Internet of Things
Department of Computer Engineering
Santa Clara University, California
Behnam Dezfouli, PhD

Santa Clara University
Internet of Things
RESEARCH LAB
SIOTLAB

Internet of Things

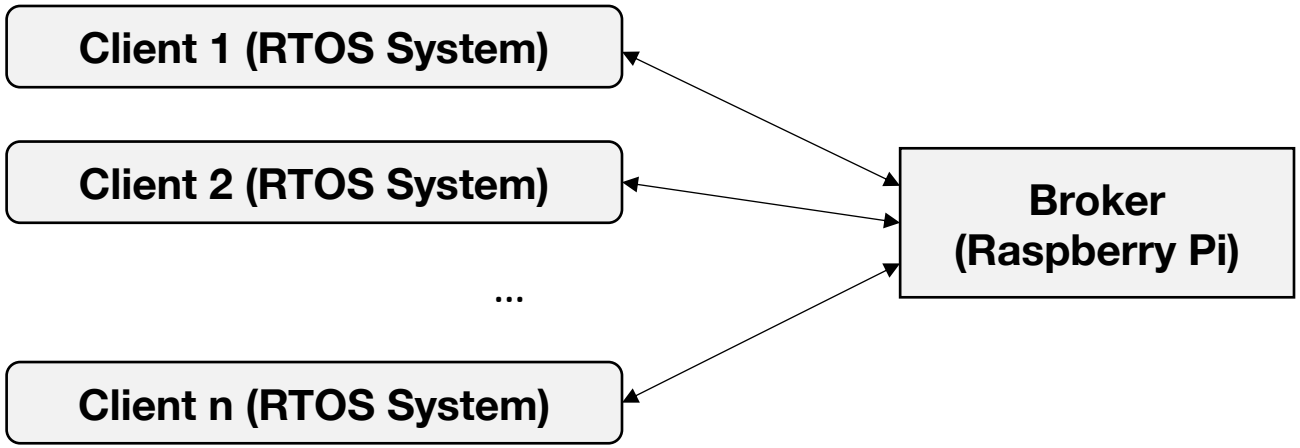
Lab 10
MQTT
(Using RTOS)

MQTT Projects

Software Development using WICED® Studio and Open-Source Tools

MQTT Project

- We are interested in implementing the following application:



- All clients subscribe to topic `light/led1`
- When button1 on the shield is pressed, the light status is changed and a messages is published to MQTT broker
- So the light status of all the clients is affected by other clients

MQTT Project

- **Before we start the installation, add the following line to the makefile**

```
$(NAME)_COMPONENTS := protocols/MQTT
```

MQTT Project

Client Side

Pseudo-code

```
mqtt_connection_event_cb ()
{
    if (publish message received from the broker) {
         print the message;
         change LED status accordingly;
    }
}


application_start() {
    do {
         establish MQTT connection;
         if (connection was unsuccessful) break;

         While(1)
         {
            wait for button semaphore;
            publish the button status;
            if (publish was unsuccessful) break;
         }

         close the connection;

       } while(1)

    house keeping;
}
```

MQTT Project

Client Side

MQTT Client
(subscriber_publisher) - Part 1

```
#include "wiced.h"
#include "mqtt_api.h"

#define MQTT_BROKER_ADDRESS                "rpi-server-mqtt-1"
#define WICED_TOPIC                        "light/led1"
#define CLIENT_ID                          "Behnam_Dezfouli"
#define MQTT_REQUEST_TIMEOUT               (5000)
#define MQTT_DELAY_IN_MILLISECONDS         (1000)
#define MQTT_PUBLISH_RETRY_COUNT           (3)
#define MQTT_SUBSCRIBE_RETRY_COUNT         (3)
#define MSG_ON                             "LIGHT ON-"
#define MSG_OFF                            "LIGHT OFF-"


static wiced_ip_address_t                  broker_address;

static wiced_mqtt_event_type_t             received_event;

static wiced_semaphore_t                   event_semaphore;
static wiced_semaphore_t                   wake_semaphore;
static uint8_t                             pub_in_progress = 0;
```

The MQTT event we just received

MQTT Project

Client Side

MQTT Client
(`subscriber_publisher`) - Part 2

- Using the below function, we can **check if a desired event happens within a given deadline**
- `WICED_ERROR` happens when no event is received or the event is not what we want

```
/*
 * A blocking call to an expected event.
 */
static wiced_result_t wait_for_response(
        wiced_mqtt_event_type_t expected_event, uint32_t timeout )
{
    if ( wiced_rtos_get_semaphore( &event_semaphore, timeout ) != WICED_SUCCESS )
    {
        return WICED_ERROR;
    }

    else
    {
        if (expected_event != received_event )
        {
            return WICED_ERROR;
        }
    }
    return WICED_SUCCESS;
}
```

- The `event_semaphore` is set when we receive any event

- Check if the received event is what we are waiting for
- `received_event` is a global variable

MQTT Project

Client Side

MQTT Client
(subscriber_publisher) - Part 3

- This function **handles the MQTT events received**
- This function is passed to the `mqtt_conn_open` function to **serve as the event callback function**, which is used for notifying the events from library

```
static wiced_result_t mqtt_connection_event_cb( wiced_mqtt_object_t mqtt_object,
                                                wiced_mqtt_event_info_t *event )
{
    static char data[30];

    switch ( event->type )
    {
    case WICED_MQTT_EVENT_TYPE_SUBCRIBED:
    {
        WPRINT_APP_INFO(( "\nSubscription acknowledged!\n" ));
        received_event = event->type;
        wiced_rtos_set_semaphore( &event_semaphore );
    }
        break;
    case WICED_MQTT_EVENT_TYPE_CONNECT_REQ_STATUS:
    case WICED_MQTT_EVENT_TYPE_DISCONNECTED:
    case WICED_MQTT_EVENT_TYPE_PUBLISHED:
    case WICED_MQTT_EVENT_TYPE_UNSUBSCRIBED:
    {
        received_event = event->type;
        wiced_rtos_set_semaphore( &event_semaphore );
    }
        break;
```

See the list of events in the next slide

We set the `recived_event` global variable and the semaphore so that function `wait_for_response()` can check the event validity

MQTT Project

Client Side

Documentation

`WICED_MQTT_EVENT_TYPE_CONNECT_REQ_STATUS`	Event sent when broker accepts CONNECT request
`WICED_MQTT_EVENT_TYPE_DISCONNECTED`	Event sent when broker accepts DISCONNECT request, or when there is any network issue
`WICED_MQTT_EVENT_TYPE_PUBLISHED`	Event sent for QoS-1 and QoS-2 for the published *when successfully delivered*. No event will be sent for QOS-0.
`WICED_MQTT_EVENT_TYPE_SUBCRIBED`	Event sent when broker accepts SUBSCRIBED request
`WICED_MQTT_EVENT_TYPE_UNSUBSCRIBED`	Event sent when broker accepts UNSUBSCRIBED request
`WICED_MQTT_EVENT_TYPE_PUBLISH_MSG_RECEIVED`	Event sent when PUBLISH message is received from the broker for a subscribed topic

MQTT Project

Client Side

Documentation

```
/* MQTT Event info */
typedef struct wiced_mqtt_event_info_s
{
    wiced_mqtt_event_type_t  type; /* Message event type */

    union
    {
        /* Valid only for WICED_MQTT_EVENT_TYPE_CONNECT_REQ_STATUS event.
        Indicates the error identified while connecting to Broker */
        wiced_mqtt_conn_err_code_t      err_code;

        /* Valid only for WICED_MQTT_EVENT_TYPE_PUBLISHED,
        WICED_MQTT_EVENT_TYPE_SUBCRIBED, WICED_MQTT_EVENT_TYPE_UNSUBSCRIBED
        events. Indicates message ID */
        wiced_mqtt_msgid_t    msgid;

        /* Valid only for WICED_MQTT_EVENT_TYPE_PUBLISH_MSG_RECEIVED event.
        Indicates the message received from Broker */
        wiced_mqtt_topic_msg_t   pub_recvd;

    } data;                     /* Event data */

} wiced_mqtt_event_info_t;
```

See the previous table

When the event type is `WICED_MQTT_EVENT_TYPE_PUBLISH_MSG_RECEIVED`, we need to get the actual received data from `pub_recvd`

MQTT Project

Client Side

Documentation

```
/**
 * Contains the message received for a topic from the Broker
 */
typedef struct wiced_mqtt_topic_msg_s
{
 /* Name of the topic associated with the message. It's not 'null' terminated */
    uint8_t*    topic;

    uint32_t    topic_len; /* Length of the topic */

    uint8_t*    data;      /* Payload of the message */

    uint32_t    data_len;  /* Length of the message payload */

} wiced_mqtt_topic_msg_t;
```

MQTT Project

Client Side

MQTT Client (`subscriber_publisher`) – Part 4

Function `mqtt_connection_event_cb()` (cont'd)

```
case WICED_MQTT_EVENT_TYPE_PUBLISH_MSG_RECEIVED:
    {
        WPRINT_APP_INFO(( "\nReceived message from broker!\n" ));
        wiced_mqtt_topic_msg_t msg = event->data.pub_recvd;
        memcpy( data, msg.data, msg.data_len );
        data[ msg.data_len + 1 ] = '\0';
        if ( !strncmp( data, "LIGHT ON", 8 ) )
        {
            wiced_gpio_output_high( WICED_SH_LED1 );
            WPRINT_APP_INFO(( "LIGHT ON\n" ));
        }
        else
        {
            wiced_gpio_output_low( WICED_SH_LED1 );
            WPRINT_APP_INFO(( "LIGHT OFF\n" ));
        }
    }
        break;
    default:
        break;
    }
    return WICED_SUCCESS;
}
```

Event sent when PUBLISH message is received from the broker for a subscribed topic

MQTT Project

Client Side

MQTT Client (subscriber_publisher) – Part 5

This function is **called to establish a connection**

```
/*
 * Open a connection and wait for MQTT_REQUEST_TIMEOUT period to receive a
connection open OK event
 */
static wiced_result_t mqtt_conn_open( wiced_mqtt_object_t mqtt_obj,
         wiced_ip_address_t *address, wiced_interface_t interface,
            wiced_mqtt_callback_t callback, wiced_mqtt_security_t *security )
{
    wiced_mqtt_pkt_connect_t conninfo;
    wiced_result_t ret = WICED_SUCCESS;

    memset( &conninfo, 0, sizeof( conninfo ) );

    conninfo.port_number = 1883;
    conninfo.mqtt_version = WICED_MQTT_PROTOCOL_VER4;
    conninfo.clean_session = 1;
    conninfo.client_id = (uint8_t*) CLIENT_ID;
    conninfo.keep_alive = 5;
    conninfo.username = (uint8_t*)"iotstudent";
    conninfo.password = (uint8_t*)"coen";
```

MQTT Project

Client Side

MQTT Client (subscriber_publisher) – Part 6

```
    ret = wiced_mqtt_connect( mqtt_obj, address, interface, callback,
                                                security, &conninfo );
```

See the next slide for the documentation

```
    if ( ret != WICED_SUCCESS )
    {
        return WICED_ERROR;
    }

    if ( wait_for_response( WICED_MQTT_EVENT_TYPE_CONNECT_REQ_STATUS,
                            MQTT_REQUEST_TIMEOUT ) != WICED_SUCCESS )
    {
        return WICED_ERROR;
    }

    return WICED_SUCCESS;
}
```

Wait to receive WICED_MQTT_EVENT_TYPE_CONNECT_REQ_STATUS

MQTT Project

Client Side

Documentation

```
/** Contains information related to establishing connection with Broker */
typedef struct wiced_mqtt_pkt_connect_s
{
    /* Indicates mqtt broker port number to which publisher/subscriber want to
    communicate ( 1883 as open port, 8883 as secure port) */
    uint16_t    port_number;

    /* Indicates mqtt version number. Supported versions are 3 and 4. Any value
    other than 4 will be treated as 3 (default)*/
    uint8_t     mqtt_version;

    /* Indicates keep alive interval to Broker */
    uint16_t    keep_alive;

    /* Indicates if the session to be cleanly started */
    uint8_t     clean_session;

    uint8_t*    client_id;   /* Client ID */
    uint8_t*    username;    /* Username to connect to Broker */
    uint8_t*    password;    /* Password to connect to Broker */
    uint8_t*    peer_cn;

} wiced_mqtt_pkt_connect_t;
```

MQTT Project

Client Side

Documentation

```
wiced_result_t wiced_mqtt_connect (wiced_mqtt_object_t  mqtt_obj,
        wiced_ip_address_t *    address,
        wiced_interface_t   interface,
        wiced_mqtt_callback_t   callback,
        wiced_mqtt_security_t *     security,
        wiced_mqtt_pkt_connect_t *  conninfo
    )
```

Establishes connection with MQTT broker

NOTE: This is an asynchronous API. Connection status will be notified using callback function.

`WICED_MQTT_EVENT_TYPE_CONNECTED` event will be sent using callback function

Parameters

- **`mqtt_obj`**: Contains address of a memory location which is passed during MQTT init
- **`address`**: IP address of the Broker
- **`interface`**: Network interface to be used for establishing connection with Broker
- **`callback`**: Event callback function which is used for notifying the events from library
- **`security`**: Security related information for establishing secure connection with Broker. If NULL, connection with Broker will be unsecured.
- **`conninfo`**: MQTT connect message related information

Returns

- **`wiced_result_t`**
- NOTE: Allocate memory for `conninfo->client_id`, `conninfo->username`, `conninfo->password` in non-stack area. And free/resuse them after getting event `WICED_MQTT_EVENT_TYPE_CONNECT_REQ_STATUS` or `WICED_MQTT_EVENT_TYPE_DISCONNECTED`

MQTT Project

Client Side

This function is called to subscribe to a topic

MQTT Client
(`subscriber_publisher`) – Part 7

```
/*
 * Subscribe to WICED_TOPIC and wait for 5 seconds to receive an ACK.
 */
static wiced_result_t mqtt_app_subscribe( wiced_mqtt_object_t mqtt_obj,
                                          char *topic, uint8_t qos )
{
    wiced_mqtt_msgid_t pktid;
    pktid = wiced_mqtt_subscribe( mqtt_obj, topic, qos );

    if ( pktid == 0 )
    {
        return WICED_ERROR;
    }

    if ( wait_for_response( WICED_MQTT_EVENT_TYPE_SUBCRIBED,
                            MQTT_REQUEST_TIMEOUT ) != WICED_SUCCESS )
    {
        return WICED_ERROR;
    }

    return WICED_SUCCESS;
}
```

Wait to receive `WICED_MQTT_EVENT_TYPE_SUBCRIBED`

MQTT Project

Client Side

Documentation

Subscribe for a topic with MQTT Broker

```
 *
 * NOTE: This is an asynchronous API. Subscribe status will be notified
 * using callback function.
 * WICED_MQTT_EVENT_TYPE_SUBCRIBED event will be sent using callback function
 *
 * @param[in] mqtt_obj          : Contains address of a memory location which is
 * passed during MQTT init
 * @param[in] topic             : Contains the topic to be subscribed to
 * @param[in] qos               : QoS level to be used for receiving the message
 * on the given topic
 *
 * @return wiced_mqtt_msgid_t   : ID for the message being subscribed
 * NOTE: Allocate memory for topic in non-stack area.
 *       And free/resuse them after getting event WICED_MQTT_EVENT_TYPE_SUBCRIBED
 *       or WICED_MQTT_EVENT_TYPE_DISCONNECTED for given message ID
(wiced_mqtt_msgid_t)
 */
wiced_mqtt_msgid_t wiced_mqtt_subscribe( wiced_mqtt_object_t mqtt_obj,
                                         char *topic, uint8_t qos );
```

MQTT Project

Client Side

This function is called to publish to a topic

MQTT Client
(subscriber_publisher) – Part 8

```
/*
 * Publish (send) message to WICED_TOPIC and wait for 5 seconds to receive a
PUBCOMP (as it is QoS=2).
 */
static wiced_result_t mqtt_app_publish( wiced_mqtt_object_t mqtt_obj,
                 uint8_t qos, uint8_t *topic, uint8_t *data, uint32_t data_len )
{
    wiced_mqtt_msgid_t pktid;

    pktid = wiced_mqtt_publish( mqtt_obj, topic, data, data_len, qos );

    if ( pktid == 0 )
    {
        return WICED_ERROR;
    }

    if ( wait_for_response( WICED_MQTT_EVENT_TYPE_PUBLISHED,
                                MQTT_REQUEST_TIMEOUT ) != WICED_SUCCESS )
    {
        return WICED_ERROR;
    }
    return WICED_SUCCESS;
}
```

Wait to receive event
WICED_MQTT_EVENT_TYPE_PUBLISHED

MQTT Project

Client Side

MQTT Client
(subscriber_publisher) – Part 9

This function is called to close the MQTT connection

```
/*
 * Close a connection and wait for 5 seconds to receive a connection close OK
event
 */
static wiced_result_t mqtt_conn_close( wiced_mqtt_object_t mqtt_obj )
{
    if ( wiced_mqtt_disconnect( mqtt_obj ) != WICED_SUCCESS )
    {
        return WICED_ERROR;
    }
    if ( wait_for_response( WICED_MQTT_EVENT_TYPE_DISCONNECTED,
                                    MQTT_REQUEST_TIMEOUT ) != WICED_SUCCESS )
    {
        return WICED_ERROR;
    }
    return WICED_SUCCESS;
}
```

MQTT Project

Client Side

```
wiced_mqtt_msgid_t wiced_mqtt_publish (
        wiced_mqtt_object_t mqtt_obj, uint8_t *topic,
                uint8_t *data,  uint32_t data_len, uint8_t qos
)
```

Publish message to MQTT Broker on the given Topic.

- NOTE: This is **an asynchronous API**. Publish status will be notified using using callback function. `WICED_MQTT_EVENT_TYPE_PUBLISHED` event will be sent using callback function

Parameters

- **`mqtt_obj`**: Contains address of a memory location which is passed during MQTT init
- **`topic`**: Contains the topic on which the message to be published
- **`message`**: Pointer to the message to be published
- **`msg_len`**: Length of the message pointed by 'message' pointer
- **`qos`**: QoS level to be used for publishing the given message

Returns

`wiced_mqtt_msgid_t`: ID for the message being published

MQTT Project

Client Side

MQTT Client (`subscriber_publisher`) – Part 10

- This is the callback function of the button interrupt
- When the button is pressed, this function sets a semaphore that unlocks MQTT publishing

```
static void button_callback( void* arg )
{
    if(pub_in_progress == 0)
    {
        pub_in_progress = 1;
        wiced_rtos_set_semaphore( &wake_semaphore );
    }
}
```

MQTT Project

Client Side

The main thread

MQTT Client
(subscriber_publisher) – Part 11

```
void application_start( void )
{
    wiced_mqtt_object_t     mqtt_object;
    wiced_result_t          ret = WICED_SUCCESS;
    int                     connection_retries = 0;
    int                     pub_sub_retries = 0;
    int                     count = 0;
    char                    msg[30];

    wiced_init( );


    /* Bring up the network interface */
    ret = wiced_network_up( WICED_STA_INTERFACE,
                            WICED_USE_EXTERNAL_DHCP_SERVER, NULL );

    if ( ret != WICED_SUCCESS )
    {
        WPRINT_APP_INFO( ( "\nNot able to join the requested AP\n\n" ) );
        return;
    }

    /* configure push button to publish a message */
    wiced_gpio_input_irq_enable( WICED_SH_MB1, IRQ_TRIGGER_RISING_EDGE,
                                 button_callback, NULL );
```

The MQTT object is used for connection establishment, subscribing, publishing, and closing connection

MQTT Project

Client Side

MQTT Client
(subscriber_publisher) – Part 12

```
/* Allocate memory for MQTT object*/
mqtt_object = (wiced_mqtt_object_t) malloc(
                        WICED_MQTT_OBJECT_MEMORY_SIZE_REQUIREMENT );
if ( mqtt_object == NULL )
{
    WPRINT_APP_ERROR("Don't have memory to allocate for MQTT object...\n");
    return;
}

WPRINT_APP_INFO( ( "Resolving IP address of MQTT broker...\n" ) );
ret = wiced_hostname_lookup( MQTT_BROKER_ADDRESS, &broker_address, 10000,
                                                    WICED_STA_INTERFACE);

WPRINT_APP_INFO(("Resolved Broker IP: %u.%u.%u.%u\n\n",
        (uint8_t)(GET_IPV4_ADDRESS(broker_address) >> 24),
        (uint8_t)(GET_IPV4_ADDRESS(broker_address) >> 16),
        (uint8_t)(GET_IPV4_ADDRESS(broker_address) >> 8),
        (uint8_t)(GET_IPV4_ADDRESS(broker_address) >> 0)));

if ( ret == WICED_ERROR || broker_address.ip.v4 == 0 )
{
    WPRINT_APP_INFO(("Error in resolving DNS\n"));
    return;
}
```

MQTT Project

Client Side

MQTT Client (subscriber_publisher) – Part 13

```
wiced_rtos_init_semaphore( &wake_semaphore );
wiced_mqtt_init( mqtt_object );
wiced_rtos_init_semaphore( &event_semaphore );

do
{
    WPRINT_APP_INFO(("[MQTT] Opening connection..."));
    do
    {
        ret = mqtt_conn_open( mqtt_object, &broker_address,
                              WICED_STA_INTERFACE, mqtt_connection_event_cb, NULL );

        connection_retries++ ;
    } while ( ( ret != WICED_SUCCESS ) && ( connection_retries <
                                WICED_MQTT_CONNECTION_NUMBER_OF_RETRIES ) );

    if ( ret != WICED_SUCCESS )
    {
        WPRINT_APP_INFO(("Failed connection!\n"));
        break;
    }
    WPRINT_APP_INFO(("Successful connection!\n"));
```

Whenever the inner loop fails, this loop retries connection establishment

Note that waiting to receive a proper event occurs inside this function, which we have previously defined

MQTT Project

Client Side

MQTT Client (`subscriber_publisher`) – Part 14

```
WPRINT_APP_INFO(("[MQTT] Subscribing..."));

do
{

    ret = mqtt_app_subscribe( mqtt_object, WICED_TOPIC,
                              WICED_MQTT_QOS_DELIVER_AT_MOST_ONCE );
    pub_sub_retries++ ;

} while ( ( ret != WICED_SUCCESS ) &&
                  ( pub_sub_retries < MQTT_SUBSCRIBE_RETRY_COUNT ) );


if ( ret != WICED_SUCCESS )
{
    WPRINT_APP_INFO(( Failed subscribing!\n"));
    return;
}
```

Note that waiting to receive a proper event occurs inside this function, which we have previously defined

MQTT Project

Client Side

In this loop, we wait for a button press, and then publish to the broker

MQTT Client (subscriber_publisher) – Part 15

```
while ( 1 )   {

            wiced_rtos_get_semaphore( &wake_semaphore, WICED_NEVER_TIMEOUT );

            if ( pub_in_progress == 1 )
            {
                WPRINT_APP_INFO(("[MQTT] Publishing..."));
                if ( count % 2 )
                {
                    strcpy(msg, MSG_ON);
                    strcat(msg, CLIENT_ID);
                }
                else
                {
                    strcpy(msg, MSG_OFF);
                    strcat(msg, CLIENT_ID);
                }
                pub_sub_retries = 0;
               // reset pub_sub_retries to 0 before going
               // into the loop so that the next publish after a
               // failure will still work
```

MQTT Project

Client Side

Note that waiting to receive a proper event occurs inside this function, which we have previously defined

MQTT Client (subscriber_publisher) – Part 16

```
do
{
   ret = mqtt_app_publish( mqtt_object, WICED_MQTT_QOS_DELIVER_AT_LEAST_ONCE,
                           (uint8_t*) WICED_TOPIC, (uint8_t*) msg, strlen( msg ) );
   pub_sub_retries++ ;
} while ( ( ret != WICED_SUCCESS ) &&
                         ( pub_sub_retries < MQTT_PUBLISH_RETRY_COUNT ) );

   if ( ret != WICED_SUCCESS )
   {
     WPRINT_APP_INFO(( Failed publishing!\n));
     break;//break the loop and reconnect
   }
   else
   {
     WPRINT_APP_INFO(( Successful publishing!\n));
   }

     pub_in_progress = 0;
     count++ ;
}

 wiced_rtos_delay_milliseconds( 100 );
}
```

MQTT Project

Client Side

MQTT Client
(subscriber_publisher) – Part 17

```
        pub_in_progress = 0; // Reset flag if we got a failure so that another
         // button push is needed after a failure

        WPRINT_APP_INFO(("[MQTT] Closing connection..."));
        mqtt_conn_close( mqtt_object );

        wiced_rtos_delay_milliseconds( MQTT_DELAY_IN_MILLISECONDS * 2 );
    } while ( 1 );
```

The main loop is repeated to re-establish the connection if necessary

```
    wiced_rtos_deinit_semaphore( &event_semaphore );
    WPRINT_APP_INFO(("[MQTT] Deinit connection...\n"));
    ret = wiced_mqtt_deinit( mqtt_object );
    wiced_rtos_deinit_semaphore( &wake_semaphore );
    free( mqtt_object );
    mqtt_object = NULL;

    return;
}
```

Deliverable

Student Work

❖**Task 10-1.**

- **Modify the code to try DNS resolve for up to 5 times**

❖**Task 10-2.**

- Install MQTT server on your Raspberry Pi and configure it
- Run the server in verbose mode to see the outputs
- **What happens when you reduce the keep alive duration?**
- **Install a MQTT client on your laptop/phone, connect to the server, and use your phone/laptop to change the LED status**

✓The rest of the slides provide you with some instructions regarding the installation of MQTT

Installation and Configuration of MQTT Server

MQTT Project

Installation and Configuration of MQTT Server

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

eclipse

- We use the **open-source MQTT implementation** from Eclipse
- This implementation is called **Mosquitto**

- Install **Mosquitto** on Raspberry Pi
- First import the repository package signing key:

➢ `wget http://repo.mosquitto.org/debian/mosquitto-repo.gpg.key`

➢ `sudo apt-key add mosquitto-repo.gpg.key`

- To make the repository available to apt:

➢ `cd /etc/apt/sources.list.d`

MQTT Project

Installation and Configuration of MQTT Server

- Then run:

➢ `sudo wget http://repo.mosquitto.org/debian/mosquitto-jessie.list`

- Update apt information:

➢ `sudo apt-get update`

➢ `sudo apt-get install libwebsockets-dev`

➢ `sudo apt-get upgrade`

- Install Mosquitto:

➢ `sudo aptitude install libmosquitto-dev mosquitto mosquitto-clients`

Follow the prompt, and answer "n" (meaning no) until all 4 packages are selected for install AND are version *jessie* (DO NOT INSTALL Stable)
Ask the TA if you are confused!

Installation and Configuration of MQTT Server

Configuring the server

- You can find the documentation of the configuration file at: `https://mosquitto.org/man/mosquitto-conf-5.html`

- Open the configuration file as follows

➢ `sudo nano /etc/mosquitto/mosquitto.conf`

- Delete the line `include_dir /etc/mosquitto/conf.d` and add your configuration

Installation and Configuration of MQTT Server

`persistence [ true | false ]`

- If true, connection, subscription and message data will be written to the disk in `mosquitto.db` at the location dictated by `persistence_location`
- When mosquitto is restarted, it will reload the information stored in `mosquitto.db`
- The **data will be written to disk** when mosquitto closes and also at periodic intervals as defined by `autosave_interval`

`allow_anonymous [ true | false ]`

Boolean value that determines whether clients that connect without providing a username are allowed to connect

`password_file` *`filepath`*

Set the path to a password file. If defined, the contents of the file are used to control client access to the broker. If mosquitto is compiled without TLS support, then the password file should be a text file with each line in the format "username:password"

Installation and Configuration of MQTT Server

Configuration file:

```
pid_file /var/run/mosquitto.pid

persistence true
persistence_location /var/lib/mosquitto/

log_dest file /var/log/mosquitto/mosquitto.log

#include_dir /etc/mosquitto/conf.d

allow_anonymous false
password_file /etc/mosquitto/pwfile
listener 1883
```

MQTT Project

Installation and Configuration of MQTT Server

- We create password file using the `mosquitto_passwd` command
- The documentation of this file can be found at: `https://mosquitto.org/man/mosquitto_passwd-1.html`

- We create the password file and add a username/password to that:

➢ `sudo mosquitto_passwd -c /etc/mosquitto/pwfile iotstudent`

- Then we enter the password: "`coen`"

- Next, we need to restart the Pi:

➢ `sudo reboot`

MQTT Project

Installation and Configuration of MQTT Server

- We can run the MQTT server by:

➢ `sudo mosquitto -c /etc/mosquitto/mosquitto.conf`

- You can subscribe to a topic using:

➢ `mosquitto_sub  -t  light/#`

- You can publish to a topic using:

➢ `mosquitto_pub  -t  'light/led1' –m 'LIGHT OFF'`

Sample Python Client Code

Use this code if you want to use your laptop instead of an app on your phone!

MQTT Project

```
import paho.mqtt.client as mqtt
import time


# The callback for when the client receives a CONNACK response
from the server.
def on_connect(client, userdata, flags, rc):
    print("Connected with result code "+str(rc))


# Subscribing in on_connect() means that if we lose the
connection and reconnect then subscriptions will be renewed.
    client.subscribe("light/led1")


# The callback for when a PUBLISH message is received from the
server.
def on_message(client, userdata, msg):
    print(msg.topic+" "+str(msg.payload))
```

MQTT Project

```
client = mqtt.Client()
client.on_connect = on_connect
client.on_message = on_message


broker_ip = "YOUR_RPI_IP_ADDRESS"
client.username_pw_set("iotstudent", "coen")
client.connect(broker_ip, 1883, 60)


while True:
    # Get input message here! ("LIGHT ON" or "LIGHT OFF")


    # Make sure the client stays connected!
    client.reconnect()

    # Publish the message here! Use the following function
    # client.publish("topic", payload="your_message_to_send")
```