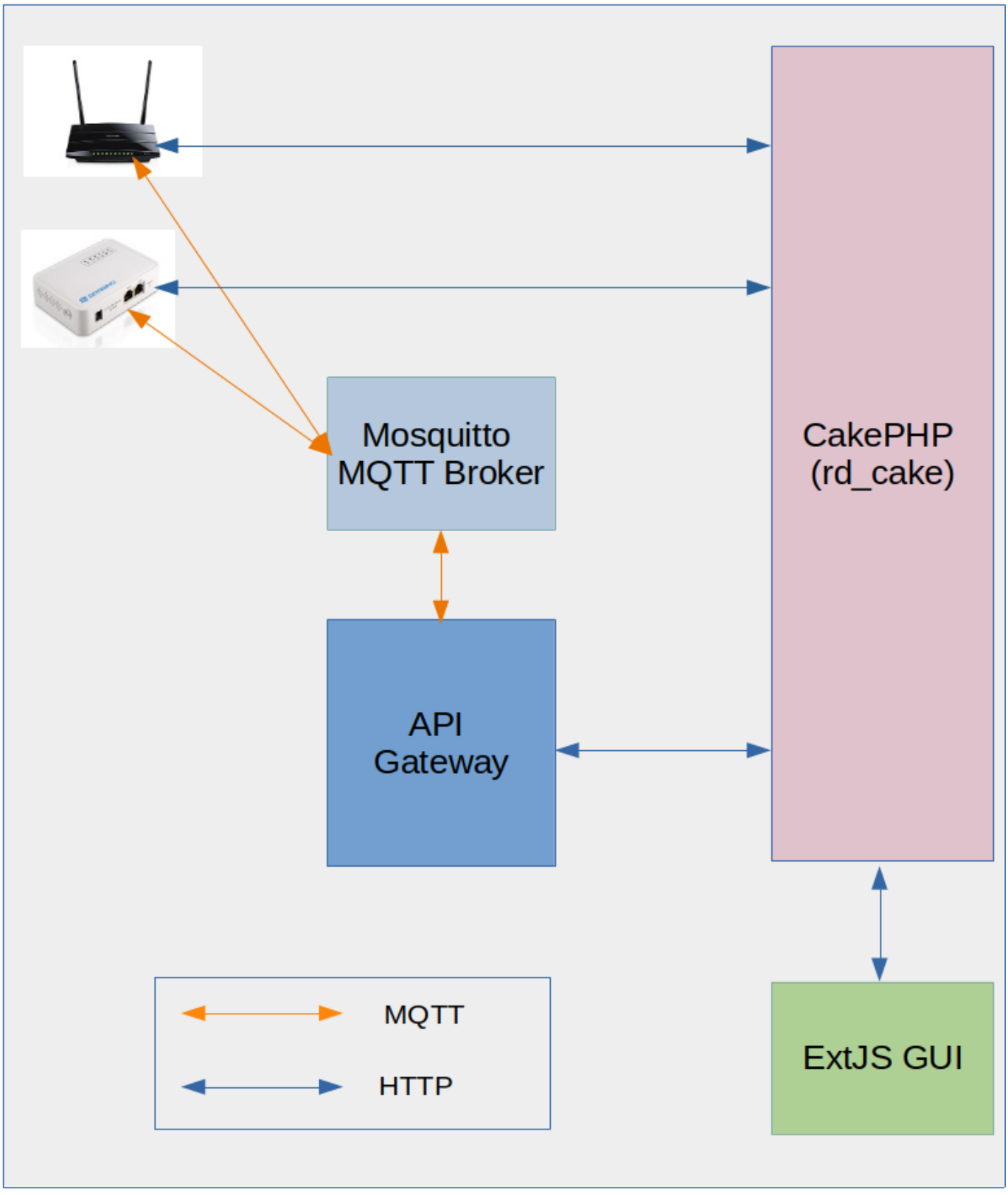
RADIUSdesk MQTT Implementation

Introduction

- MESHdesk and APdesk traditionally makes use of a heartbeat system to communicate and report to the back-end.
- We now also include a MQTT based implementation to allow real-time communication between the mesh nodes or access points and the back-end.
- This implementation is used as **a compliment** to the heartbeat system, making it more robust while offering you added real-time communication.
- The MQTT implementation is not compulsory in order to have a working deployment but it does offer a lot of advantages.
- It is ideal for hardware that is used in a IOT environment where you need immediate execution of commands.

Architecture

- Consider the following diagram and then the subsequent discussion of each of the components.



ExtJS GUI

- The ExtJS GUI can be used to send commands to the mesh nodes and access points managed by MESHdesk and APdesk respectively.
- The communication between ExtJS and the CakePHP application consists of REST-like API calls using HTTP or HTTPS.
- This means essentially that these actions can easily be automated or done with another GUI should the need arise.

CakePHP

- If MQTT support is enabled on the system and someone initiate a command execution action from the GUI, the controller code handling this request will send the request to the API Gateway.
- This communication between the CakePHP controller and the API Gateway also consist of REST-like API calls using HTTP or HTTPS.

API Gateway

- The API Gateway is a Node.js based web service that acts as a middle man.
- The MQTT implementation uses a command and response principle.
- The API Gateway
    - Receive instructions from CakePHP and translate them to MQTT **publish** actions (Command) which are published to the Mosquitto MQTT Broker.
    - Subscribe to MQTT topics (Response) on the Mosquitto MQTT Broker which will get input from the mesh nodes and access points and translate them to HTTP/HTTPS based API calls to CakePHP.

Mesh nodes and access points

- The mesh nodes and access points communicate with the CakePHP back-end using HTTP/HTTPS to fetch its configuration and do reporting.
- If the system has MQTT support enabled the mesh node or access point will configure itself to publish and subscribe to certain topics on the Mosquitto MQTT Broker.
- The system works on a **command** and **response** principle.
    - The mesh node or access point **subscribe** to a topic where it will expect **commands** from the API Gateway.
    - The mesh node or access point will **publish** to a topic where the API Gateway expect **responses**.
    - The API Gateway will **publish** to a topic where the mesh node or access point expect **commands**.
    - The API Gateway will **subscribe** to a topic where the mesh node or access points **publish** their **responses**.

Enable MQTT

- There are two components of the MQTT setup that needs to be configured
    - Configuration settings for mesh nodes and access points (MESHdesk and APdesk)
    - Configuration settings for the MQTT API Gateway.

Looking at the code

Command -> CakePHP Controller

- Lets look at the **/var/www/html/cake3/rd_cake/src/Controller/NodeActionsController.php** file.
- When an action is added to a node and MQTT is enabled on the system this code is executed:

```
if ($cfg['api_mqtt_enabled'] == "1"){
    //Talk to MQTT Broker
     $data = $this->_get_node_mac_mesh_id($formData['node_id']);
     $payload = [
```

```
            'mode'      => 'mesh',
            'node_id'   => $formData['node_id'],
            'mac'       => strtoupper($data['mac']),
            'mesh_id'   => strtoupper($data['ssid']),
            'cmd_id'    => $entity->id,
            'cmd'       => $formData['command'],
            'action'    => $formData['action'],
        ];

        if($this->_check_server($client, $cfg['api_gateway_url'], 5)){
            try {
                $client->request('POST', $cfg['api_gateway_url'] .
'/rd/mesh/command', ['json' => ['message' => $payload]]);
            } catch (\Exception $e) {
                // Do Nothing
            }
        }
}
```

Command -> API Gateway

- The API call to the API Gateway will execute this piece of code in the **/opt/Rdcore-API-Gateway/routes/rdmesh.js** file

```
router.post('/mesh/command', function(req, res){
    //var data = JSON.parse(req.body.message);
    var data = req.body.message;
    var message = JSON.stringify(data);
    console.log(message);
    client.publish('/RD/MESH/' + data.node_id + '/COMMAND', message);
    console.log("Published command to Mesh node: " + data.mac + " MODE
"+data.mode);
    res.json(message);
});
```

Command -> mqtt.lua

- Here is a snippet in **/etc/MESHdesk/mqtt.lua** which shows what it will do when a message is published from the API Gateway.
- This is the command which it will then respond to.

```
client.ON_MESSAGE = function(mid, topic, payload)
    -- Parse/Decode JSON Payload
    local jsonStr           = luci_json.parse(payload)
    -- Check if message belongs to us (MAC Address)
```

Response -> mqtt.lua

- Depending on the type of command the code in Lua will determine the correct response.
- Here is a snippet in **/etc/MESHdesk/mqtt.lua** which respond to the **execute** action.
- This is part of the code which are processing the command that the mesh node or access point received (inside the **ON_MESSAGE** event)

```
--Here depending on the value of jsonStr['action'] we will either just
execute the command or execute and report the output
if(jsonStr['action'] == 'execute')then
    print("MODE IS "..mode);
    if(mode == 'mesh')then
        message =
luci_json.stringify({mode=mode,node_id=nodeId,mesh_id=meshId,mac=macAddr,cmd
_id=cmdId,status='os_command'});
    end
    if(mode == 'ap')then
        message =
luci_json.stringify({mode=mode,ap_id=apId,mac=macAddr,cmd_id=cmdId,status='o
s_command'});
    end

    local cl_execute = mqtt.new();
    cl_execute:login_set(MQTT_USER, MQTT_PASS)
    cl_execute:connect(MQTT_HOST)
    --Connected now publish
    cl_execute.ON_CONNECT = function()
        cl_execute:publish(cmdTopic, message, qos, retain);
    end
    --Done publishing - now execute command
    cl_execute.ON_PUBLISH = function()
        cl_execute:disconnect();
        os.execute(jsonStr['cmd']);
    end
    cl_execute:loop_forever();
end
```

Response -> API Gateway

- The API Gateway subscribe to the topic which the mesh node or access point publishes to.
- Here is a snippet from the **/opt/Rdcore-API-Gateway/routes/rdmesh.js** file that execute some code when a message is received on that topic

```
default:
    request.put({
            url: mesh_controller + '/cake3/rd_cake/node-actions/node-
command.json',
            form: data
        },
        function (err, res, body) {
            if (err) {
                console.error('Error Occurred: ' + err);
            }

            console.log(body);
        }
    );
    break;
```

Response -> CakePHP Controller

- Finally we can look at the CakePHP code that process the response so our system know and can indicate the mesh node or access point did receive the instruction.
- Lets look at the **/var/www/html/cake3/rd_cake/src/Controller/NodeActionsController.php** file.

```
//--This comes from the NodeJS API Gateway Application in response to
'execute' type node_actions
//--This comes from the NodeJS API Gateway Application in FIRST response to
'execute_and_reply' type node_actions
public function nodeCommand(){

    if($this->request->is('put')){
        $data = $this->request->data;
        if((!empty($data['node_id']))||(!empty($data['ap_id']))){
            // update command status to fetched
            $model = 'NodeActions';
            if($data['mode'] == 'ap'){
                $model = 'ApActions';
            }

            $entity  = $this->{$model}->find()->where(['id' =>
$data['cmd_id']])->first();
            if($entity){
                $entity->status = 'fetched';
                $this->{$model}->save($entity);
            }

            $this->set(array(
                'data'          => $data,
                'success'       => true,
                '_serialize'    => array('data','success')
            ));
        } else {
            $this->set(array(
                'message'     => 'Node ID not found',
                'success'       => false,
                '_serialize'    => array('message','success')
            ));
        }

    } else {
        $this->set(array(
            'message'         => 'Send only PUT request',
            'success'       => false,
            '_serialize'    => array('message','success')
        ));
    }
}
```