**WOT LAB MQTT Protocol**

**ECM1004**

Name: Nagharjun M

Registration Number: 18BLC1129

Date: 07/04/2021

**CODE:**

#include "contiki.h"

#include "net/routing/routing.h"

#include "mqtt.h"

#include "mqtt-prop.h"

#include "net/ipv6/uip.h"

#include "net/ipv6/uip-icmp6.h"

#include "net/ipv6/sicslowpan.h"

#include "sys/etimer.h"

#include "sys/ctimer.h"

#include "lib/sensors.h"

#include "dev/button-hal.h"

#include "dev/leds.h"

#include "os/sys/log.h"

#include "mqtt-client.h"

#include <string.h>

#include <strings.h>

#include <stdarg.h>

/\*---------------------------------------------------------------------------\*/

#define LOG\_MODULE "mqtt-client"

#ifdef MQTT\_CLIENT\_CONF\_LOG\_LEVEL

#define LOG\_LEVEL MQTT\_CLIENT\_CONF\_LOG\_LEVEL

#else

#define LOG\_LEVEL LOG\_LEVEL\_NONE

#endif

/\*---------------------------------------------------------------------------\*/

/\* Controls whether the example will work in IBM Watson IoT platform mode \*/

#ifdef MQTT\_CLIENT\_CONF\_WITH\_IBM\_WATSON

#define MQTT\_CLIENT\_WITH\_IBM\_WATSON MQTT\_CLIENT\_CONF\_WITH\_IBM\_WATSON

#else

#define MQTT\_CLIENT\_WITH\_IBM\_WATSON 0

#endif

/\*---------------------------------------------------------------------------\*/

/\* MQTT broker address. Ignored in Watson mode \*/

#ifdef MQTT\_CLIENT\_CONF\_BROKER\_IP\_ADDR

#define MQTT\_CLIENT\_BROKER\_IP\_ADDR MQTT\_CLIENT\_CONF\_BROKER\_IP\_ADDR

#else

#define MQTT\_CLIENT\_BROKER\_IP\_ADDR "fd00::1"

#endif

/\*---------------------------------------------------------------------------\*/

/\*

\* MQTT Org ID.

\*

\* If it equals "quickstart", the client will connect without authentication.

\* In all other cases, the client will connect with authentication mode.

\*

\* In Watson mode, the username will be "use-token-auth". In non-Watson mode

\* the username will be MQTT\_CLIENT\_USERNAME.

\*

\* In all cases, the password will be MQTT\_CLIENT\_AUTH\_TOKEN.

\*/

#ifdef MQTT\_CLIENT\_CONF\_ORG\_ID

#define MQTT\_CLIENT\_ORG\_ID MQTT\_CLIENT\_CONF\_ORG\_ID

#else

#define MQTT\_CLIENT\_ORG\_ID "quickstart"

#endif

/\*---------------------------------------------------------------------------\*/

/\* MQTT token \*/

#ifdef MQTT\_CLIENT\_CONF\_AUTH\_TOKEN

#define MQTT\_CLIENT\_AUTH\_TOKEN MQTT\_CLIENT\_CONF\_AUTH\_TOKEN

#else

#define MQTT\_CLIENT\_AUTH\_TOKEN "AUTHTOKEN"

#endif

/\*---------------------------------------------------------------------------\*/

#if MQTT\_CLIENT\_WITH\_IBM\_WATSON

/\* With IBM Watson support \*/

static const char \*broker\_ip = "0064:ff9b:0000:0000:0000:0000:b8ac:7cbd";

#define MQTT\_CLIENT\_USERNAME "use-token-auth"

#else /\* MQTT\_CLIENT\_WITH\_IBM\_WATSON \*/

/\* Without IBM Watson support. To be used with other brokers, e.g. Mosquitto \*/

static const char \*broker\_ip = MQTT\_CLIENT\_BROKER\_IP\_ADDR;

#ifdef MQTT\_CLIENT\_CONF\_USERNAME

#define MQTT\_CLIENT\_USERNAME MQTT\_CLIENT\_CONF\_USERNAME

#else

#define MQTT\_CLIENT\_USERNAME "use-token-auth"

#endif

#endif /\* MQTT\_CLIENT\_WITH\_IBM\_WATSON \*/

/\*---------------------------------------------------------------------------\*/

#ifdef MQTT\_CLIENT\_CONF\_STATUS\_LED

#define MQTT\_CLIENT\_STATUS\_LED MQTT\_CLIENT\_CONF\_STATUS\_LED

#else

#define MQTT\_CLIENT\_STATUS\_LED LEDS\_GREEN

#endif

/\*---------------------------------------------------------------------------\*/

#ifdef MQTT\_CLIENT\_CONF\_WITH\_EXTENSIONS

#define MQTT\_CLIENT\_WITH\_EXTENSIONS MQTT\_CLIENT\_CONF\_WITH\_EXTENSIONS

#else

#define MQTT\_CLIENT\_WITH\_EXTENSIONS 0

#endif

/\*---------------------------------------------------------------------------\*/

/\*

\* A timeout used when waiting for something to happen (e.g. to connect or to

\* disconnect)

\*/

#define STATE\_MACHINE\_PERIODIC (CLOCK\_SECOND >> 1)

/\*---------------------------------------------------------------------------\*/

/\* Provide visible feedback via LEDS during various states \*/

/\* When connecting to broker \*/

#define CONNECTING\_LED\_DURATION (CLOCK\_SECOND >> 2)

/\* Each time we try to publish \*/

#define PUBLISH\_LED\_ON\_DURATION (CLOCK\_SECOND)

/\*---------------------------------------------------------------------------\*/

/\* Connections and reconnections \*/

#define RETRY\_FOREVER 0xFF

#define RECONNECT\_INTERVAL (CLOCK\_SECOND \* 2)

/\*---------------------------------------------------------------------------\*/

/\*

\* Number of times to try reconnecting to the broker.

\* Can be a limited number (e.g. 3, 10 etc) or can be set to RETRY\_FOREVER

\*/

#define RECONNECT\_ATTEMPTS RETRY\_FOREVER

#define CONNECTION\_STABLE\_TIME (CLOCK\_SECOND \* 5)

static struct timer connection\_life;

static uint8\_t connect\_attempt;

/\*---------------------------------------------------------------------------\*/

/\* Various states \*/

static uint8\_t state;

#define STATE\_INIT 0

#define STATE\_REGISTERED 1

#define STATE\_CONNECTING 2

#define STATE\_CONNECTED 3

#define STATE\_PUBLISHING 4

#define STATE\_DISCONNECTED 5

#define STATE\_NEWCONFIG 6

#define STATE\_CONFIG\_ERROR 0xFE

#define STATE\_ERROR 0xFF

/\*---------------------------------------------------------------------------\*/

#define CONFIG\_ORG\_ID\_LEN 32

#define CONFIG\_TYPE\_ID\_LEN 32

#define CONFIG\_AUTH\_TOKEN\_LEN 32

#define CONFIG\_EVENT\_TYPE\_ID\_LEN 32

#define CONFIG\_CMD\_TYPE\_LEN 8

#define CONFIG\_IP\_ADDR\_STR\_LEN 64

/\*---------------------------------------------------------------------------\*/

/\* A timeout used when waiting to connect to a network \*/

#define NET\_CONNECT\_PERIODIC (CLOCK\_SECOND >> 2)

#define NO\_NET\_LED\_DURATION (NET\_CONNECT\_PERIODIC >> 1)

/\*---------------------------------------------------------------------------\*/

/\* Default configuration values \*/

#define DEFAULT\_TYPE\_ID "mqtt-client"

#define DEFAULT\_EVENT\_TYPE\_ID "status"

#define DEFAULT\_SUBSCRIBE\_CMD\_TYPE "+"

#define DEFAULT\_BROKER\_PORT 1883

#define DEFAULT\_PUBLISH\_INTERVAL (30 \* CLOCK\_SECOND)

#define DEFAULT\_KEEP\_ALIVE\_TIMER 60

#define DEFAULT\_RSSI\_MEAS\_INTERVAL (CLOCK\_SECOND \* 30)

/\*---------------------------------------------------------------------------\*/

#define MQTT\_CLIENT\_SENSOR\_NONE (void \*)0xFFFFFFFF

/\*---------------------------------------------------------------------------\*/

/\* Payload length of ICMPv6 echo requests used to measure RSSI with def rt \*/

#define ECHO\_REQ\_PAYLOAD\_LEN 20

/\*---------------------------------------------------------------------------\*/

PROCESS\_NAME(mqtt\_client\_process);

AUTOSTART\_PROCESSES(&mqtt\_client\_process);

/\*---------------------------------------------------------------------------\*/

/\*\*

\* \brief Data structure declaration for the MQTT client configuration

\*/

typedef struct mqtt\_client\_config {

char org\_id[CONFIG\_ORG\_ID\_LEN];

char type\_id[CONFIG\_TYPE\_ID\_LEN];

char auth\_token[CONFIG\_AUTH\_TOKEN\_LEN];

char event\_type\_id[CONFIG\_EVENT\_TYPE\_ID\_LEN];

char broker\_ip[CONFIG\_IP\_ADDR\_STR\_LEN];

char cmd\_type[CONFIG\_CMD\_TYPE\_LEN];

clock\_time\_t pub\_interval;

int def\_rt\_ping\_interval;

uint16\_t broker\_port;

} mqtt\_client\_config\_t;

/\*---------------------------------------------------------------------------\*/

/\* Maximum TCP segment size for outgoing segments of our socket \*/

#define MAX\_TCP\_SEGMENT\_SIZE 32

/\*---------------------------------------------------------------------------\*/

/\*

\* Buffers for Client ID and Topic.

\* Make sure they are large enough to hold the entire respective string

\*

\* d:quickstart:status:EUI64 is 32 bytes long

\* iot-2/evt/status/fmt/json is 25 bytes

\* We also need space for the null termination

\*/

#define BUFFER\_SIZE 64

static char client\_id[BUFFER\_SIZE];

static char pub\_topic[BUFFER\_SIZE];

static char sub\_topic[BUFFER\_SIZE];

/\*---------------------------------------------------------------------------\*/

/\*

\* The main MQTT buffers.

\* We will need to increase if we start publishing more data.

\*/

#define APP\_BUFFER\_SIZE 512

static struct mqtt\_connection conn;

static char app\_buffer[APP\_BUFFER\_SIZE];

/\*---------------------------------------------------------------------------\*/

#define QUICKSTART "quickstart"

/\*---------------------------------------------------------------------------\*/

static struct mqtt\_message \*msg\_ptr = 0;

static struct etimer publish\_periodic\_timer;

static struct ctimer ct;

static char \*buf\_ptr;

static uint16\_t seq\_nr\_value = 0;

/\*---------------------------------------------------------------------------\*/

/\* Parent RSSI functionality \*/

static struct uip\_icmp6\_echo\_reply\_notification echo\_reply\_notification;

static struct etimer echo\_request\_timer;

static int def\_rt\_rssi = 0;

/\*---------------------------------------------------------------------------\*/

static mqtt\_client\_config\_t conf;

/\*---------------------------------------------------------------------------\*/

#if MQTT\_CLIENT\_WITH\_EXTENSIONS

extern const mqtt\_client\_extension\_t \*mqtt\_client\_extensions[];

extern const uint8\_t mqtt\_client\_extension\_count;

#else

static const mqtt\_client\_extension\_t \*mqtt\_client\_extensions[] = { NULL };

static const uint8\_t mqtt\_client\_extension\_count = 0;

#endif

/\*---------------------------------------------------------------------------\*/

/\* MQTTv5 \*/

#if MQTT\_5

static uint8\_t PUB\_TOPIC\_ALIAS;

struct mqtt\_prop\_list \*publish\_props;

/\* Control whether or not to perform authentication (MQTTv5) \*/

#define MQTT\_5\_AUTH\_EN 0

#if MQTT\_5\_AUTH\_EN

struct mqtt\_prop\_list \*auth\_props;

#endif

#endif

/\*---------------------------------------------------------------------------\*/

PROCESS(mqtt\_client\_process, "MQTT Client");

/\*---------------------------------------------------------------------------\*/

static bool

have\_connectivity(void)

{

if(uip\_ds6\_get\_global(ADDR\_PREFERRED) == NULL ||

uip\_ds6\_defrt\_choose() == NULL) {

return false;

}

return true;

}

/\*---------------------------------------------------------------------------\*/

static int

ipaddr\_sprintf(char \*buf, uint8\_t buf\_len, const uip\_ipaddr\_t \*addr)

{

uint16\_t a;

uint8\_t len = 0;

int i, f;

for(i = 0, f = 0; i < sizeof(uip\_ipaddr\_t); i += 2) {

a = (addr->u8[i] << 8) + addr->u8[i + 1];

if(a == 0 && f >= 0) {

if(f++ == 0) {

len += snprintf(&buf[len], buf\_len - len, "::");

}

} else {

if(f > 0) {

f = -1;

} else if(i > 0) {

len += snprintf(&buf[len], buf\_len - len, ":");

}

len += snprintf(&buf[len], buf\_len - len, "%x", a);

}

}

return len;

}

/\*---------------------------------------------------------------------------\*/

static void

echo\_reply\_handler(uip\_ipaddr\_t \*source, uint8\_t ttl, uint8\_t \*data,

uint16\_t datalen)

{

if(uip\_ip6addr\_cmp(source, uip\_ds6\_defrt\_choose())) {

def\_rt\_rssi = sicslowpan\_get\_last\_rssi();

}

}

/\*---------------------------------------------------------------------------\*/

static void

publish\_led\_off(void \*d)

{

leds\_off(MQTT\_CLIENT\_STATUS\_LED);

}

/\*---------------------------------------------------------------------------\*/

static void

pub\_handler(const char \*topic, uint16\_t topic\_len, const uint8\_t \*chunk,

uint16\_t chunk\_len)

{

LOG\_DBG("Pub Handler: topic='%s' (len=%u), chunk\_len=%u, chunk='%s'\n", topic,

topic\_len, chunk\_len, chunk);

/\* If we don't like the length, ignore \*/

if(topic\_len != 23 || chunk\_len != 1) {

LOG\_ERR("Incorrect topic or chunk len. Ignored\n");

return;

}

/\* If the format != json, ignore \*/

if(strncmp(&topic[topic\_len - 4], "json", 4) != 0) {

LOG\_ERR("Incorrect format\n");

}

if(strncmp(&topic[10], "leds", 4) == 0) {

LOG\_DBG("Received MQTT SUB\n");

if(chunk[0] == '1') {

leds\_on(LEDS\_RED);

} else if(chunk[0] == '0') {

leds\_off(LEDS\_RED);

}

return;

}

}

/\*---------------------------------------------------------------------------\*/

static void

mqtt\_event(struct mqtt\_connection \*m, mqtt\_event\_t event, void \*data)

{

switch(event) {

case MQTT\_EVENT\_CONNECTED: {

LOG\_DBG("Application has a MQTT connection\n");

timer\_set(&connection\_life, CONNECTION\_STABLE\_TIME);

state = STATE\_CONNECTED;

break;

}

case MQTT\_EVENT\_DISCONNECTED:

case MQTT\_EVENT\_CONNECTION\_REFUSED\_ERROR: {

LOG\_DBG("MQTT Disconnect. Reason %u\n", \*((mqtt\_event\_t \*)data));

state = STATE\_DISCONNECTED;

process\_poll(&mqtt\_client\_process);

break;

}

case MQTT\_EVENT\_PUBLISH: {

msg\_ptr = data;

/\* Implement first\_flag in publish message? \*/

if(msg\_ptr->first\_chunk) {

msg\_ptr->first\_chunk = 0;

LOG\_DBG("Application received publish for topic '%s'. Payload "

"size is %i bytes.\n", msg\_ptr->topic, msg\_ptr->payload\_chunk\_length);

}

pub\_handler(msg\_ptr->topic, strlen(msg\_ptr->topic),

msg\_ptr->payload\_chunk, msg\_ptr->payload\_chunk\_length);

#if MQTT\_5

/\* Print any properties received along with the message \*/

mqtt\_prop\_print\_input\_props(m);

#endif

break;

}

case MQTT\_EVENT\_SUBACK: {

#if MQTT\_31

LOG\_DBG("Application is subscribed to topic successfully\n");

#else

struct mqtt\_suback\_event \*suback\_event = (struct mqtt\_suback\_event \*)data;

if(suback\_event->success) {

LOG\_DBG("Application is subscribed to topic successfully\n");

} else {

LOG\_DBG("Application failed to subscribe to topic (ret code %x)\n", suback\_event->return\_code);

}

#if MQTT\_5

/\* Print any properties received along with the message \*/

mqtt\_prop\_print\_input\_props(m);

#endif

#endif

break;

}

case MQTT\_EVENT\_UNSUBACK: {

LOG\_DBG("Application is unsubscribed to topic successfully\n");

break;

}

case MQTT\_EVENT\_PUBACK: {

LOG\_DBG("Publishing complete.\n");

break;

}

#if MQTT\_5\_AUTH\_EN

case MQTT\_EVENT\_AUTH: {

LOG\_DBG("Continuing auth.\n");

struct mqtt\_prop\_auth\_event \*auth\_event = (struct mqtt\_prop\_auth\_event \*)data;

break;

}

#endif

default:

LOG\_DBG("Application got a unhandled MQTT event: %i\n", event);

break;

}

}

/\*---------------------------------------------------------------------------\*/

static int

construct\_pub\_topic(void)

{

int len = snprintf(pub\_topic, BUFFER\_SIZE, "iot-2/evt/%s/fmt/json",

conf.event\_type\_id);

/\* len < 0: Error. Len >= BUFFER\_SIZE: Buffer too small \*/

if(len < 0 || len >= BUFFER\_SIZE) {

LOG\_INFO("Pub Topic: %d, Buffer %d\n", len, BUFFER\_SIZE);

return 0;

}

#if MQTT\_5

PUB\_TOPIC\_ALIAS = 1;

#endif

return 1;

}

/\*---------------------------------------------------------------------------\*/

static int

construct\_sub\_topic(void)

{

int len = snprintf(sub\_topic, BUFFER\_SIZE, "iot-2/cmd/%s/fmt/json",

conf.cmd\_type);

/\* len < 0: Error. Len >= BUFFER\_SIZE: Buffer too small \*/

if(len < 0 || len >= BUFFER\_SIZE) {

LOG\_INFO("Sub Topic: %d, Buffer %d\n", len, BUFFER\_SIZE);

return 0;

}

return 1;

}

/\*---------------------------------------------------------------------------\*/

static int

construct\_client\_id(void)

{

int len = snprintf(client\_id, BUFFER\_SIZE, "d:%s:%s:%02x%02x%02x%02x%02x%02x",

conf.org\_id, conf.type\_id,

linkaddr\_node\_addr.u8[0], linkaddr\_node\_addr.u8[1],

linkaddr\_node\_addr.u8[2], linkaddr\_node\_addr.u8[5],

linkaddr\_node\_addr.u8[6], linkaddr\_node\_addr.u8[7]);

/\* len < 0: Error. Len >= BUFFER\_SIZE: Buffer too small \*/

if(len < 0 || len >= BUFFER\_SIZE) {

LOG\_ERR("Client ID: %d, Buffer %d\n", len, BUFFER\_SIZE);

return 0;

}

return 1;

}

/\*---------------------------------------------------------------------------\*/

static void

update\_config(void)

{

if(construct\_client\_id() == 0) {

/\* Fatal error. Client ID larger than the buffer \*/

state = STATE\_CONFIG\_ERROR;

return;

}

if(construct\_sub\_topic() == 0) {

/\* Fatal error. Topic larger than the buffer \*/

state = STATE\_CONFIG\_ERROR;

return;

}

if(construct\_pub\_topic() == 0) {

/\* Fatal error. Topic larger than the buffer \*/

state = STATE\_CONFIG\_ERROR;

return;

}

/\* Reset the counter \*/

seq\_nr\_value = 0;

state = STATE\_INIT;

/\*

\* Schedule next timer event ASAP

\*

\* If we entered an error state then we won't do anything when it fires.

\*

\* Since the error at this stage is a config error, we will only exit this

\* error state if we get a new config.

\*/

etimer\_set(&publish\_periodic\_timer, 0);

#if MQTT\_5

LIST\_STRUCT\_INIT(&(conn.will), properties);

mqtt\_props\_init();

#endif

return;

}

/\*---------------------------------------------------------------------------\*/

static int

init\_config()

{

/\* Populate configuration with default values \*/

memset(&conf, 0, sizeof(mqtt\_client\_config\_t));

memcpy(conf.org\_id, MQTT\_CLIENT\_ORG\_ID, strlen(MQTT\_CLIENT\_ORG\_ID));

memcpy(conf.type\_id, DEFAULT\_TYPE\_ID, strlen(DEFAULT\_TYPE\_ID));

memcpy(conf.auth\_token, MQTT\_CLIENT\_AUTH\_TOKEN,

strlen(MQTT\_CLIENT\_AUTH\_TOKEN));

memcpy(conf.event\_type\_id, DEFAULT\_EVENT\_TYPE\_ID,

strlen(DEFAULT\_EVENT\_TYPE\_ID));

memcpy(conf.broker\_ip, broker\_ip, strlen(broker\_ip));

memcpy(conf.cmd\_type, DEFAULT\_SUBSCRIBE\_CMD\_TYPE, 1);

conf.broker\_port = DEFAULT\_BROKER\_PORT;

conf.pub\_interval = DEFAULT\_PUBLISH\_INTERVAL;

conf.def\_rt\_ping\_interval = DEFAULT\_RSSI\_MEAS\_INTERVAL;

return 1;

}

/\*---------------------------------------------------------------------------\*/

static void

subscribe(void)

{

/\* Publish MQTT topic in IBM quickstart format \*/

mqtt\_status\_t status;

#if MQTT\_5

status = mqtt\_subscribe(&conn, NULL, sub\_topic, MQTT\_QOS\_LEVEL\_0,

MQTT\_NL\_OFF, MQTT\_RAP\_OFF, MQTT\_RET\_H\_SEND\_ALL,

MQTT\_PROP\_LIST\_NONE);

#else

status = mqtt\_subscribe(&conn, NULL, sub\_topic, MQTT\_QOS\_LEVEL\_0);

#endif

LOG\_DBG("Subscribing!\n");

if(status == MQTT\_STATUS\_OUT\_QUEUE\_FULL) {

LOG\_ERR("Tried to subscribe but command queue was full!\n");

}

}

/\*---------------------------------------------------------------------------\*/

static void

publish(void)

{

/\* Publish MQTT topic in IBM quickstart format \*/

int len;

int remaining = APP\_BUFFER\_SIZE;

int i;

char def\_rt\_str[64];

#if MQTT\_5

static uint8\_t prop\_err = 1;

#endif

seq\_nr\_value++;

buf\_ptr = app\_buffer;

len = snprintf(buf\_ptr, remaining,

"{"

"\"d\":{"

"\"Platform\":\""CONTIKI\_TARGET\_STRING"\","

#ifdef CONTIKI\_BOARD\_STRING

"\"Board\":\""CONTIKI\_BOARD\_STRING"\","

#endif

"\"Seq #\":%d,"

"\"Uptime (sec)\":%lu",

seq\_nr\_value, clock\_seconds());

if(len < 0 || len >= remaining) {

LOG\_ERR("Buffer too short. Have %d, need %d + \\0\n", remaining,

len);

return;

}

remaining -= len;

buf\_ptr += len;

/\* Put our Default route's string representation in a buffer \*/

memset(def\_rt\_str, 0, sizeof(def\_rt\_str));

ipaddr\_sprintf(def\_rt\_str, sizeof(def\_rt\_str), uip\_ds6\_defrt\_choose());

len = snprintf(buf\_ptr, remaining,

",\"Def Route\":\"%s\",\"RSSI (dBm)\":%d",

def\_rt\_str, def\_rt\_rssi);

if(len < 0 || len >= remaining) {

LOG\_ERR("Buffer too short. Have %d, need %d + \\0\n", remaining,

len);

return;

}

remaining -= len;

buf\_ptr += len;

for(i = 0; i < mqtt\_client\_extension\_count; i++) {

len = snprintf(buf\_ptr, remaining, ",%s",

mqtt\_client\_extensions[i]->value());

if(len < 0 || len >= remaining) {

LOG\_ERR("Buffer too short. Have %d, need %d + \\0\n", remaining,

len);

return;

}

remaining -= len;

buf\_ptr += len;

}

len = snprintf(buf\_ptr, remaining, "}}");

if(len < 0 || len >= remaining) {

LOG\_ERR("Buffer too short. Have %d, need %d + \\0\n", remaining,

len);

return;

}

#if MQTT\_5

/\* Only send full topic name with the first PUBLISH

\* Afterwards, only use topic alias

\*/

if(seq\_nr\_value == 1) {

mqtt\_publish(&conn, NULL, pub\_topic, (uint8\_t \*)app\_buffer,

strlen(app\_buffer), MQTT\_QOS\_LEVEL\_0, MQTT\_RETAIN\_OFF,

PUB\_TOPIC\_ALIAS, MQTT\_TOPIC\_ALIAS\_OFF,

publish\_props);

prop\_err = mqtt\_prop\_register(&publish\_props,

NULL,

MQTT\_FHDR\_MSG\_TYPE\_PUBLISH,

MQTT\_VHDR\_PROP\_TOPIC\_ALIAS,

PUB\_TOPIC\_ALIAS);

} else {

mqtt\_publish(&conn, NULL, pub\_topic, (uint8\_t \*)app\_buffer,

strlen(app\_buffer), MQTT\_QOS\_LEVEL\_0, MQTT\_RETAIN\_OFF,

PUB\_TOPIC\_ALIAS, (mqtt\_topic\_alias\_en\_t) !prop\_err,

publish\_props);

}

#else

mqtt\_publish(&conn, NULL, pub\_topic, (uint8\_t \*)app\_buffer,

strlen(app\_buffer), MQTT\_QOS\_LEVEL\_0, MQTT\_RETAIN\_OFF);

#endif

LOG\_DBG("Publish!\n");

}

/\*---------------------------------------------------------------------------\*/

static void

connect\_to\_broker(void)

{

/\* Connect to MQTT server \*/

mqtt\_connect(&conn, conf.broker\_ip, conf.broker\_port,

(conf.pub\_interval \* 3) / CLOCK\_SECOND,

#if MQTT\_5

MQTT\_CLEAN\_SESSION\_ON,

MQTT\_PROP\_LIST\_NONE);

#else

MQTT\_CLEAN\_SESSION\_ON);

#endif

state = STATE\_CONNECTING;

}

/\*---------------------------------------------------------------------------\*/

#if MQTT\_5\_AUTH\_EN

static void

send\_auth(struct mqtt\_prop\_auth\_event \*auth\_info, mqtt\_auth\_type\_t auth\_type)

{

mqtt\_prop\_clear\_prop\_list(&auth\_props);

if(auth\_info->auth\_method.length) {

(void)mqtt\_prop\_register(&auth\_props,

NULL,

MQTT\_FHDR\_MSG\_TYPE\_AUTH,

MQTT\_VHDR\_PROP\_AUTH\_METHOD,

auth\_info->auth\_method.string);

}

if(auth\_info->auth\_data.len) {

(void)mqtt\_prop\_register(&auth\_props,

NULL,

MQTT\_FHDR\_MSG\_TYPE\_AUTH,

MQTT\_VHDR\_PROP\_AUTH\_DATA,

auth\_info->auth\_data.data,

auth\_info->auth\_data.len);

}

/\* Connect to MQTT server \*/

mqtt\_auth(&conn, auth\_type, auth\_props);

if(state != STATE\_CONNECTING) {

LOG\_DBG("MQTT reauthenticating\n");

}

}

#endif

/\*---------------------------------------------------------------------------\*/

static void

ping\_parent(void)

{

if(have\_connectivity()) {

uip\_icmp6\_send(uip\_ds6\_defrt\_choose(), ICMP6\_ECHO\_REQUEST, 0,

ECHO\_REQ\_PAYLOAD\_LEN);

} else {

LOG\_WARN("ping\_parent() is called while we don't have connectivity\n");

}

}

/\*---------------------------------------------------------------------------\*/

static void

state\_machine(void)

{

switch(state) {

case STATE\_INIT:

/\* If we have just been configured register MQTT connection \*/

mqtt\_register(&conn, &mqtt\_client\_process, client\_id, mqtt\_event,

MAX\_TCP\_SEGMENT\_SIZE);

/\*

\* If we are not using the quickstart service (thus we are an IBM

\* registered device), we need to provide user name and password

\*/

if(strncasecmp(conf.org\_id, QUICKSTART, strlen(conf.org\_id)) != 0) {

if(strlen(conf.auth\_token) == 0) {

LOG\_ERR("User name set, but empty auth token\n");

state = STATE\_ERROR;

break;

} else {

mqtt\_set\_username\_password(&conn, MQTT\_CLIENT\_USERNAME,

conf.auth\_token);

}

}

/\* \_register() will set auto\_reconnect. We don't want that. \*/

conn.auto\_reconnect = 0;

connect\_attempt = 1;

#if MQTT\_5

mqtt\_prop\_create\_list(&publish\_props);

/\* this will be sent with every publish packet \*/

(void)mqtt\_prop\_register(&publish\_props,

NULL,

MQTT\_FHDR\_MSG\_TYPE\_PUBLISH,

MQTT\_VHDR\_PROP\_USER\_PROP,

"Contiki", "v4.5+");

mqtt\_prop\_print\_list(publish\_props, MQTT\_VHDR\_PROP\_ANY);

#endif

state = STATE\_REGISTERED;

LOG\_DBG("Init MQTT version %d\n", MQTT\_PROTOCOL\_VERSION);

/\* Continue \*/

case STATE\_REGISTERED:

if(have\_connectivity()) {

/\* Registered and with a public IP. Connect \*/

LOG\_DBG("Registered. Connect attempt %u\n", connect\_attempt);

ping\_parent();

connect\_to\_broker();

} else {

leds\_on(MQTT\_CLIENT\_STATUS\_LED);

ctimer\_set(&ct, NO\_NET\_LED\_DURATION, publish\_led\_off, NULL);

}

etimer\_set(&publish\_periodic\_timer, NET\_CONNECT\_PERIODIC);

return;

break;

case STATE\_CONNECTING:

leds\_on(MQTT\_CLIENT\_STATUS\_LED);

ctimer\_set(&ct, CONNECTING\_LED\_DURATION, publish\_led\_off, NULL);

/\* Not connected yet. Wait \*/

LOG\_DBG("Connecting (%u)\n", connect\_attempt);

break;

case STATE\_CONNECTED:

/\* Don't subscribe unless we are a registered device \*/

if(strncasecmp(conf.org\_id, QUICKSTART, strlen(conf.org\_id)) == 0) {

LOG\_DBG("Using 'quickstart': Skipping subscribe\n");

state = STATE\_PUBLISHING;

}

/\* Continue \*/

case STATE\_PUBLISHING:

/\* If the timer expired, the connection is stable. \*/

if(timer\_expired(&connection\_life)) {

/\*

\* Intentionally using 0 here instead of 1: We want RECONNECT\_ATTEMPTS

\* attempts if we disconnect after a successful connect

\*/

connect\_attempt = 0;

}

if(mqtt\_ready(&conn) && conn.out\_buffer\_sent) {

/\* Connected. Publish \*/

if(state == STATE\_CONNECTED) {

subscribe();

state = STATE\_PUBLISHING;

} else {

leds\_on(MQTT\_CLIENT\_STATUS\_LED);

ctimer\_set(&ct, PUBLISH\_LED\_ON\_DURATION, publish\_led\_off, NULL);

LOG\_DBG("Publishing\n");

publish();

}

etimer\_set(&publish\_periodic\_timer, conf.pub\_interval);

/\* Return here so we don't end up rescheduling the timer \*/

return;

} else {

/\*

\* Our publish timer fired, but some MQTT packet is already in flight

\* (either not sent at all, or sent but not fully ACKd).

\*

\* This can mean that we have lost connectivity to our broker or that

\* simply there is some network delay. In both cases, we refuse to

\* trigger a new message and we wait for TCP to either ACK the entire

\* packet after retries, or to timeout and notify us.

\*/

LOG\_DBG("Publishing... (MQTT state=%d, q=%u)\n", conn.state,

conn.out\_queue\_full);

}

break;

case STATE\_DISCONNECTED:

LOG\_DBG("Disconnected\n");

if(connect\_attempt < RECONNECT\_ATTEMPTS ||

RECONNECT\_ATTEMPTS == RETRY\_FOREVER) {

/\* Disconnect and backoff \*/

clock\_time\_t interval;

#if MQTT\_5

mqtt\_disconnect(&conn, MQTT\_PROP\_LIST\_NONE);

#else

mqtt\_disconnect(&conn);

#endif

connect\_attempt++;

interval = connect\_attempt < 3 ? RECONNECT\_INTERVAL << connect\_attempt :

RECONNECT\_INTERVAL << 3;

LOG\_DBG("Disconnected. Attempt %u in %lu ticks\n", connect\_attempt, interval);

etimer\_set(&publish\_periodic\_timer, interval);

state = STATE\_REGISTERED;

return;

} else {

/\* Max reconnect attempts reached. Enter error state \*/

state = STATE\_ERROR;

LOG\_DBG("Aborting connection after %u attempts\n", connect\_attempt - 1);

}

break;

case STATE\_CONFIG\_ERROR:

/\* Idle away. The only way out is a new config \*/

LOG\_ERR("Bad configuration.\n");

return;

case STATE\_ERROR:

default:

leds\_on(MQTT\_CLIENT\_STATUS\_LED);

/\*

\* 'default' should never happen.

\*

\* If we enter here it's because of some error. Stop timers. The only thing

\* that can bring us out is a new config event

\*/

LOG\_ERR("Default case: State=0x%02x\n", state);

return;

}

/\* If we didn't return so far, reschedule ourselves \*/

etimer\_set(&publish\_periodic\_timer, STATE\_MACHINE\_PERIODIC);

}

/\*---------------------------------------------------------------------------\*/

static void

init\_extensions(void)

{

int i;

for(i = 0; i < mqtt\_client\_extension\_count; i++) {

if(mqtt\_client\_extensions[i]->init) {

mqtt\_client\_extensions[i]->init();

}

}

}

/\*---------------------------------------------------------------------------\*/

PROCESS\_THREAD(mqtt\_client\_process, ev, data)

{

PROCESS\_BEGIN();

printf("MQTT Client Process\n");

if(init\_config() != 1) {

PROCESS\_EXIT();

}

init\_extensions();

update\_config();

def\_rt\_rssi = 0x8000000;

uip\_icmp6\_echo\_reply\_callback\_add(&echo\_reply\_notification,

echo\_reply\_handler);

etimer\_set(&echo\_request\_timer, conf.def\_rt\_ping\_interval);

/\* Main loop \*/

while(1) {

PROCESS\_YIELD();

if(ev == button\_hal\_release\_event &&

((button\_hal\_button\_t \*)data)->unique\_id == BUTTON\_HAL\_ID\_BUTTON\_ZERO) {

if(state == STATE\_ERROR) {

connect\_attempt = 1;

state = STATE\_REGISTERED;

}

}

if((ev == PROCESS\_EVENT\_TIMER && data == &publish\_periodic\_timer) ||

ev == PROCESS\_EVENT\_POLL ||

(ev == button\_hal\_release\_event &&

((button\_hal\_button\_t \*)data)->unique\_id == BUTTON\_HAL\_ID\_BUTTON\_ZERO)) {

state\_machine();

}

if(ev == PROCESS\_EVENT\_TIMER && data == &echo\_request\_timer) {

ping\_parent();

etimer\_set(&echo\_request\_timer, conf.def\_rt\_ping\_interval);

}

}

PROCESS\_END();

}

/\*---------------------------------------------------------------------------\*/

**OUTPUT:**









