

Engineering an IoT-Edge-Cloud Computing System Architecture: Lessons Learnt from An Undergraduate Lab Course

Jasenka Dizdarević and Admela Jukan
 Technische Universität Braunschweig, Germany
 {j.dizdarevic, a.jukan}@tu-bs.de

Abstract—With the rapid advances in IoT, edge and cloud computing solutions, it is critical to educate and train students in computer science and engineering in various aspects of IoT-edge-cloud (IoT-E-C) system architecture implementations. We outline the design and development of an undergraduate laboratory course that sets the goal of implementing various interfaces and communication protocols to connect IoT, edge and cloud computing systems and evaluating their performance. The lab setup is modular and based on open source tools. In the IoT context, it consists of low-cost processing platforms with various sensors and actuators. In the edge and cloud computing context, we implement and deploy single board computers and Firebase cloud solutions, respectively. The modular lab setup allows students to engineer and integrate various communication protocol solutions, including MQTT, COAP and HTTP. In addition to the system implementation, students can evaluate and benchmark the performance of the entire system.

Index Terms—Edge and cloud computing, communication protocols, IoT

I. INTRODUCTION

The unstoppable trend towards the combined IoT, edge and cloud computing systems has lead to an increased demand for educated workforce in corresponding areas of computer science and engineering. This has furthermore driven efforts in creating computer science and computer engineering courses that can implement and integrate IoT devices in cloud and edge computing systems [1]–[4]. Currently available courses reflect the ongoing quests towards the specific focus of the course, such as testing different hardware boards, or developing domain applications with IoT devices, such as in healthcare applications [5]. Today, in addition to a number of cloud and edge based IoT commercial platforms, numerous open-source solutions can be used to further broaden the participation in such courses [6]. Since the technology choice is both broad and diverse, and the technology trends and the contents covered are constantly evolving, scoping such courses towards specific goals is rather critical.

We propose and outline the design and development of an undergraduate laboratory course, referred to as Network-of-Things Engineering Lab (NoteLab), - that sets the goal of implementing various interfaces and communication protocols to connect IoT, edge and cloud computing systems and evaluating their performance. Unlike other courses that usually cover individual and separate areas of either IoT, or edge, or cloud computing subsystems, our course offers the implementation

of the entire system, and the subsequent evaluation and benchmarking of the end-to-end system performance. The lab setup is highly-modular and based on open source tools. It includes three contexts: IoT, edge and cloud, as a model for separation of concerns implying that any device can be deployed in a specific context. This is in contrast to a commonly used computing hierarchical architecture, with cloud computing at the top of hierarchy [7]. The modular lab setup allows students to engineer and integrate various communication protocol and interface solutions, including MQTT, COAP and HTTP. We define the following main learning objectives in the course:

- O1. Engineering the interfaces and the related communication protocols in an IoT embedded system setup, as an integral part of the integrated IoT-edge-cloud computing architecture;
- O2. Integrating a diverse set of the specific hardware solutions in IoT and edge computing context (e.g., sensors, actuators, single-board-computers, and microcontroller platforms);
- O3. Applying open-source software solutions for resource-constrained embedded devices (including Arduino IDE sketches, ARM compatible docker images, mqtt proxy, coap proxy and http proxy python scripts);
- O4. Executing and testing sample sensing and actuation applications (e.g., temperature, humidity and motion sensing);
- O5. Measuring and benchmarking system performance (e.g., latency and power consumption).

The rest of the paper is organized as follows. Section II presents the related work. Section III describes the NoteLab architecture design and the learning scope. Section IV presents the details behind the individual task units in the lab, along with a discussion on lessons learnt. Section V provides conclusion and outlook.

II. RELATED WORK AND OUR CONTRIBUTION

The driving force behind the innovation and implementation of IoT system solutions in recent years, which has also led to an increase in the related electrical and computer engineering courses, is the availability of low-cost, yet highly performant IoT devices [1]. Many hands-on courses developed to date focus on IoT laboratories [8]. Paper [2] presents an extensive IoT courses survey defining different categories, ranging from

introductory IoT courses, more advanced IoT certification multi-courses, over to most common approach of studying and developing domain-specific IoT applications, such as in agriculture, transportation or healthcare. Paper [9] examines different hardware platforms for IoT-centric courses, and emphasizes the challenges that the course instructors need to pay attention to when choosing specific hardware and software solutions. Papers [10] and [11] describe various approaches to teaching new IoT concepts through elective courses.

Paper [4] presents a newly developed embedded system IoT course, explaining to students and their instructors specific aspects of data processing in IoT systems. As the data processing has shifted more towards cloud computing context, major efforts in computer engineering also shifted focus to only collecting data with IoT devices, and their processing in the cloud [12]. More recently, data processing is done in edge and fog computing [13], which has also brought a notable interest in combined IoT-edge-cloud solutions, both in industry and academia [14], [15]. Paper [16] presents a system oriented graduate course, covering the concepts of edge and cloud computing in integrated IoT systems. This more system oriented direction has successfully integrated aspects of networking and communication protocols, even in undergraduate IoT lab courses, such as in [3].

In terms of our specific novel contributions, we greatly benefitted from previous work, especially in our initial choices of hardware platforms, such as from [4], [9]. We adopted similar goals set in some courses, such as to creating an industrially relevant IoT system in [16]. We took a different approach, however, and worked with open source and non-proprietary hardware and software, with the goal of enabling easy reproducibility of the course, albeit possibly of a lesser industrial interest. Also notable is related work [3], addressing not only the integrated IoT-edge-cloud computing but also introducing different IoT protocol solutions. This course however uses a different methodology and software systems. In our approach, we do not use stand-alone lab units, but design all units as inter-related and building upon each other, where students can gain significant problem-solving skills as they are trying to plan and connect individual tasks. In addition, we use containerization, including Docker, but also as applied to Kafka, Firebase and Mosquitto, which provides additional training in software engineering. Our paper does not include a detailed taxonomy based evaluation of student learning outcomes [17], as the focus has been mainly on technical aspects of the course implementation. In future work we plan an extension that will include this kind of evaluation, as it will help improve our learning framework and outcomes.

III. NOTELAB ARCHITECTURE DESIGN

A. Reference architecture

We start with the envisioned system architecture shown in Fig. 1, representing both the high-level architectural concept as well the implemented system solution. We divide the architecture into various *context*: IoT-, edge computing- and cloud computing context. In our approach, we found it useful

to refer to a *context* as a separation of concerns, as opposed to *layers* that often imply hierarchy. In other words, there is no assumption on any specific hierarchy in the architecture, and computing and data processing can happen in all three context, and on any device put in context.

In the *IoT context*, we envision resource constrained devices and low-cost processing platforms, including various sensors and actuators. These sensors and actuators are the principal generators of data in the architecture and will have a client role (Fig. 1a). The way how these clients are going to send the data to other devices depends on the implementation, assuming that both the communication protocols that are based on publish/subscribe and client/server (also known as request/reply) interaction model are potential choices.

In the *edge computing context*, we define two types of devices, one with the role of connecting the edge with the cloud database service (edge-cloud connector in Fig. 1a), for which in the implementation we will use an already developed open source solution; and the other with the role of a proxy, which will be developed by the students with the intent of forwarding data from the devices in IoT context to the connector. In order to establish communication with IoT devices, the proxy solution is going to include a server or a broker implementation (depending on the interaction model of the communication protocol) as its southbound interface. It will also include a northbound interface for forwarding data to the edge-cloud connector. For the latter communication, as it is based on an open-source pre-developed solution, a publish/subscribe based protocol is used.

The third context is the *cloud computing context*, where we envisioned a simple cloud database service for client's data storage and synchronization purposes. The communication between the database and the edge-cloud connector will be maintained through proprietary or open source protocols, depending on the implementation choice of the cloud service.

The architectural representation containing implementation details is shown in Fig. 1b, illustrating various devices put in IoT, edge or cloud context. Here, for the IoT context implementation, we use six microcontroller based devices, with the corresponding sensors and actuators, including temperature, humidity, and motion sensors as well as LED actuators. Localized in the edge context, we differentiate between two types of devices: single board computers (Raspberry Pi, RPi) and off-the-shelf desktop computers.

As a part of the edge context implementation Fig. 1b illustrates three devices implemented with single board computers, which are used for our proxy solutions (mqtt, http and coap proxy), each running the corresponding software. The connection between the microcontrollers and single board computers is established through their WiFi interfaces. The other type of edge context device, implemented on the desktop computer is used for the edge-cloud connector implementation, shown as *Kafka-Firebase connector* in Fig. 1b, an open source solution available from github [18]. The connection between two types of edge context devices is through a switch over their ethernet interfaces. This particular edge-cloud connector

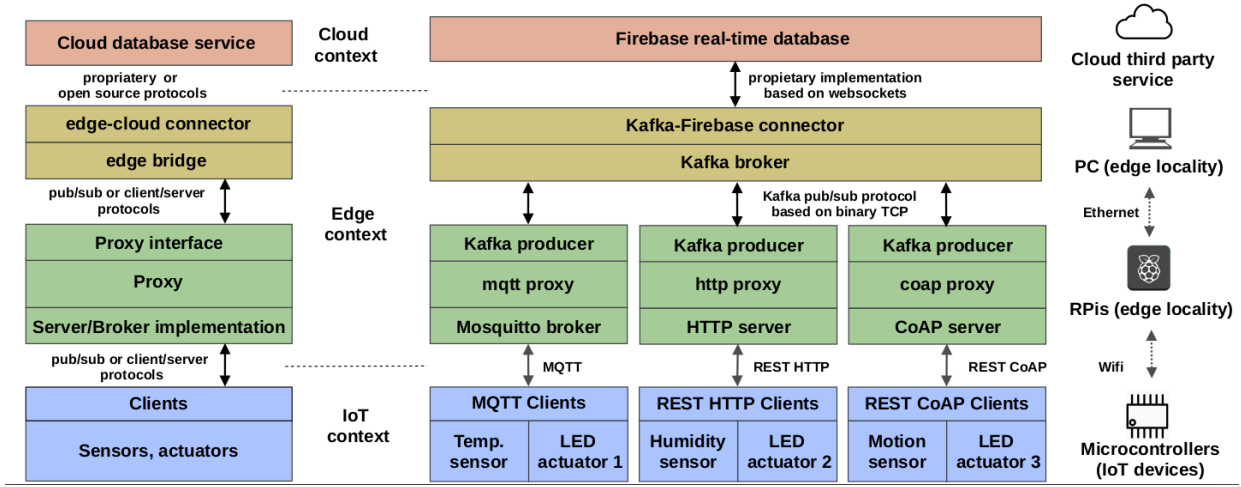


Fig. 1: NoteLab reference architecture: a) high-level overview and b) implementation

implementation was selected due to our choice of the Firebase as the cloud database service for NoteLab.

And finally, for the cloud context implementation, as mentioned, we deploy Google’s Firebase cloud solution [19]. The Firebase was selected due to being an open source and its ability of storing the client’s data locally through the edge context’s connector, and later automatically synchronizing with the cloud [20]. In addition, setting up a Firebase Database is easy and takes only few steps, providing quicker learning curve for students.

The setup and implementation of a few distinct interface and communication protocol solutions to establish communication between devices used in various contexts is one of the salient features of NoteLab. Current state of the art points at MQTT (Message Queue Telemetry Transport) as communication protocol of choice due to its maturity and performance. This is followed closely by HTTP (HyperText Transfer Protocol) as the widely adopted solution and a second best choice of developers [21]. Finally, CoAP (Constrained Application Protocol) is a well-known IoT messaging standard communication alternative to these two protocols [22] due to its lightweight characteristics and comparatively better performances in resource-constrained environments [23]. In the implementation part of architecture shown in Fig. 1b, we illustrate the choice of interfaces at the context boundaries based on few possible choices for related communication protocols: MQTT, HTTP and CoAP. When using publish-subscribe MQTT communication protocol, at the boundaries between IoT and edge context, RPi edge localized device interfaces the lower level devices of the IoT context (micro-controller boards with their attached sensors and actuators) with a MQTT Mosquitto broker. These devices serve as MQTT publisher and subscriber clients. Since the architecture is extensible, other protocols can be modularly implemented in parallel, where edge computing context would implement the server (HTTP or CoAP) while the IoT context would implement corresponding interfaces for client applications, based on REST standard software architecture.

In the remaining parts of the reference architecture, the interface between the Kafka based edge-cloud connector and Firebase database is based on a Firebase proprietary communication implementation over websockets. In the high-level overview we saw that the communication protocols and interfaces between proxy solution and edge-cloud connector are based on publish/subscribe paradigm. In the implementation this is achieved with interfacing the Kafka based connector (comprised of multiple components, one being a Kafka broker) with Kafka producer as the northbound interface on the implemented proxy solutions. The communication protocol used here is Kafka’s native binary TCP protocol, an integral part of Kafka implementation which as such is not required for students to understand all the details. Instead, students are instructed that when the software tools require native protocols to be used, it is necessary to develop the corresponding *proxies*, as described in detail in the following sections.

B. Scoping the learning framework

To appreciate the broadness of the subject matter, we now briefly give an overview of the learning scope, and outline the reasons behind our choices made. Table I gives an overview of the hardware development kit, with device-corresponding operating systems or firmware within a specific context as previously described.

TABLE I: NoteLab hardware with corresponding operating systems

Context	Hardware	Operating system or firmware
IoT	Single board ESP8266 microcontrollers: NodeMCU and WeMos D1 R2	optional-NodeMCU firmware
	Sensors: DHT11 temperature and humidity sensor, HC-SR04 ultrasonic sensor, Passive Infrared Sensor, LED actuators, set of resistors and jumper wires	/
Edge	Single board computers: RPi 3/4 Model B	Raspberry Pi OS
	Desktop computer	Ubuntu 20.04
Cloud	Desktop computer (Firebase Console)	Ubuntu 20.04

The choice of IoT hardware platforms is based on how widely available and used they are, including the availability of adequate tutorials and developer's support [24], [25]. In the IoT context, we chose two types of microcontroller development boards [26], i.e., NodeMCU and WeMos D1 R2. The practical reasons behind this choice is their WiFi connectivity support (as they are based on ESP8266) critical to building a networked system of IoT devices. In addition, these microcontrollers are programmable using Arduino IDE, a user-friendly environment as it comes with an abundance of online programming examples freely available to students. Finally, numerous low-cost sensors and actuators that were initially manufactured for the Arduino platform are also compatible with these microcontrollers. In NoteLab we use DHT11 temperature and humidity sensor, HC-SR04 ultrasonic sensor, Passive Infrared Sensor (PIR) motion sensor, LED actuators, along with a set of jumper wires and resistors for connecting the circuits. For microcontroller configurations, the course manual includes instructions on how to configure the relevant ESP8266 board parameters in Arduino IDE. This way, students can easily use the code programmed on microcontrollers.

In the edge context, as mentioned we use two types of devices, i.e., single board computer and off-the-shelf desktop computer. For single board computers, each student is provided with either a Raspberry Pi 3 Model B+ or Raspberry Pi 4 Model B, with their corresponding operating systems pre-installed. RPis are generally very popular among students, due to being low-cost, with powerful computing and interfacing features, both in hardware and software. In NoteLab, students use RPis both in the IoT context as simple workstations to program the ESP8266 boards and as a computing resource for receiving and processing sensor data. The off-the-shelf desktop computer serves as the edge-cloud connector. Its southbound interface (Kafka's broker) communicates with RPis, while its northbound interface (Firebase interface) connects to the cloud real-time database through a Firebase Console. The OS for the desktop computer is always chosen as open-source, in our case the latest Ubuntu version.

In the cloud context, students are expected to create a real-time database through Firebase console and to generate an authorization key to connect the said database to edge-cloud connector. In other words, the usage of the cloud is scoped for data storage purposes only.

A more detailed overview of the software development solutions used to scoping the course is outlined in Table II, with MQTT as the protocol of choice. (Other protocols would result in different table entries and are not listed here for brevity). It is first important to consider that also students without programming skills are able to learn the basics of different programming languages, different software solutions, and finally, an increasingly important concepts of virtualization and containerization techniques. The importance of introducing these concepts, which have long been the critical skills in high-tech industry, cannot be overstated [27], [28].

When MQTT protocol is used, different types of sensors and actuators are implemented using Arduino IDE for program-

ming MQTT clients. Programming MQTT in Arduino IDE is well documented by numerous tutorials and research activities and hence easy to use [29]–[32]. In NoteLab, students are given specific instructions on how to create the so-called Arduino sketch codes (based on C/C++) for MQTT publisher and MQTT subscriber. (The examples of this sketch code will be given in the following section.) While this code is based on C/C++, for programming in Arduino IDE, students are not required to actually know C or C++ programming. In other words, for developing simple sketches to be used in NoteLab, students are given specific scripts and instructions.

In terms of virtualization techniques, we choose Docker containerization [33]. One of the known benefits of Docker-based containerizations is its performance in CPU and memory utilization, which students can experience as especially relevant in combination with resource constrained IoT devices [34], [35]. We include in Table II a separate column that shows which system components in the reference architecture use containerization. To acquire *dockerization* skills, students use two pre-installed docker images in the edge computing context. One image is used to implement a Mosquitto MQTT broker (developed in C, [36]) on RPi devices, while the other one to implement *Kafka-Firebase connector* (developed in Java) on the desktop computer. Since these images will be used as pre-installed solutions, students only focus on modifying the related configuration parameters. On the other hand, students are requested to actually develop a *mqtt proxy*, albeit based on detailed instructions provided to them. This is critical, since the objective is to establish communication between the two docker based system edge components, i.e., Mosquitto broker and Kafka based edge-cloud connector. To program this application students are requested to use a Python script, and dockerize it.

TABLE II: NoteLab system components with MQTT as protocol of choice

Context	Device	System component (software solution)	Programming language	Dockerized
IoT	Sensor attached to ESP8266 board	Client (MQTT publisher)	Arduino sketch (code unit based on C/C++)	No
	Actuator attached to ESP8266 board	Client (MQTT subscriber)	Arduino sketch (code unit based on C/C++)	No
Edge	RPi 3/4 Model B	Broker (Mosquitto)	C	Yes
		mqtt proxy	Python script	Yes
	Desktop computer	Edge-cloud connector (<i>Kafka-Firebase connector</i>)	Java program	Yes
Cloud	Firebase real-time database			

Summarized over Tables I and II, we design the NoteLab to include the following hardware and software systems, i.e.,

Hardware systems

- ESP8266 complete development boards (Arduino compatible e.g. NodeMCU or WeMos D1 R2)
- Arduino sensor and actuator kit

- Micro USB connector (uploading the code, update firmware, charging the battery)
- Breadboard, Jumper wires and Resistor Kit
- Raspberry Pi single-board computers
- Off-the-shelf desktop computer
- WiFi router

Software systems

- Arduino IDE with added support for ESP8266 boards
- Arduino IDE libraries for WiFi support, communication protocols and different sensors(e.g. DHT11 library for temperature and humidity sensor; PubSubClient library for MQTT)
- Docker for the ARM architecture
- Docker and docker-compose for the x86 architecture
- Python programming language.

IV. LABORATORY SETUP AND TASK UNITS IN CONTEXT

Before going into details of each task unit, a summarized mapping of task units, learning objectives, student's learning outcomes and NoteLab architecture is outlined in Table III.

TABLE III: Mapping of task units with learning outcomes, learning objectives and the reference architecture

Task unit	Learning objectives	Learning outcome	Computing context
1	O2	Configure a microcontroller	IoT
2	O2	Connect sensors and actuators based on circuit layouts	IoT
3	O3, O4	Demonstrate programming of three sensors with Arduino IDE sketches	IoT
4	O3, O4	Setup WiFi network using RPi as an access point	IoT and edge
5	O3	Use containerization tool to run a proxy solution on Raspberry Pi	edge
6	O1	Establish communication between IoT devices and edge localized proxy	IoT-edge interface
7	O5	Statistical performance analysis	IoT and edge
8	O3	Configure a cloud database service	cloud
9	O1, O3	Configure and run a containerized edge-cloud connector	IoT-edge-cloud interfaces
10	O5	Statistical performance analysis of the entire system	IoT, edge and cloud

A. IoT Context (Task Units 1-4)

The familiarization with IoT devices is the foundation of this course. In IoT context, students are learning to configure microcontrollers and sensors, to wire the IoT devices with the devices in edge computing context and to writing and running the software that collects the measurements. Here, each student receives two microcontrollers and one single board computer (RPi). In addition, each student receives a set of three types of sensors and three LED actuators, with jumper wires and set of resistors that will help them connect the circuits. This is shown in the left part of Fig. 2, and it represents the basic kit for developing an IoT device with sensing and actuating functions. In this context also single board computers are used, serving as workstations to connect the microcontrollers via MicroUSB cable for energy supply (i.e., battery-less operation).

Task Unit 1 (IoT device setup): The first task unit starts with detailed instructions and all necessary information about

microcontrollers, sensors and actuators, based on the corresponding survey of their specifications, supported interfaces, circuit physical layout, pin definition, etc. After connecting the microcontrollers to single board computers, students are required to actually use the single board computer's *command-line* (the instructions will depend on the OS running on the device, pre-installed by instructors) to configure microcontrollers by flashing the NodeMCU firmware. (It should be noted that this step is not necessary for programming in Arduino IDE). The version of the firmware used is to be pre-downloaded on RPi and it can be download from [37] under a name such as *nodemcu-master-7-modules-x-float.bin*. Another requirement is to install the firmware flashing tool, called *esptool*. The *esptool* installation and utilization instructions from *commandline* are illustrated in the code block below. Independently of the board used, all microcontrollers need to be manually flashed with NodeMCU firmware. This enables the students to find out which serial port of RPi connects to the microcontroller. Using that serial port (for example *ttyUSB0*) it is possible to flash the firmware. The code block consists of following commands:

```
$ sudo apt-get install esptool
$ dmesg //Output:
[15705.320141] usb 1-1: cp210x converter now
attached to ttyUSB0
$ sudo esptool --port /dev/ttyUSB0
write_flash 0 nodemcu-master-7-modules-x-float.bin
```

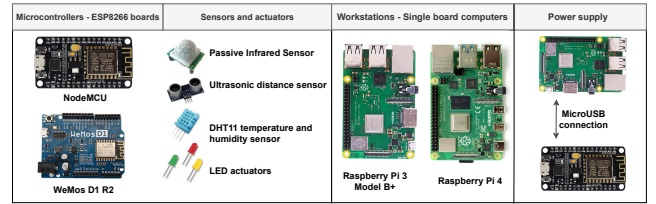


Fig. 2: Basic kit in IoT context

Task Unit 2 (Circuit setup): In this task unit, students learn how to connect each of the sensors and actuators to the microcontroller. Students receive a breadboard circuit layout and a pin layout. This is illustrated in Fig. 3 with LED actuator and DHT11 temperature and humidity sensors connected to NodeMCU board. Based on the circuit scheme and pin layout table, microcontrollers, sensors and actuators are placed on their corresponding breadboards and connected with jumper wires. The instructions provided to students include information about which type of resistor is needed for component protection. It should be noted that the circuits are different for different boards. In the example shown, LEDs have two pin interfaces, one with a long leg (anode) for positive supply that is connected with a yellow wire to one of the GPIO (General Purpose Input/Output) pins of NodeMCU, and which is then used to carry digital or analog signals. The smaller leg (cathode) is used for negative supply that will be connected with a black wire to the ground pin (GND) of a microcontroller. The resistor of value $200\ \Omega$ is added in series with the LED. In other example, when connecting NodeMCU with DHT11, three of the pins on each side are to

be connected with jumper wires: DHT11's pin that supplies power for the sensor - VCC pin connects to +3.3 V pin of NodeMCU (red wire) and DHT11's ground pin to the ground pin of the NodeMCU (black wire). The Data pin on the DHT11 sensor connects to one of the GPIO pins of NodeMCU (here, pin D3) with a green jumper wire. Finally, a 10 kΩ resistor is added between VCC and Data pin of DHT11, as can be seen in Fig. 3. This task unit needs to be repeated for all sensors.

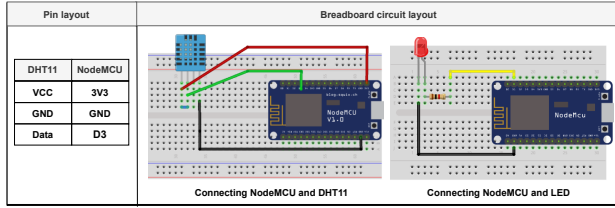


Fig. 3: Interfacing microcontrollers with sensors and actuators

Task Unit 3: (Programming microcontrollers) In this task unit, students are introduced to Arduino IDE and to programming microcontrollers through writing, compiling, and uploading the code. First, students install the latest version of Arduino IDE on the RPi used to configure ESP8266 boards. To test how DHT11 sensor can be used, students test Arduino IDE sketch code provided to them, as illustrated in Fig. 4. To upload the code, however, students are requested to find under *Tools* an option to manage libraries in Arduino IDE. This is necessary to making sure that *DHT.h* library, which is the corresponding library for sensor exemplified here, was installed. To read the output of Arduino IDE, - in this case temperature and humidity values, students use *Serial Monitor* - a separate pop-up window from Arduino IDE that acts as terminal, also to be found under *Tools*. This *Serial Monitor* is used for IoT setup verifications, since in case that the circuit components have not been correctly connected, the expected outputs after compiling and uploading the code will not be shown. In case there were no errors, the output on the monitor will show sensor readings, which for DHT11 are the values measured on temperature and humidity sensors. Arduino IDE sketch code is also tested for LED, which is commonly used as a first tutorial in Arduino IDE, since it does not require inclusion of libraries. This sketch code is to be uploaded to the second of the two microcontrollers given to students.

Task Unit 4 (Wireless network setup): In this task unit, the previously used single board computers are used to configure their wireless interfaces. To this end, students are instructed on how to program microcontrollers to receiving an IP address from the network, as illustrated in Fig. 4. To set up WiFi network, each student is given an access to a NETGEAR router, which is pre-configured with default settings. Students are first asked to connect the router to a RPi in a wired fashion, i.e., via an Ethernet cable, in order to change the default settings and be able to setup a network name of choice in *Name(SSID)* field to a value *netwX*, and the password in *Password(NetworkKey)* field to a value *passwordX*. The value noted as *X* will be replaced with a randomly generated

number for each of the RPi devices. Following the same approach the network address is to be set to 192.168.X.0 (e.g., 192.168.1.0, 192.168.2.0, ...) with subnet mask 255.255.255.0. Table IV illustrates network configurations on the example of two students working in the lab (each with their own devices).

Let us illustrate in Fig. 4 how an RPi 1 is assigned an IP address from 192.168.1.0 network. Student uses the WiFi credentials illustrated in Table IV to connect their microcontrollers to the network and in that way obtain IP addresses from the same network. In this example, it is necessary to include a few Arduino IDE WiFi libraries for ESP8266 boards as shown in the Arduino code block (the libraries included are *WiFiClient.h* and *ESP8266WiFi.h*). After uploading the code, the IoT devices (assigned IP addresses 192.168.1.y and 192.168.1.z and the RPi device are all connected to the same WiFi network, where the exact assigned IP addresses of microcontrollers can be verified in the Arduino IDE's *Serial Monitor*. This code is now merged with the sensor and actuator sketches from the previous task (with the example for DHT11 provided by instructors), which in turn finally allows students to send the data measured on sensors over a wireless network.

TABLE IV: Wireless network configurations in IoT context

Group	SSID	Password	Network address	Subnet mask
Student1(RPi 1)	netw1	password1	192.168.1.0	255.255.255.0
Student2(RPi 2)	netw2	password2	192.168.2.0	255.255.255.0

B. Edge computing context (Task Units 4-7)

The devices used in edge computing context include primarily single board computers (RPi 3 Model B+ or RPi 4) but also a desktop computer running an edge-cloud connector. RPIs are configured to run MQTT communication protocol software, including MQTT broker and the related processing software. The main purpose of the edge-cloud connector is as the name says to connecting the devices in edge and cloud context. To understand the role of communication protocols, students are given instruction, including the learning material about state of the art application layer protocols that are currently being implemented in IoT resource constrained environments. This group of Task Units also connects IoT and edge computing context, and students are tasked with reading and measuring performance of the data published by sensors in IoT context. As previously noted, each student creates a separate group of edge devices. However, all groups of edge devices stream the data measured to the same edge-cloud connector.

Task Unit 5 (MQTT broker installation): The devices in the edge context are here to be setup as MQTT brokers. The broker is responsible for receiving the data generated in form of MQTT messages, and then publishing the received messages to all subscribed clients. To install the MQTT broker, students use an open source broker Mosquitto, pre-installed as docker image made available in the official repository of container images, Docker Hub. This allows also students to get introduced to the containerization concept [38]. For pre-installation on RPIs, instructors need to download the newest ARM compatible official Eclipse Mosquitto docker image.

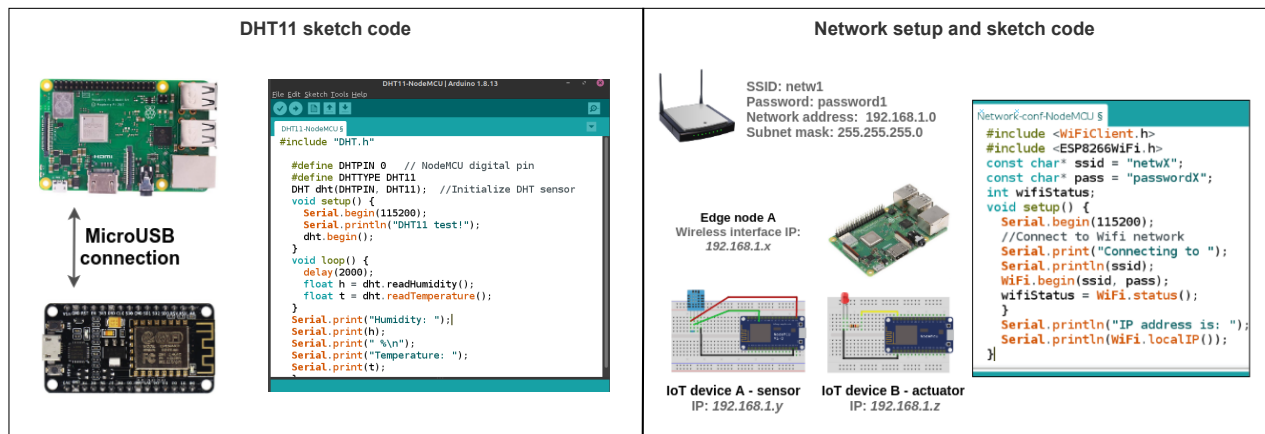


Fig. 4: IoT context - Programming microcontrollers with Arduino IDE

This in turn requires to install Docker software first, with the installation steps notably different for various models of RPis (3 or 4). With Docker installed and running, students run MQTT broker on port 1883 (this port is usually used for MQTT brokers) with a simple command *sudo docker run -p 1883:1883 eclipse-mosquitto*. The output of will be a message indicating that broker is listening for incoming messages on the port 1883. Upon completion of this task unit, the edge devices are ready for microcontrollers to connect to the MQTT broker and send the sensor data from the IoT context.

Task Unit 6 (MQTT communication): In this task unit, students learn how to program MQTT publisher and subscriber clients on microcontrollers and establish the communication with the broker, over the previously set WiFi network from Task Unit 1.4. The subscriber and publisher clients are implemented on two microcontroller boards. The subscriber is implemented on the board connected to the LED actuator while publisher connects to the sensors. (Recall that the setup of the board circuits was completed in Task Unit 1.2). The MQTT communication exchange between a publisher and a subscriber is shown in Fig. 5, on the examples of DHT11 sensor and LED actuator, respectively. Publisher, which is implemented on microcontroller connected to DHT11 sensor (IoT device A), publishes the information based on the sensor's readings, i.e., temperature and humidity values. The data is published to a topic *DHTsensor\Temp_humidity* and subscriber subscribes to this data. All communications goes through MQTT broker on RPi which listens for MQTT messages on port 1883. For the subscriber - actuator, a condition needs to be added that LED is to be turned on in case the temperature readings from the publisher exceed the threshold values. For example, this can be defined as "turn on the LED if temperature exceeds 22 degrees Celsius"). Based on the wireless network setup (Task Unit 1.3), publisher and subscriber are assigned the IP addresses. In this case, the broker can be reached on IP address of the Raspberry Pi's wireless interface, based on network configuration from Table IV. In the example of the student working on the RPi 1 with SSID and password corresponding to *netw1* and *password1*, IP address of the Raspberry Pi

wireless interface, as well as of the MQTT broker will be assigned from the network 192.168.1.0/24. By simply typing *ifconfig* in their Raspberry Pi's terminal they will be able to find out broker's IP (e.g., 192.168.1.1), which is necessary to program publishers and subscribers, as described next.

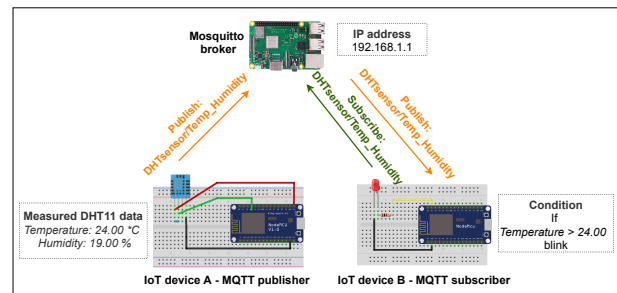


Fig. 5: MQTT communication exchange between IoT and edge context

The said programming of MQTT based communication in Fig. 5 is illustrated in Fig. 6. The main function and parameters for Arduino IDE sketches of MQTT publisher and subscriber in Arduino IDE are shown on the examples of DHT11 sensor and LED actuator, respectively. The outputs of running the code for subscriber and publisher are as follows:

```
// Publisher output:
Wifi connected
IP address is:
192.168.1.2
Connecting to Mosquitto Broker [DONE]
Humidity: 19.00 %
Temperature: 22.00 *C
Sending temperature and humidity :
[22.00, 19.00] ->
{"temperature":22.00, "humidity":18.00}
// Subscriber output:
Wifi connected
IP address is:
192.168.1.3
Connecting to Mosquitto Broker [DONE]
Message received in topic:
DHTsensor/Temp_humidity
Message: {"temperature":25.00, "humidity":36.00}
```

Task Unit 7 (Latency measurements) In this task unit, students measure one of the most critical performance indicators

```

DHT11-MQTTpub-NodeMCU 5
#include <DHT.h>
#include <PubSubClient.h>
#include <WiFiClient.h>
#include <ESP8266WiFi.h>
const char* ssid = "netw1";
const char* pass = "password 1";
const char* mqttBroker = "192.168.1.1";
const int mqttPort = 1883;
WiFiClient espClient1;
PubSubClient client(espClient1);
#define DHTPIN 0 // NodeMCU digital pin D3
#define DHTTYPE DHT11 // DHT 11
DHT dht(DHTPIN, DHTTYPE); // Initialize DHT sensor.

void setup() {
  Serial.begin(115200);
  InitWifi(); //Connects the microcontroller to the wifi netw
  dht.begin();
  delay(10);
  client.setServer(mqttBroker, mqttPort); //Connect to the broker
}

void getAndSendTemperatureAndHumidityData() {
  Serial.println("Collecting temperature and humidity data.");
  float h = dht.readHumidity();
  float t = dht.readTemperature(); // Temperature is in Celsius
  String temperature = String(t);
  String humidity = String(h);
  // Prepare a JSON payload string
  String payload = "{";
  payload += "\"temperature\": "; payload += temperature; payload += ",";
  payload += "\"humidity\": "; payload += humidity;
  payload += "}";

  // Send payload
  char attributes[100];
  payload.toCharArray(attributes, 100);
  client.publish("/DHTsensor/Temp_humidity", attributes);
  Serial.println(attributes);
}

```

(a) MQTT publisher

```

DHT11-MQTTsub-NodeMCU 5
#include <PubSubClient.h>
#include <WiFiClient.h>
#include <ESP8266WiFi.h>
const char* ssid = "netw1";
const char* pass = "password 1";
const char* mqttBroker = "192.168.1.1";
const int mqttPort = 1883;
const int LED = 2; // NodeMCU pin D4
WiFiClient espClient2;
PubSubClient client(espClient2);

void setup()
{
  Serial.begin(115200);
  InitWifi(); //Connects the microcontroller to the wifi netw
  delay(10);
  pinMode(LED, OUTPUT);
  client.setServer(mqttBroker, mqttPort); //connect to the broker
  client.subscribe("/DHTsensor/Temp_humidity");
}

void ReceivedMessage(char* topic, byte* payload, unsigned int length) {
  Serial.print("Message received in topic: ");
  Serial.println(topic);
  Serial.print("Message:");
  String temp;
  for(int i = 0; i < length; i++) {
    Serial.print((char)payload[i]);
  }
  //add a condition for turning on a LED
  Serial.println();
  if (payload[15] == '2' && payload[16] == '2') {
    digitalWrite(LED, HIGH);
  }
  else {
    digitalWrite(LED, LOW);
  }
}

```

(b) MQTT subscriber

Fig. 6: Arduino IDE programming of MQTT communication

that is the communication latency. We start with the publisher creating the message load and creating a timestamp before sending the data to the broker. Student can measure the time for the message to reach the subscriber by creating another timestamp. The time that passes between the two timestamps is the latency value measured. The experiment can be repeated for different message sizes, and the results can be statistically evaluated. Fig. 7 shows statistical results (minimum, maximum, mean values and standard deviation) of one such experiment in form of a table and a boxplot, for message sizes of 10B, 100B, and 1KB. From these measurements, students can learn that the mean value of MQTT latency is around 0.004 sec and that the size of the message did not have a major effect. The boxplot results also show that the latency for 10B message size is in the range of 0.00307s - 0.00425s, for 100B message size 0.00318s - 0.00422s, and for 1KB it is 0.00368s - 0.00462s. This simple statistical analysis introduces students also to research in this field.

Additional measurement assignments for students that completed the previous task units in shorter time-frame include measuring power consumption for different application layer protocols. This is particularly interesting for the most critical component regarding its resource capabilities, which is the device chosen for the IoT context - microcontroller. Here, students were instructed to estimate ESP8266 microcontroller board RAM utilization for sending the sensor data, by measuring its dynamic memory allocation (free Heap size). This was achieved by sending a fixed number of messages containing sensor measured data in JSON format (i.e 1000 temperature values) to the Mosquito broker and returning as the output the

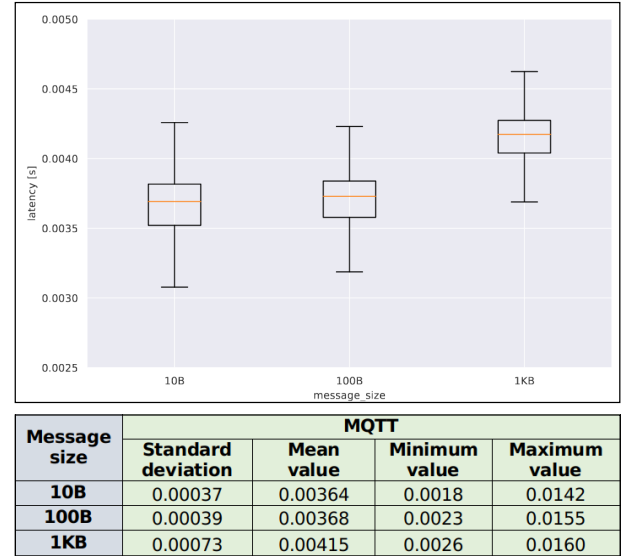


Fig. 7: MQTT latency measurements in edge context

free heap size in bytes. To ensure a correct estimation several measurements were taken and the mean value calculated. The experiment was then repeated with the HTTP protocol implementation. The obtained values in table V indicate that the free heap size is lower for HTTP, which means that for sending the same number of messages containing sensor data, using HTTP will result in microcontroller's higher RAM utilization compared to the MQTT.

TABLE V: Microcontroller RAM utilization for sending sensor data to the edge

protocol	Free Heap in Bytes
MQTT	49142
HTTP	48932

C. Cloud computing context

The cloud computing context requires the implementation of Kafka based edge-cloud connector that connects devices in edge computing context to Firebase cloud. To this end, we pre-install the desktop computer in edge context to run a dockerized *Kafka-Firebase connector* solution to all edge devices. To establish the IoT-edge-cloud communication, all edge devices in edge context need to be in the same network. We setup static IP addresses for Ethernet interfaces on each devices in edge context, as illustrated in Table VI. As the connector is not only shared by all RPi devices but also by all students controlling their corresponding RPis, the instructors manually assign static IP address on two Ethernet interfaces. One address is used for the network configured to connect to the cloud database, while the other for RPi used in the edge context, see Table VI. When all of the IP addresses have been configured students will by using the Ethernet cables connect their nodes to a switch, with one port connected with a PC, now having their entire edge domain in the same network.

TABLE VI: Configuration of static IP addresses

Edge context	IP address	Subnet mask
Desktop PC (edge-cloud connector)	192.168.5.2	255.255.255.0
Student1 (RPi 1)	192.168.5.3	255.255.255.0
Student2 (RPi 2)	192.168.5.4	255.255.255.0

Task Unit 8 (Configuring cloud database): To get started, students first sign in on the edge-cloud connector device with Google account created for NoteLab. This account allows the student to use Firebase Console (<https://firebase.google.com>) and create a new *Firebase project*. After creating a project, students select from multiple database options, and by doing so are required to opt for a real-time database. The database can be started in two modes, the so-called locked and test mode. In NoteLab, we use the test mode (the locked mode is for production solutions). Whenever a new database is created, it will have a unique URL ending in *firebaseio.com* and the URL of the database following the format *<project_id>.firebaseio.com*. This information is later used for interfacing the connector and Firebase database. In this stage, students have an empty database in Firebase cloud. The last pair of information required to connecting the connector in the edge context to the database in the cloud context are the host name and authorization key/secret key of the created project. This information can be read from *Project Settings-Service accounts* tab where the option to generate a new private key will be offered for generation. Students can save the key as *firebase-admin.json* in the connector.

Task Unit 9 (Connecting edge and cloud databases): Since this is an undergraduate course, where students are not required to learn all the development details in Java, the

connector device is pre-configured, with all relevant details saved locally, including the installation of Docker. Furthermore, the docker-compose tool is also pre-installed allowing the deployment and management of multiple containers at the same time instead of running each container individually. This is due to the fact that the Kafka based connector (*Kafka-Firebase connector* from Fig. 1b) actually includes multiple containers. Based on pre-installation, students receive the set of instructions, which consist of modifying two parameters found in *docker-compose.yml* saved locally. File *docker-compose.yml* can be opened in a text editor and two parameters need to be configured: *externalIPaddress* and *FIREBASE_URL*. This way, students have configured the database for synchronization between edge and cloud context. **Task Unit 10 (IoT-E-C performance measurements):** We illustrate end-to-end latency measurement similar to the IoT-to-edge measurements from Task Unit 7, where latency was measured as the time that passes between the two timestamps. Here, the first timestamp is also created with a MQTT publisher sending the message to the broker. The second timestamp is created when the message previously sent is received by the subscriber directly from the Firebase which represents the cloud context. The experiment is again repeated for different message sizes. Fig. 8 shows statistical results in form of a table and a boxplot, for different message sizes of 10B, 100B, and 1KB. Students can notice that compared with Fig. 7 the end-to-end latency values were noticeably higher than what was measured between IoT devices and edge.

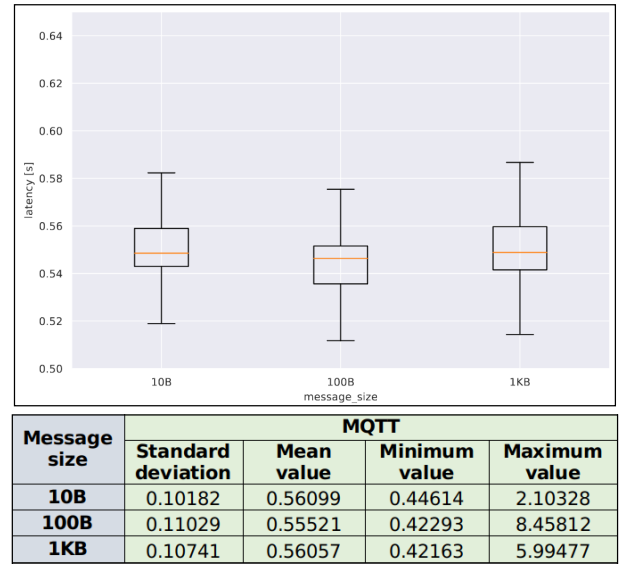


Fig. 8: MQTT latency measurements IoT-E-C

D. Lessons learnt

Undergraduate students typically come with different knowledge backgrounds and different programming and electronics class prerequisites. At the same time, most students already have specific preferences towards either software- or hardware-oriented courses. Students with knowledge in programming and software engineering performed especially

well in Task Units related to data processing and storage in IoT-E-C context (Task Units 7, 8, 9). Students more interested in electronics aspects of the course, usually excel at Task Units 1, 2, and 3. Clearly, the benefit of the course is that a number of task units is designed to combine various backgrounds and preferences (Task Units 4, 5, 6, and 10).

To motivate students without a certain background, it was essential to create a detailed instruction guidelines that cover a wide range of topics, from devices' hardware capabilities, circuit setup schematics over to programming scripts and basic instructions in various programming languages. Even then students faced challenges. For instance, when configuring hardware components, and despite following a precise circuit schematics, minor omissions in interfacing sensors and actuators inevitably happen; wrongly connected jumper wire directly leads to device damage. This was a common occurrence, and we found that it helped developing student's engineering and problem solving skills. On the flip side, the course needs to purchase backup hardware components.

In terms of developing and testing software solutions, a set of basic and detailed instructions for programming in Arduino IDE is rather critical, since also the students versed in programming may have no knowledge in low-level programming languages. While learning the programming basics with Arduino IDE is not complex in itself, students need to get used to establishing a rather strong link between software and hardware components in IoT context. In the end, students can learn that even a minor differences in hardware configurations would end up in the code being executed differently for different devices (and different students in charge of these devices). Anecdotaly, one of the common difficulties for students is to differentiate whether the device is damaged or simply disconnected, as in both cases the low-level program would just stop running.

With dockerized components, on the other hand, students can face the challenge on how to run docker images even when already prepared for them (including the instructions related to operating systems, e.g., Ubuntu and Raspbian OS). For the development of their own application scripts, we found that it was advantageous to students to have basic knowledge in Python (Python 2 and Python 3 come pre-installed on Raspbian OS). The Python programming skills are not mandatory though, as the only scripted application developed in Python was mqtt proxy. As both RPi and Desktop PC operating systems are Debian-based, the students are instructed to use *commandline* interface instead of graphical user interface for which also basic skills need to be acquired.

All challenges that students face require help of tutors and instructors. In addition, instructors also face conceptual challenges. One of the daunting challenges is to scope the learning framework, given the rapid evolution of the state-of-the-art, both in hardware and software. Instructors also need to provide a significant amount of up-to-date, detailed and workable instructions to students about the devices' hardware and software capabilities as well as the networking and communication protocols. Table VII summarizes hardware and

software ready-configurations and code that instructors need to prepare. As it can be seen, scoping the learning framework requires not only students to further challenge their hardware and software skills, but also instructors to provide the enabling hardware and software configurations.

TABLE VII: Material provided by instructors

Context	Hardware and OS	Software
IoT	<ul style="list-style-type: none"> • Pin layout, specifications of sensors, actuators and microcontrollers • Breadboard circuit scheme for connecting sensor/actuators with microcontroller 	<ul style="list-style-type: none"> • Arduino IDE configured for ESP8266 • List of libraries to program sensors and actuators • Arduino sketch code: sensors, actuators, wireless network connection, MQTT publisher/subscriber.
Edge	<ul style="list-style-type: none"> • RPi 3/4 Model B pin layout and specifications • Latest version of Raspbian OS • Breadboard circuit scheme for connecting sensor/actuators with microcontroller • Latest Ubuntu version 	<ul style="list-style-type: none"> • Mosquitto broker docker image downloaded with installation and configuration instructions • Instructions on RPi and Desktop PC static IP address configuration • mqtt proxy python script • Edge-cloud connector docker image downloaded with installation and configuration instructions
Cloud		<ul style="list-style-type: none"> • NoteLab Google account to sign in to Firebase • New Firebase project and Real-time Database created • Kafka based edge-cloud connector connected to the database

V. CONCLUSIONS AND OUTLOOK

We proposed and outlined the design and development of an undergraduate laboratory course, we named Network-of-Things Engineering Lab (NoteLab), - that set the goal of implementing various interfaces and communication protocols to connect IoT, edge and cloud computing systems and evaluate their performance. Unlike other such related courses, our course was designed to provide an implementation of the entire system, based on open-source and low-cost devices, and for the subsequent evaluation and benchmarking of the performance of the entire system. We also integrated various communication protocol and interface solutions.

In this paper, we focused on technical aspects of the course implementation. Despite receiving overwhelmingly positive feedback from students, we have not yet didactically and competently analyzed the learning outcomes and student assessments. This is also due to the course duration (one semester over two years) and due to the limited number of students per lab (six). In the future, we plan to leverage two computer science related learning taxonomies, Bloom and SOLO taxonomy, which will allow us to improve upon design of the course didactically. Future possible modular extension to our course include other application layer protocols, most notably HTTP3, security protocols, and machine learning applications.

REFERENCES

- [1] J. He, Dan Chia-Tien Lo, Y. Xie, and J. Lartigue, "Integrating internet of things (iot) into stem undergraduate education: Case study of a modern technology infused courseware for embedded system course," in *2016 IEEE Frontiers in Education Conference (FIE)*, Oct 2016, pp. 1–9.
- [2] B. Burd *et al.*, "Courses, content, and tools for internet of things in computer science education," in *Proceedings of the 2017 ITiCSE Conference on Working Group Reports*, ser. ITiCSE-WGR '17. New York, NY, USA: Association for Computing Machinery, 2018, pp. 125–139. [Online]. Available: <https://doi.org/10.1145/3174781.3174788>
- [3] S. J. Lee, A. Jung, and M. Yun, "Creative internet of things (iot) for undergraduates," in *2019 14th International Conference on Computer Science Education (ICCSE)*, Aug 2019, pp. 567–572.
- [4] A. R. Rao, D. Clarke, M. Bhadiyadra, and S. Phadke, "Development of an embedded system course to teach the internet-of-things," in *2018 IEEE Integrated STEM Education Conference (ISEC)*, 2018, pp. 154–160.
- [5] A. Farhat, T. McNeill, and J. Raven, "An interdisciplinary approach to developing an iot healthcare solution in applied higher education," in *2018 Advances in Science and Engineering Technology International Conferences (ASET)*, Feb 2018, pp. 1–5.
- [6] J. Guth *et al.*, *A Detailed Analysis of IoT Platform Architectures: Concepts, Similarities, and Differences*. Singapore: Springer Singapore, 2018, pp. 81–101. [Online]. Available: https://doi.org/10.1007/978-981-10-5861-5_4
- [7] S. Tayeb, S. Latifi, and Y. Kim, "A survey on iot communication and computation frameworks: An industrial perspective," in *2017 IEEE 7th Annual Computing and Communication Workshop and Conference (CCWC)*, 2017, pp. 1–6.
- [8] K. Karvinen and T. Karvinen, "Iot rapid prototyping laboratory setup," *International Journal of Engineering Education*, vol. 34, no. 1, pp. 263–272, 2018.
- [9] S. Kurkovsky and C. Williams, "Raspberry pi as a platform for the internet of things projects: Experiences and lessons," in *Proceedings of the 2017 ACM Conference on Innovation and Technology in Computer Science Education*, ser. ITiCSE '17. New York, NY, USA: Association for Computing Machinery, 2017, pp. 64–69. [Online]. Available: <https://doi.org/10.1145/3059009.3059028>
- [10] M. N. Barendt, N. Sridhar, and K. Loparo, "A new course for teaching internet of things: a practical, hands-on, and systems-level approach," in *ASEE annual conference and exposition*.
- [11] S. A. Nelke and M. Winokur, "Introducing iot subjects to an existing curriculum," *IEEE Design Test*, vol. 37, no. 6, pp. 24–30, 2020.
- [12] S. Maitra, A. Abdelgawad, and K. Yelamarthi, "Lab in a box : A rapidly deployable environmental monitoring iot system," in *2019 IEEE 62nd International Midwest Symposium on Circuits and Systems (MWSCAS)*, Aug 2019, pp. 472–475.
- [13] Y. Ai, M. Peng, and K. Zhang, "Edge computing technologies for internet of things: a primer," *Digital Communications and Networks*, vol. 4, no. 2, pp. 77 – 86, 2018. [Online]. Available: <http://www.sciencedirect.com/science/article/pii/S2352864817301335>
- [14] J. Pan and J. McElhannon, "Future edge cloud and edge computing for internet of things applications," *IEEE Internet of Things Journal*, vol. 5, no. 1, pp. 439–449, Feb 2018.
- [15] R. Mahmud, F. L. Koch, and R. Buyya, "Cloud-fog interoperability in iot-enabled healthcare solutions," in *Proceedings of the 19th International Conference on Distributed Computing and Networking*, ser. ICDCN '18. New York, NY, USA: Association for Computing Machinery, 2018. [Online]. Available: <https://doi.org/10.1145/3154273.3154347>
- [16] R. Grammenos and C. Poole, "Teaching the internet of things: The first three years," in *2019 26th International Conference on Telecommunications (ICT)*, April 2019, pp. 265–269.
- [17] C. C. Chan, M. Tsui, M. Y. Chan, and J. H. Hong, "Applying the structure of the observed learning outcomes (solo) taxonomy on student's learning outcomes: An empirical study," *Assessment & Evaluation in Higher Education*, vol. 27, no. 6, pp. 511–527, 2002.
- [18] GitHub. (2020) Bridge between kafka and firebase realtime database. [Online]. Available: <https://github.com/fcarp10/kafka-firebase-aggregator>
- [19] Google. (2014) Firebase realtime database. [Online]. Available: <https://firebase.google.com/products/realtime-database>
- [20] F. Carpio, M. Delgado, and A. Jukan, "Engineering and experimentally benchmarking a container-based edge computing system," 2020.
- [21] Eclipse IoT Working Group, IEEE IoT, AGILE IoT, and Open Mobile Alliance, "IoT Developer Survey," pp. 1 – 39, 2018.
- [22] A. Bhattacharjya, X. Zhong, J. Wang, and X. Li, *CoAP—Application Layer Connection-Less Lightweight Protocol for the Internet of Things (IoT) and CoAP-IPSEC Security with DTLS Supporting CoAP*. Cham: Springer International Publishing, 2020, pp. 151–175. [Online]. Available: https://doi.org/10.1007/978-3-030-18732-3_9
- [23] J. Dizdarević, F. Carpio, A. Jukan, and X. Masip-Bruin, "A survey of communication protocols for internet of things and related challenges of fog and cloud computing integration," *ACM Comput. Surv.*, vol. 51, no. 6, Jan. 2019. [Online]. Available: <https://doi.org/10.1145/3292674>
- [24] C. Kishor Kumar Reddy, P. R. Anisha, R. Shastry, and B. V. Ramana Murthy, "Comparative study on internet of things: Enablers and constraints," in *Data Engineering and Communication Technology*, K. S. Raju, R. Senkerik, S. P. Lanka, and V. Rajagopal, Eds. Singapore: Springer Singapore, 2020, pp. 677–684.
- [25] P. E. Hertzog and A. J. Swart, "Arduino-enabling engineering students to obtain academic success in a design-based module," in *2016 IEEE Global Engineering Education Conference (EDUCON)*, 2016, pp. 66–73.
- [26] I. Grokhotkov. (2017) Esp8266 arduino core's documentation - boards. [Online]. Available: <https://arduino-esp8266.readthedocs.io/en/latest/boards.html>
- [27] C. Boettiger, "An introduction to docker for reproducible research," *SIGOPS Oper. Syst. Rev.*, vol. 49, no. 1, pp. 71–79, Jan. 2015. [Online]. Available: <https://doi.org/10.1145/2723872.2723882>
- [28] A. Hess. (2019) The top 20 tech skills of 2019. [Online]. Available: <https://www.cnbc.com/2019/11/24/top-20-tech-skills-of-2019-and-the-easiest-one-to-learn-in-2020.html>
- [29] T. Keophilavong, Widyawan, and M. N. Rizal, "Quality of service of protocols performance evaluation for internet of thing applications development using low-cost devices," in *Proceedings of the 2019 2nd International Conference on Information Science and Systems*, ser. ICISS 2019. New York, NY, USA: Association for Computing Machinery, 2019, p. 166–170. [Online]. Available: <https://doi.org/10.1145/3322645.3322694>
- [30] V. A. Barros, S. A. B. Junior, S. M. Bruschi, F. J. Monaco, and J. C. Estrella, "An iot multi-protocol strategy for the interoperability of distinct communication protocols applied to web of things," in *Proceedings of the 25th Brazilian Symposium on Multimedia and the Web*, ser. WebMedia '19. New York, NY, USA: Association for Computing Machinery, 2019, pp. 81–88. [Online]. Available: <https://doi.org/10.1145/3323503.3349546>
- [31] S. Garg and M. S. Ansari, "Implementation of rest architecture in arduino based home automation system," in *2017 International Conference on Innovations in Control, Communication and Information Systems (ICICCI)*, 2017, pp. 1–3.
- [32] R. K. Kodali and V. S. K. Gorantla, "Restful motion detection and notification using iot," in *2018 International Conference on Computer Communication and Informatics (ICCCI)*, 2018, pp. 1–5.
- [33] B. B. Rad, H. J. Bhatti, and M. Ahmadi, "An introduction to docker and analysis of its performance," *International Journal of Computer Science and Network Security (IJCSNS)*, vol. 17, no. 3, p. 228, 2017.
- [34] P. Bellavista and A. Zanni, "Feasibility of fog computing deployment based on docker containerization over raspberrypi," in *Proceedings of the 18th International Conference on Distributed Computing and Networking*, ser. ICDCN '17. New York, NY, USA: Association for Computing Machinery, 2017. [Online]. Available: <https://doi.org/10.1145/3007748.3007777>
- [35] Z. Y. Thean, V. Voon Yap, and P. C. Teh, "Container-based mqtt broker cluster for edge computing," in *2019 4th International Conference and Workshops on Recent Advances and Innovations in Engineering (ICRAIE)*, 2019, pp. 1–6.
- [36] T. E. Foundation. eclipse-mosquitto docker image. [Online]. Available: https://hub.docker.com/_/eclipse-mosquitto
- [37] (2020) Nodemcu custom builds. [Online]. Available: <https://nodemcu-build.com/>
- [38] H. Knoche and H. Eichelberger, "Using the raspberry pi and docker for replicable performance experiments: Experience paper," in *Proceedings of the 2018 ACM/SPEC International Conference on Performance Engineering*, ser. ICPE '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 305–316. [Online]. Available: <https://doi.org/10.1145/3184407.3184431>