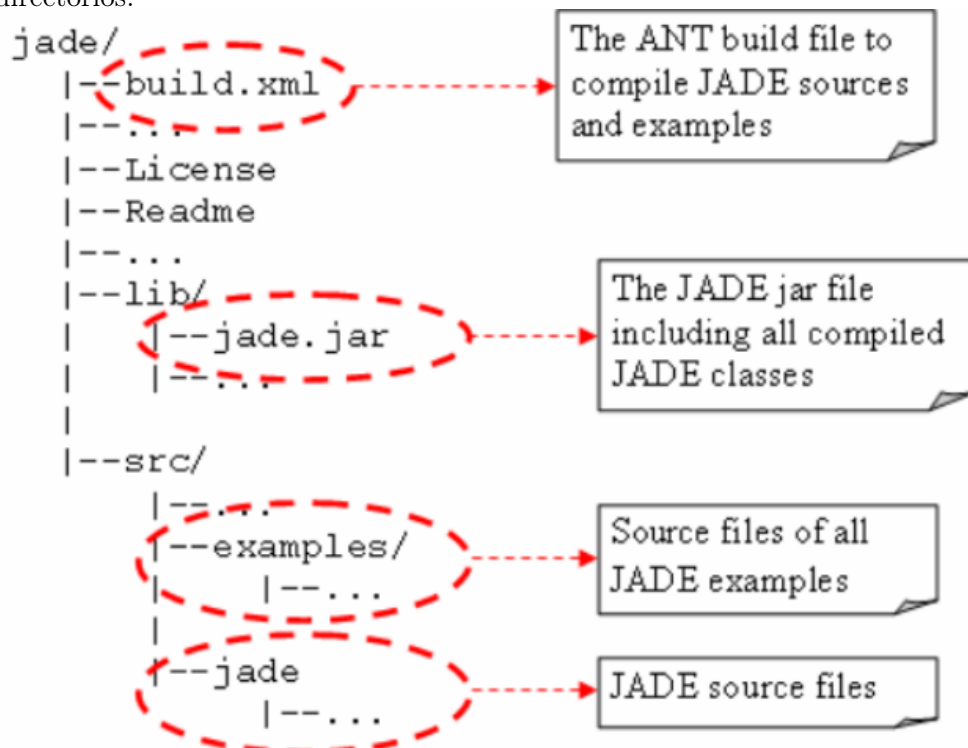
En resumen:

- **App.java:** Es en si un agente que puede ser lanzado por la plataforma JADE. Este agente no viaja a ningun destino, sino que se encarga de instanciar los objetos de lectura o escritura remota. Tampoco tiene estados (*****_State*****), sino que solo controla como instanciar los objetos de lectura/escritura y despliega el menu de opciones para seleccionar que accion realizar.
- **ReadAgent:** Es un agente de lectura remota. Este agente viaja a destino y es el encargado de realizar la lectura remota y escritura local. Este agente en su maquina de estados consta de 4(cuatro) estados.
 - *****_state=0*:** En este estado el agente se encuentra en la maquina local y se realiza la migracion hacia el host destino (container de destino) con el metodo **doMove()****.
 - *****_state=1*:** En este estado el agente se encuentra en el contenedor del host remoto. Entonces aqui comienza la lectura del archivo remoto. Se utiliza un buffer de tamaño 1024 bytes y ademas se realiza el control de un offset, ya que JADE maneja agentes con migracion debil, es decir que no maneja estados. Y por ultimo se vuelve al host de origen (local) con el metodo **doMove()****.
 - *****_state=2:** *En este estado el agente se encuentra nuevamente en el origen, con lo cual procede a la escritura del buffer que leyo remotamente, a un archivo local. La clave de este estado, es verificar si count (variable que realiza el read()) es mayor a 0. Entonces si count>0 quiere decir que leyo algo en el buffer y tiene que hacer un write() en el archivo local y poner la variable global _state=1*** o ***_state=*** nuevamente para repetir la accion de lectura remota. Y si count<0 quiere decir que no hay mas que leer, y se pone ***_state++*** o ***_state=3*** para que la maquina de estados pase de estado.*
 - *****_state=3*** :** Si se llega a este estado quiere decir que ya no hay mas que leer y se termino de leer el archivo remoto. Se procede a eliminar el agente con el metodo **doDelete()**.
- **WriteAgent:** Es un agente de escritura remota. Este agente viaja de origen a destino y se encarga de realizar una escritura remota. Este agente en su maquina de estados consta de 3(tres) estados.

- *****_state=0:** *En este estado el agente se encuentra en la maquina local o host local. Por lo cual se procede a la lectura del archivo local con la funcion `read()` de manera local. Luego el agente migra hacia el host destino (container destino) con `doMove()` y se reliza el cambio de estado `_state++` o `***_state=1***`.*
- *****_state=1***:** *En este estado el agente se encuentra en el host remoto (o container destino) y se verifica si:*
 - * La variable de lectura **count**>0 quiere decir que el buffer esta lleno y hay bytes que escribir en el archivo remoto. Se escriben con la funcion **write()** y luego se realida el **doMove()** nuevamente hacia la maquina local y se vuelve al estado anterior de la maquina de estados `***_state--` o `***_state=1***`.
 - * La variable de lectura **count**<0 quiere decir que el buffer esta vacio y termino de leer el archivo local y no hay nada que escribir remotamente. Se procede a cambiar de estado la maquina de estado `***_state++` o `***_state=2***`.
- *****_state=2*:** *Si se llega a este estado quiere decir que ya no hay mas escribir en el archivo remoto. Se procede a eliminar el agente con el metodo `doDelete()`.*

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 descomprimos 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 diferencia más evidente radica en el caso de la arquitectura cliente/servidor tenemos dos aplicaciones que conforman el sistema distribuido, mientras en el caso de agentes, es sólo una.

Por ello es que en el paradigma cliente/servidor, el problema es el protocolo de comunicaciones mientras que en el paradigma agentes el problema radica en el desarrollo de una máquina de estados que nos permita monitorear la movilidad de los agentes.

En relación al problema abordado (RFS) se distinguen algunas de las ventajas y desventajas de la implementación de una solución basada en agentes.

Ventajas

- No es necesario establecer un protocolo o interfaz entre cliente y servidor.
- La programación asíncrona es inherente al paradigma.
- En la mayoría de los casos (en nuestro caso no lo hemos probado) reduce el tráfico de red.

Desventajas

- Puede complicarse notablemente la gestión y escalabilidad en aplicaciones multiagente con requerimientos de comunicación entre ellos.
- La movilidad y ejecución de código remoto genera problemas de seguridad adicionales que pueden resultar difícil gestionar

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 Jade

- En ambos casos el “cliente” (host que consume el servicio RFS), ejecuta código en un host remoto.
- Existe bibliografía que propone implementaciones de agentes basadas en RMI (<http://ocw.uc3m.es/ingenieria-informatica/desarrollo-de-aplicaciones-distribuidas/materiales-de-clase/paradigmas>)

Diferencias entre RMI y Jade

- RMI ejecuta código que se encuentra en una máquina remota (el servidor), pero la complejidad de la programación del servicio se encuentra en ese host remoto.
- Jade, tiene la posibilidad de migrar su código a otro host, ejecutarse y regresar al sitio de origen. (también permite la clonación de código), con lo cual la complejidad se encuentra en el consumidor del servicio
- Mientras que en RMI, tal como en todas las soluciones de tipo cliente/servidor, la carga de procesamiento está en un solo host, en Jade se distribuye. Además es una gran ventaja que el host remoto no deba tener instalada una aplicación adicional sino solo la plataforma, dado que genera mucha más transparencia de localización de los archivos

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.

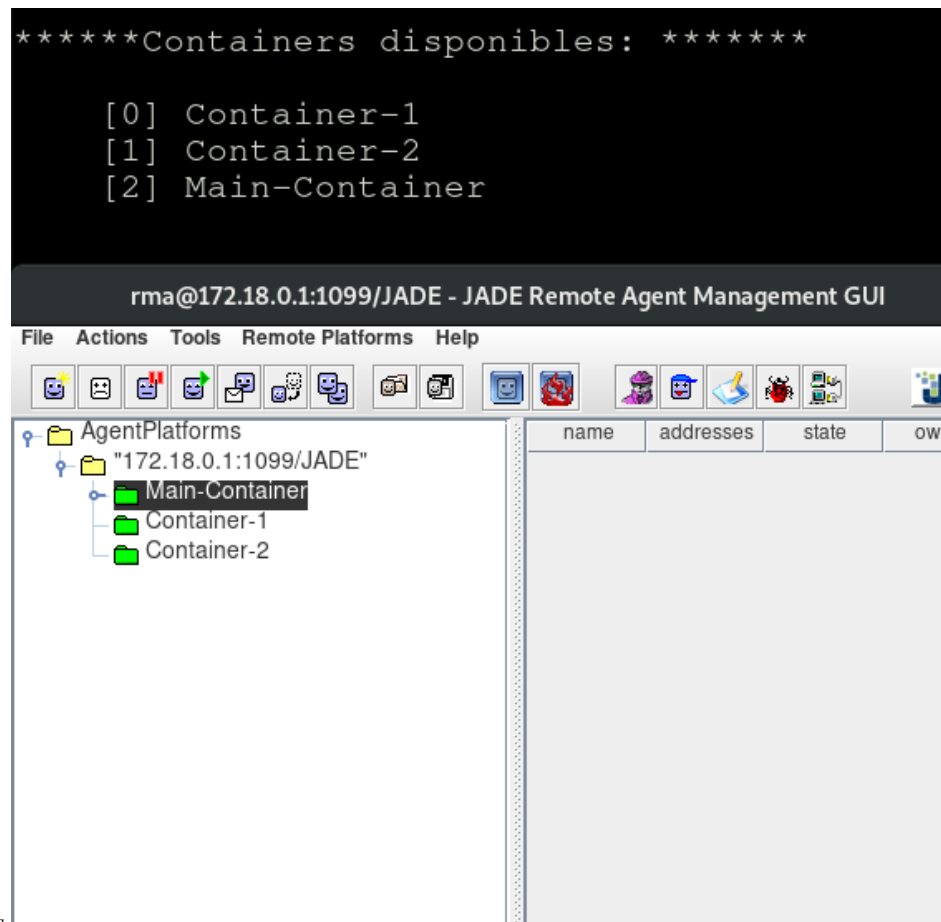
- **Opción 2:** El agente volvió al contenedor inicial, esto quiere decir que ya circulo por todos los contenedores, entonces debe imprimir en la salida estándar la información recolectada y terminar.

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 Impresión de un Host

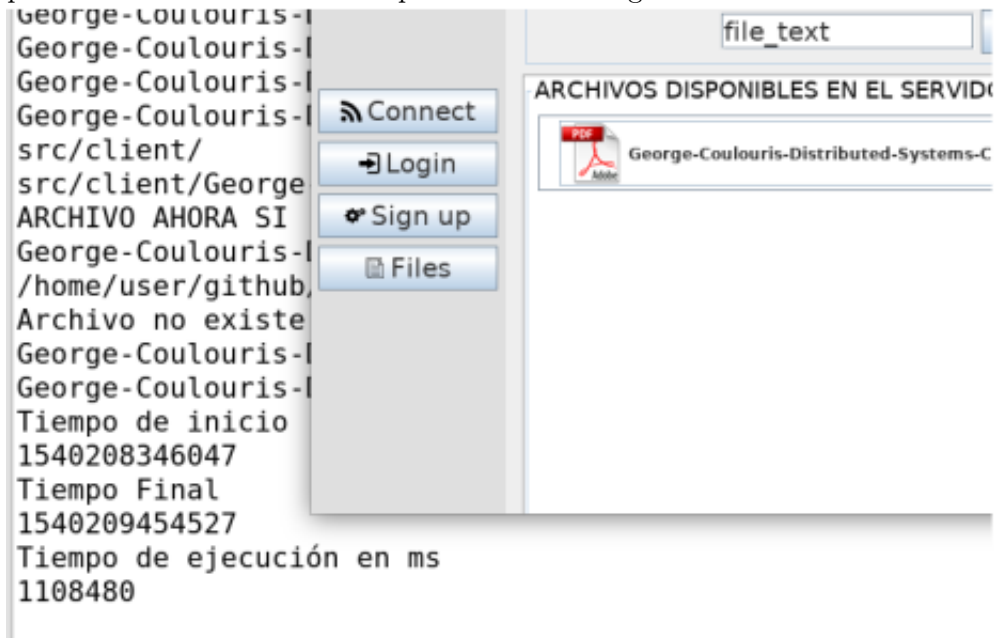
```
-----
Host Name: maxi
IP: 192.168.1.112
SO: Linux
CPU: 0.19457058914046257
Hora Llegada a Destino: Mon Dec 03 16:45:37 UTC 2018
Hora de Partida hacia el Origen: Mon Dec 03 16:45:42 UTC 2018
-----
```

1.3 3. Realizar y documentar los siguientes experimentos:

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

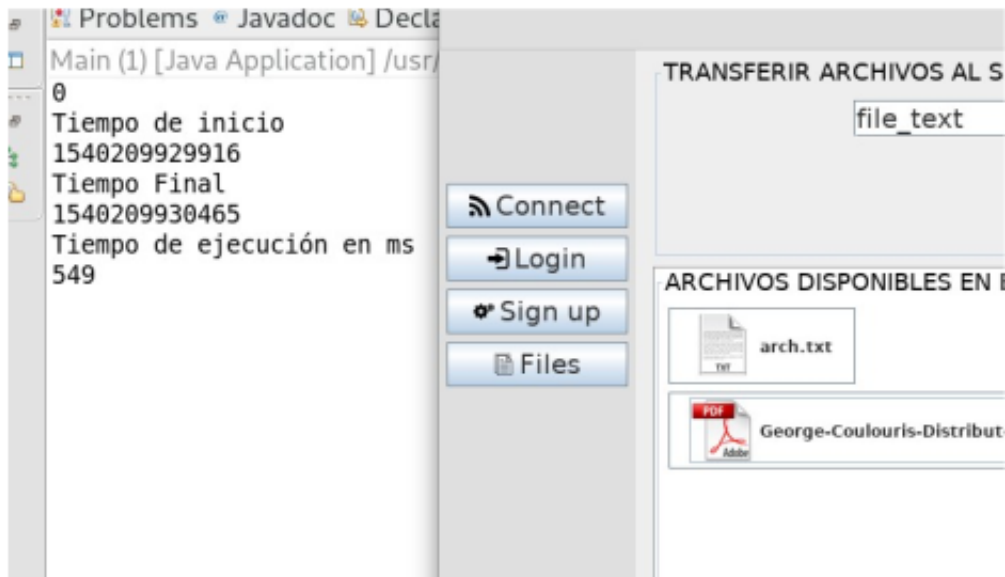
1.3.2 a) En la versión con Java Sockets del TP1.

Se utilizó el archivo pdf del libro "Distributed Systems Concepts and Design" de George Coulouris, que pesa 11,1MB y se utilizó un buffer de 1024 bytes. Como puede verse en la imágenes a continuación, la aplicación con sockets tardó un poco más de 11 seg en transferir el archivo localmente.



1.3.3 b) En la versión Java RMI del TP2.

Se utilizó el archivo pdf del libro "Distributed Systems Concepts and Design" de George Coulouris, que pesa 11,1MB y se utilizó un buffer de 1024 bytes. Como puede verse en la imagen a continuación, la aplicación con rmi demoró solamente 549 ms.



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

Se utilizó el archivo pdf del libro "Distributed Systems Concepts and Design" de George Coulouris, que pesa 11,1MB y se utilizó un buffer de 1024 bytes. Como puede verse en la imágenes a continuación, la aplicación con agentes tardó aproximadamente 5 segundos.

```

No hay mas para Leer
Tiempo de fin = 1543884560137
Tiempo Total de Ejecucion = 55597
destruye al agente --> ReadAgent@172.18.0.1:1099/JADE

```

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 nodo 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 panel 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)

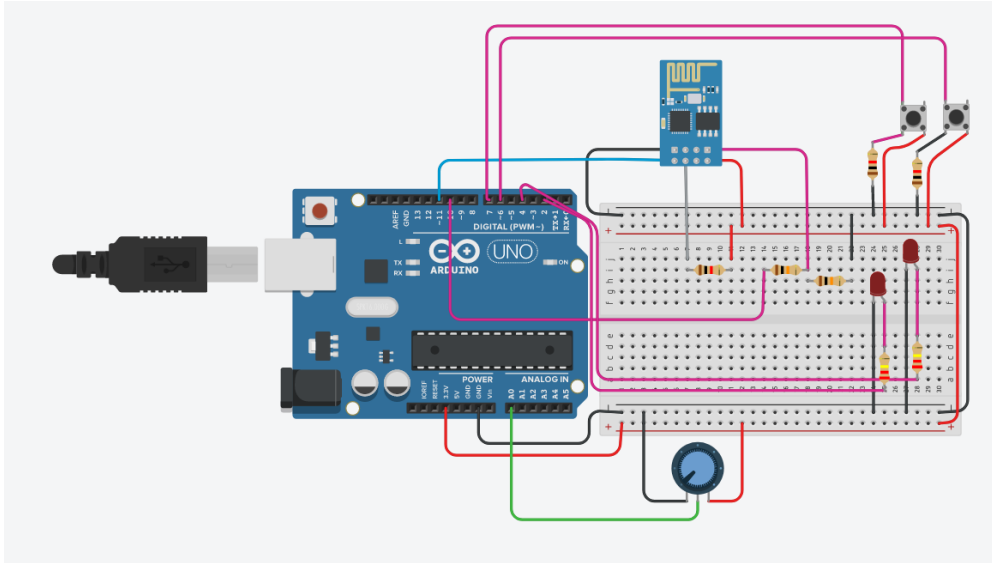
Los nodos se importan con **Ctrl+i** y pegando el contenido del archivo json correspondiente en el modal. Luego presionar ***Done*** y luego ***Deploy***. Finalmente las interfaces web estarán disponibles en las rutas:

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

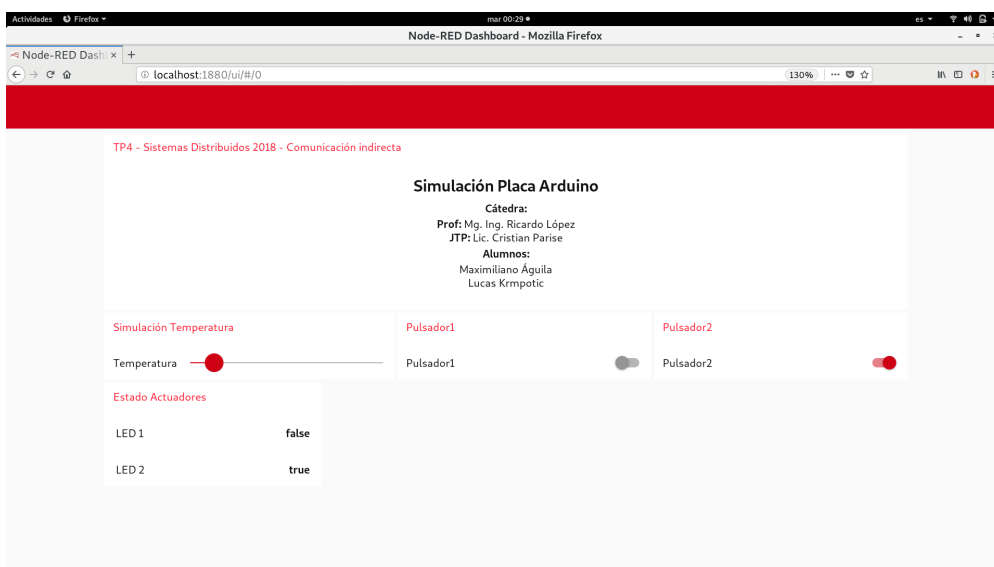
Diseño Según los requerimientos de la consigna la placa Arduino podría tener el siguiente esquema de conexiones:

- 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



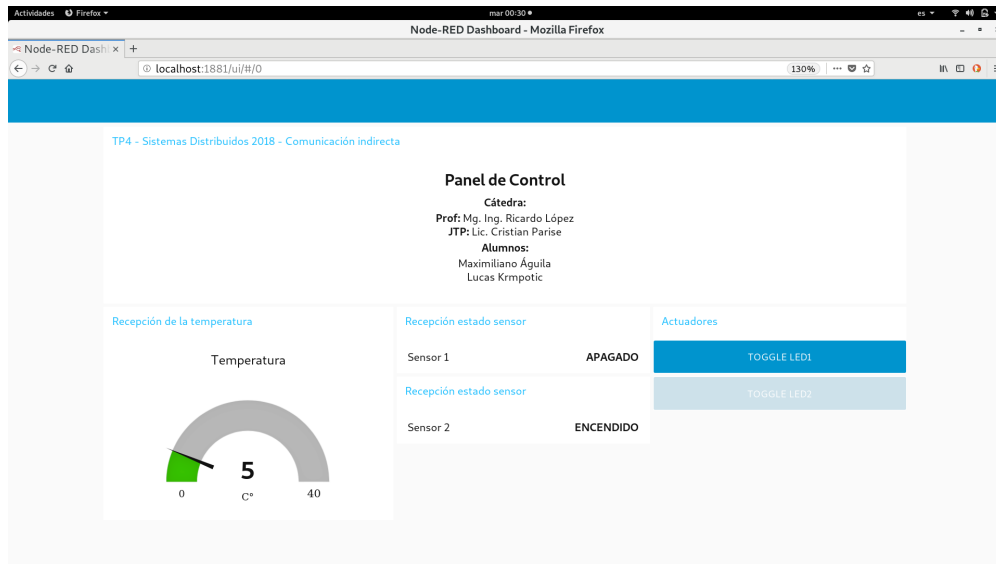
La correspondencia de este diseño arduino con la UI-web se puede ver en la siguiente imagen



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:



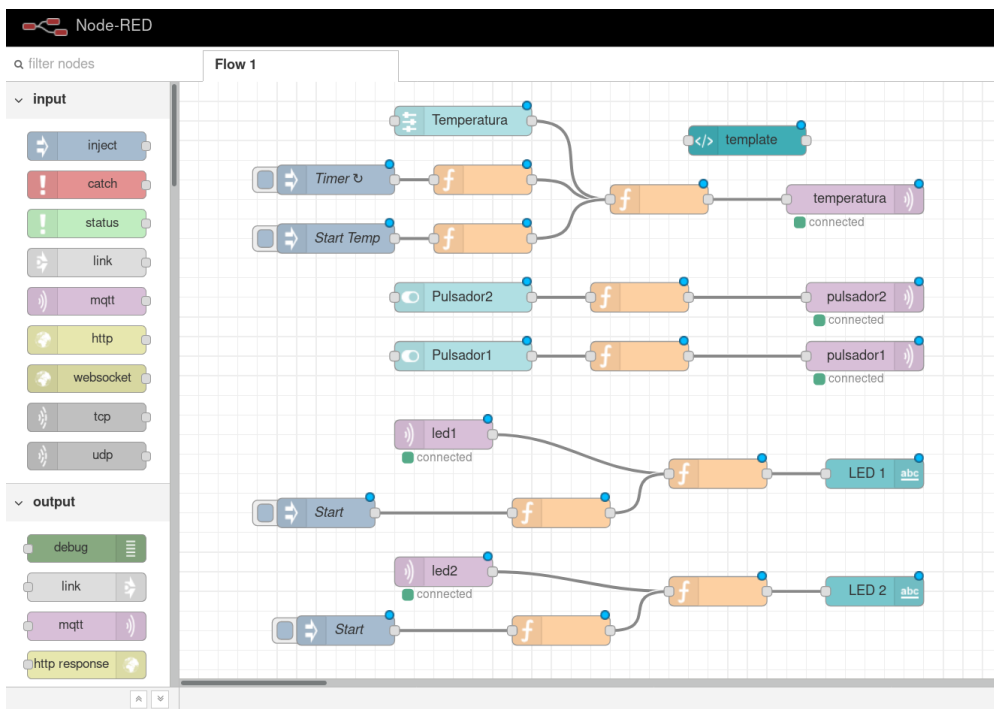
Esquema de tópicos y programación de los nodos En sistemas grandes puede resultar conveniente la utilización de un protocolo de aplicación (MARA por ejemplo) para publicar un conjunto amplio de variables sobre un conjunto limitado de tópicos, lo cierto es que en nuestro caso el problema puede resolverse 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 timestamp (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 slider 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 caída 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
        }
    }
}

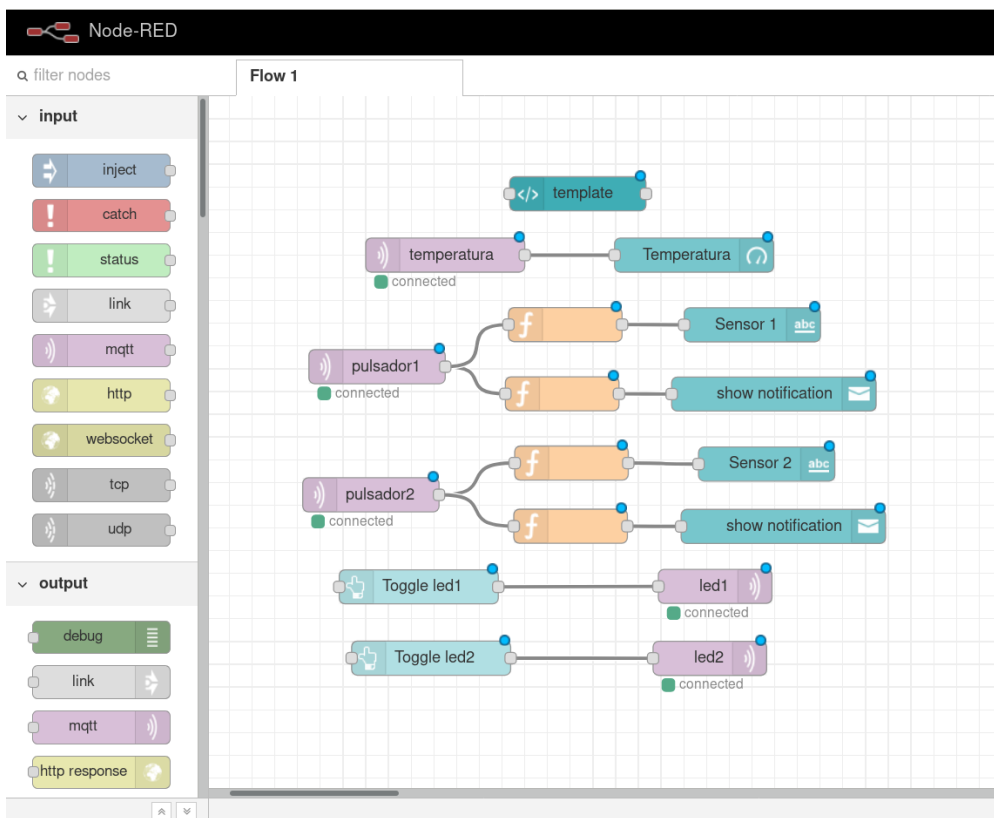
// si el payload es true es evento del timer
if (msg.payload === true){

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

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

Aplicación Panel de Control Esta aplicación es sumamente 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 sensores 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 comportamiento 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 tópico “*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 escribirán 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 mqttt

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

        class MQTTSniffer():

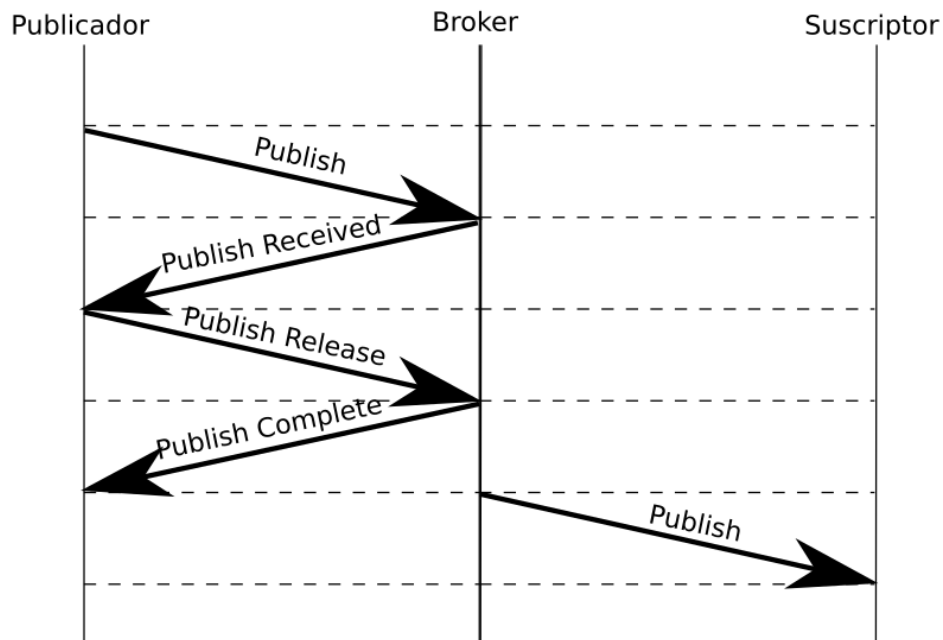
            def __init__(self):
                self.packets = []

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

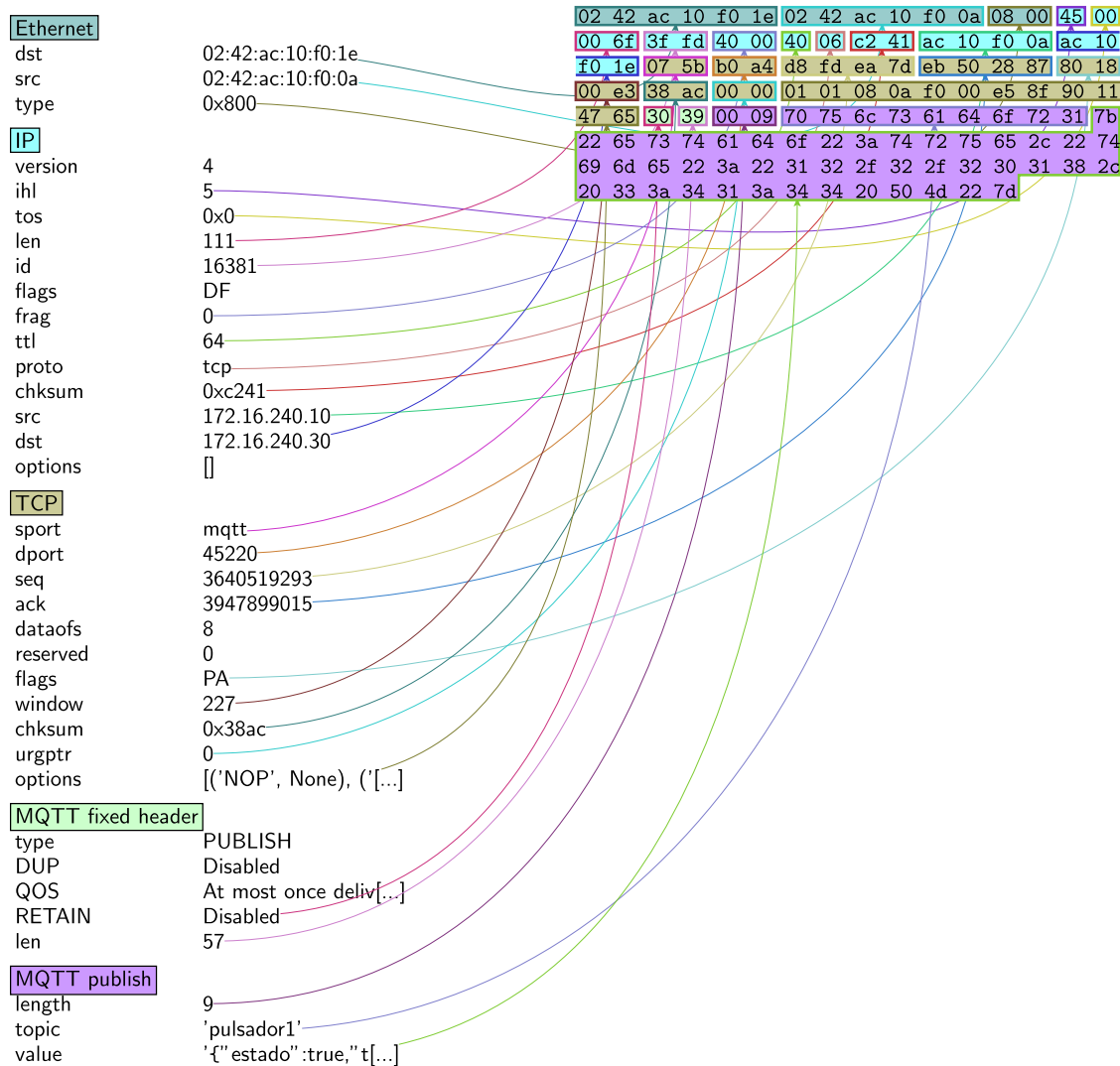
Finalmente planteamos la disección del último paquete que corresponde al envío del mensaje desde el broker al suscriptor. En la imagen podemos ver que el campo QoS tiene el valor **At most once delivery**, versión verbosa de QoS=0.

```

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

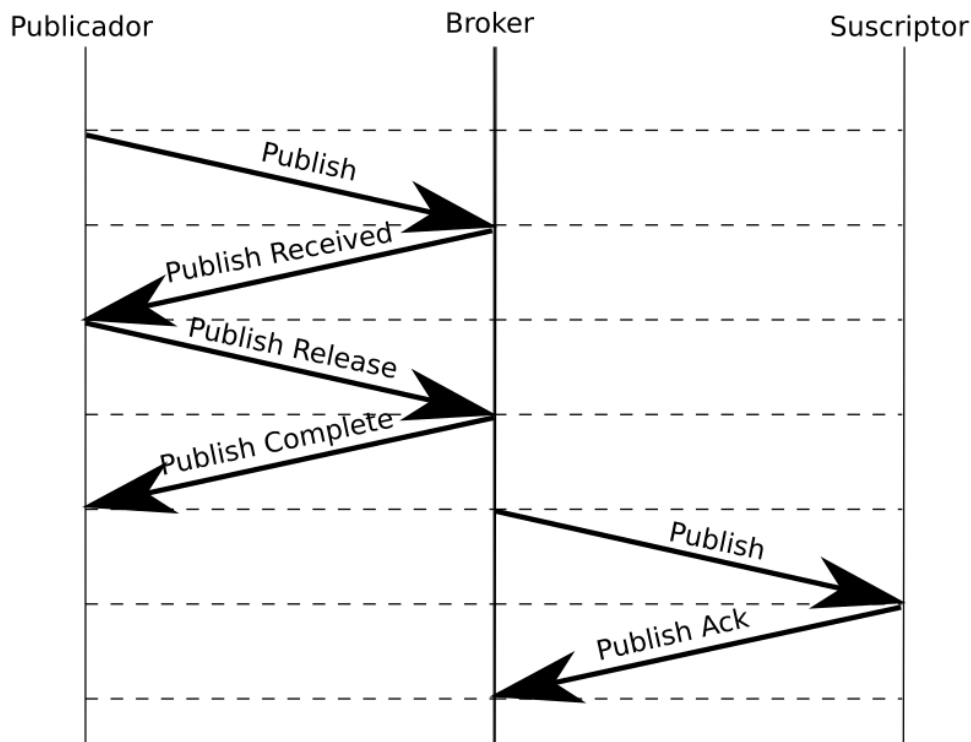
Ignoring line 5841 in mapping file 'psfonts.map': Unknown token '<DSSerif-Bold'
 Ignoring line 5843 in mapping file 'psfonts.map': Unknown token '<DSSerifUni-Bold'
 Ignoring line 5841 in mapping file 'psfonts.map': Unknown token '<DSSerif-Bold'
 Ignoring line 5843 in mapping file 'psfonts.map': Unknown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

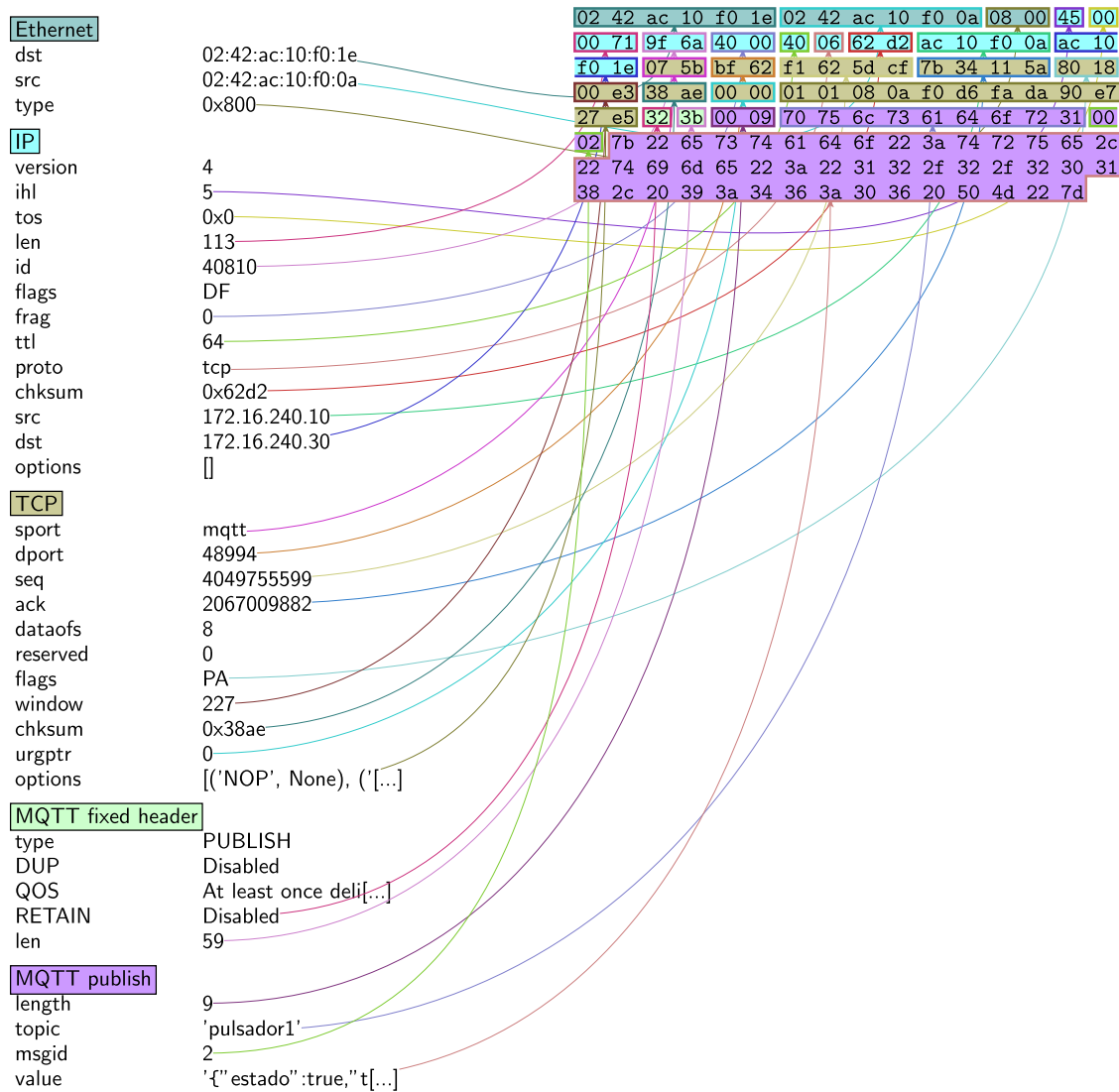
Nuevamente el escenario es el esperado y en la imagen del mensaje ***“Publish”*** desde el broker al suscriptor podemos comprobar que el campo QoS tiene el valor ***“At least once delivery”*** (QoS=1).

```

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

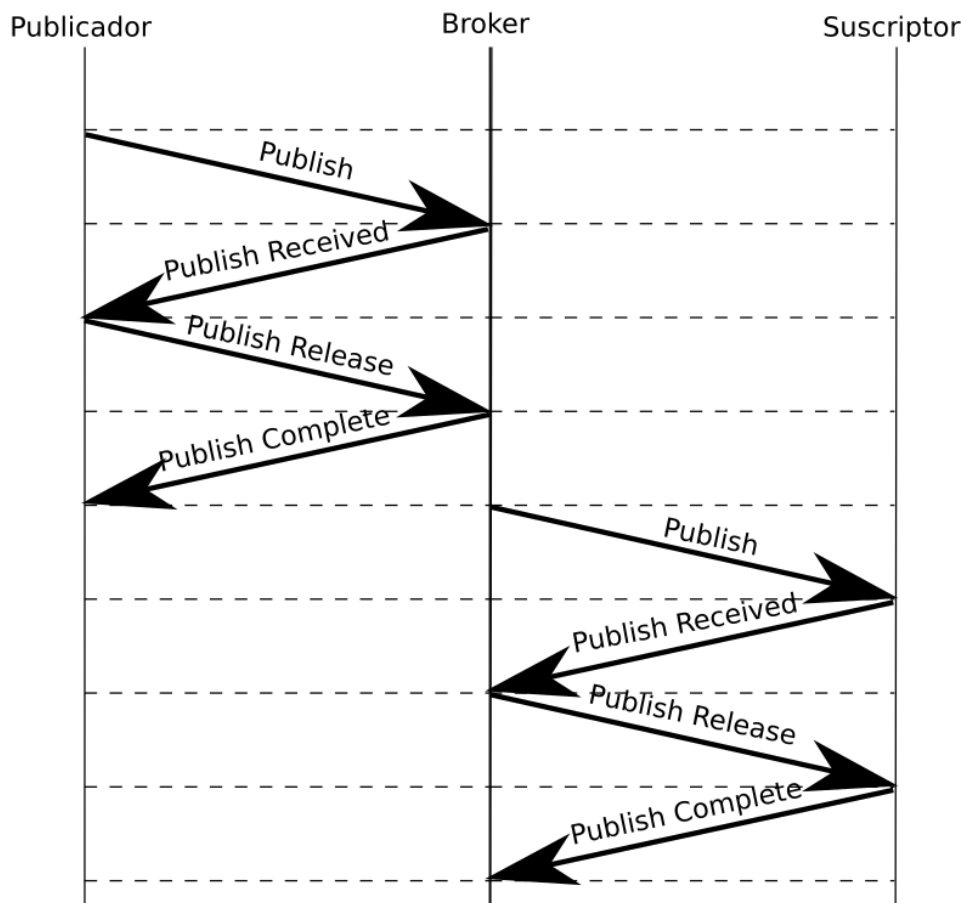
Ignoring line 5841 in mapping file 'psfonts.map': Unknown token '<DSSerif-Bold'
Ignoring line 5843 in mapping file 'psfonts.map': Unknown token '<DSSerifUni-Bold'
Ignoring line 5841 in mapping file 'psfonts.map': Unknown token '<DSSerif-Bold'
Ignoring line 5843 in mapping file 'psfonts.map': Unknown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

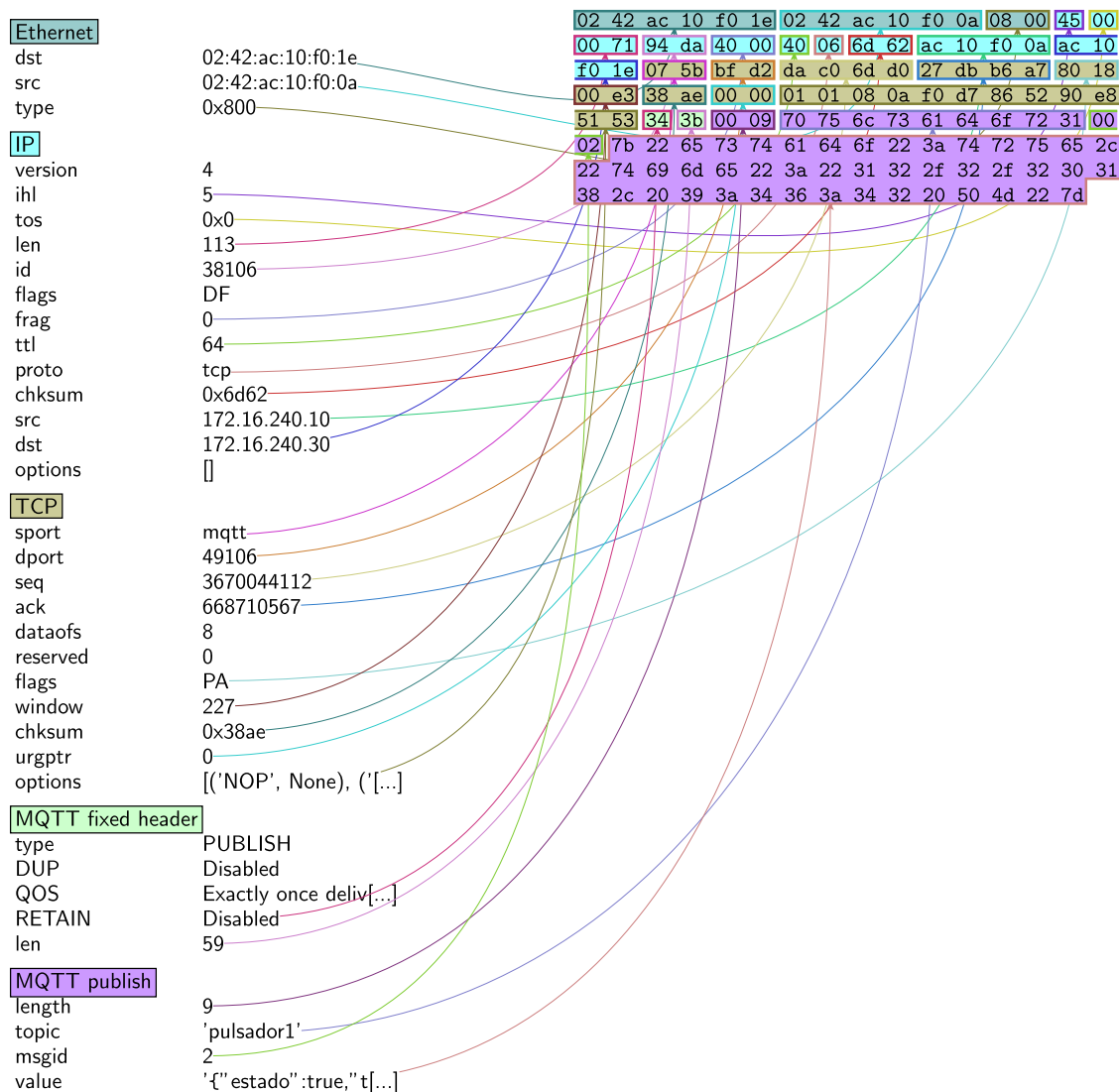
Efectivamente 8 paquetes han sido capturados y el ploteo del mensaje muestra que el campo QoS tiene el valor ***“Exactly once delivery”*** (QoS=2).

```

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'

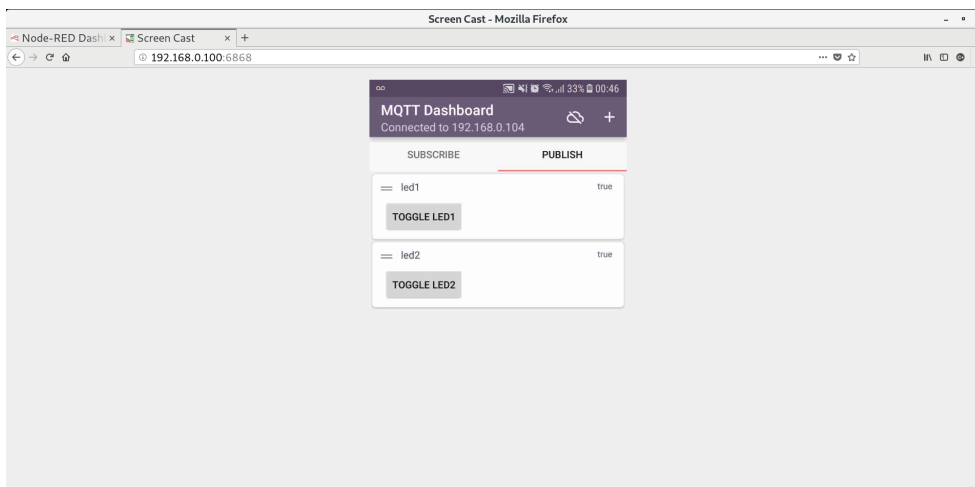
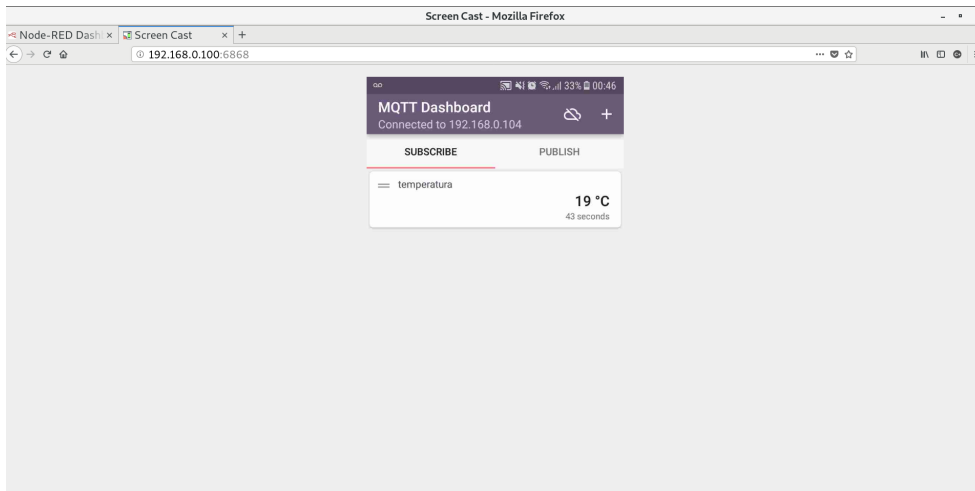
Out[3]:



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 inciso 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/suscripción en los tópicos.