Comentario técnico: CTC-095
Componente: **RPC de Mongoose-OS por WebSocket con ESP32**
Autor: Sergio R. Caprile, Senior R&D Engineer

Revisiones	Fecha	Comentarios
0	12/05/20	

Mongoose-OS incorpora una API del tipo RPC (Remote Procedural Call) que podemos utilizar a través de diversos transportes. El servidor HTTP que provee también incluye soporte para WebSocket, por lo que la utilización de RPC sobre WebSocket resulta ideal para la implementación de interfaces de usuario en dispositivos sin visualización, en situaciones en las que disponemos de acceso por red al dispositivo. 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.

WebSocket

Se trata de un protocolo binario de comunicación full-duplex. Su principal ventaja aparece por oposición a HTTP, que es “coloquial” en el sentido que los mensajes son texto legible por un humano y half-duplex: debe establecerse una conexión, pedir un elemento, y éste es devuelto. La comparación es válida en tanto que WebSocket no compite sino que amplía HTTP en el sentido que un navegador puede solicitar a un servidor web el “upgrade” de la conexión a WebSocket, con el fin de hacer la comunicación más fluida y evitar no sólo el parsing de HTTP sino toda la serie de establecimientos de conexiones y posteriores desconexiones, a la vez que el navegador puede enviar información al servidor y hacerlo en forma simultánea. Como podemos imaginar, esto resulta mucho más eficiente a la hora de implementar interfaces de usuario en relación con otros esquemas como AJAX. Esto obviamente, dado que el dispositivo opera como servidor, requiere disponer de un acceso por red al mismo, es decir, si pretendemos operar la interfaz a través de la Internet deberemos instrumentar el networking necesario, cosa que no sucede en un esquema desacoplado como MQTT. Una característica interesante adicional es que WebSocket sigue funcionando por los mismos puertos que HTTP y HTTPS, y es soportado por muchos esquemas de proxy, lo cual significa que no dificulta el atravesar esquemas de seguridad tradicionales y hace relativamente simple la implementación del networking necesario, en oposición a otras alternativas tal vez más modernas.
El protocolo además soporta múltiples canales sobre la misma conexión, de modo de hacer del intercambio de diversos tipos de información entre ambas entidades, algo más fluido.

Configuración

Configuramos el ESP32 con Mongoose-OS para operar como un Access Point con un web server. La operación como AP 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/http-server # requiere el web server
  - origin: https://github.com/mongoose-os-libs/rpc-ws      # provee RPC sobre WebSocket
```

Si deseamos agregar soporte para WSS (Secure WebSocket), deberemos seguir los lineamientos que vimos en el CTC-092 para agregar soporte TLS al servidor web.

Si deseamos agregar autorización a las RPC, nos referiremos al CTC-094 para ello.

Operación

Luego de compilado el código (`mos build`) y grabado el microcontrolador (`mos flash`) mediante *mos tool*, observaremos en el log la dirección del Access Point y el nombre.

```
[Apr 21 18:39:50.935] esp32_wifi.c:450        WiFi AP: SSID myESP_807A98, channel 6
[Apr 21 18:39:51.802] esp32_wifi.c:507        WiFi AP IP: 192.168.4.1/255.255.255.0 gw [...]
[Apr 21 18:39:51.813] mgos_http_server.c:343  HTTP server started on [80]
[Apr 21 18:39:51.880] init.js:6               ### init script started ###
```

A continuación, nos conectaremos a esa red con un cliente que soporte WebSocket. En los archivos provistos hemos incluido tres ejemplos:

- JavaScript en un navegador[1]
- JavaScript en *node.js*, requiere el módulo *websocket*[2]
- Python, requiere el módulo *websocket-client*[3] y dependiendo de la versión, algún módulo JSON[4]

En general lo que haremos será conectarnos al URL correspondiente al servicio de RPC en esa dirección, por ejemplo: *ws://192.168.4.1/rpc*. Establecida la conexión, intercambiaremos objetos JSON con el dispositivo. Por ejemplo, para pedir el estado del dispositivo usamos `Sys.GetInfo`, entonces enviamos:

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

donde *id* es un identificador para asociar la respuesta y *method* es la RPC que queremos llamar.
En respuesta recibiremos:

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

donde *result* es el objeto que contiene la respuesta a nuestro pedido.
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*[5], y le pedimos una parte en particular de la configuración, la estructura es:

```
"params": {
  "key": "wifi.ap"
}
```

1 Modificamos este ejemplo: https://github.com/mdn/samples-server/tree/master/s/websocket-chat
2 https://www.npmjs.com/package/websocket
3 http://pypi.python.org/pypi/websocket-client
4 Por ejemplo *simplejson*, https://simplejson.readthedocs.io/
5 https://mongoose-os.com/docs/mongoose-os/api/rpc/rpc-service-config.md

El objeto completo es entonces:

```
{
  "id": 123,
  "method": "Config.Get",
  "params": {
    "key": "wifi.ap"
  }
}
```

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.

La herramienta *mos tool* soporta el transporte por WebSocket, podemos operar sobre un dispositivo indicando el transporte a emplear con el parámetro *port*, de la siguiente forma:

```
$ mos --port ws://192.168.4.1/rpc call FS.List
$ mos --port wss://192.168.4.1/rpc call FS.List
```

Si las RPC se configuran para usar 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 un navegador
- 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.