1 Problem: BLE Connection

1. Use your IMU and proximity sensors on Arduino to read data every 1 second.
2. Set your Arduino as a BLE sever publishing the accelerometer and the proximity sensor data in two separate characteristics. (For acceleration only publish the measurement on the x-axis. So each characteristic would report one number per measurement)

```
#include <ArduinoBLE.h>
#include <Arduino_LSM9DS1.h>
#include <Arduino_APDS9960.h>

BLEService ServiceUUID("4fafc201-1fb5-459e-8fcc-c5c9c331914b");
BLEByteCharacteristic ProximityCharUuid("beb5483e-36e1-4688-b7f5-ea07361b26a8", BLERead | BLENotify | BLEWrite);
BLEFloatCharacteristic XAccCharUuid("2ea70455-850e-498b-86c0-bd0f89d76b86", BLERead | BLENotify | BLEWrite);

unsigned long initial_acc = 0;
unsigned long initial_prox = 0;
unsigned long previousMillis = 0;
float x, y, z;
int interval = 1000;

void setup() {
  Serial.begin(9600);
//  while (!Serial);

  if (!BLE.begin()) {
    Serial.println("* Starting BLE module failed!");
    while (1);
  }
  if (!IMU.begin()) {
    Serial.println("The IMU could not be initialized");
    while (1);
  }
  if (!APDS.begin()) {
    Serial.println("The proximity sensor could not be initialized");
    while (1);
  }

  BLE.setLocalName("Nano 33 BLE (Central) Carl");
  BLE.setAdvertisedService(ServiceUUID);

  ServiceUUID.addCharacteristic(ProximityCharUuid);
  ServiceUUID.addCharacteristic(XAccCharUuid);

  BLE.addService(ServiceUUID);
  ProximityCharUuid.writeValue(initial_prox);
  XAccCharUuid.writeValue(initial_acc);

  BLE.advertise();

  Serial.println("Arduino Nano 33 BLE Sense (Central Device)");
}

void getdata(){

  if (IMU.accelerationAvailable() & APDS.proximityAvailable()) {
      IMU.readAcceleration(x,y,z);
      XAccCharUuid.writeValue(x);
      Serial.print("Acceleration:");
      Serial.print(x);
      int proximity = APDS.readProximity();
      ProximityCharUuid.writeValue(proximity);
      Serial.println("Proximity:");
      Serial.print(proximity);
  }
}

void loop() {
    BLEDevice central = BLE.central();
    if (central){
      Serial.print("Connected to central");
```

```
65        Serial.print(central.address());
66        digitalWrite(LED_BUILTIN, HIGH);
67        while (central.connected())
68        {
69          unsigned long currentMillis = millis();
70          if (currentMillis - previousMillis >= interval)
71          {
72            getdata();
73            previousMillis = currentMillis;
74          }
75        }
76        digitalWrite(LED_BUILTIN, LOW);
77        Serial.print("diconnected from central");
78  }
79  }
```

3. Set your ESP32 as a BLE client and make sure to receive both characteristics

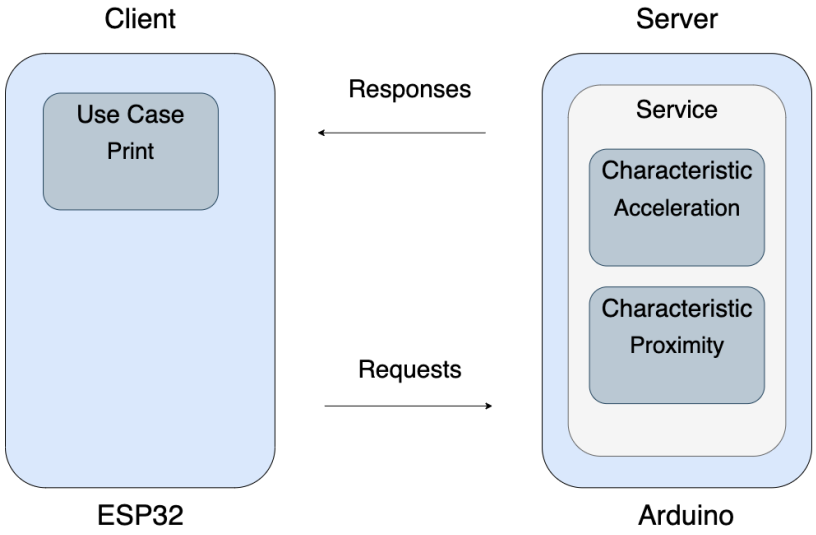
```
1  #include "BLEDevice.h"
2  #include "BLEScan.h"
3
4  float AccelerationX = 0;
5  int Proximity = 0;
6
7  boolean regen_acc = false;
8  boolean regen_prox = false;
9
10 static BLEUUID serviceUUID("4fafc201-1fb5-459e-8fcc-c5c9c331914b");
11 static BLEUUID CharProximityUUID("beb5483e-36e1-4688-b7f5-ea07361b26a8");
12 static BLEUUID CharAccelerationUUID("2ea70455-850e-498b-86c0-bd0f89d76b86");
13
14 static boolean doConnect = false;
15 static boolean connected = false;
16 static boolean doScan = false;
17 static BLERemoteCharacteristic* aRemoteCharacteristic;
18 static BLERemoteCharacteristic* pRemoteCharacteristic;
19 static BLEAdvertisedDevice* myDevice;
20
21 static void notifyCallbackp( BLERemoteCharacteristic* pBLERemoteCharacteristic, uint8_t* ↩
       pDatap, size_t length, bool isNotify) {
22     Proximity = *pDatap;
23     regen_prox = true;
24 }
25
26 static void notifyCallbacka( BLERemoteCharacteristic* aBLERemoteCharacteristic, uint8_t* ↩
       pDataa, size_t length, bool isNotify) {
27     AccelerationX  = *(float*)pDataa;
28     regen_acc = true;
29 }
30
31 class MyClientCallback : public BLEClientCallbacks {
32   void onConnect(BLEClient* pclient) {
33   }
34
35   void onDisconnect(BLEClient* pclient) {
36     connected = false;
37     Serial.println("onDisconnect");
38   }
39 };
40
41 bool connectToServer() {
42     Serial.print("Forming a connection to ");
43     Serial.println(myDevice->getAddress().toString().c_str());
44
45     BLEClient*  pClient  = BLEDevice::createClient();
46     Serial.println(" - Created client");
47
48     pClient->setClientCallbacks(new MyClientCallback());
49
50     pClient->connect(myDevice);
51     Serial.println(" - Connected to server");
52
53     BLERemoteService* pRemoteService = pClient->getService(serviceUUID);
54     Serial.print("passed");
```

```
55      if (pRemoteService == nullptr) {
56        Serial.print("Failed to find our service UUID: ");
57        Serial.println(serviceUUID.toString().c_str());
58        pClient->disconnect();
59        return false;
60      }
61      Serial.println(" - Found our service");
62
63      aRemoteCharacteristic = pRemoteService->getCharacteristic(CharAccelerationUUID);
64      if (aRemoteCharacteristic == nullptr) {
65        Serial.print("Failed to find our characteristic UUID: ");
66        Serial.println(CharAccelerationUUID.toString().c_str());
67        pClient->disconnect();
68        return false;
69      }
70      Serial.println(" - Found our acceleration characteristic");
71
72      if(aRemoteCharacteristic->canRead()) {
73        std::string value = aRemoteCharacteristic->readValue();
74        Serial.print("The acceleration characteristic value was: ");
75        Serial.println(value.c_str());
76      }
77
78      pRemoteCharacteristic = pRemoteService->getCharacteristic(CharProximityUUID);
79      if (pRemoteCharacteristic == nullptr) {
80        Serial.print("Failed to find our characteristic UUID: ");
81        Serial.println(CharProximityUUID.toString().c_str());
82        pClient->disconnect();
83        return false;
84      }
85      Serial.println(" - Found our proximity characteristic");
86
87      if(pRemoteCharacteristic->canRead()) {
88        std::string value = pRemoteCharacteristic->readValue();
89        Serial.print("The proximity characteristic value was: ");
90        Serial.println(value.c_str());
91      }
92
93      if(aRemoteCharacteristic->canNotify())
94        aRemoteCharacteristic->registerForNotify(notifyCallbacka);
95      if(pRemoteCharacteristic->canNotify())
96        pRemoteCharacteristic->registerForNotify(notifyCallbackp);
97
98      connected = true;
99
100 }
101
102 class MyAdvertisedDeviceCallbacks: public BLEAdvertisedDeviceCallbacks {
103
104   void onResult(BLEAdvertisedDevice advertisedDevice) {
105     Serial.print("BLE Advertised Device found: ");
106     Serial.println(advertisedDevice.toString().c_str());
107
108     if (advertisedDevice.haveServiceUUID() && advertisedDevice.isAdvertisingService(↩
          serviceUUID)) {
109
110       BLEDevice::getScan()->stop();
111       myDevice = new BLEAdvertisedDevice(advertisedDevice);
112       doConnect = true;
113       doScan = true;
114
115     }
116   }
117 };
118
119
120 void setup() {
121   Serial.begin(115200);
122   while(!Serial);
123   Serial.println("Starting Arduino BLE Client application...");
124   BLEDevice::init("");
125
126   BLEScan* pBLEScan = BLEDevice::getScan();
127   pBLEScan->setAdvertisedDeviceCallbacks(new MyAdvertisedDeviceCallbacks());
128   pBLEScan->setInterval(1349);
129   pBLEScan->setWindow(449);
```

```
130   pBLEScan->setActiveScan(true);
131   pBLEScan->start(300, false);
132 }
133
134 void loop() {
135
136   if (doConnect == true) {
137     if (connectToServer())
138     { Serial.println("We are now connected to the BLE Server.");
139
140     } else {
141       Serial.println("We have failed to connect to the server; there is nothin more we will ↩
              do.");
142     }
143       doConnect = false;
144   }
145
146   if (regen_acc && regen_prox) {
147     Serial.print("Acceleration:");
148     Serial.println(AccelerationX);
149     Serial.print("Proximity:");
150     Serial.println(Proximity);
151     regen_acc = false;
152     regen_prox = false;
153   }
154
155 }
```

4. Draw a diagram of your system and explain, in less than 4 sentences, how your BLE connection works.

   The ESP32 acting as a peripheral(client), will broadcast signals about itself to the surrounding area. The Arduino picks up on this broadcast and establishes a connection which allows for reading to occur. During reading, the peripheral (ESP32) asks the central device (Arduino) for information like a specific characteristic of a service. As the server, the Arduino acts as the master and the client acts as the slave.



2 Problem

1. The data that you received from your Arduino is now sent to your cloud. Use two MQTT topics to publish your sensor data (one for each of your sensors)

```
#include "BLEDevice.h"
#include "BLEScan.h"
#include <UbiConstants.h>
#include <UbiTypes.h>
#include <UbidotsEsp32Mqtt.h>
#include <WiFi.h>

const char *UBIDOTS_TOKEN = "BBFF-uPnUOyq1G4EwIXVOnQqEmmQxxKyH67";
const char *WIFI_SSID = "iPhone";
const char *WIFI_PASS = "sasa8888";
const char *DEVICE_LABEL = "ESP32";
const char *ubiaccpoint = "AccelerationX";
const char *ubiproxpoint = "Proximity";

const int PUBLISH_FREQUENCY = 5000;
unsigned long previousMillis = 0;

float AccelerationX = 0;
int Proximity = 0;

boolean regen_acc = false;
boolean regen_prox = false;

static BLEUUID serviceUUID("4fafc201-1fb5-459e-8fcc-c5c9c331914b");
static BLEUUID CharProximityUUID("beb5483e-36e1-4688-b7f5-ea07361b26a8");
static BLEUUID CharAccelerationUUID("2ea70455-850e-498b-86c0-bd0f89d76b86");

static boolean doConnect = false;
static boolean connected = false;
static boolean doScan = false;
static BLERemoteCharacteristic* aRemoteCharacteristic;
static BLERemoteCharacteristic* pRemoteCharacteristic;
static BLEAdvertisedDevice* myDevice;
Ubidots ubidots(UBIDOTS_TOKEN);

static void notifyCallbackp( BLERemoteCharacteristic* pBLERemoteCharacteristic, uint8_t* pDatap, size_t length, bool isNotify) {
    Proximity = *pDatap;
    regen_prox = true;
}

static void notifyCallbacka( BLERemoteCharacteristic* aBLERemoteCharacteristic, uint8_t* pDataa, size_t length, bool isNotify) {
    AccelerationX  = *(float*)pDataa;
    regen_acc = true;
}

class MyClientCallback : public BLEClientCallbacks {
  void onConnect(BLEClient* pclient) {
  }

  void onDisconnect(BLEClient* pclient) {
    connected = false;
    Serial.println("onDisconnect");
  }
};

bool connectToServer() {
    Serial.print("Forming a connection to ");
    Serial.println(myDevice->getAddress().toString().c_str());

    BLEClient*  pClient  = BLEDevice::createClient();
    Serial.println(" - Created client");

    pClient->setClientCallbacks(new MyClientCallback());

    pClient->connect(myDevice);
    Serial.println(" - Connected to server");

    BLERemoteService* pRemoteService = pClient->getService(serviceUUID);
```

```
69      Serial.print("passed");
70      if (pRemoteService == nullptr) {
71        Serial.print("Failed to find our service UUID: ");
72        Serial.println(serviceUUID.toString().c_str());
73        pClient->disconnect();
74        return false;
75      }
76      Serial.println(" - Found our service");
77
78      aRemoteCharacteristic = pRemoteService->getCharacteristic(CharAccelerationUUID);
79      if (aRemoteCharacteristic == nullptr) {
80        Serial.print("Failed to find our characteristic UUID: ");
81        Serial.println(CharAccelerationUUID.toString().c_str());
82        pClient->disconnect();
83        return false;
84      }
85      Serial.println(" - Found our acceleration characteristic");
86
87      if(aRemoteCharacteristic->canRead()) {
88        std::string value = aRemoteCharacteristic->readValue();
89        Serial.print("The acceleration characteristic value was: ");
90        Serial.println(value.c_str());
91      }
92
93      pRemoteCharacteristic = pRemoteService->getCharacteristic(CharProximityUUID);
94      if (pRemoteCharacteristic == nullptr) {
95        Serial.print("Failed to find our characteristic UUID: ");
96        Serial.println(CharProximityUUID.toString().c_str());
97        pClient->disconnect();
98        return false;
99      }
100     Serial.println(" - Found our proximity characteristic");
101
102     if(pRemoteCharacteristic->canRead()) {
103       std::string value = pRemoteCharacteristic->readValue();
104       Serial.print("The proximity characteristic value was: ");
105       Serial.println(value.c_str());
106     }
107
108     if(aRemoteCharacteristic->canNotify())
109       aRemoteCharacteristic->registerForNotify(notifyCallbacka);
110     if(pRemoteCharacteristic->canNotify())
111       pRemoteCharacteristic->registerForNotify(notifyCallbackp);
112
113     connected = true;
114 }
115
116 class MyAdvertisedDeviceCallbacks: public BLEAdvertisedDeviceCallbacks {
117
118   void onResult(BLEAdvertisedDevice advertisedDevice) {
119     Serial.print("BLE Advertised Device found: ");
120     Serial.println(advertisedDevice.toString().c_str());
121
122     if (advertisedDevice.haveServiceUUID() && advertisedDevice.isAdvertisingService(↩
          serviceUUID)) {
123
124       BLEDevice::getScan()->stop();
125       myDevice = new BLEAdvertisedDevice(advertisedDevice);
126       doConnect = true;
127       doScan = true;
128
129     }
130   }
131 };
132
133
134 void setup() {
135   Serial.begin(115200);
136   while(!Serial);
137   Serial.println("Starting Arduino BLE Client application...");
138   BLEDevice::init("");
139   ubidots.setDebug(true);
140   ubidots.connectToWifi(WIFI_SSID, WIFI_PASS);
141
142   ubidots.setup();
143   ubidots.reconnect();
```

```
144
145   BLEScan* pBLEScan = BLEDevice::getScan();
146   pBLEScan->setAdvertisedDeviceCallbacks(new MyAdvertisedDeviceCallbacks());
147   pBLEScan->setInterval(1349);
148   pBLEScan->setWindow(449);
149   pBLEScan->setActiveScan(true);
150   pBLEScan->start(300, false);
151 }
152
153
154 void loop() {
155
156   if (doConnect == true) {
157     if (connectToServer()) {
158       Serial.println("We are now connected to the BLE Server.");
159     }
160     else {
161       Serial.println("We have failed to connect to the server; there is nothin more we will ↩
              do.");
162     }
163       doConnect = false;
164   }
165   if (!ubidots.connected()){
166     ubidots.reconnect();
167     }
168
169   if (regen_acc && regen_prox) {
170     Serial.print("Acceleration:");
171     Serial.println(AccelerationX);
172     Serial.print("Proximity:");
173     Serial.println(Proximity);
174     regen_acc = false;
175     regen_prox = false;
176   }
177
178   unsigned long currentMillis = millis();
179   if (currentMillis - previousMillis >= PUBLISH_FREQUENCY) {
180     ubidots.add(ubiaccpoint, AccelerationX);
181     ubidots.add(ubiproxpoint, Proximity);
182     previousMillis = currentMillis;
183     ubidots.publish(DEVICE_LABEL);
184   }
185 ubidots.loop();
186
187 }
```

2. Using the Ubidots platform now you can check your sensor data in real-time.

3. Complete your system diagram from problem 1. Add all your MQTT components and explain the overall system in less than 6 sentences.

   Ubidots acts as a MQTT broker, that means that the ESP32 publishes the acceleration float and the proximity string to the broker using the MQTT publish-and-subscribe protocol. This protocol works by first connecting to the broker and then publishing all using the existing local internet home network. Once Ubidots has received the data, the data from the bluetooth characteristic is stored online to a variable. The data can be then accessed and read using a smartphone or computer using the Wi-Fi protocol. Using radio waves Wi-Fi connects the device to the nearest router, which sends the request for the data to the ubidots.com host.