Universität Stuttgart

Institute of Parallel and
Distributed Systems (IPVS)

Universitätsstraße 38
D-70569 Stuttgart

Mobile Computing Lab
Assignment 2

Bluetooth Low Energy (BLE)

Frank Dürr, Saravana Murthy Palanisamy, Ahmad Slo, Zohaib Riaz

Outline

- Background: Bluetooth Low Energy (BLE)
- Task 1: Android BLE App: Weather App
- Task 2: Android BLE App: Fan Control App
- Organizational issues

Universität Stuttgart

Institute of Parallel and
Distributed Systems (IPVS)

Universitätsstraße 38
D-70569 Stuttgart

Bluetooth Low Energy

Motivation

The Internet of Things: Everything connected

Wireless sensors and actuators will be everywhere

- “Quantified Self”: monitor everything about your life
  - Fitness trackers, blood pressure, glucometers
- Environmental and urban monitoring
  - Air quality, noise level, temperature
- Home automation
- Industry 4.0
- Smart watches, wearables
- Proximity sensors (iBeacon)

Motivation

The Internet of Things: Everything connected

Wireless sensors and actuators will be everywhere

- "Quantified Self": monitor everything about your life
  - Fitness trackers, blood pressure, glucometers
- Environmental and urban monitoring
  - Air quality, noise level, temperature
- Home automation
- Industry 4.0
- Smart watches, wearables
- Proximity sensors (iBeacon)

Bluetooth Low Energy

- 2.4 GHz wireless communication technology
- Low range
  - ~ 10 meters
- Ultra low energy consumption
  - Run from coin cells for months or years
  - No need for chargers; rather replace device
- Low cost
  - Less than 1$
- Low latency
  - Connect and acknowledge data within 3 ms
  - Can send data without connection
- High data rate not a goal
  - Standard Bluetooth faster and more efficient for high data rates

Achieving Low Energy Consumption

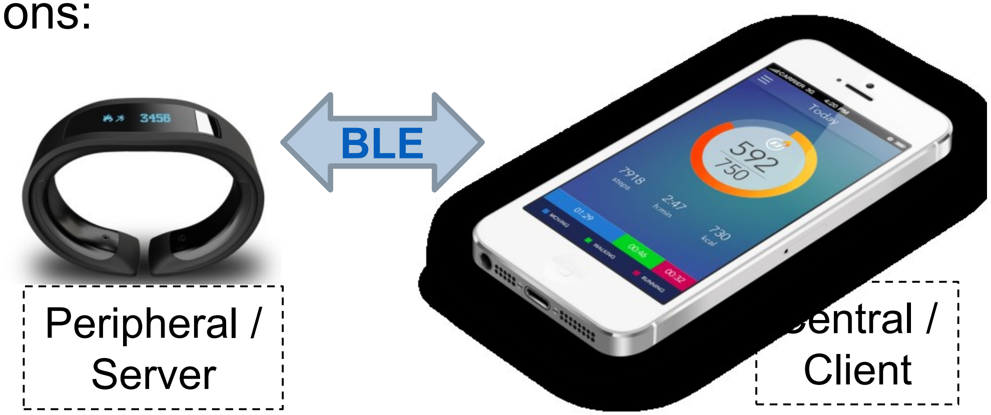
- **Minimize duty cycles**
  - µA in sleep mode vs. mA in active mode
  - Active only every 7.5 ms to 4 s (connection interval)

- **Fast connection setup**
  - Bluetooth uses frequency hopping on channels
  - BLE only uses 3 channels for advertising: radio on for only 1.2 ms
    - Standard Bluetooth uses 16 to 32 channels: radio on for 22.5 ms
  - Only 3 ms between connecting and acknowledgement of packet
    - Standard Bluetooth might take up to 100 ms for connection setup
  - BLE can also broadcast data without any connection setup

Device Roles

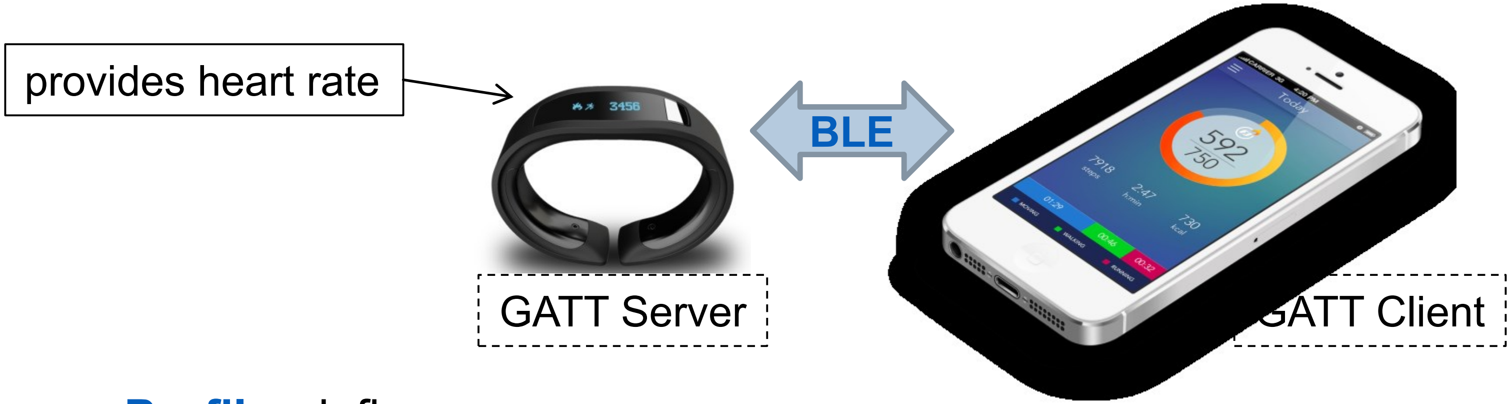
- Devices supporting connections:
  - **Peripheral**
    - Only one connection to one central
  - **Central**
    - Possibly multiple connections to different peripherals
    - Initiates connection to peripheral

- Devices not supporting connections:
  - **Broadcaster:** only sender
  - **Observer:** only receiver



Generic Attribute Profile (GATT): Profiles, Services, Characteristics (1)

- **Generic Attribute Profile (GATT):** Describes how GATT **servers** can provide small pieces of data to GATT **clients**



provides heart rate

BLE

GATT Server

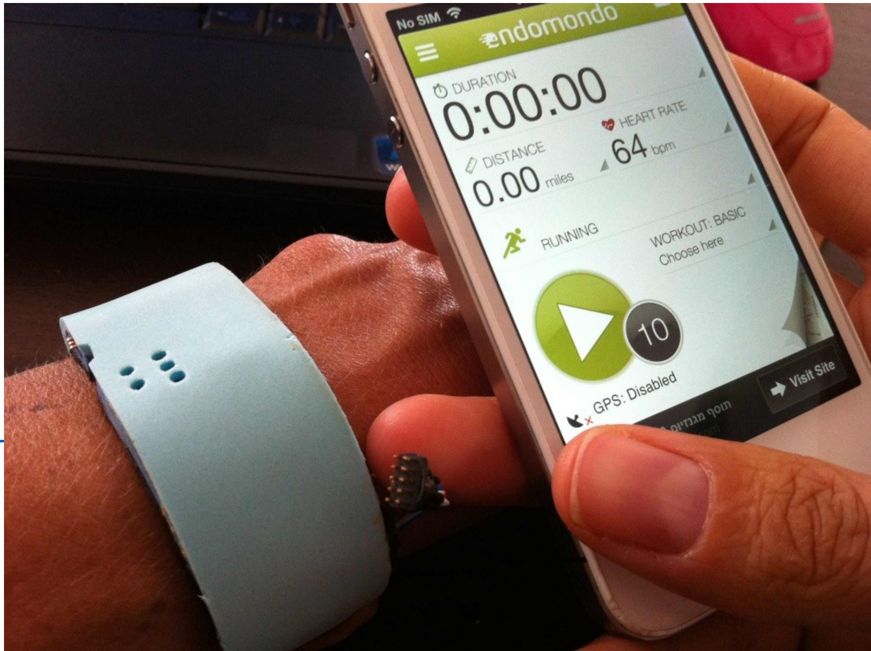
GATT Client

- **Profile:** defines a use case
  - Includes services to implement use case
  - Example: heart rate profile
    - “This profile enables a Collector device to connect and interact with a Heart Rate Sensor for use in fitness applications.” [https://developer.bluetooth.org]
    - Used services: org.bluetooth.service.heart_rate

Generic Attribute Profile (GATT): Profiles, Services, Characteristics (2)

- **Service:** Collection of data items (called characteristics) and behavior
  - Which characteristics are provided?
  - Which operations are supported on characteristics?
    - read, write, notify (see next slides)

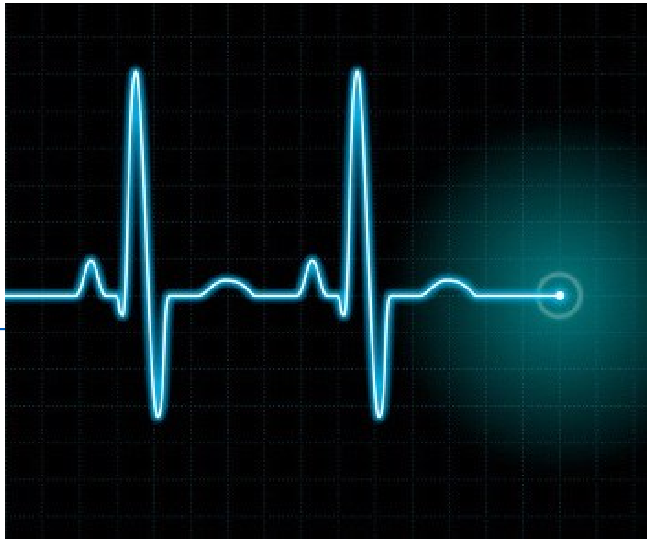
- **Example:** heart rate service (org.bluetooth.service.heart_rate)
  - Characteristic: heart rate measurement
    - Supported operation: indication
  - Characteristic: body sensor location
    - Supported operation: read



Generic Attribute Profile (GATT): Profiles, Services, Characteristics (3)

- **Characteristic:** Data item
  - Data structure declaring fields and defining data layout
  - Descriptors describing value

- **Example:** heart rate measurement (org.bluetooth.characteristic.heart_rate_measurement)
  - Flags (8 bits)
    - Bit 0: 0 = heart rate defined as uint8; 1 = heart rate defined as uint16
  - Heart rate measurement (uint8 or uint16)
  - etc.



Standard and Custom Services and Characteristics

- BLE defines sets of standard …
  - … profiles:
    - https://www.bluetooth.com/specifications/gatt/services
  - … services:
    - https://www.bluetooth.com/specifications/gatt/characteristics
  - … characteristics:
    - https://www.bluetooth.com/specifications/assigned-numbers/units
  - etc.
- Everyone can define custom profiles, services, characteristics
  - … you will use two custom services of IPVS ☺

Unique Identifiers

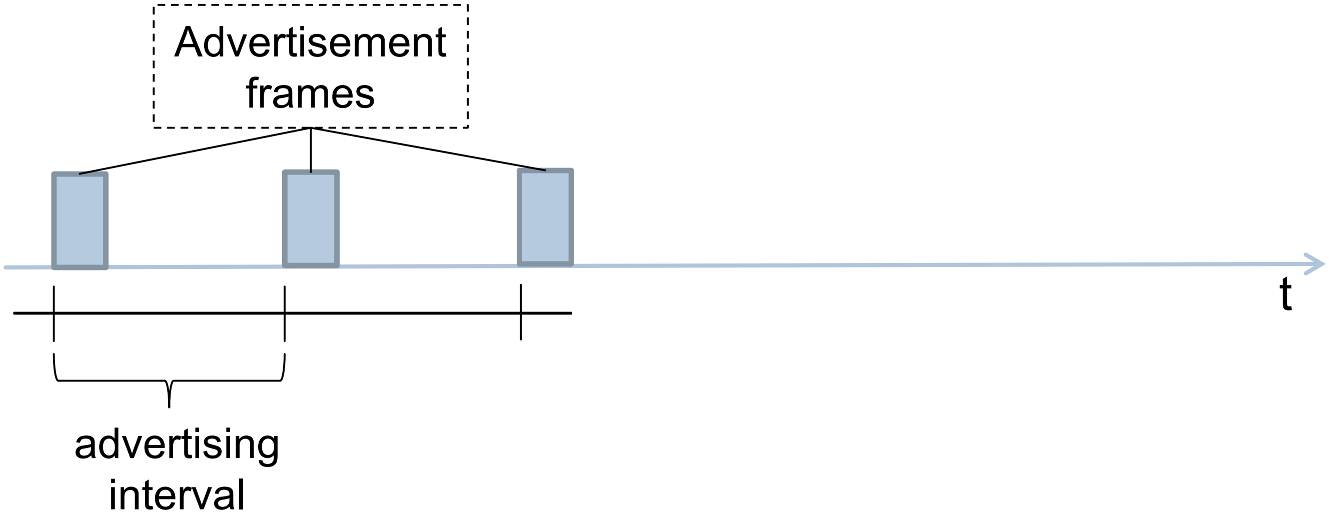
Services and characteristics are identified by **globally unique identifiers**

- **16 bit and 32 bit UUID for standard services and characteristics**
  - Mapped to 128 bit UUIDs:
    - BaseUUID = 00000000-0000-1000-8000-00805F9B34FB
    - $UUID_{128bit} = UUID_{16bit} * 2^{96} + BaseUUID$
      - 16 bit UUID blood pressure measurement: 0x2A35
      - 128 bit UUID: 00002A35-0000-1000-8000-00805F9B34FB
    - $UUID_{128bit} = UUID_{32bit} * 2^{96} + BaseUUID$
- **Must use 128 bit UUIDs for custom services and characteristics**
  - Created independently without coordination
    - Unix tool `uuidgen;` tons of generators on websites
  - Must use values outside reserved range!
    - Use your own base UUID different from standard base UUID

Advertisements

Peripherals periodically send **advertisements**

- Centrals can
  - discover peripherals in range,
  - discover services implemented by peripheral,
  - receive broadcast data without connection (e.g., iBeacon ID).
- Advertising intervals: 20 ms to 10 s
- Payload: up to 31 bytes
  - Peripheral name, UUIDs of implemented services, broadcast data



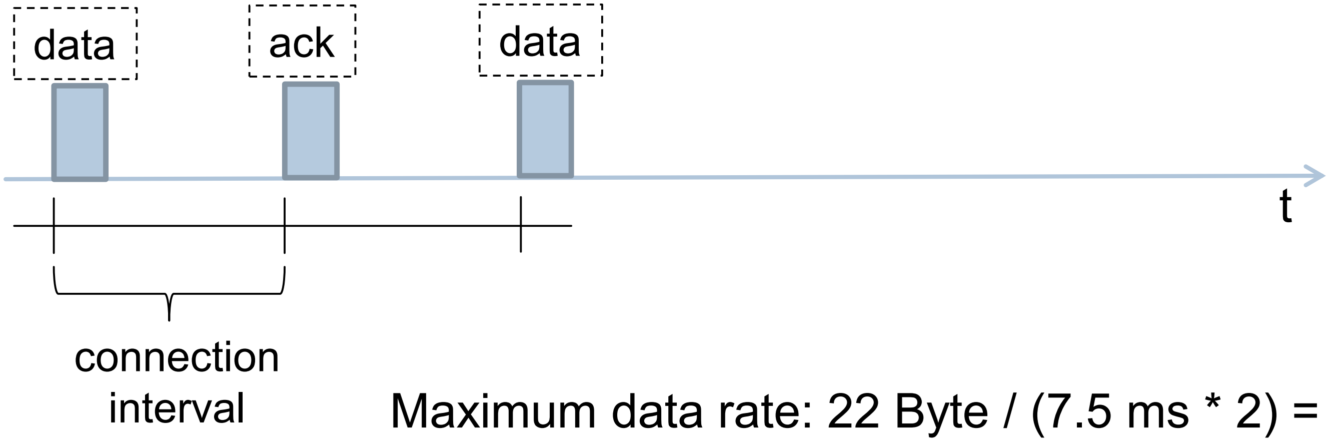
Transferring Characteristics Data between Client and Server (1)

Possible **operations on characteristics**:

- **read** data from server
  - Acknowledged
  - Payload size: 22 Byte
- **write** data to server
  - Acknowledged
  - Payload size: 20 Byte
- **write without response**
  - Unacknowledged
  - Payload size: 20 Byte
- **notification** (no ACK) and **indication** (ACK) from server to client
  - Payload size: 20 Byte

Transferring Characteristics Data between Client and Server (1)

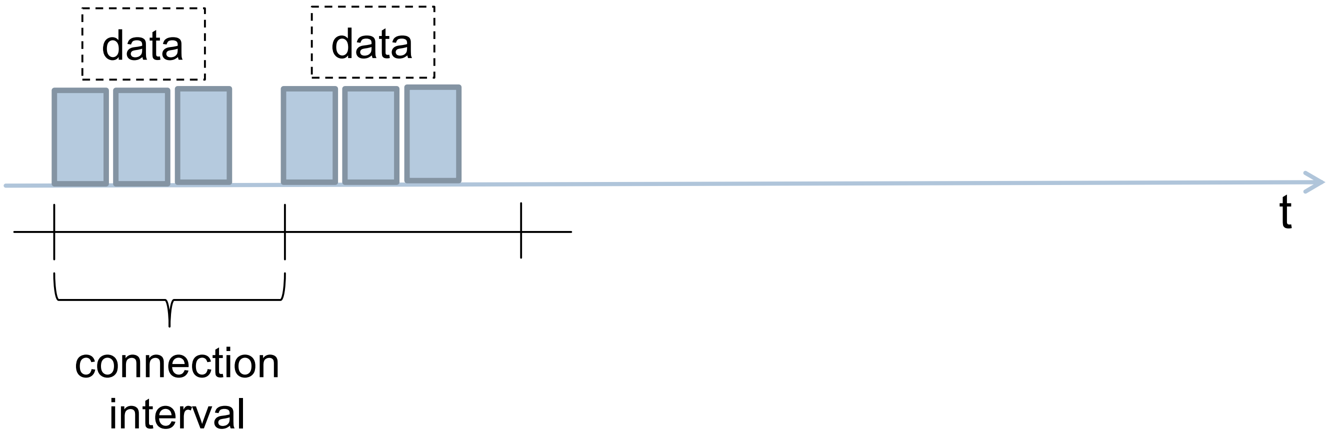
- Packets are sent in **connection intervals**
  - 7.5 ms – 4 s
  - Deep sleep (radio off) between intervals
- Acknowledged operations wait for ack before sending next packet
  - Only one packet (data or ack) per interval
  - Two intervals required for data & ack for one read or write operation



Maximum data rate: 22 Byte / (7.5 ms * 2) = 11.7 kbps

Transferring Data between Client and Server (3)

- Unacknowledged operations can send several packets in one interval
  - Number of packets depends on send buffer size of peripheral
    - Typically only few frames



- Maximum data rate assuming send buffer size of 8 packets:
  - 20 Byte * 8 / 7.5 ms = 170.67 kbps

Let‘s get practical: BLE in Android

BLE in Android – device discovery

- `BluetoothManager` class manages BLE
- Methods for scanning devices
  - `startLeScan(callback)` or `startLeScan(UUID [], callback)`
  - callback is instance of `LeScanCallback`
  - Second method to specify Array of UUIDS to scan for
- If device is found, the `onLeScan(..)` of `callback` instance will be called
  - RSSI is given as parameter
- Stop scan with `stopLeScan(callback)`

BLE in Android - GATT

- Connect to GATT server: call `device.connectGatt(..)`
  - `device` provided as parameter to `onLeScan(..)`
  - needs callback instance as parameter
  - returns `BluetoothGatt` instance
- Following methods can be implemented in callback instance (among others):
  - `onConnectionStateChange`: called on connect/disconnect
  - `onServicesDiscovered`: called when service, characteristics, descriptors have been updated
  - `onCharactersiticRead`: result of read operation
  - `onCharactersiticChanged`: used for notifications
- Requesting notifications using `setCharcteristicNotification(..)` on the `BluetoothGatt` instance

BLE in Android – New API

Scanning:

- `BluetoothManager`
- ~~`startLeScan()`~~ → `startScan()`
- ~~`stopLeScan()`~~ → `stopScan()`
- `LeScanCallback`

Connect to GATT Server

- `onConnectionStateChange()`
- `onServicesDiscovered()`
- `onCharacteristicRead()`
- `BluetoothGattCharacteristic`
- `readCharacteristic()`
- `writeCharacteristic()` …

Recommended Reading

- Android application fundamentals:
  https://developer.android.com/guide/components/fundamentals.html
- Location information in Android:
  https://developer.android.com/guide/topics/location/index.html
- User interfaces:
  https://developer.android.com/guide/topics/ui/index.html
- HelloWorld App example:
  https://developer.android.com/training/basics/firstapp/index.html
- BLE:
  https://developer.android.com/guide/topics/connectivity/bluetooth-le.html

Universität Stuttgart

Institute of Parallel and
Distributed Systems (IPVS)

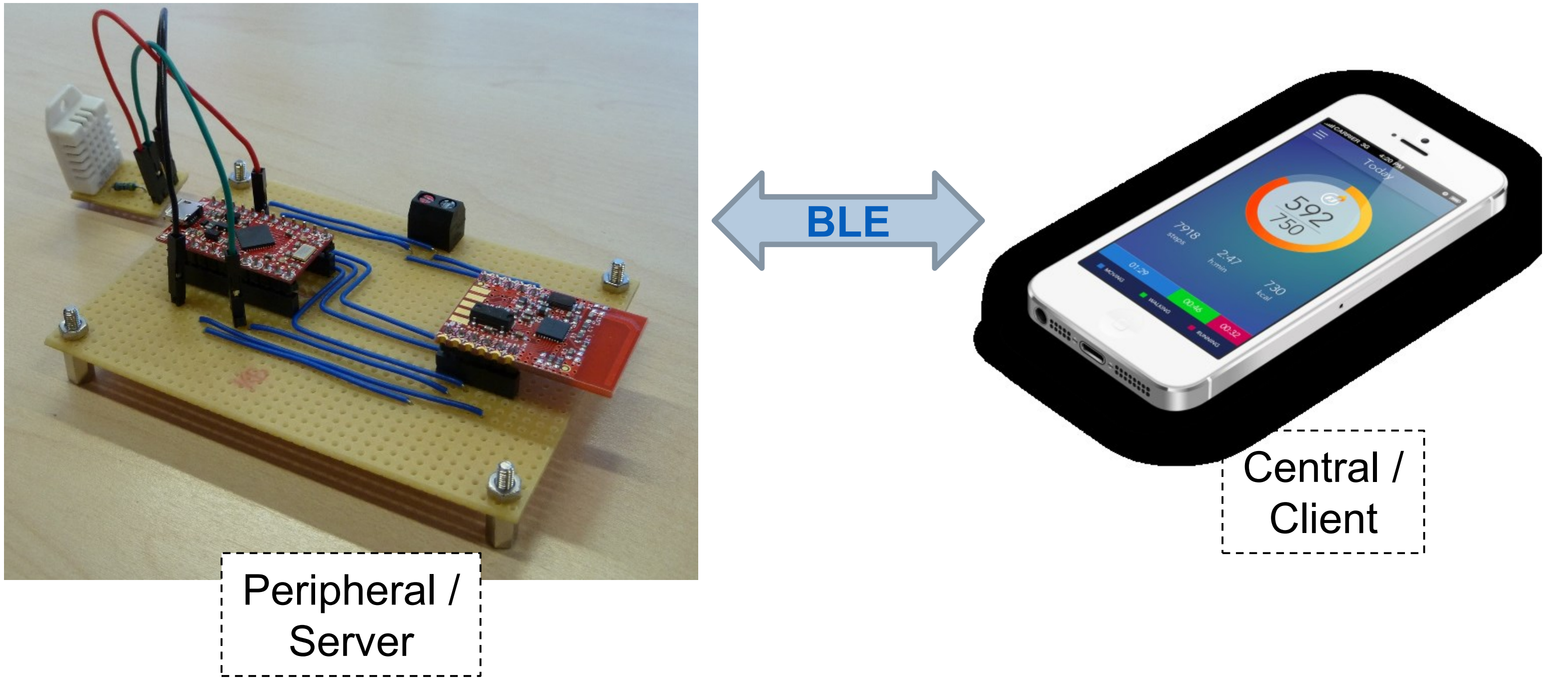
Universitätsstraße 38
D-70569 Stuttgart

Task 1
Android BLE App: Weather App

Task

Implement an **Android App for retrieving weather data from a BLE sensor**

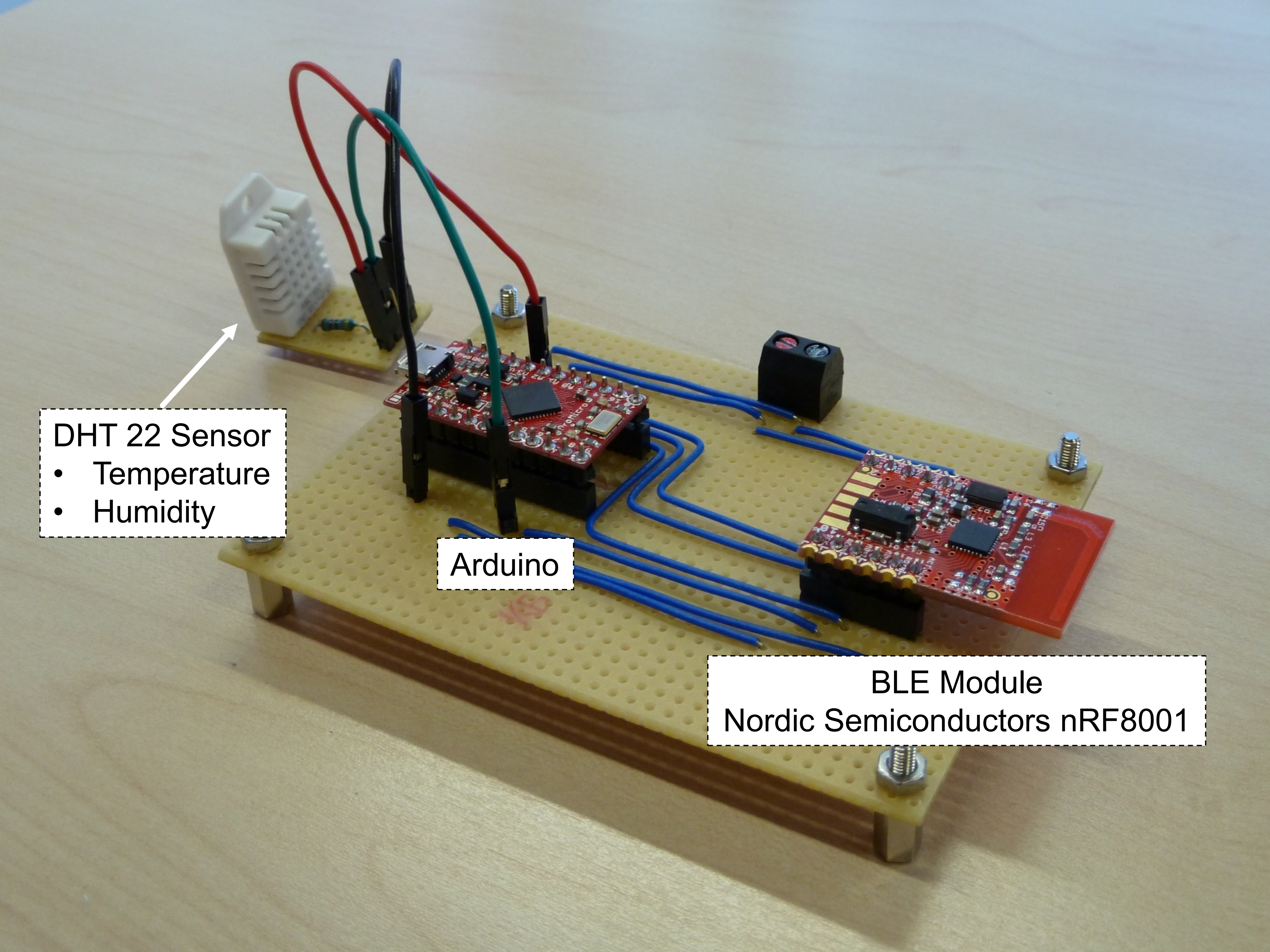
- Peripheral (sensor + Arduino + BLE radio) is provided by us
- You need to implement the **central** role on Android smartphone



BLE

Peripheral / Server

Central / Client



DHT 22 Sensor

- Temperature
- Humidity

Arduino

BLE Module
Nordic Semiconductors nRF8001

BLE Weather Service

- **Service UUID:** 00000002-0000-0000-FDFD-FDFDFDFDFDFD
- **Characteristics:**
  - Temperature Measurement
    - Standard BLE characteristic
    - https://www.bluetooth.com/specifications/gatt/viewer?attributeXmlFile=org.bluetooth.characteristic.temperature_measurement.xml
  - Humidity
    - Standard BLE characteristic
    - https://www.bluetooth.com/specifications/gatt/viewer?attributeXmlFile=org.bluetooth.characteristic.humidity.xml&u=org.bluetooth.characteristic.humidity.xml
- **Supported operations:** Both characteristics support read and notify
  - Your App should implement functions for querying (reading) and subscribing to notifications

Universität Stuttgart

Institute of Parallel and
Distributed Systems (IPVS)

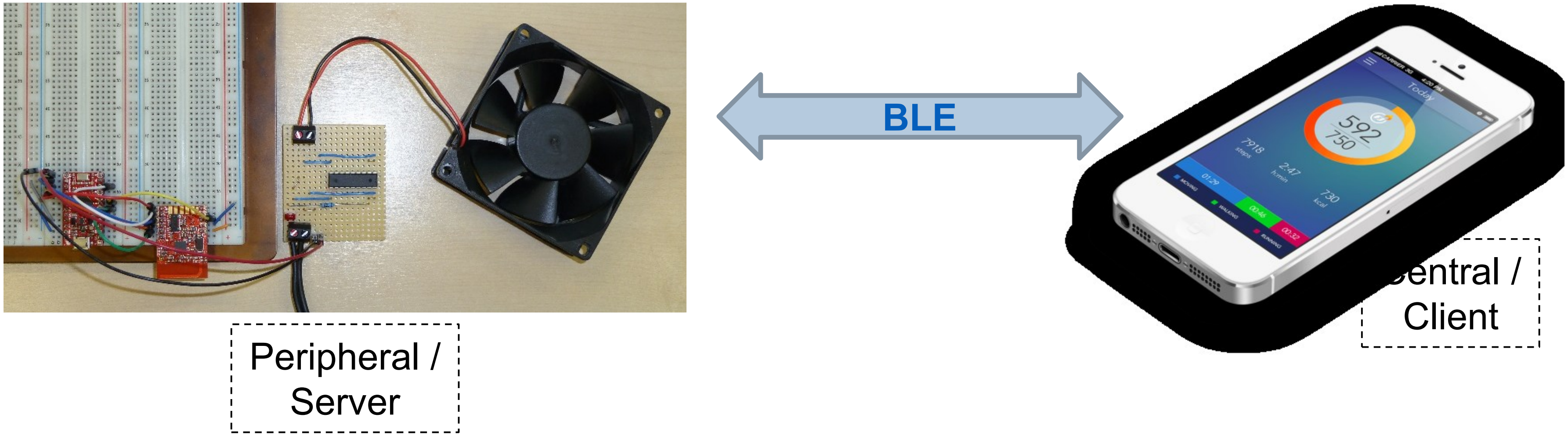
Universitätsstraße 38
D-70569 Stuttgart

Task 2:
Android BLE App: Fan Control App

Task

Implement an **Android App for controlling the speed of a fan / light-intensity of an LED**

- Peripheral (Fan/LED + Arduino + BLE radio) is provided by us
- You need to implement the **central** role on Android smartphone

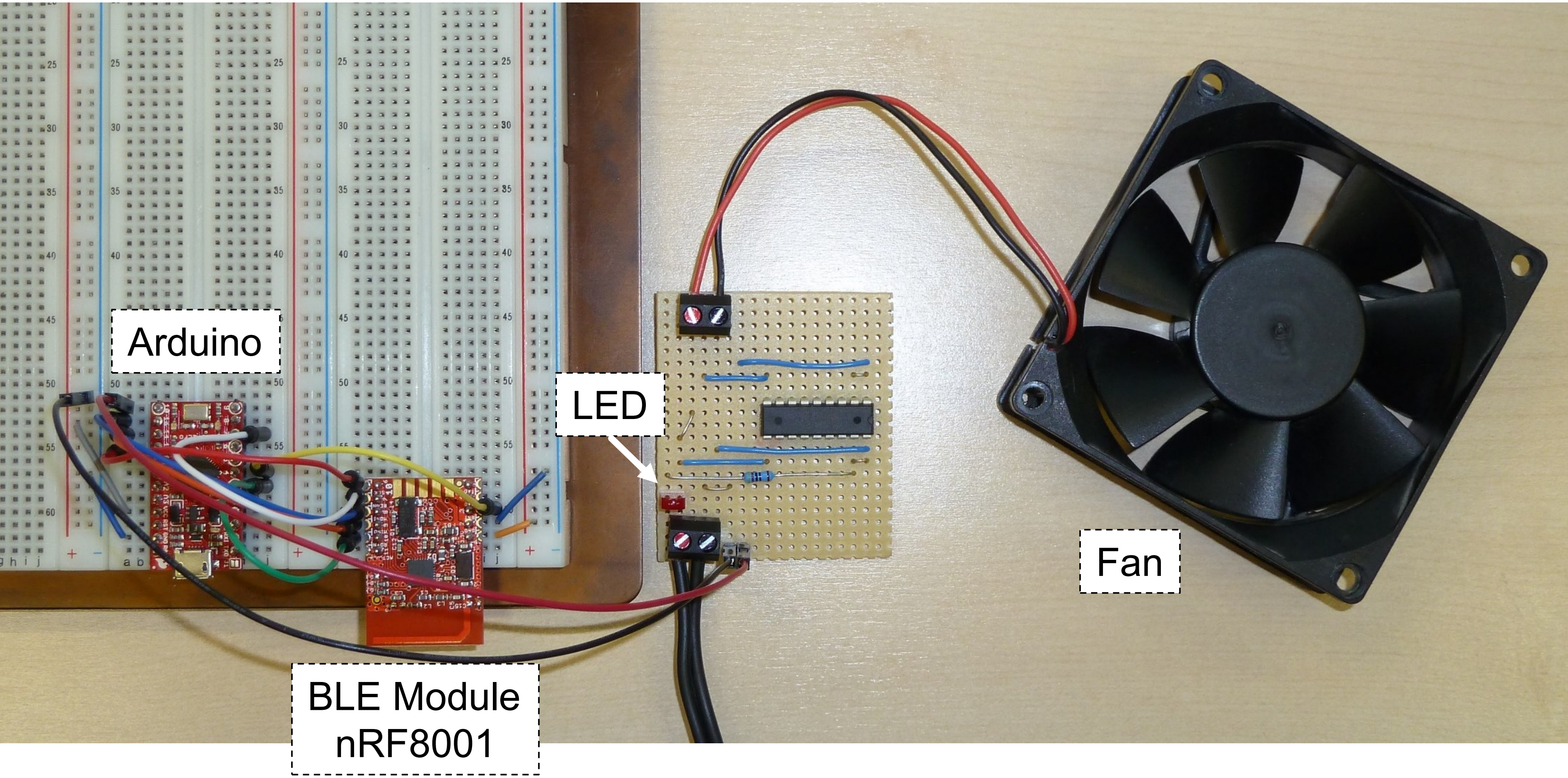


BLE

Peripheral / Server

Central / Client

Peripheral / Central



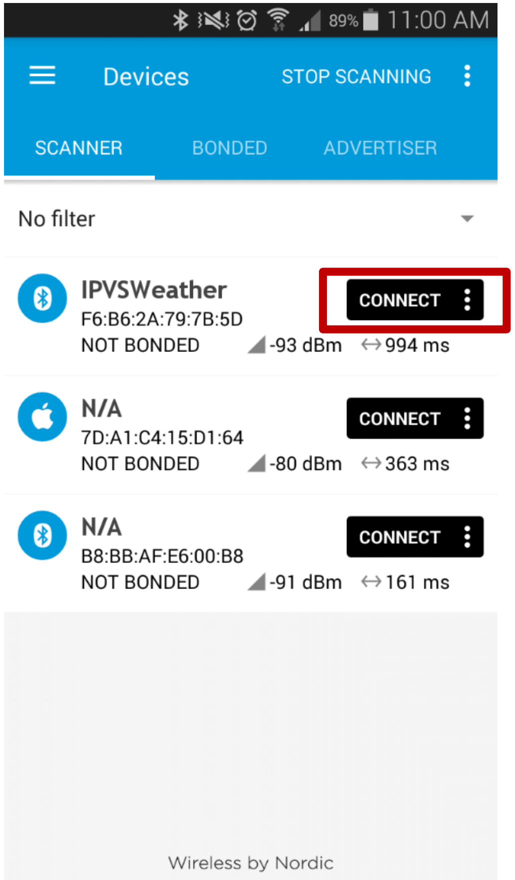
BLE Fan Control Service

- **Service UUID:** 00000001-0000-0000-FDFD-FDFDFDFDFDFD
- **Characteristics:**
  - Intensity
    - UUID: 10000001-0000-0000-FDFD-FDFDFDFDFDFD
    - Format: uint16 (0 min intensity, 65535 max intensity)
    - Exponent: 0
    - Unit: none
- **Supported operations:** Write

Some Help: BLE on Android App store

- The “nRF Connect for Mobile” app:

Step 1



11:00 AM 89%

Devices STOP SCANNING

SCANNER BONDED ADVERTISER

No filter

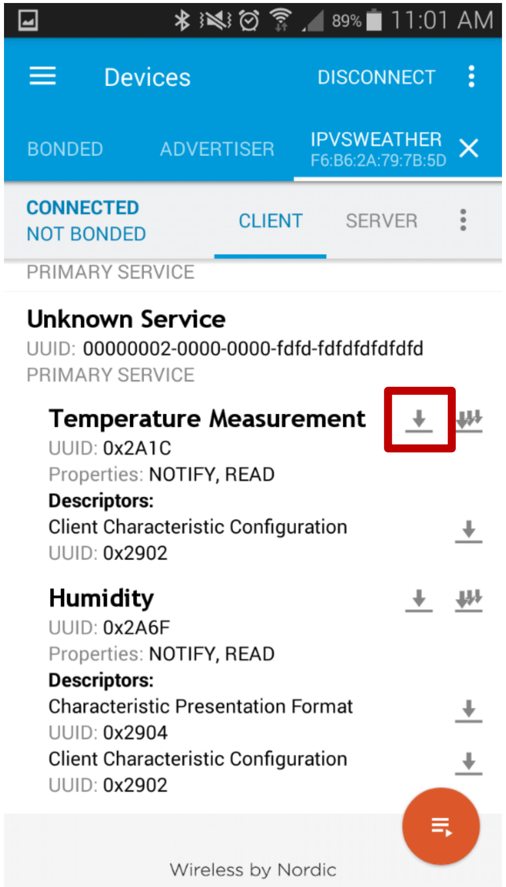
IPVSWeather
F6:B6:2A:79:7B:5D
NOT BONDED -93 dBm ↔994 ms
CONNECT

N/A
7D:A1:C4:15:D1:64
NOT BONDED -80 dBm ↔363 ms
CONNECT

N/A
B8:BB:AF:E6:00:B8
NOT BONDED -91 dBm ↔161 ms
CONNECT

Wireless by Nordic

Step 2



11:01 AM 89%

Devices DISCONNECT

BONDED ADVERTISER IPVSWEATHER F6:B6:2A:79:7B:5D

CONNECTED
NOT BONDED
CLIENT SERVER

PRIMARY SERVICE

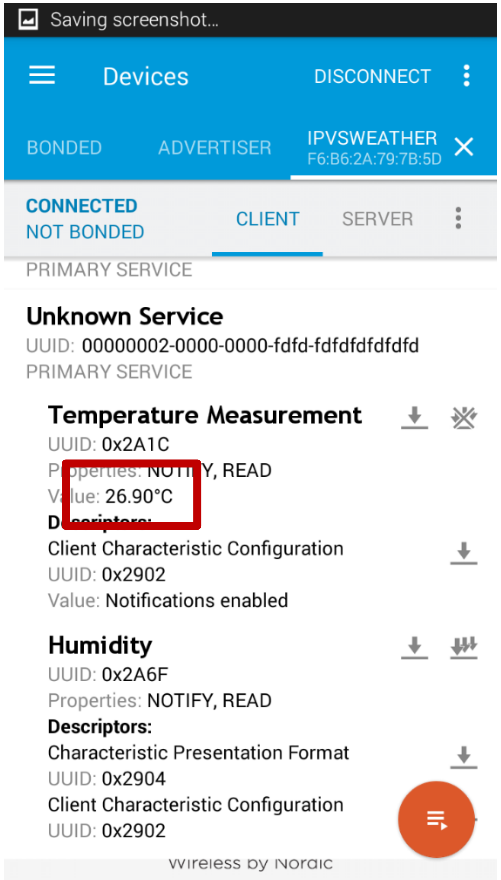
Unknown Service
UUID: 00000002-0000-0000-fdfd-fdfdfdfdfdfd
PRIMARY SERVICE

Temperature Measurement
UUID: 0x2A1C
Properties: NOTIFY, READ
Descriptors:
Client Characteristic Configuration
UUID: 0x2902

Humidity
UUID: 0x2A6F
Properties: NOTIFY, READ
Descriptors:
Characteristic Presentation Format
UUID: 0x2904
Client Characteristic Configuration
UUID: 0x2902

Wireless by Nordic

Step 3



Saving screenshot...

Devices DISCONNECT

BONDED ADVERTISER IPVSWEATHER F6:B6:2A:79:7B:5D

CONNECTED
NOT BONDED
CLIENT SERVER

PRIMARY SERVICE

Unknown Service
UUID: 00000002-0000-0000-fdfd-fdfdfdfdfdfd
PRIMARY SERVICE

Temperature Measurement
UUID: 0x2A1C
Properties: NOTIFY, READ
Value: 26.90°C
Descriptors:
Client Characteristic Configuration
UUID: 0x2902
Value: Notifications enabled

Humidity
UUID: 0x2A6F
Properties: NOTIFY, READ
Descriptors:
Characteristic Presentation Format
UUID: 0x2904
Client Characteristic Configuration
UUID: 0x2902

Wireless by Nordic

Submission & Next Meeting

- You have **2 weeks time** to work on this assignment until the final date of submission
  - Demonstration of your results scheduled on **Wednesday May 30st 2018**
  - Same place (room 0.153)
  - Time-slots will be uploaded appx. 2 days before your demonstrations

- If you have questions, post them on ILIAS

- **Submit via Ilias** at least the night before the demonstration meeting
  - **Source code** of you evaluation results
  - Group submission!

Questions?