Comentario técnico: CTC-096
Componente: **RPC de Mongoose-OS por MQTT con ESP32**
Autor: Sergio R. Caprile, Senior R&D Engineer

Revisiones	Fecha	Comentarios
0	15/05/20	

Mongoose-OS incorpora una API del tipo RPC (Remote Procedural Call) que podemos utilizar a través de diversos transportes. Entre ellos disponemos de MQTT. La utilización de RPC sobre MQTT resulta ideal para control y monitoreo de un dispositivo en situaciones en las que no disponemos de un acceso por red al mismo ya que se encuentra detrás de un firewall, o incluso para desacoplar la aplicación. Todo lo que hemos desarrollado en comentarios técnicos anteriores es válido también para este transporte, por lo que dispondremos además de autorización y encripción TLS si así lo deseamos.

Bidireccionalidad

Hemos visto en CTC-086 y CTC-087 diversos esquemas de red y como MQTT nos permitía desacoplar la aplicación y concentrarnos en bloques independientes. Comentamos además que se trataba de un esquema en esencia unidireccional pero que había algunas alternativas interesantes. Pues bien, Mongoose-OS incorpora un esquema en el que el dispositivo se suscribe a un tópico donde escucha mensajes que transportan pedidos de RPC. Dentro de dicho mensaje, un parámetro especifica el tópico donde el dispositivo publicará la respuesta, de modo que el interesado pueda obtenerla. De este modo, es posible establecer un diálogo donde el controlador publica en un tópico en el que el dispositivo está subscripto, y se subscribe a un tópico el cual a su vez le indica al dispositivo para que éste le responda publicando en dicho tópico.

Configuración

Configuramos el ESP32 con Mongoose-OS para tener un cliente MQTT y conectarse como cliente a una red WiFi. Dado que debemos conectarnos a un servidor, ésta nos resulta la forma tal vez más rápida y simple de realizar las pruebas y explicar la operación. La configuración puede realizarse manualmente mediante RPC, en un archivo de configuración JSON; o definirla en el archivo YAML que describe el proyecto. Para las pruebas elegimos esta última opción.

```
libs:
  - origin: https://github.com/mongoose-os-libs/mqtt        # incorpora el cliente MQTT
  - origin: https://github.com/mongoose-os-libs/rpc-mqtt    # provee soporte RPC sobre MQTT
config_schema:
  - ["mqtt.enable", true]                    # Habilita el cliente MQTT
  - ["mqtt.server", "address:port"]          # Dirección IP del broker a utilizar
```

El puerto comúnmente utilizado es 1883. El servidor puede además solicitar que utilicemos nombre de usuario y password, los detalles de la configuración completa los podemos encontrar en la página de Mongoose-OS[1].
Si deseamos agregar encripción, deberemos seguir los lineamientos que vimos en el CTC-093 para agregar soporte TLS al cliente MQTT.
Si deseamos agregar autorización a las RPC, nos referiremos al CTC-094 para ello.

1 https://mongoose-os.com/docs/mongoose-os/api/net/mqtt.md

Operación

Luego de compilado el código (`mos build`) y grabado el microcontrolador (`mos flash`) mediante *mos tool*, observaremos en el log si todo funciona como debe, o los errores que se hayan producido. Recordemos que debemos configurar las credenciales para conectarnos por WiFi a nuestra red (SSID y clave) y la dirección y port del broker MQTT que vayamos a utilizar.

```
[May 15 14:33:48.882] mgos_mqtt_conn.c:471    MQTT0 connecting to 192.168.5.1:1883
[May 15 14:33:48.911] mgos_mqtt_conn.c:231    MQTT0 TCP connect ok (0)
[May 15 14:33:48.987] mgos_mqtt_conn.c:275    MQTT0 CONNACK 0
[May 15 14:33:48.995] init.js:34              MQTT connected
[May 15 14:33:49.009] init.js:26              Published:OK topic:/this/test/esp32_807A98 msg:CONNECTED!
[May 15 14:33:49.014] mgos_mqtt_conn.c:212    MQTT0 sub esp32_807A98/rpc/# @ 1
[May 15 14:33:49.019] mgos_mqtt_conn.c:212    MQTT0 sub esp32_807A98/rpc @ 1
```

Observemos en el log los tópicos en los que el dispositivo se ha suscripto, allí le enviaremos los mensajes.
A continuación, nos conectaremos al broker y realizaremos la operatoria que indicamos:

- nos suscribimos al tópico que elegimos para recibir las respuestas
- publicamos un mensaje en el tópico donde escucha el dispositivo; un objeto JSON conteniendo el pedido, e indicando que responda en el tópico anterior
- recibiremos un mensaje conteniendo un objeto JSON con la respuesta a nuestro pedido

Por ejemplo, queremos pedir el estado del dispositivo esp32_807A98[2]; para ello usaremos `Sys.GetInfo`. Entonces, luego de suscribirnos a (por ejemplo) */somewhere/#*, publicamos el siguiente objeto en el tópico *esp32_807A98/rpc*:

```
{
  "src": "/somewhere",
  "id": 123,
  "method": "Sys.GetInfo"
}
```

donde *id* es un identificador para asociar la respuesta, *method* es la RPC que queremos llamar y *src* es el tópico donde el dispositivo debe publicar la respuesta.
En respuesta el dispositivo publicará en el tópico indicado (*/somewhere*), bajo el subtópico configurado (que por defecto es */rpc*), y como nos hemos suscripto a ese tópico y todos sus subtópicos, lo recibiremos como un mensaje:

```
{
  "id": 123,
  "src": "esp32_807A98",
  "result": {
    "id": "esp32_807A98",
    "app": "rpcws",
   [...]
  }
}
```

donde *result* es el objeto que contiene la respuesta a nuestro pedido.

La captura a continuación muestra como podemos simular este diálogo mediante los clientes del broker (como los que provee Mosquitto):

2 Por defecto el nombre se forma con la directiva *esp32_??????*, donde los seis signos de interrogación son reemplazados por los correspondientes últimos dígitos de la MAC del dispositivo. Esto es configurable. Podemos observarlo en el log o por ejemplo podemos configurar al dispositivo para publicar algo al inicializar y así observaremos su identificador.

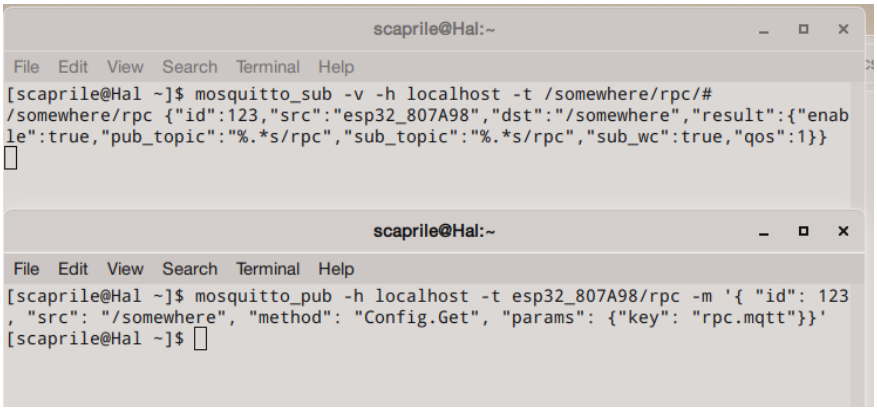
Este esquema de RPC puede además transportar parámetros dentro de un objeto JSON de nombre *params*, con las claves que defina la implementación del método a utilizar. Si por ejemplo llamamos a `Config.Get`, definida por el servicio *service-config*[3], y le pedimos una parte en particular de la configuración:

```
"params": {
  "key": "rpc.mqtt"
}
```

El objeto completo es entonces:

```
{
  "src": "/somewhere",
  "id": 123,
  "method": "Config.Get",
  "params": {
    "key": "rpc.mqtt"
  }
}
```

La captura a continuación muestra una simulación de este diálogo:



En los archivos provistos hemos incluido ejemplos de esta operación:

- JavaScript en *node.js*, requiere el módulo *mqtt*[4]

3 https://mongoose-os.com/docs/mongoose-os/api/rpc/rpc-service-config.md
4 https://www.npmjs.com/package/mqtt

- Python, requiere el módulo *paho-mqtt*[5] y dependiendo de la versión, algún módulo JSON[6]

Como hemos indicado en el CTC-094, podemos escribir nuestras propias RTC tanto en JavaScript (mJS) como en C, enviar información como parámetros y recibir las respuestas.

A los fines didácticos como hemos mostrado, o para ayudar en el desarrollo, podemos utilizar los clientes del broker (como los que provee Mosquitto), tanto para observar el tráfico como para generar los objetos JSON; por ejemplo:

```
$ mosquitto_sub -v -h 192.168.5.3 -t /#  --cafile ca.crt --cert sandboxclient.crt --key
sandboxclient.key –insecure

$ mosquitto_pub -h 192.168.5.3 -t esp32_807A98/rpc -m '{ "id": 123, "src": "/somewhere",
"method": "Config.Get", "params": {"key": "rpc.mqtt"}}' --cafile ca.crt --cert
sandboxclient.crt --key sandboxclient.key –insecure
```

Los parámetros en bastardilla corresponden a lo que debe agregarse para la operación bajo SSL, que de otro modo en principio no podríamos visualizar. En particular, *insecure* es una solución rápida debido a que utilizamos nuestra propia CA, como explicamos en CTC-093.
En realidad, en un ambiente de pruebas el cliente en una computadora no necesariamente debe usar la conexión autenticada, es posible tener un ambiente donde los dispositivos que generan los datos se conectan por MQTT sobre TLS y las PCs y navegadores que los consumen usan otro tipo de conexión, incluso abierta. En general, para una aplicación en la vida real, es deseable minimizar la posibilidad de que equipos no autorizados accedan a la información o puedan generar datos inválidos, por lo cual es poco probable disponer de un acceso abierto.

La herramienta *mos tool* soporta el transporte por MQTT y MQTT sobre TLS. Podemos operar sobre un dispositivo indicando el transporte a emplear con el parámetro *port*, uniendo el tópico al URL de la siguiente forma:

```
$ mos --port mqtt://192.168.5.1/esp32_807A98 call Sys.GetInfo
$ mos --port mqtts://192.168.5.3/esp32_807A98 --cert-file sandboxclient.crt --key-file
sandboxclient.key call Sys.GetInfo
```

Si las RPC se configuran para usarse con autorización, deberemos agregar a esto la implementación de Digest Authentication, tal como la detallamos en el CTC-094. En los archivos provistos hemos incluido también ejemplos para dichos lenguajes de programación:

- JavaScript en *node.js*, requiere el módulo *md5*
- Python, requiere los módulos *hashlib* y *random*

De igual modo, la herramienta *mos tool* la utilizaremos también como indica el CTC-094, agregando el parámetro *port* como ya comentamos.

Configuración adicional

Los tópicos en los que el dispositivo recibe pedidos de RPC y publica las respuestas son configurables, podemos dejarlos como subtópicos del identificador de dispositivo y lo provisto en *src*, respectivamente, o configurarlos como un tópico fijo.
Si configuramos, por ejemplo, "*pub_topic*": "*/algo*", no se requiere enviar el parámetro *src*, la respuesta se publica en el tópico especificado (*/algo*). Caso contrario, si lo dejamos como subtópico, si omitimos el parámetro *src* observaremos un error en el log:

5 https://pypi.org/project/paho-mqtt/
6 Por ejemplo *simplejson*, https://simplejson.readthedocs.io/

```
[May 15 16:09:46.678] mgos_rpc_channel_mq:129 Cannot reply to RPC over MQTT, no dst:
[{"id":123,"src":"esp32_807A98","result":
{"enable":true,"pub_topic":"%.*s/rpc","sub_topic":"%.*s/rpc","sub_wc":true,"qos":1}}]
```

Dichos parámetros los podemos observar si pedimos la parte de la configuración correspondiente como hicimos en el ejemplo que deriva en la segunda captura de imagen. Lo repetimos aquí de forma genérica:

```
$ mos config-get rpc.mqtt
{
  "enable": true,
  "pub_topic": "%.*s/rpc",
  "qos": 1,
  "sub_topic": "%.*s/rpc",
  "sub_wc": true
}
```

Como vemos, también podemos configurar el nivel de calidad de servicio a utilizar.