TK3: Mobile & Ubiquitous Computing

Exercise E2: Bluetooth, SSDP & MQTT

Telecooperation Lab

Submission Date: 24.06.2019 by 14:00 MESZ

By handing in a solution, you confirm that you are the exclusive author(s) of all the materials.

Introduction

This year, the exercises will be all about creating the IoTK, a simplified Internet-of-Things platform, from scratch. In this exercise, we focus on Auto IP, simple service discovery, and RFID communication.

Hand-In

You have to hand-in the code before the submission date specified above. No late submissions will be allowed and results in 0 points. After the submission, each group will present their solution to one of the tutors.

Please note the presentation instructions on top of the slot registration (will be available before the presentations here: https://moodle.informatik.tu-darmstadt.de/mod/organizer/view.php?id=20279).

Task 1. GAP, SOA & SSDP (10 points)

An IoT network has to provide basic network services to enable communication between the devices. Hence, the first task focuses on GAP and SSDP.

Task 1.1

The IoTK network should support the gardening work this summer. Therefore, some sensor nodes are deployed outside to measure moisture, temperature and solar irradiation. Every time the user walks through the garden, the sensor nodes should transfer their data to your smartphone, using the "scan response request" feature of the BTLE GAP protocol.

Provide a message flow diagram to illustrate the message flow between the GAP peripheral devices and the central device.

Task 1.2

a) The so-called "Service Oriented Architecture" (SOA) is used to generally structure service-oriented systems. Please explain, in the context of SOA, the terms "Service Lookup" and "Service Discovery".

b) What types of "Service Registry" do you know from the lecture? Please describe them in one sentence each.

Task 1.3

In the lecture, you got to know SSDP. Imagine now that you have three control points: Alice's PC, Bob's PC, and Eve's PC, as well as two devices: a printer and a scanner. The printer offers two services: 1) printing black/white and 2) printing color pages. The scanner only offers the service to scan a document.

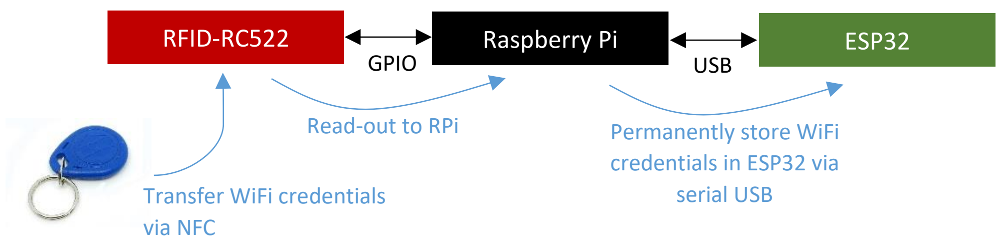
The printer advertises itself and its services to Alice and Bob via Notify messages, while Eve is actively performing a multicast search via M-Search to all devices to identify the scanner. Draw the whole SSDP system as flow chart including all components, control points, devices, and services.

Task 2. Plug ’n’ Wi-Fi extension, mDNS & MQTT (10 points)

In the previous exercise the Plug’n’WiFi service works with hard coded network information. As this is not scalable in environments with many different WiFi networks and lacks of security, we extend the setup with an RFID token as secure storage. Afterwards, the devices within one WiFi network should communicate using the widely used pub-sub protocol MQTT.

Task 2.1

Write code for the ESP32 that receives WiFi (i.e. SSID, and password suitable for encrypted WPA2 personal/psk as in E1) from the serial console and stores the credentials in a permanent memory on the ESP (as illustrated below). Moreover, the ESP should initiate a Wi-Fi connection on each boot using the stored credentials.


RFID-RC522
GPIO
Raspberry Pi
USB
ESP32
Read-out to RPi
Permanently store WiFi credentials in ESP32 via serial USB
Transfer WiFi credentials via NFC

In contrast to the previous programming task, the credential transfer is not started by connecting the ESP32, but with holding an RFID token (containing the WiFi credentials) to the RFID RC522 reader. To that end, the RFID reader (via GPIO) and the ESP32 (via USB) are both already connected to the Raspberry Pi.

The credentials should be stored using an NDEF record called “Wi-Fi Configuration Token” which contains unencrypted credentials (see https://ndeflib.readthedocs.io/en/stable/records/wifi.html#configuration-token). To store the credentials on RFID token you can use your NFC-enabled smartphone (see below for a list of suitable apps), write a separate writing program (not graded), or ask the tutors.

The following requirements must be fulfilled for this task:

- The ESP32 should signal success through permanently activating its LED1 for 10 seconds and failure through blinking LED1 for 10 seconds
- The WiFi credentials need to be permanently stored in the ESP32.
- The credential transfer starts with holding the respective RFID tag to the reader (the ESP32 should be already connected via USB).
- The ESP32 connects to a WiFi on each boot if valid credentials were stored.
- The Raspberry Pi automatically starts all necessary code on boot.
- It is only required to support credentials that allow to connect to encrypted WPA2 personal with SSID and password

Task 2.2

After connecting the ESP32s to the WiFi network (you may use the Raspberry Pi as an access point this time), they should register themselves at a discoverable MQTT broker using the PubSub Arduino library. To that end, you have to install “Mosquitto” on the Raspberry Pi. The IP of the broker should be automatically discovered by all ESP32s using mDNS (avahi). The ESP32 then automatically connects to the MQTT broker.

Task 2.3

To demonstrate the working MQTT connection, the IoTK should support "Mensa polling" (i.e., n participants want to vote if it is time to go to Mensa or not). Therefore, we use the infrastructure build up in task 2.1 and 2.2 to implement the following application:

- When no poll is active and a user long presses (> 3s) the "KEY" button on the ESP32 a poll is initiated:
    - After releasing the "KEY" button, the user shortly presses the "KEY" button **n** times to define the number of required positive/accepted participants in the poll
    - After no key input for 2 seconds, the led blinks as many times as specified by the user (1s on, 1s off for **n** times)
    - After blinking, the esp32 then sends a "POLL" message (including the required minimum number of accepts **n**) to all other MQTT clients (using a preconfigured MQTT topic) and starts blinking (2s on, 2s off) its on-board LED
- All ESP32 clients start blinking (1s on, 1s off) the on-board LED when they receive a "POLL" message
- A user can accept a poll by short pressing the "KEY" button. As a result, an "ACCEPT" message is sent and the LED is permanently enabled.
- A user can deny a poll by long pressing (> 3s) the "KEY" button. As a result, the LED of the ESP32 is disabled.
- When **n** participants (see POLL message above) agreed, the LED of all clients start fast blinking (0.5s on, 0.5s off). The initiator's ESP32 sends a CLEAR message after 5 minutes.
- While running a poll, the initiating user can also manually end the poll by long pressing the "KEY" button. The ESP32 then sends a CLEAR message.
- When receiving a CLEAR message, all receiving ESP32s go into the initial "no-poll" state (i.e. LED off, no poll active).

Links & Tutorials

ESP32 Libraries for mDNS & MQTT

- https://github.com/espressif/arduino-esp32/tree/master/libraries/ESPmDNS
- https://github.com/knolleary/pubsubclient

RPi + MQTT

- https://smarthome-blogger.de/tutorial/mqtt-raspberry-pi-einfuehrung/
- https://www.instructables.com/id/Installing-MQTT-BrokerMosquitto-on-Raspberry-Pi/

RPi + RFID RC522

- https://pimylifeup.com/raspberry-pi-rfid-rc522/
- For python implementations using MFC522: https://github.com/mxgxw/MFRC522-python
    - If you observe an AUTH error, this branch may help: https://github.com/mikicaivosevic/MFRC522-python

NFC Links

- NFC Writing Apps
    - NFC Tools
        - Android https://play.google.com/store/apps/details?id=com.wakdev.wdnfc
        - Ios https://itunes.apple.com/us/app/nfc-tools/id1252962749?mt=8
- Windows Library
    - https://github.com/andijakl/ndef-nfc (with NdefDemo app, no wifi credential support)