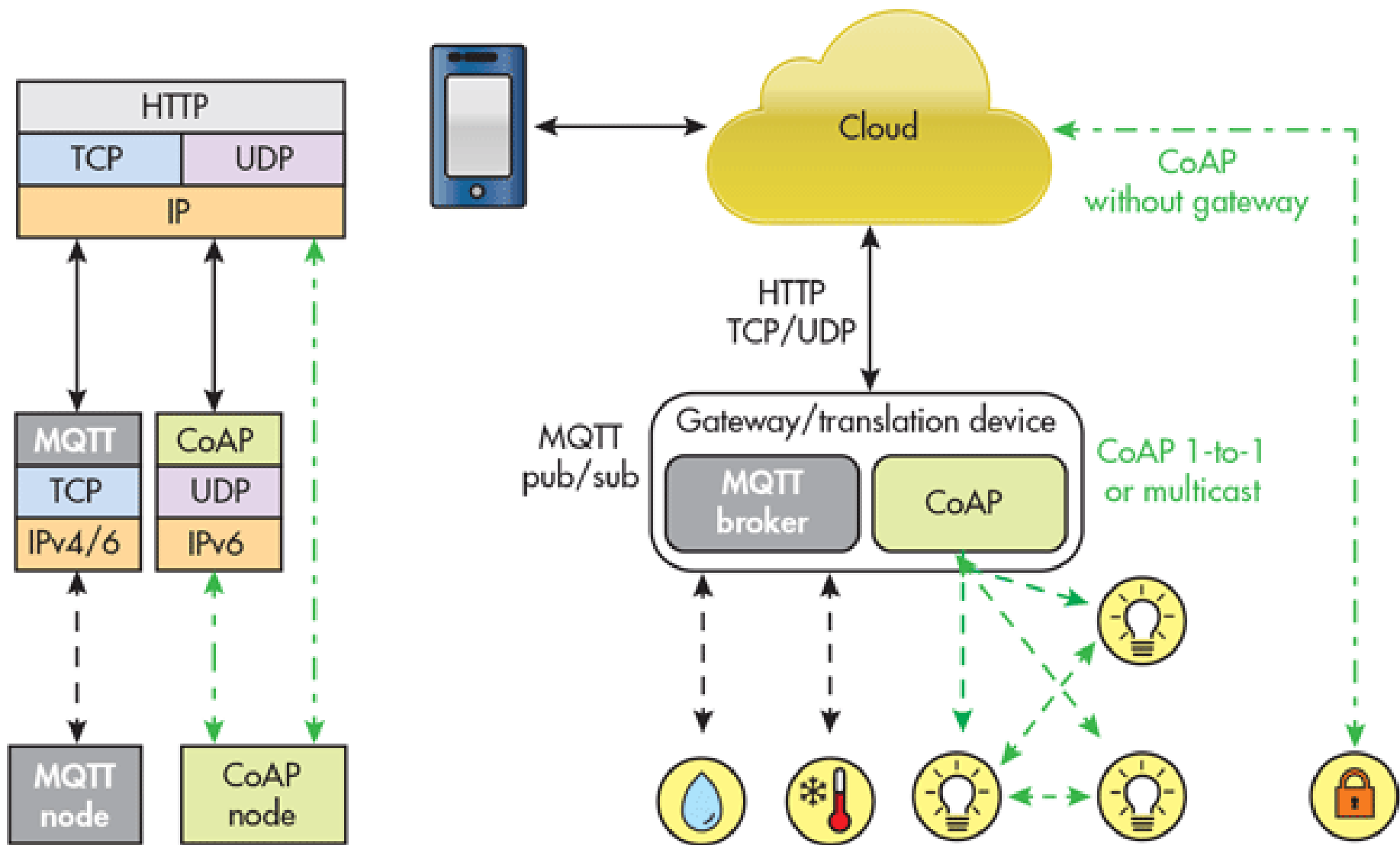




# CoAP and MQTT

Antonio Liñán Colina,  
Zolertia





# 03-coap

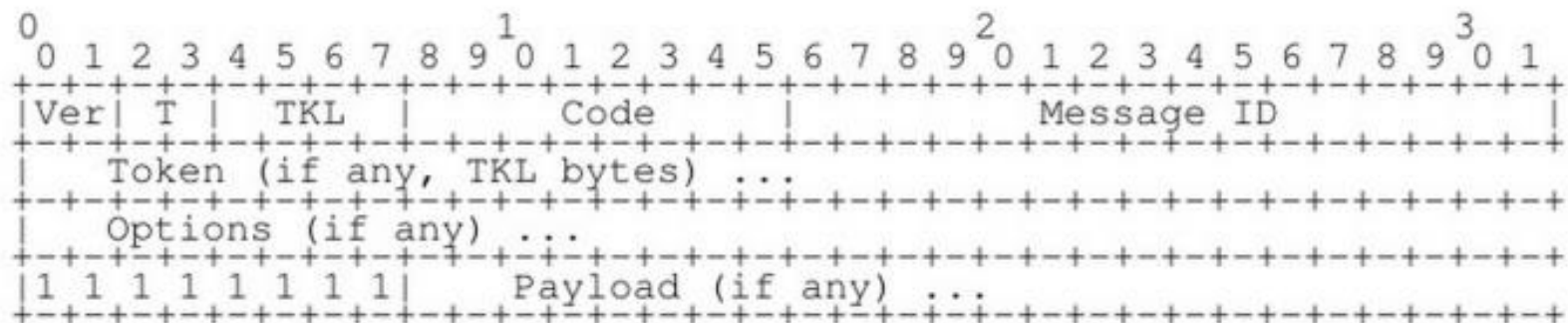
# CoAP

## RFC 7252 Constrained Application Protocol

“The Constrained Application Protocol (CoAP) is a specialized web transfer protocol for use with constrained nodes and constrained networks in the **Internet of Things**.

The protocol is designed for machine-to-machine (M2M) applications such as smart energy and building automation.”

- UDP-*reliable* (confirmable), SMS supported
- CoRE *Link-format* (GET /.well known/core)
- Client/Server
- IANA Registered (error codes, content format)
- Resource Discovery and asynchronous subscription
- Four-bytes compact header
- Multicast and one-to-one supported
- HTTP verbs GET, PUT, POST, DELETE
- HTTP-like header (*Options*)
- URI (Uniform Resource Identifier)



**Ver** - Version (1)

**T** - Message Type (Confirmable, Non-Confirmable, Acknowledgement, Reset)

**TKL** - Token Length, if any, the number of Token bytes after this header

**Code** - Request Method (1-10) or Response Code (40-255)

**Message ID** - 16-bit identifier for matching responses

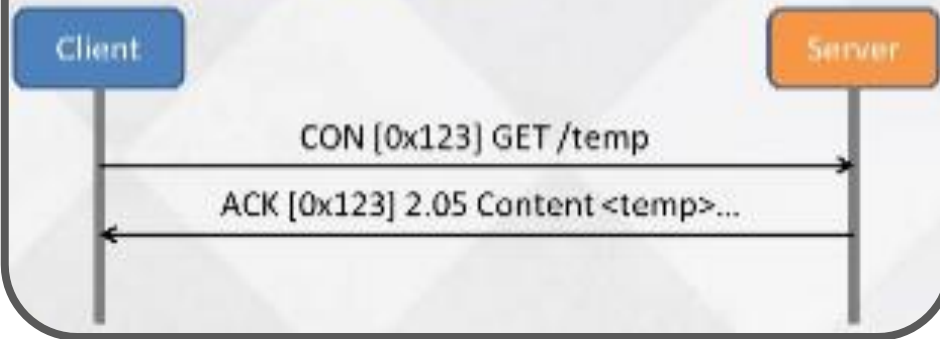
**Token** - Optional response matching token

# CoAP URI

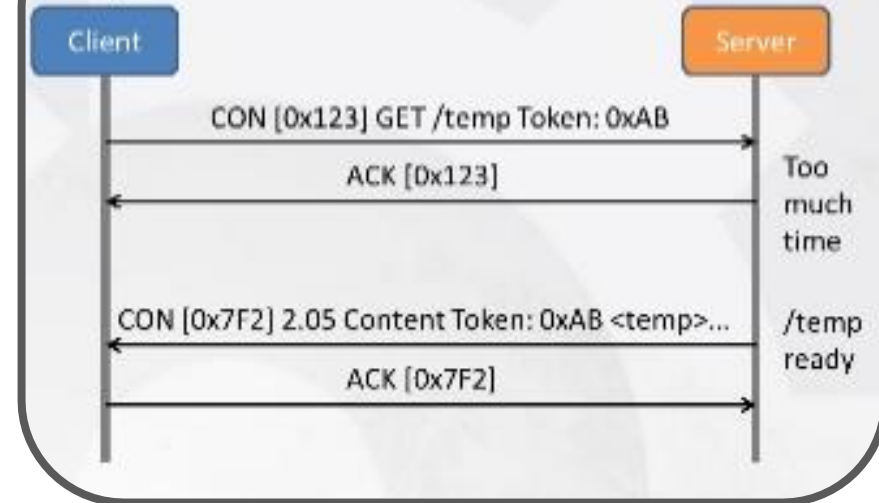
coap://[aaaa::c30c:0:0:1234]:5683/actuators/leds?color=b

Host	Port	Path	Query
------	------	------	-------

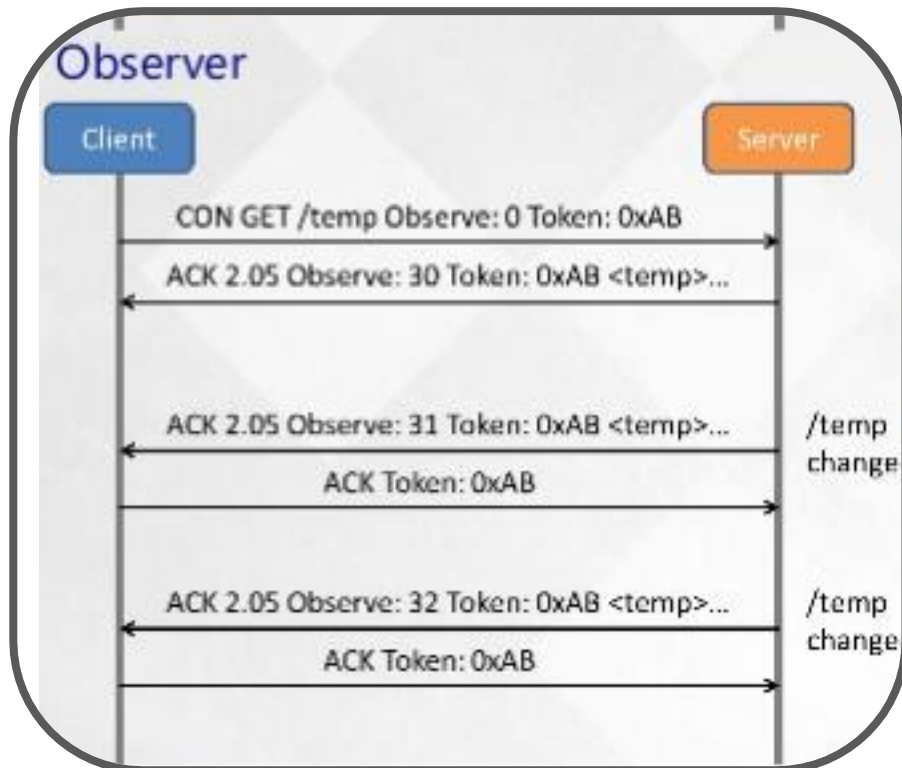
## Confirmable request



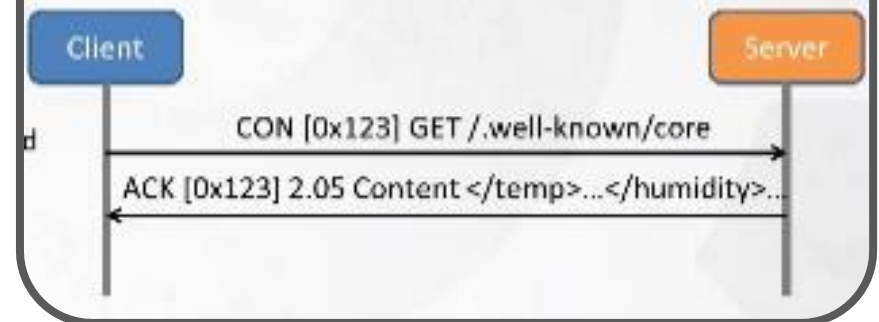
## Response back after a while



## Observer



## Resource discovery



A **normal resource** is defined by a static Uri-Path that is associated with a resource handler function. This is the basis for all other resource types.

```
#define RESOURCE(name, attributes, get_handler, post_handler, put_handler,  
    delete_handler) \  
    resource_t name = { NULL, NULL, NO_FLAGS, attributes, get_handler, post_handler,  
        put_handler, delete_handler, { NULL } }
```

A **parent resource** manages several sub-resources by evaluating the Uri-Path, which may be longer than the parent resource.

```
#define PARENT_RESOURCE(name, attributes, get_handler, post_handler, put_handler,  
    delete_handler) \  
    resource_t name = { NULL, NULL, HAS_SUB_RESOURCES, attributes, get_handler,  
        post_handler, put_handler, delete_handler, { NULL } }
```



If the server is not able to respond immediately to a **CON** request, it simply responds with an empty ACK message so that the client can stop re-transmitting the request. After a while, when the server is ready with the response, it sends the response as a **CON** message. The following macro allows to create a CoAP resource with **separate response**:

```
#define SEPARATE_RESOURCE(name, attributes, get_handler, post_handler, put_handler,
    delete_handler, resume_handler) \
    resource_t name = { NULL, NULL, IS_SEPARATE, attributes, get_handler,
        post_handler, put_handler, delete_handler, { .resume = resume_handler } }
```

An **event resource** is similar to a periodic resource, but the second handler is called by a non periodic event such as the pressing of a button.

```
#define EVENT_RESOURCE(name, attributes, get_handler, post_handler, put_handler,
    delete_handler, event_handler) \
    resource_t name = { NULL, NULL, IS_OBSERVABLE, attributes, get_handler,
        post_handler, put_handler, delete_handler, { .trigger = event_handler } }
```

If we need to declare a **periodic resource**, for example to poll a sensor and publish a changed value to subscribed clients, then we should use:


```
#define PERIODIC_RESOURCE(name, attributes, get_handler, post_handler, put_handler,  
    delete_handler, period, periodic_handler) \  
    periodic_resource_t periodic_##name; \  
    resource_t name = { NULL, NULL, IS_OBSERVABLE | IS_PERIODIC, attributes,  
    get_handler, post_handler, put_handler, delete_handler, { .periodic =  
    &periodic_##name } }; \  
    periodic_resource_t periodic_##name = { NULL, &name, period, { { 0 } },  
    periodic_handler };
```

Notice that the `PERIODIC_RESOURCE` and `EVENT_RESOURCE` can be observable, meaning a client can be notified of any change in a given resource.

## Resource declaration

```
/* GET method */
RESOURCE(res_adxl345,
    "title=\"Accelerometer 3-axis\";rt=\"adxl345\"",
    res_get_handler,
    NULL,
    NULL,
    NULL);
```

URI Query



## Resource implementation

```
static void
res_get_handler(void *request, void *response, uint8_t *buffer,
    uint16_t preferred_size, int32_t *offset)
```


```
{
    uint8_t x_axis = adxl345.value(X_AXIS);
    uint8_t y_axis = adxl345.value(Y_AXIS);
    uint8_t z_axis = adxl345.value(Z_AXIS);
    unsigned int accept = -1;

    REST.get_header_accept(request, &accept);

    if(accept == REST.type.APPLICATION_JSON) {
        REST.set_header_content_type(response, REST.type.APPLICATION_JSON);
        snprintf((char *)buffer, REST_MAX_CHUNK_SIZE,
            "{ 'adxl345': { 'X': %d, 'Y': %d, 'Z': %d } }", x_axis, y_axis, z_axis);
        REST.set_response_payload(response, buffer, strlen((char *)buffer));
    }
```

Function to invoke whenever there's a GET request

The CoAP Accept option can be used to indicate which Content-Format is acceptable to the client.



## Importing the Resource

```

/*
 * Resources to be activated need to be imported through the extern keyword.
 * The build system automatically compiles the resources in the corresponding
 * sub-directory.
 */
extern resource_t
    res_hello,
    res_leds,
    res_toggle,
    res_adxl345,
    res_event,
    res_separate;

```

## Resource activation

```

/*
 * Bind the resources to their Uri-Path.
 * WARNING: Activating twice only means alternate path, not two instances!
 * All static variables are the same for each URI path.
 */
rest_activate_resource(&res_hello, "test/hello");
rest_activate_resource(&res_leds, "actuators/leds");
rest_activate_resource(&res_toggle, "actuators/toggle");
rest_activate_resource(&res_adxl345, "sensors/adxl345");
rest_activate_resource(&res_event, "sensors/button");
rest_activate_resource(&res_separate, "test/separate");

```

```
all: er-example-server
```

```
CONTIKI = ../../../../..
```

```
CFLAGS += -DPROJECT_CONF_H=\"project-conf.h\"
```

Search resources to include  
in the “resources” directory

```
# Automatically build RESTful resources
```

```
REST_RESOURCES_DIR = ./resources
```

```
REST_RESOURCES_FILES = $(notdir $(shell find $(REST_RESOURCES_DIR) -name '*.c'))
```

```
PROJECTDIRS += $(REST_RESOURCES_DIR)
```

```
PROJECT_SOURCEFILES += $(REST_RESOURCES_FILES)
```

```
# linker optimizations
```

```
SMALL = 1
```

```
# REST Engine shall use Erbium CoAP implementation
```

```
APPS += er-coap
```

```
APPS += rest-engine
```

```
CONTIKI_WITH_IPV6 = 1
```

```
include $(CONTIKI)/Makefile.include
```

REST engine and CoAP libraries



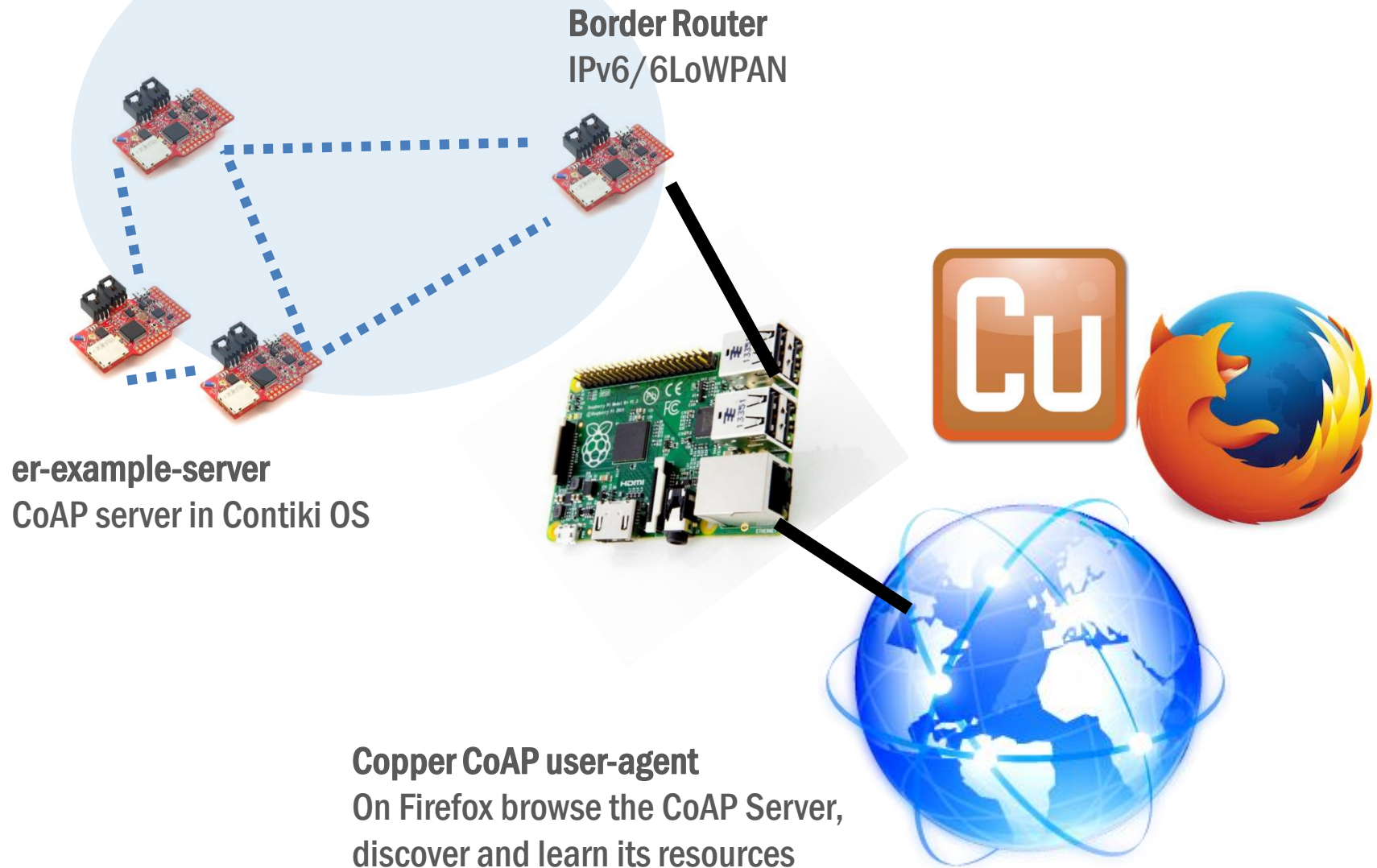
## *Copper (Cu) 0.18.4.1-signed*

by [Matthias Kovatsch](#)

The Copper (Cu) CoAP user-agent for Firefox installs a handler for the 'coap' URI scheme and allows users to browse and interact with Internet of Things devices.

**Only with Firefox — Get Firefox Now!**

This add-on has been preliminarily reviewed by Mozilla. [Learn more](#)



**TIP: enable the DEBUG to 1 to print more information about the processes taking place inside the CoAP and REST libraries (all .c files inside!)**

```
#define DEBUG 0
#if DEBUG
#define PRINTF(...) PRINTF(__VA_ARGS__)
#else
#define PRINTF(...)
#endif
```





# Pong! – check the CoAP server is online

The screenshot shows the ContikiRPL web interface. The browser address bar displays `coap://[aaaa::c30c:0:0:13c8]:5683/test/hello`. A blue arrow points to the 'Ping' button in the top navigation bar. The main content area shows a successful response from the CoAP server at `[aaaa::c30c:0:0:13c8]:5683` with an RTT of 3647ms. The response is titled 'Pong: Remote responds to CoAP'. The left sidebar shows a tree view of resources, with 'test' selected. The main area displays the response details, including headers and payload. The right sidebar shows the 'Debug Control' panel with various options.

ContikiRPL [aaaa::c30c:0:0:13c8] x +

coap://[aaaa::c30c:0:0:13c8]:5683/test/hello Search

Discover Ping PUT DELETE Observe Payload Text Behavior CoAP 18

[aaaa::c30c:0:0:13c8]:5683 (RTT: 3647ms)

## Pong: Remote responds to CoAP

▼ [aaaa::c30c:0:0:13c8]:5683

- well-known
  - core
- actuators
  - leds
  - toggle
- sensors
  - adxl345
  - button
- test
  - hello
  - separate

Header	Value	Option	Value	Info
Type	Reset			
Code	EMPTY			
Message ID	34787			
Token	empty			

**Payload**

Incoming Rendered Outgoing

Debug Control Reset

Token: 0x7ACF

Request Options

Accept: application/json

Content-Format: application/json

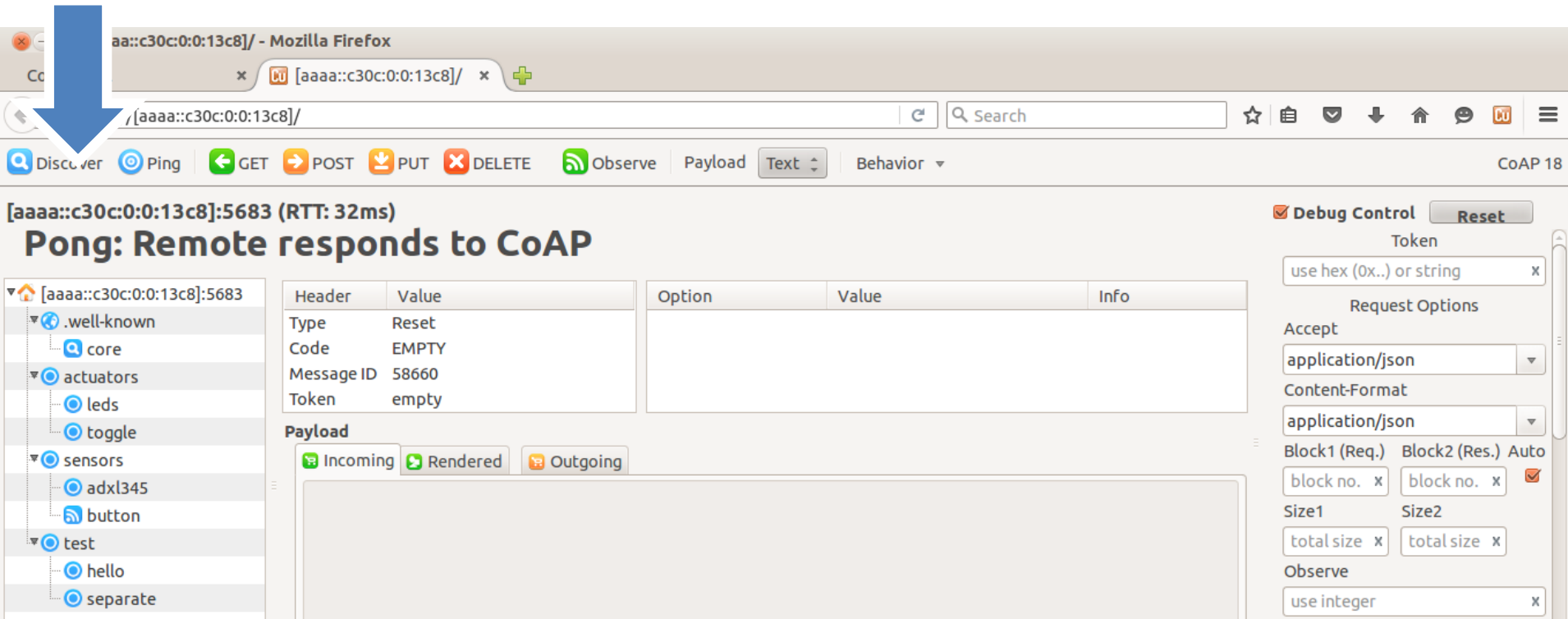
Block1 (Req.): block no. x Size1: total size x

Block2 (Res.): block no. x Size2: total size x

Observe: use integer x

ETag: use hex (0x..) or string x

# Discover – learn the Resources the CoAP server has



CoAP 18

**[aaaa::c30c:0:0:13c8]:5683 (RTT: 32ms)**  
**Pong: Remote responds to CoAP**

▼ [aaaa::c30c:0:0:13c8]:5683

- .well-known
  - core
- actuators
  - leds
  - toggle
- sensors
  - adxl345
  - button
  - test
    - hello
    - separate

Header	Value	Option	Value	Info
Type	Reset			
Code	EMPTY			
Message ID	58660			
Token	empty			

**Payload**

Incoming Rendered Outgoing

**Debug Control** **Reset**

Token  
 use hex (0x..) or string x

Request Options

Accept  
 application/json

Content-Format  
 application/json

Block1 (Req.) Block2 (Res.) Auto  
 block no. x block no. x ☒

Size1 Size2  
 total size x total size x

Observe  
 use integer x

# Hover over the resources to learn its attributes

ContikiIRPL [aaaa::c30c:0:0:13c8] x [aaaa::c30c:0:0:13c8] x

coap://[aaaa::c30c:0:0:13c8]:5683/actuators/leds

Discover Ping GET POST PUT DELETE Observe Payload Text Behavior CoAP 18

**[aaaa::c30c:0:0:13c8]:5683**

- [aaaa::c30c:0:0:13c8]:5683
  - .well-known
    - core
  - actuators
    - leds (selected)
    - toggle
  - sensor
    - adx1545
    - button
    - test
    - hello
    - separate

title="LEDs: ?color=r|g|b, POST/PUT mode=on|off" rt="Control"

Header	Value	Option	Value	Info
Type				
Code				
Message ID				
Token				

Payload

Outgoing

**Debug Control** **Reset**

Token

use hex (0x..) or string x

Request Options

Accept

application/json

Content-Format

application/json

Block1 (Req.) Block2 (Res.) Auto

block no. x block no. x

Size1 Size2

total size x total size x

Observe

use integer x

ETag

use hex (0x..) or string x

# POST/PUT – change the LEDs state (on or off)

ContikiRPL [aaaa::c30c:0:0:13c8] x [aaaa::c30c:0:0:13c8] x

coap://[aaaa::c30c:0:0:13c8]:5683/actuators/leds?color=b

Discover Ping GET POST PUT DELETE Observe Payload Text Behavior CoAP 18

[aaaa::c30c:0:0:13c8]:5683 (RTT: 46ms) 2.05 Content

↑

Header

Type	Code	Message	Token
Application	2.05 Content	13308	empty

Payload

Incoming Rendered Outgoing

mode=on

Debug Control Reset

Token

use hex (0x..) or string x

Request Options

Accept

application/json

Content-Format

application/json

Block1 (Req.) Block2 (Res.) Auto

block no. x block no. x

Size1 Size2

total size x total size x

Observe

use integer x

# GET – read the ADXL345 with JSON format (application/json)

The screenshot shows the ContikiRPL web interface in a Mozilla Firefox browser. The address bar displays the URL: `coap://[aaaa::c30c:0:0:13c8]:5683/sensors/adxl345`. The interface includes a navigation bar with buttons for Discover, Ping, GET, POST, PUT, DELETE, Observe, Payload, and Behavior. The main content area shows the result of a GET request, with a large blue arrow pointing to the '2.05 Content' status. The response is displayed in a table format, showing the Content-Format as 'application/json' and the Block2 as '1 (32 B/block)'. The combined payload is shown as a JSON object containing the ADXL345 sensor data.

**2.05 Content (application/json) (Download finished)**

Header	Value	Option	Value	Info
Type	Acknowledgment			
Code	2.05 Content	Content-Format	application/json	50
Message...	36998	Block2	1 (32 B/block)	1 byte
Token	empty			

**Combined Payload (37)**

Incoming Rendered Outgoing

```

{
  adxl345:
  {
    X: 24
    Y: 135
    Z: 18
  }
}
  
```

**Debug Control** **Reset**

Token: use hex (0x..) or string x

**Request Options**

Accept: application/json

Content-Format: application/json

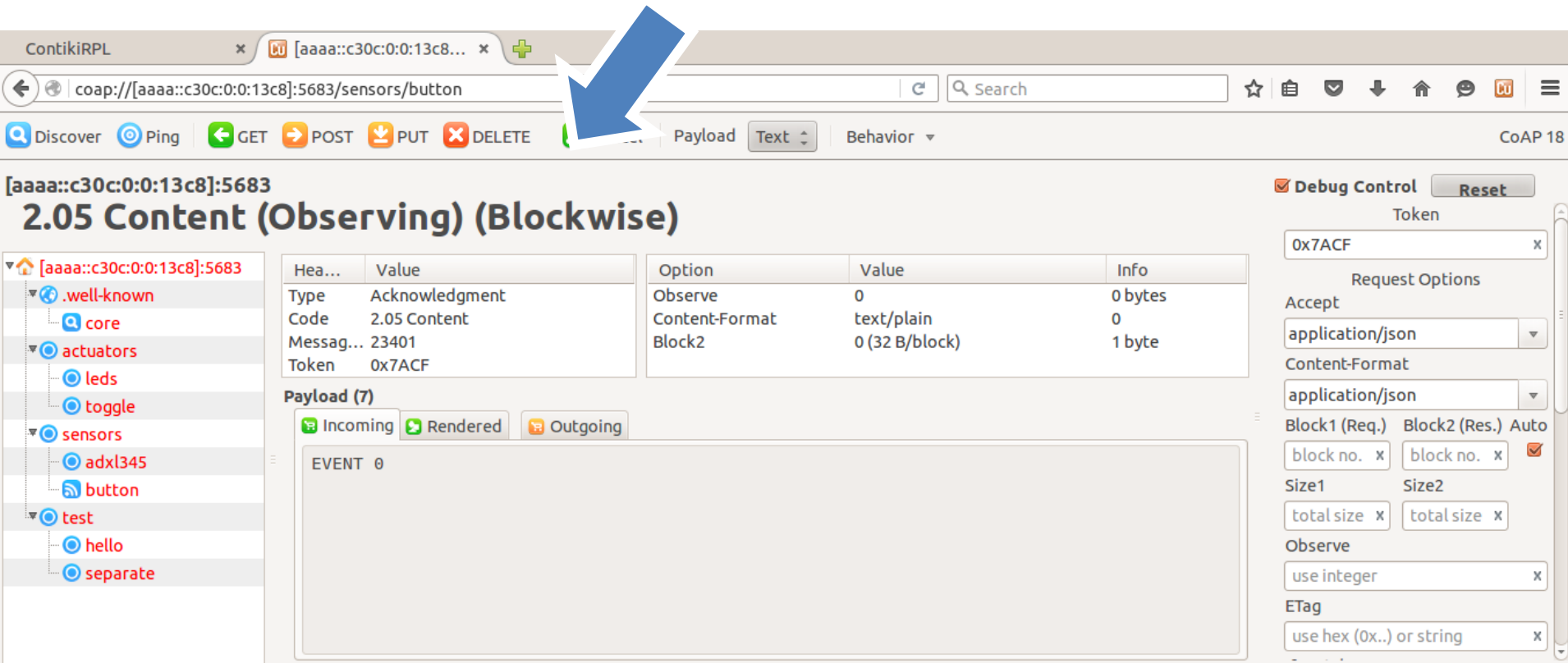
Block1 (Req.): block no. x Block2 (Res.): block no. x Auto: ☒

Size1: total size x Size2: total size x

Observe: use integer x

ETag: use hex (0x..) or string x

# OBSERVE – get notifications about an event (press the user button)



ContikiRPL [aaaa::c30c:0:0:13c8...]

coap://[aaaa::c30c:0:0:13c8]:5683/sensors/button

Discover Ping GET POST PUT DELETE Payload Text Behavior CoAP 18

## [aaaa::c30c:0:0:13c8]:5683 2.05 Content (Observing) (Blockwise)

▼ [aaaa::c30c:0:0:13c8]:5683

- ▼ .well-known
  - core
- ▼ actuators
  - leds
  - toggle
- ▼ sensors
  - adxl345
  - button
  - test
    - hello
    - separate

Header	Value	Option	Value	Info
Type	Acknowledgment	Observe	0	0 bytes
Code	2.05 Content	Content-Format	text/plain	0
Message-ID	23401	Block2	0 (32 B/block)	1 byte
Token	0x7ACF			

**Payload (7)**

Incoming Rendered Outgoing

EVENT 0

**Debug Control** **Reset**

Token: 0x7ACF

**Request Options**

Accept: application/json

Content-Format: application/json

Block1 (Req.) Block2 (Res.) Auto

block no. x block no. x ☒

Size1 Size2

total size x total size x

Observe: use integer x

ETag: use hex (0x..) or string x

# 04-mqtt



- On top of TCP/IP
- Publish/Subscribe messaging pattern
- Message Broker distributes topics to clients
- Topics are UTF-8 string-based with hierarchical structure
- No direct connection between clients
- Quality of Service
- Retain-Flag: new subscribed clients will received last value
- Last Will: notify other clients when disconnected ungracefully
- KeepAlive: ping request messages to the bróker
- Clients have to know beforehand the structure of the data published to a topic
- MQTT is data-agnostic



## CONNECT

Waits for a connection to be established with the server

## DISCONNECT

Waits for the MQTT client to finish any pending task and closes the TCP session

## SUBSCRIBE

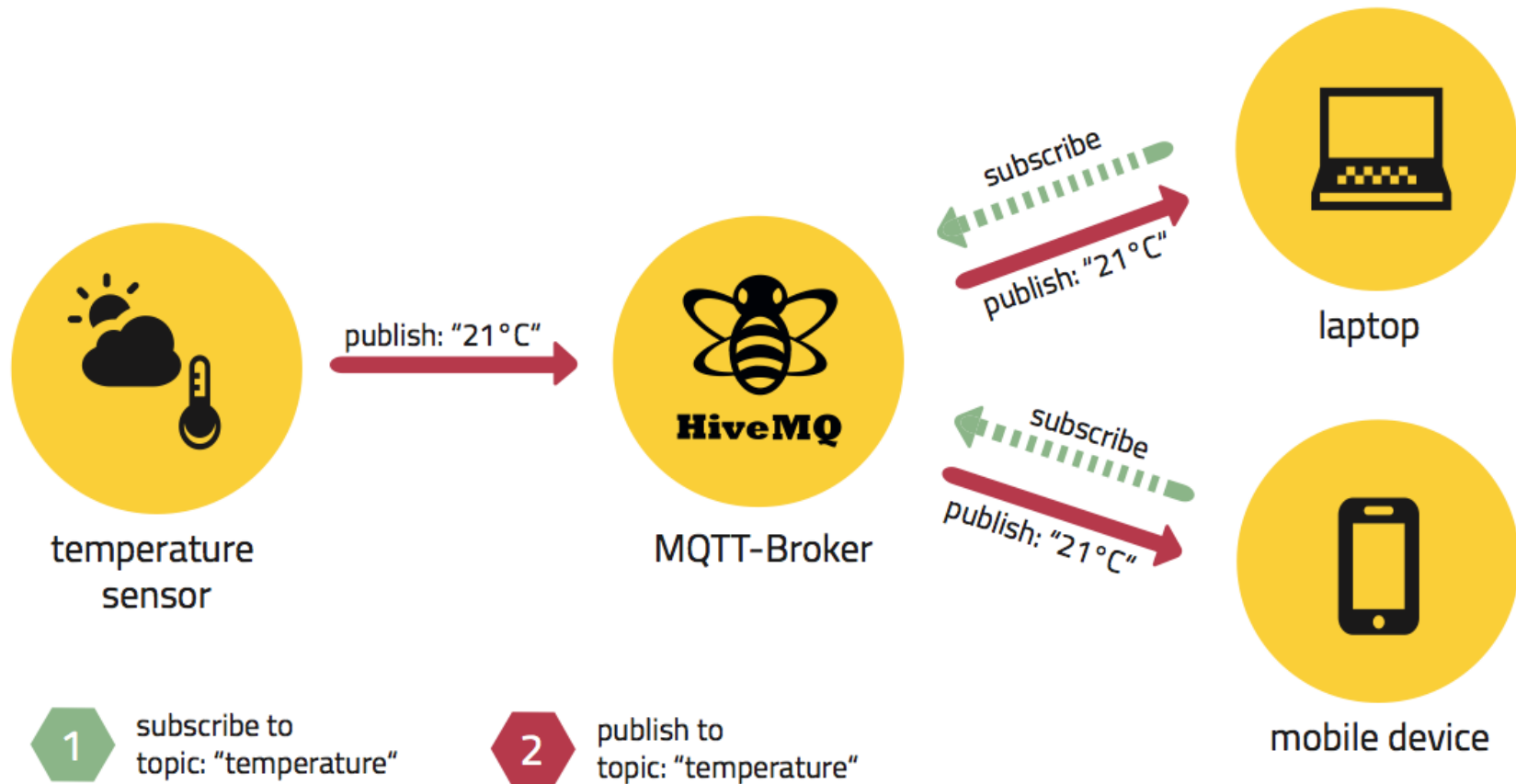
Request the server to subscribe the client to one or more topics

## UNSUBSCRIBE

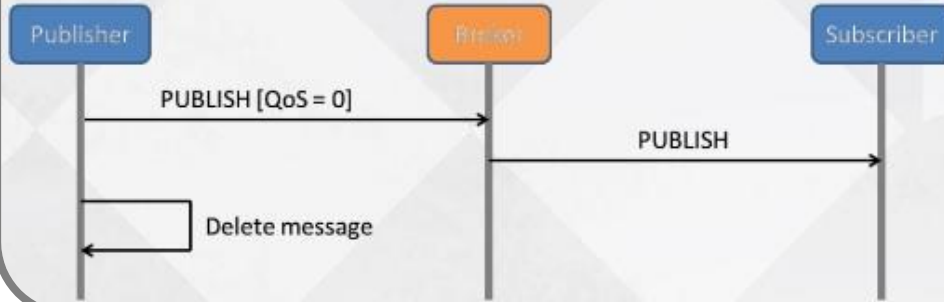
Request the server to unsubscribe the client to one or more topics

## PUBLISH

Updates a topic with data



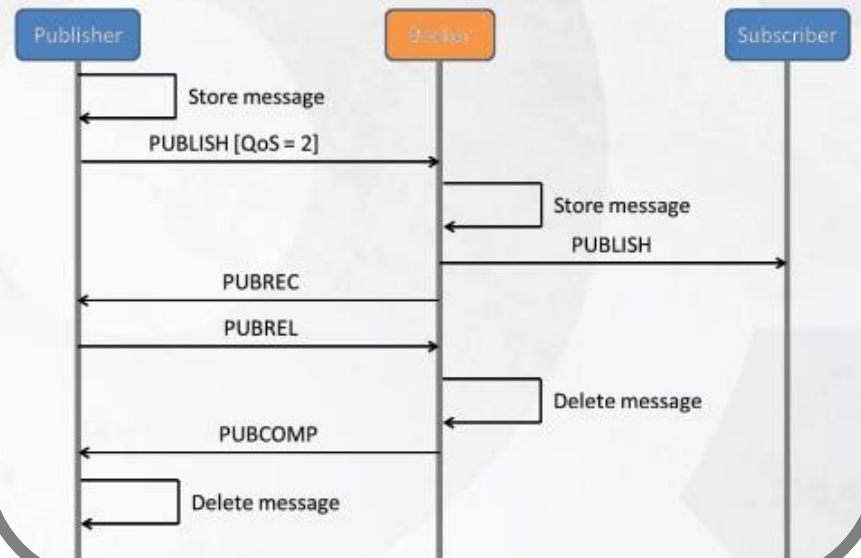
## QoS 0 : At most once (fire and forget)



## QoS 1 : At least once



## QoS 2 : Exactly once





Topics starting with \$ are special  
These are reserved for the broker  
statistics

\$SYS/broker/clients/connected  
\$SYS/broker/clients/disconnected  
\$SYS/broker/clients/total  
\$SYS/broker/messages/sent  
\$SYS/broker/uptime



```

/*-----*/
/**
 * \brief      MQTT event callback function
 * \param m    A pointer to a MQTT connection
 * \param event The event number
 * \param data  A user-defined pointer
 *
 * The MQTT socket event callback function gets called whenever there is an
 * event on a MQTT connection, such as the connection getting connected
 * or closed.
 */
typedef void (*mqtt_event_callback_t)(struct mqtt_connection *m,
                                     mqtt_event_t event,
                                     void *data);

```

Required to be included in the process using MQTT

```

/**
 * \brief Initializes the MQTT engine.
 * \param conn A pointer to the MQTT connection.
 * \param app_process A pointer to the application process handling the MQTT
 * connection.
 * \param client_id A pointer to the MQTT client ID.
 * \param event_callback Callback function responsible for handling the
 * callback from MQTT engine.
 * \param max_segment_size The TCP segment size to use for this MQTT/TCP
 * connection.
 * \return MQTT_STATUS_OK or MQTT_STATUS_INVALID_ARGS_ERROR
 *
 * This function initializes the MQTT engine and shall be called before any
 * other MQTT function.
 */
mqtt_status_t mqtt_register(struct mqtt_connection *conn,
                           struct process *app_process,
                           char *client_id,
                           mqtt_event_callback_t event_callback,
                           uint16_t max_segment_size);

```

To start the MQTT client this function should be called first

The max\_segment\_size is the TCP chunk of data to be sent (default is 32 bytes)

The client\_id is a string identifying the client

```

/**
 * \brief Connects to a MQTT broker.
 * \param conn A pointer to the MQTT connection.
 * \param host IP address of the broker to connect to.
 * \param port Port of the broker to connect to, default is MQTT port is 1883.
 * \param keep_alive Keep alive timer in seconds. Used by broker to handle
 *         client disc. Defines the maximum time interval between two messages
 *         from the client. Shall be min 1.5 x report interval.
 * \return MQTT_STATUS_OK or an error status
 *
 * This function connects to a MQTT broker.
 */
mqtt_status_t mqtt_connect(struct mqtt_connection *conn,
                           char *host,
                           uint16_t port,
                           uint16_t keep_alive);

```

```

/**
 * \brief Disconnects from a MQTT broker.
 * \param conn A pointer to the MQTT connection.
 *
 * This function disconnects from a MQTT broker.
 */
void mqtt_disconnect(struct mqtt_connection *conn);

```

The keep\_alive value is used by a timer waiting a PINGRES from the broken, if expired and no response is obtained, it triggers a disconnection

```

/**
 * \brief Subscribes to a MQTT topic.
 * \param conn A pointer to the MQTT connection.
 * \param mid A pointer to message ID.
 * \param topic A pointer to the topic to subscribe to.
 * \param qos_level Quality Of Service level to use. Currently supports 0, 1.
 * \return MQTT_STATUS_OK or some error status
 *
 * This function subscribes to a topic on a MQTT broker.
 */
mqtt_status_t mqtt_subscribe(struct mqtt_connection *conn,
                             uint16_t *mid,
                             char *topic,
                             mqtt_qos_level_t qos_level);

```

```

/**
 * \brief Unsubscribes from a MQTT topic.
 * \param conn A pointer to the MQTT connection.
 * \param mid A pointer to message ID.
 * \param topic A pointer to the topic to unsubscribe from.
 * \return MQTT_STATUS_OK or some error status
 *
 * This function unsubscribes from a topic on a MQTT broker.
 */
mqtt_status_t mqtt_unsubscribe(struct mqtt_connection *conn,
                                uint16_t *mid,
                                char *topic);

```

Message ID (mid) is zero for QoS=0



```

/**
 * \brief Publish to a MQTT topic.
 * \param conn A pointer to the MQTT connection.
 * \param mid A pointer to message ID.
 * \param topic A pointer to the topic to subscribe to.
 * \param payload A pointer to the topic payload.
 * \param payload_size Payload size.
 * \param qos_level Quality Of Service level to use. Currently supports 0, 1.
 * \param retain If the RETAIN flag is set to 1, in a PUBLISH Packet sent by a
 *               Client to a Server, the Server MUST store the Application Message
 *               and its QoS, so that it can be delivered to future subscribers whose
 *               subscriptions match its topic name
 * \return MQTT_STATUS_OK or some error status
 *
 * This function publishes to a topic on a MQTT broker.
 */
mqtt_status_t mqtt_publish(struct mqtt_connection *conn,
                          uint16_t *mid,
                          char *topic,
                          uint8_t *payload,
                          uint32_t payload_size,
                          mqtt_qos_level_t qos_level,
                          mqtt_retain_t retain);

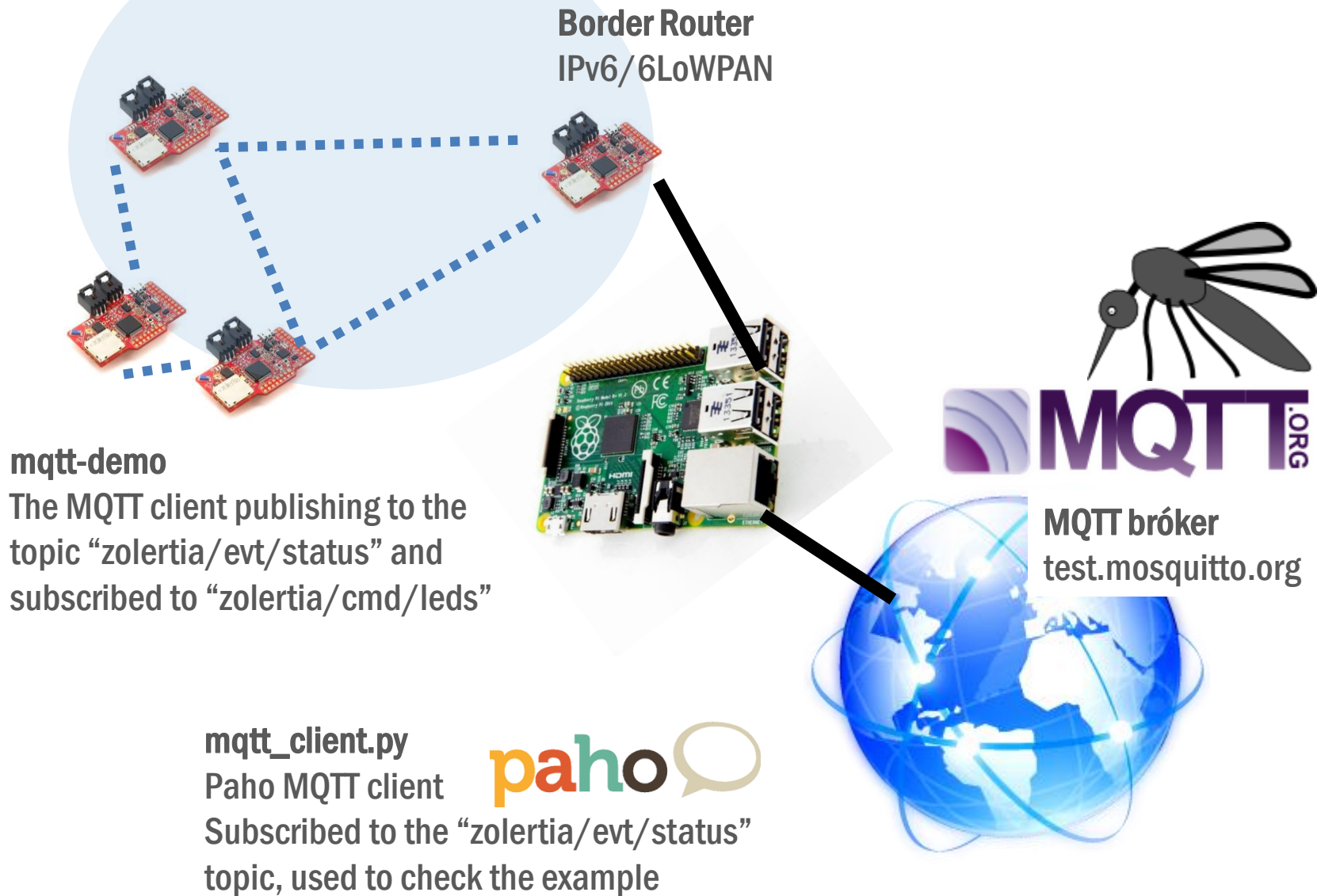
```

```

/**
 * \brief Set the user name and password for a MQTT client.
 * \param conn A pointer to the MQTT connection.
 * \param username A pointer to the user name.
 * \param password A pointer to the password.
 *
 * This function sets clients user name and password to use when connecting to
 * a MQTT broker.
 */
void mqtt_set_username_password(struct mqtt_connection *conn,
                               char *username,
                               char *password);

/**
 * \brief Set the last will topic and message for a MQTT client.
 * \param conn A pointer to the MQTT connection.
 * \param topic A pointer to the Last Will topic.
 * \param message A pointer to the Last Will message (payload).
 * \param qos The desired QoS level.
 *
 * This function sets clients Last Will topic and message (payload).
 * If the Will Flag is set to 1 (using the function) this indicates that,
 * if the Connect request is accepted, a Will Message MUST be stored on the
 * Server and associated with the Network Connection. The Will Message MUST
 * be published when the Network Connection is subsequently closed.
 *
 * This functionality can be used to get notified that a device has
 * disconnected from the broker.
 */
void mqtt_set_last_will(struct mqtt_connection *conn,
                       char *topic,
                       char *message,
                       mqtt_qos_level_t qos);

```





# Mosquitto

An Open Source MQTT v3.1/v3.1.1 Broker

**Domain Dossier**
Investigate domains and IP addresses

domain or IP address

☒ domain whois record
☒ DNS records
☐ traceroute

☒ network whois record
☐ service scan

user: anonymous [88.87.214.38]  
balance: 49 units  
[log in](#) | [account info](#)

*Central Ops .net*

**Address lookup**

canonical name **test.mosquitto.org.**  
aliases  
addresses **2001:ba8:1f1:f271::2**  
**85.119.83.194**

```

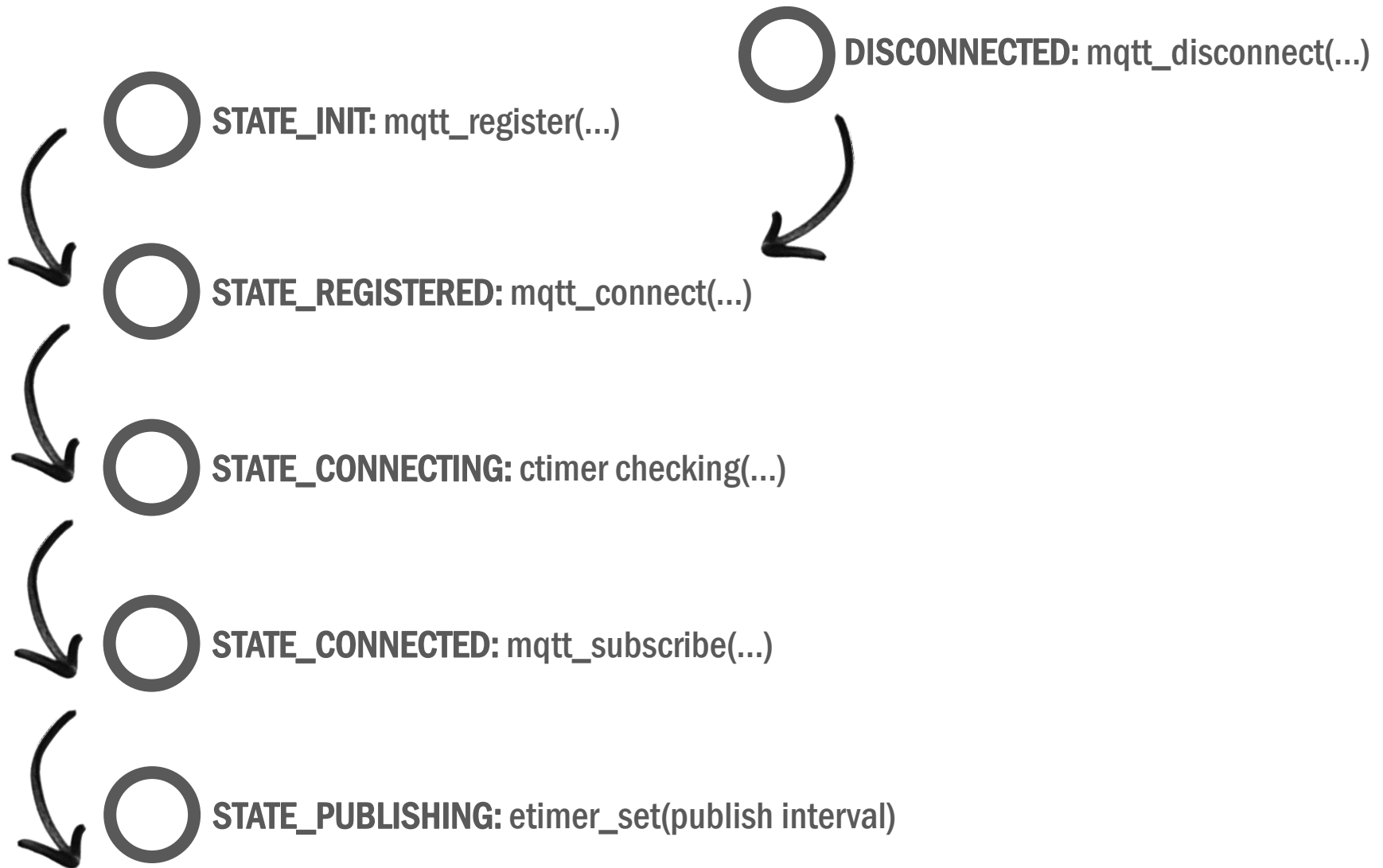
/*
 * Publish to an MQTT broker
 */
#define MQTT_DEMO_BROKER_IP_ADDR "2001:ba8:1f1:f271::2"
static const char *broker_ip = MQTT_DEMO_BROKER_IP_ADDR;
#define DEFAULT_ORG_ID "mqtt-demo"

```

```

static void
mqtt_event(struct mqtt_connection *m, mqtt_event_t event, void *data)
{
    switch(event) {
        case MQTT_EVENT_CONNECTED: {
            printf("APP - Application has a MQTT connection\n");
            state = STATE_CONNECTED;
            break;
        }
        case MQTT_EVENT_DISCONNECTED: {
            printf("APP - MQTT Disconnect. Reason %u\n", *((mqtt_event_t *)data));
            state = STATE_DISCONNECTED;
            process_poll(&mqtt_demo_proc);
            break;
        }
        case MQTT_EVENT_PUBLISH: {
            pub_handler(msg_ptr->topic, str,
                        msg_ptr->payload_length);
            break;
        }
        case MQTT_EVENT_SUBACK: {
            printf("APP - Application is subscribed to topic successfully\n");
            break;
        }
        case MQTT_EVENT_UNSUBACK: {
            printf("APP - Application is unsubscribed to topic successfully\n");
            break;
        }
        case MQTT_EVENT_PUBACK: {
            printf("APP - Publishing complete.\n");
            break;
        }
    }
}

```



Set default configuration values

```

/*-----*/
PROCESS_THREAD(mqtt_demo_process, ev, data)
{
    PROCESS_BEGIN();
    if(init_config() != 1) {
        PROCESS_EXIT();
    }
    update_config();
    while(1) {
        PROCESS_YIELD();
        if((ev == PROCESS_EVENT_TIMER && data == &publish_periodic_timer) ||
            ev == PROCESS_EVENT_POLL ||
            (ev == sensors_event && data == &button_sensor)) {
            state_machine();
        }
    }
    PROCESS_END();
}

```

Create topic/subscription/id strings (STATE\_INIT)

Polls the state machine as described before

```

* The main MQTT buffers.
* We will need to increase if we start publishing more data.
*/
#define APP_BUFFER_SIZE 256
static struct mqtt_connection conn;
static char app_buffer[APP_BUFFER_SIZE];

```

**TIP: enable the DEBUG to 1 to print more information about the processes taking place inside the MQTT library**

```
#define DEBUG 0
#if DEBUG
#define PRINTF(...) PRINTF(__VA_ARGS__)
#else
#define PRINTF(...)
#endif
```





## MQTT demo client running on the Z1 mote

```
Starting 'MQTT Demo'
MQTT Demo Process
Subscription topic zolertia/cmd/leds
Init
Registered. Connect attempt 1
Connecting (1)
APP - Application has a MQTT connection
APP - Subscribing!
APP - Application is subscribed to topic successfully
Publishing
APP - Publish to zolertia/evt/status
```



## Paho MQTT client in Python subscribed

```
$ python mqtt_client.py
connecting to test.mosquitto.org
Connected with result code 0
Subscribed to zolertia/evt/status
Subscribed to zolertia/cmd/leds
zolertia/evt/status {"d":{"myName":"Zolertia Z1 Node","Seq #":3,"Uptime (sec)":141,"Def
zolertia/evt/status {"d":{"myName":"Zolertia Z1 Node","Seq #":4,"Uptime (sec)":186,"Def
```

## Mosquitto publishing to turn a LED on

```
mosquitto_pub -h "test.mosquitto.org" -t "zolertia/cmd/led" -m "1" -q 1 -r
```



# Antonio Liñán Colina

[alinan@zolertia.com](mailto:alinan@zolertia.com)

[antonio.lignan@gmail.com](mailto:antonio.lignan@gmail.com)



Twitter: @4Li6NaN



LinkedIn: Antonio Liñán Colina



[github.com/alignan](https://github.com/alignan)



[hackster.io/alinan](https://hackster.io/alinan)