Crowdnet: crowd-powered network

Nuno Miguel Ribeiro Silva
nuno.m.ribeiro.silva@tecnico.ulisboa.pt

Instituto Superior Técnico, Lisboa, Portugal

September 2020

Abstract

The Arduino platform has gained popularity and is used to teach students about microcontrollers and embedded systems in laboratory environments. To connect various Arduino boards together, a wired connection, such the I^2C bus, can be used. However, wired connections require running wires between each device, complicating the setup of many devices, and only allow connecting devices in the same room. Moreover, due to the recent COVID-19 pandemic, allowing students to work remotely and out of a laboratory is becoming an important consideration for schools and universities. In this case, a simple wireless interface would also not be sufficient to allow communication between devices in different rooms or homes, as it would offer a complicated interface, among other issues. Crowdnet intends to provide a middleware for communication between Arduino boards, supporting communications both in the same room and in different rooms over the Internet, while offering a simple I^2C-like programming interface. It consists on an Arduino library that connects to an Android application via BLE. In turn, the smartphone connects to an MQTT Broker via the Internet. This creates a logical network that allows sketches running on various Arduino boards to exchange messages with each other. Evaluation showed that Crowdnet is able to achieve reasonable throughput speeds up to 4.4 kB/s and latencies of 25–136 milliseconds, even though results varied depending on the smartphones that were tested. Nonetheless, the achieved results are adequate for simple laboratory projects, and using MQTT in conjunction with BLE provides greater throughput than just BLE.

Keywords: Communication middleware, Arduino library, Android smartphone, BLE, MQTT, I2C

1. Introduction

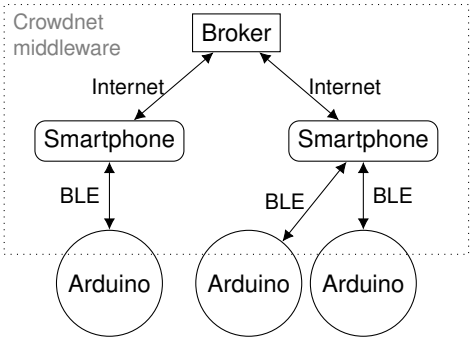
With the expansion of the Internet of Things (IoT), the Arduino platform has gained popularity, arguably because it enables beginners to develop simple projects using a microcontroller, and because it is a great tool to teach students about embedded systems in laboratory environments.

Communication in IoT devices is crucial. For devices close to each other, one solution is to use a wired connection, such the Inter-Integrated Circuit (I^2C) bus, which is one of the easiest communication solutions supported in the Arduino environment. However, these require wires between each device, which is particularly cumbersome when more than a few devices need to be connected, and even more so in laboratory environments involving many students. A wireless network such Wi-Fi, Zigbee, etc, could be used, but would offer a very different and complicated interface than what beginners are used to, among other problems.

Due to the COVID-19 pandemic (Rahiem, 2020), allowing students to work remotely is becoming an important consideration for schools and universities. Using a wired bus, students could work at home, but would be unable to have their projects interact with each other. In this case, a simple wireless interface would also not be sufficient to allow communication between devices in different homes, and a more elaborate solution is needed.

At the same time, most people own a smartphone. Therefore, we have built a middleware that uses the communication interfaces of smartphones and uses them as a means of communication between Arduino boards, as per Figure 1. By connecting Arduino boards via Bluetooth Low Energy (BLE) to smartphones and enabling smartphones to communicate with each other via the Internet, Crowdnet effectively allows Arduino boards to exchange arbitrary messages with each other indirectly using a wireless interface, while also offering a familiar I^2C-like programming interface that

Figure 1: Components of the Crowdnet middleware



is not limited to a single room, allowing students to test and integrate their work with other students remotely.

1.1. Context and motivation

This work is primarily intended to be used in laboratory classes at Instituto Superior Técnico (IST), though its applications are not limited to this. At IST, Arduino Uno boards are used for teaching in courses attended by Computer Science students.

In the Applications and Computation for the Internet of Things (ACIC) course[1], students are given an Arduino board to be used in their projects (Cunha, 2017). After a few introductory exercises, students are assigned a larger project which consists on implementing a traffic lights system.

Students can take the Arduino board home during project development. However, since the exercises and project they need to complete focuses on communicating using the I^{2}C bus and integrating projects of different students together, students are not able to properly test their project at home or outside the laboratory, and are limited by its tight schedule. Moreover, connecting many Arduino boards together using I2C quickly becomes impractical, for example, when 10 groups of students are in the same laboratory. Crowdnet solves these problems by allowing Arduino boards to communicate with each other in the same room or remotely via the Internet, using the students' smartphones. This allows students to test their projects at any time, in their homes, and requires no wires.

Crowdnet may also be useful in the Ambient Intelligence (AI) course[2], where students develop a project of their choosing, and many choose to use the Arduino platform. Therefore, they could use Crowdnet to provide them with more advanced communication between various Arduino boards, for example in a Smart Home application.

[1] `https://fenix.tecnico.ulisboa.pt/disciplinas/ACPIC7/2020-2021/1-semestre`, accessed on 1st Sep, 2020

[2] `https://fenix.tecnico.ulisboa.pt/disciplinas/AI514/2019-2020/2-semestre/`, accessed on 2nd Sep, 2020

Since these courses already use the Arduino platform, Crowdnet was developed using Arduino.

1.2. Requirements and Objectives

Crowdnet aims to create a logical network that:

- enables Arduino boards to exchange arbitrary messages between each other: a) in the same room or in a laboratory environment, and b) in different rooms, buildings, or homes;
- offers a simple API for Arduino sketches;
- is compatible with the Arduino IDE;
- uses the smartphones of the students to avoid needing additional hardware and to enable future expansion of the system.

In terms of communication parameters, and having in mind the use cases for which Crowdnet is intended, the requirements of the system are: a) latencies in the order of a few hundred milliseconds, but less than one second, and b) speeds in the order of a few kilobytes per second.

These requirements mean that Crowdnet may not be suitable for critical real time communications. In particular, since it is intended to be used by students in a laboratory environment, the flexibility of the system is more important than the performance that could be achieved using a wired communication bus such as I^{2}C.

This document is organized as follows: Section 2 describes a few communication methods we considered. In Section 3, the architecture of the system is described. Then, Section 4 describes the system implementation and communication in the various components. Section 5 describes the system evaluation and results. Finally, Section 6 concludes the paper.

2. State of the Art

In this Section the state of the art is described.

2.1. Arduino communications

This section presents some existing alternatives for communicating with an Arduino board.

2.1.1 I2C bus

I^{2}C is a low-speed serial communication bus that is widely used to connect microcontrollers to peripheral Integrated Circuits (ICs) or to other microcontrollers within a maximum distance of a few meters (Leens, 2009).

Devices in the bus are identified by a 7-bit address, and the typical raw bit speed is 100–400 kHz, depending on device support.

I^{2}C is simple, widely supported, and easy to use in the Arduino Integrated Development Environment (IDE), since it is supported by the official Ar-

duino Wire library [3]. However, because it requires short physical connections, I^2C is limited to a single room. Since we want to support students working in different rooms, outside the lab, or even from their homes, a central bus such as I^2C is not appropriate. Nonetheless, since Arduino users are mostly already familiar with the Wire interface, the Crowdnet Arduino library was based on it, as described in Section 3.1.

2.1.2 Wi-Fi

Wi-Fi is a standard for Wireless Local Area Networks (WLANs) intended to replace traditional wired Local Area Networks (LANs) in some cases (Lee et al., 2007). Reaching speeds in the order of hundreds and even thousands of megabits per second, it is most commonly used to provide Internet access to users connected to a central Access Point (AP) located within a range of few meters, though it also supports ad hoc communications.

In our project, using Wi-Fi would require a complex and relatively heavy Internet Protocol (IP) stack implementation to run on the Arduino. Moreover, connecting an Arduino and smartphone to the same network would require connecting them to the same AP or using an ad hoc connection, which may leave them without access to other networks. Therefore, another wireless communication method is preferred.

2.1.3 BLE

BLE is a low-power Wireless Personal Area Network (WPAN) aimed at short-range communications for IoT applications (Dian et al., 2018), operating in the 2.4GHz radio frequency with a maximum application layer throughput of about 221 kbps. Nowadays, both BLE and classic Bluetooth are implemented in most smartphones (Gomez et al., 2012). BLE support on Arduino is possible either using modules such as the *Bluefruit LE* from Adafruit [4], or the more recent and cheap *ESP32* System on Chip (SoC) from Espressif Systems.

In a BLE connection, exactly two devices communicate with each other. To keep the connection alive, devices must periodically exchange a packet. These periods where devices talk to each other are known as a connection event. To minimize power usage, BLE allows tuning the time between each connection event — the connection interval — in a range between 7.5 milliseconds and 4 seconds (Gomez et al., 2012). A higher value will use less power, but will result in higher latency and lower throughputs.

There are several advantages to using BLE:

- in a laboratory setting, the BLE interface could be used for other applications, such as having the Arduino interact with user interfaces running on a smartphone;
- BLE uses less energy than other methods;
- contrary to Wi-Fi, BLE does not depend on an existing or central AP to operate; this allows students to work remotely and, using only local Nodes, they can still work without an Internet connection;
- also contrary to Wi-Fi, a complex IP stack is not necessary to use BLE;
- finally, BLE is present on most smartphones.

Therefore, BLE was used in this work.

2.2. Internet communications

This section describes a few ways to communicate with other devices over the Internet. Though some of these methods can also be used for communication between Arduino boards using an appropriate shield, our main goal is to discuss some of the existing options for smartphones to communicate with each other.

2.2.1 HTTP

Hypertext Transfer Protocol (HTTP) is an application protocol designed primarily for transferring information in the World Wide Web. It uses a client-server model and runs on top of the Transmission Control Protocol (TCP) protocol.

When HTTP is used for communication in IoT applications it suffers from the following issues (Yokotani and Sasaki, 2016):

- its text-based nature requires more bandwidth;
- it requires more processing power and is more complex to encode and decode requests;
- since it uses a new short-lived TCP connection, the TCP 3-way handshake for connection establishment is potentially repeated for every application data block that needs to be transferred.

2.2.2 MQTT

Message Queuing Telemetry Transport (MQTT) is a lightweight network protocol based on a publish-subscribe communication pattern (Yassein et al., 2017). Every device establishes a Transmission Control Protocol over IP (TCP/IP) connection with a central Broker, which delivers messages to their final destinations.

[3] `https://www.arduino.cc/en/reference/wire`, accessed on 6th Jul, 2020

[4] `https://www.adafruit.com/product/1697`, accessed on 11th Aug, 2020

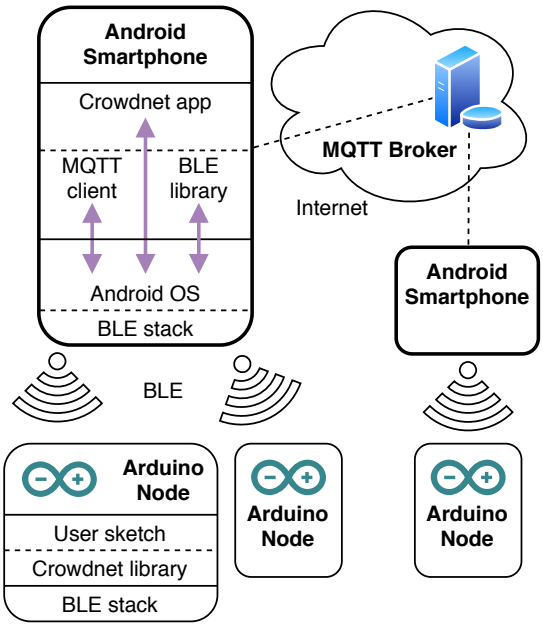
The publish-subscribe pattern provides a simple, flexible and efficient method of exchanging messages. MQTT organizes messages in *topics*, on which a client device can *publish* messages consisting of arbitrary binary data. These messages are sent to the Broker, which distributes them to other clients that have previously *subscribed* to the *topic*. This is done without clients being aware of which clients are subscribed to which *topics*. Clients do not need to connect directly to each other, eliminating the need for address discovery when publishing Messages. Furthermore, the publish-subscribe pattern combined with a long-lived TCP/IP connection eliminates the need to periodically poll the server.

These characteristics make MQTT very suitable to support efficient communications between smartphone devices in this work.

3. Architecture

This Section presents the architecture of Crowdnet. This system consists of three main components, as depicted in Figure 2: an Arduino library, an Android smartphone application (App), and a server back-end.

Figure 2: Overview of the system architecture



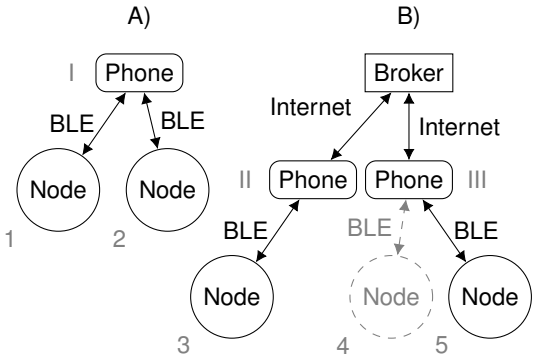
The Arduino library connects to a smartphone via BLE. In turn, the smartphone connects to the server (Broker) via the Internet. This creates a logical network that allows sketches running on the various Arduino Nodes to communicate with each other.

Using this architecture, we are able to support two main message delivery strategies, as described in Figure 3:

A) **Local Nodes** – two or more local Nodes communicating with each other via a single smartphone. Nodes can not communicate directly with each other via BLE. This is because Nodes are acting as BLE peripherals, which can only establish a connection with a single device at a time, as further explained in Section 4.2. The smartphone, however, is acting as a BLE central and is able to connect to multiple peripherals at the same time. Nodes are connected to the same smartphone, which is able to forward messages between the Nodes using only BLE communication, and with no intervention from the server.

B) **Remote Nodes** – one Node (5) communicating with a remote Node (3) via their two corresponding smartphones. In this case, smartphones connect, via the Internet, to a common server in order to forward messages between each other. When Nodes are not in reach of the same smartphone — because they are in different rooms or belong to different students — their corresponding smartphones can still communicate indirectly with each other via the Internet, enabling communication between remote Nodes.

Figure 3: Main Crowdnet use cases



A combination of both use cases is also supported. This is represented by the dashed Node (4) in Figure 3, which is able to communicate with both the local Node (5) that is connected to the same smartphone, and, via the Broker, with the remote Node (3) connected to another smartphone. In this case, smartphone II would subscribe to messages destined to Node 3, while smartphone III would subscribe to messages destined to Nodes 4 and 5. As explained in Section 2.2.2, to send a message to a remote Node, a smartphone does not need to be aware of which smartphone the remote Node is connected to. It simply sends the message to the Broker.

3.1. Arduino Library

The first component of the system is an Arduino library which users can include while developing their own sketches using the Arduino IDE[5].

The Crowdnet Arduino library sits on the edge of the Crowdnet middleware, as per Figure 1, communicating with the Android App via BLE. An Arduino board running a user sketch that communicates using this library is called a *Node*. The library allows a Node to indirectly communicate with another Node via the Crowdnet middleware, and uses an Application Programming Interface (API) which closely matches the interface offered by the Arduino Wire library [6] for the I^{2}C protocol. This is because the target users are already familiar with Wire and will be able to easily port their sketches to use Crowdnet instead of Wire. It also hides the complexities of using BLE away from users.

For I^{2}C compatibility, Nodes are assigned a 7-bit identifier (ID), similar to the device address used by I^{2}C. This is further detailed in Section 4.2.

3.2. Android smartphone application

The Android App supports communication between local Nodes, which are within range of the BLE signal of a smartphone; and remote Nodes, which are not directly accessible by a given Node or smartphone, but can be reached via the server.

Therefore, the Android App serves as gateway in the following ways:

- it receives packets from local Crowdnet-enabled Nodes, containing a message and the address (or ID) of a destination Node; if the destination Node is a local Node, the message is sent to it without server intervention; otherwise, it is sent to the server;
- using the server, it subscribes to messages that are destined for local Nodes;
- when notified by the server of a new message from a remote Node, it receives and sends the message to the corresponding local Node.

Communication with the Arduino library running on the Nodes is done using BLE, while communication with the server is done via the Internet using the MQTT protocol, as described, respectively, in Section 4.2 and Section 4.3.

3.3. Server back-end

To communicate with each other, instances of the Android App on each smartphone interact with a server back-end: an MQTT Broker that receives, temporarily stores, and forwards messages between smartphones. The implementation of the server is described in Section 4.3.

4. Implementation

This Section explains the implementation details of the Crowdnet architecture and its components.

4.1. Arduino library

The Crowdnet Arduino library was implemented to run on an *ESP32* SoC, which is available in a number of form factors (Maier et al., 2017), such as the `ESP32-WROOM-32` module that we used.

The ESP32 was chosen instead of an Arduino board with an additional shield because it was not possible to acquire a BLE shield for the existing Arduino boards in a timely manner due to budget and stock constraints. The ESP32, though, is very cheap and was readily available to be acquired by the author. It also incorporates both functionalities in the same board, simplifying the setup.

To use BLE hardware available on the ESP32, the library depends on a BLE wrapper library[7] that is part of the Arduino core for ESP32. However, other BLE APIs are not 100% compatible and therefore, even though we are using the Arduino environment, our implementation will naturally need some porting if another BLE shield or module is to be used. Nonetheless, the protocol we implemented on top of BLE can be used.

Note that these implementation details are completely transparent to users of the Crowdnet Arduino library, since all they need to do on their sketches is to interface the library using the I^{2}C-like API it offers.

4.2. BLE communication

To enable the Arduino library to communicate with the Android App, and vice versa, a custom protocol was developed on top of BLE.

Using the Crowdnet BLE protocol, the Nodes and the Android App are able to send arbitrary messages to each other. The length of the messages is limited by the underlying BLE protocol to 512 bytes, minus the length of the Crowdnet meta-data (2–8 bytes). For this to work, the App changes the BLE Maximum Transmission Unit (MTU) from the default of 20 bytes (Dian et al., 2018) to 512 bytes.

In the Crowdnet middleware, Nodes are identified by the EUI-48 Media Access Control (MAC) address of their BLE interface. However, for I^{2}C compatibility, a 7-bit Node identifier (ID) is also supported and is mapped to the corresponding MAC address. When an Arduino sketch uses the

[5] `https://www.arduino.cc/en/main/software`, accessed on 8th Sep, 2020

[6] `https://www.arduino.cc/en/reference/wire`, accessed on 6th Jul, 2020

[7] `https://www.arduino.cc/reference/en/libraries/esp32-ble-arduino/`, accessed on 25th Aug, 2020

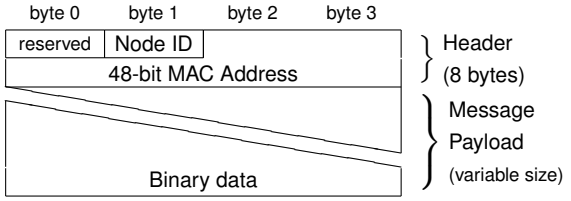
Crowdnet library, the user can choose to send a message using either the ID or the MAC address. Since Nodes only communicate directly with the smartphone they are connected to, Nodes do not need to be aware of the *ID to MAC* mappings in order to transmit a message. Smartphones, though, may receive messages destined to a given ID, and need determine which BLE device to forward the message to. Therefore, smartphones keep track of the *ID to MAC* mappings of their local Nodes. Mapping for remote Nodes is handled by the smartphones and the server, as per Section 4.3.

BLE devices can serve one of two Connection Roles: the Central Role, used by the device that scans for advertisements and initiates connections; the Peripheral Role, used by the device which advertises itself and accepts connections from Central devices. A Central device can connect to multiple Peripherals, but one Peripheral can only connect to one Central. The way two BLE devices transfer data back and forth is governed by the Generic Attribute Profile (GATT), which defines a few concepts that enable communications (Dian et al., 2018). Independently of the Connection Roles, a device can act either as a GATT Server or Client. The Server stores data locally, while the Client accesses or sends data to the Server. In Crowdnet, Nodes act as a Peripheral with one GATT Server, while the smartphone, to be able to connect to multiple devices, assumes the Central Role as a GATT Client. Upon user request, smartphones scan for, and connect to, Nodes that advertise the Crowdnet Service.

4.2.1 Packet structure

The Crowdnet Arduino library and the Android Application exchange messages by sending packets with a well-defined structure. A packet, as per Fig. 4, is a Crowdnet message, plus some metadata, that is sent over the `Node Tx` and `Node Rx` characteristics described in Section 4.2.2.

Figure 4: Crowdnet BLE Packet structure



The packet begins with one byte that is reserved for future use and must be zero. The next byte stores an I^{2}C-like Node ID. An ID equal to zero means that no ID was assigned. The next 48-bits (6 bytes) encode a MAC address. The Node ID and MAC address fields correspond to:

- the *destination* address, when the packet is transmitted *from* an Arduino; In this case, the MAC address field is omitted when the ID is not zero and the payload starts at byte 0x002.
- the *source* address, when the packet is transmitted *to* an Arduino. In this case, the MAC field is always present.

Next are the actual message contents — the payload — which can have a variable size between 0 and $512 - 8 = 504$ bytes. The exact size of the payload field is not part of the packet, as it can be derived in runtime from the total size of the packet, as received by the BLE stack.

Since Nodes can only communicate directly with a smartphone, our BLE packets only need to carry one address (source or destination) at a time, as the missing address is always the Node itself.

4.2.2 GATT Service

In BLE communications, devices serving the Server Role offer one or more GATT Services identified by a Universally Unique Identifier (UUID). These services can then expose one or more Characteristics.

A disconnected BLE peripheral device can send BLE advertisements of a few bytes, which are broadcast to whichever devices are listening. These advertisements include the device's name, MAC address and, among other things, a list of custom UUIDs identifying the Services offered by the peripheral.

Crowdnet Nodes offer and advertise one Service — the Crowdnet GATT Service, which is assigned a 128-bit UUID hard-coded in the library. If a device is advertising this Service UUID, then the Crowdnet App knows that it is a Crowdnet Node and can connect to it. Therefore, this UUID is used by the smartphone to scan for, and connect to, BLE devices which are running the Crowdnet GATT Service. During the scan, Nodes can also be filtered by a *Group ID*, as per Section 4.2.3.

GATT Characteristics The Crowdnet GATT Service is created by the Crowdnet Arduino library to enable communication with the smartphone via BLE. Each Characteristic is identified by a UUID and contains a value. The following GATT Characteristics are exposed by the Crowdnet Service:

Node Tx – Node to Gateway communication. A Node writes to this characteristic when it needs to transmit a packet to the smartphone. This characteristic supports notifications, meaning that the smartphone does not need to poll it for new data.

Node Rx – Gateway to Node communication. A smartphone writes to this characteristic when it needs to send a packet to the Node.

Node Attributes – Contains read-only information about the Node, such as its I²C-compatible address — the Node ID — and its Group ID .

Values passed in the `Node Tx` and `Node Rx` characteristics use the Crowdnet packet format described in Section 4.2.1.

4.2.3 Group ID filtering

When using the Arduino library, users can assign a *Group ID* to the Node. Not to be confused with the *Node ID*, the *Group ID* is not related to the workings of I²C or BLE, but it is used by the smartphone to filter Nodes found during a scan, allowing the Android App to only connect to Nodes belonging to a Group ID that was previously chosen by the user in the App. This allows multiple smartphones and Nodes to operate in the same room without trying to connect to Nodes belonging to another group. This is useful in a laboratory setting involving various groups of students, each with a different smartphone and set of Nodes.

Because the *Group ID* needs to be read by the smartphone during scanning and before actually connecting to the device, it is concatenated to the device name, which is part of the BLE advertisements. To facilitate parsing, the *Group ID* is encoded as a string in base-10, prefixed by a space and the letter "`G`". For example, a Node given *Group ID* `123` would advertise the name "`Node G123`". To double check that the ID was parsed correctly, it is also included in the `Node Attributes`, which can only be read after a connection is made.

4.3. Server communication

The Android App instances, running on various smartphones, communicate with each other using an MQTT client that connects to an MQTT Broker.

MQTT uses a lightweight *publish-subscribe* model to deliver arbitrary *messages* via TCP, as explained in Section 2.2.2. Crowdnet messages are organized by the Broker in *topics*. A client can *subscribe* to topics it is interested in or *publish* messages on a given *topic* in the Broker. These messages are sent to, or *published* on, the server (Broker), which distributes them to clients that have previously *subscribed* to the *topic*. This is done without clients being aware of which clients are subscribed to which topics.

In Crowdnet, two topics are reserved for each destination Node: one topic for sending messages to a Node addressed by MAC (`"crowdnet/messages/mac/<MAC>"`) and another topic for addressing Nodes by ID (`"crowdnet/messages/id/<ID>"`), where `<MAC>` and `<ID>` correspond, respectively, to the BLE MAC address or I²C-compatible ID of the Node to which the message should be forwarded to. By organizing topics in this manner, the Broker is able to deliver messages to the correct smartphone, which in turn delivers it to the Node.

Whenever the smartphone receives a message for an unknown Node, the message is published to the server on one of the topics corresponding to the destination Node. Similarly, when the smartphone connects to a Node via BLE, the former subscribes to the topic corresponding to this Node's ID, if one was assigned, and MAC address. This way, the smartphone is notified whenever a new message for this Node is published in the server by another smartphone, allowing smartphones to forward messages between each other and deliver them to the corresponding Nodes.

Additionally, since smartphones subscribe to both ID and MAC topics belonging to their local Nodes, the former need not be aware of the *ID to MAC* mapping of remote Nodes: they can simply use the ID topic to publish a message without knowing the corresponding MAC address, and the message will be delivered.

The Crowdnet MQTT Broker server is implemented by an instance of *Eclipse Mosquitto* (Light, 2017), a Free and Open-Source Software (FOSS) lightweight MQTT Broker implementation.

4.4. Android application

This section describes the components of the Crowdnet Android application (App). It runs on Android 6.0 Marshmallow — API level 23 — or newer, which, at the time of writing, should cover 85% of devices that are active in the Google Play Store.

The App is separated into a few separate Java packages or components: the Crowdnet domain, an Android Service, and the User Interface (UI).

4.4.1 External dependencies

The Crowdnet Android App depends on the following external libraries, which are bundled with the App and are automatically installed: the MQTT client implementation provided by the *Eclipse Paho* project[8], used to create a separate Android Service to handle the MQTT connection to the Broker; the BLESSED[9] library, a wrapper around the standard Android BLE classes used for interfac-

[8] `https://www.eclipse.org/paho/`, accessed on 12th Aug, 2020

[9] `https://github.com/weliem/blessed-android`, accessed on 1st Sep, 2020

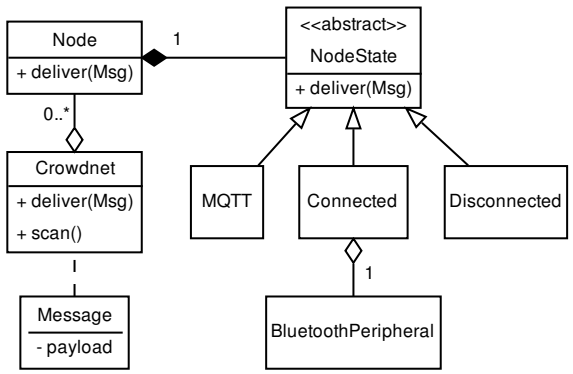
ing the BLE adapter; and the Android Jetpack[10] library, which is used to make the application code compatible with older Android versions.

4.4.2 Domain

The Crowdnet domain is a separate Java package that abstracts all the details of the Crowdnet middleware in the Android App. It interfaces with both BLE and MQTT using, respectively, the BLESSED and Paho libraries mentioned above.

The domain contains the following main classes, as per Figure 5:

Figure 5: Simplified UML Class diagram of the Crowdnet Domain in the Android application
Note: some classes and methods are omitted for simplicity.



Crowdnet is a Singleton Facade class for the domain, which is used by the Crowdnet Service.

Message represents a message that needs to be forwarded to a `Node`. It contains the source address, destination, and payload. Can either be created from a packet received via BLE or when data is received from the MQTT Broker.

Node represents an Arduino `Node`, either remote or local, that the App is aware of; `Node` instances are created either by performing a BLE scan or by sending `Messages` to an unknown destination.

NodeState is an abstract class implementing the State and Strategy Patterns; each `Node` is associated with one `NodeState`; the operation of delivering a `Message` to a `Node` is delegated to the `Node`'s concrete `NodeState`:

Connected `Nodes` transition to this state when connected via BLE to the smartphone, and if the MQTT Broker is connected, the `Node` is subscribed to its MAC and ID topics, as per Section 4.3. This state stores a reference to a BLE Peripheral and can deliver `Messages` by generating a BLE packet that is sent over the `Node Rx` Characteristic, as per Section 4.2.

[10] `https://developer.android.com/jetpack/`, accessed on 4th Sep, 2020

MQTT used by remote `Nodes` when the smartphone is connected to the Broker and the `Node` is not connected via BLE. This state delivers `Messages` by publishing them to the Broker.

Disconnected used by `Nodes` found in a BLE scan that are not connected, or by remote `Nodes` if the MQTT connection dies.

To deliver a Message, `Crowdnet` tries to find the corresponding destination `Node` instance by MAC or ID. If one is not found, it is assumed to be a remote Node, and a new `Node` instance is created and transitioned to the MQTT state. Once `Crowdnet` finds the `Node` instance, its `deliver(Msg)` method is called, which delegates the Message delivery process to the current Node State. Then, depending on the State, the Message is delivered via BLE or MQTT. The advantage of this delegation using the Strategy pattern is that the delivery algorithm is automatically selected depending on the current state of the Node.

To receive Messages from remote Nodes, the `MQTT` Node State first subscribes to topics for the corresponding Node, as explained in Section 4.3. Since it is the Node class itself that receives callbacks for these topics, in this case there is no need for intervention from the `Crowdnet` class: when a Message is published to the topic, the Broker notifies the smartphone and a callback is received by the corresponding `Node` instance, which delivers the Message using its own state.

5. Evaluation

This Section describes a few tests that were performed to evaluate the performance of Crowdnet. We begin by evaluating the speed throughput of Crowdnet using BLE and then both MQTT+BLE. Then the latency of Crowdnet is evaluated and the results are compared with reference values for I^2C.

5.1. Communication throughput

This section describes various tests using two separate setups to compare the throughput of Crowdnet using both BLE and MQTT, using different payload sizes. Two ESP32 Nodes were programmed to communicate with one another under different conditions. One Node is configured to send packets, while the other receives packets.

In each setup, one Node sent a burst of 9 packets. Each burst is one sample, and each sample uses a different payload size. Samples in the graphs were computed by the receiver Node by measuring the total number of bytes it received in the payloads and then dividing by the difference between the timestamp (in milliseconds) when the last and first packets were received in each burst. The value is then multiplied by 1000, resulting in

the average speed in bytes per second. Since the time measured by the receiver corresponds to the instant when the transmission of the packet ended, the payload size of the first packet is excluded from the measurements, as it was exclusively transmitted before the measurement started. Also, since the time between packets is also included in the measurement, the computed value also accounts for the packet processing time in the smartphone (and server), not only the air time of the packets.

5.1.1 BLE only

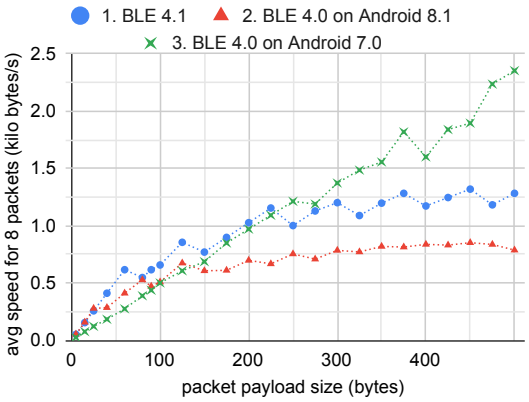
First, we tested the throughput of Crowdnet while using two local ESP32 Nodes connected to a single smartphone via BLE.

The same test was ran a few times on each of three different smartphones, using the same setup. These particular devices were chosen simply because they are the only ones we had access to:

1. an OnePlus One with Android 6.0.1, BLE 4.1;
2. an Asus ZC520TL with Android 8.1, BLE 4.0;
3. an Asus ZC520TL with Android 7.0, BLE 4.0.

The results are show in Figure 6. Each sample (dot, triangle, or cross) in the graph is the average speed of a burst of 8 packets of equal payload size, calculated as explained above.

Figure 6: Average receiving speed vs payload size for BLE



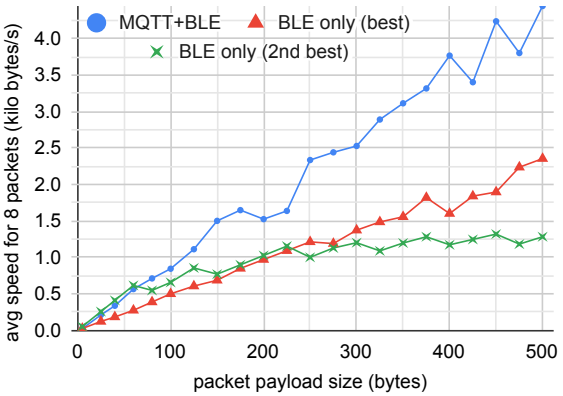
As expected, the throughput increases when using greater payload sizes. This is likely because greater packet sizes help reducing periods of radio silence, which are considered in our average calculation. Maximum measured speed was about 2353 bytes/s while using a payload size of 500 bytes per packet using phone number 3. This corresponds to a speed of about 294 kbps (kilobits per second), which is about 1.3 times faster than the maximum BLE 4.2 application layer throughput of about 221 kbps measurement by Dian et al. (2018). Note that Android does not let us change or read the connection interval value directly, so we can not be certain that the value we used matches the one used by Dian et al. (2018).

Using the other two smartphones, however, the maximum observed speed is significantly lower, and contrary to what was expected, BLE 4.1 performed worse than 4.0. It is therefore clear that, in practice, the throughput of BLE will depend on the devices that are used. This is even more evident in the two Asus ZC520TL that were tested, which, despite using the exact same hardware, achieved a very different throughput.

5.1.2 BLE and MQTT

Next we tested how Crowdnet performs when using MQTT in conjunction with BLE. In this test we used the two best performing devices (1 and 3) from the BLE-only test, each connected to its own ESP32. Results are shown in Figure 7.

Figure 7: Average Crowdnet receiving speed versus payload size for BLE and MQTT



The red line corresponds to the smartphone that better performed in the BLE-only test. The blue line is the speed over BLE+MQTT measured by connecting each Node to a different smartphone in different rooms. Smartphones were connected to an MQTT Broker running on the same LAN via the same Wi-Fi AP. These two setups correspond, respectively, to use cases A) and B) in Figure 3.

As expected, when using MQTT+BLE, the maximum achieved speed is about 2–3 times greater than when using only BLE.

5.2. Communication latency

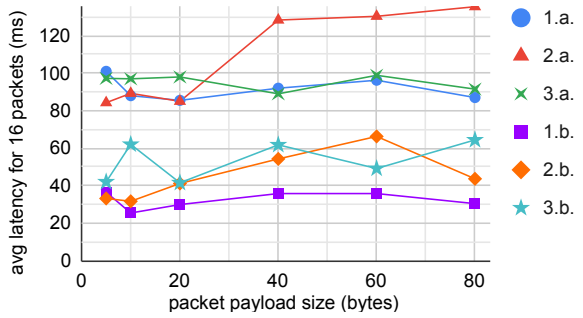
This section compares the latency of Crowdnet while communicating via BLE. In order to accurately measure time intervals, an accurate and synchronized clock is needed. Since it is not trivial to synchronize clocks in multiple Nodes, only one Node was used in these tests.

A single ESP32 Node was programmed to send packets destined to itself via Crowdnet, and was

connected to a single smartphone via BLE. Latency values for each packet include the total air time of the packet for two round trips, and processing time on the smartphone and Arduino.

For each payload size that was tested, 16 packets were sent. Each test was repeated on each of the three smartphones used to evaluate the throughput of BLE in Section 5.1.1, respectively numbered 1–3. For each smartphone, the test was repeated using different connection priority values on Android: (a) `CONNECTION_PRIORITY_BALANCED`, which uses the connection parameters recommended by the Bluetooth Special Interest Group (SIG); (b) `CONNECTION_PRIORITY_HIGH`, which requests a low latency connection.

Figure 8: Average Crowdnet latency vs payload size using BLE



Results are shown in Figure 8. As expected, the latency is reduced when using a high connection priority. The minimum observed latency value was 25 ms using a payload size of 10 bytes on the OnePlus One (1.b). However, just like in the BLE throughput tests in Section 5.1.1, results start to show significant variations between devices when larger payload sizes are used.

From these tests we can conclude that, in practice, the expected latency between Nodes using BLE will be in the range of 25 to about 136 milliseconds, depending on the smartphone, the connection priority, and the payload size that is used. These values are well above what is expected of an I^2C bus, which, at 400 kHz, is able to reach a maximum theoretical throughput of 316 kbps and, using 10 byte packets, a latency of 0.5 milliseconds. Crowdnet, though, using the same payload size, only achieved a throughput of 96 bytes/s (768 bps) when using MQTT+BLE, and a much higher round-trip latency of 25 milliseconds.

However, the performance offered by Crowdnet is still suitable for the use cases for which Crowdnet is intended, and the conveniences offered by the system when compared to I^2C are a trade-off that users must consider.

6. Conclusion

This work introduced a middleware for communication between Arduino boards using BLE and Internet connectivity of smartphones. It enables communication between both local and remote Nodes, allowing students to work in and out of the laboratory, as well as from their homes. The Crowdnet Arduino library API is similar to that of I^2C, making it simple to use. It was implemented and evaluated using the ESP32 SoC, an Android smartphone App, and the Mosquitto MQTT Broker.

Evaluation showed Crowdnet can achieve reasonable speeds and latency, even though results vary considerably depending on the smartphone that is used. Nonetheless, the achieved throughput is adequate for the use cases for which Crowdnet is intended, and using MQTT in conjunction with BLE provides greater throughputs.

Future improvements to Crowdnet include porting the Arduino library to additional BLE shields on Arduino boards other than the ESP32, and adding more features to the Android App, such as the ability to send sensor or input data to the Arduino.

References

Cunha, A. R. (November 2017). Software for Embedded Systems Laboratory Guide. Lisbon. Instituto Superior Técnico.

Dian, F. J., A. Yousefi, and S. Lim (2018). A practical study on Bluetooth Low Energy (BLE) throughput. In *2018 IEEE 9th Annu. Inf. Technol. Electron. Mob. Commun. Conf. (IEMCON), Inf. Technol. Electron. Mob. Commun. Conf. (IEMCON), 2018 IEEE 9th Annu.*, pp. 768–771.

Gomez, C., J. Oller, and J. Paradells (2012). Overview and evaluation of bluetooth low energy: An emerging low-power wireless technology. *Sensors (Switzerland) 12*(9), 11734–11753.

Lee, J.-s., Y.-w. Su, and C.-c. Shen (2007). A Comparative Study of Wireless Protocols: Bluetooth, UWB, ZigBee, and Wi-Fi. In *IECON 2007 - 33rd Annu. Conf. IEEE Ind. Electron. Soc.*, pp. 46–51. IEEE.

Leens, F. (2009, 02). An introduction to I2C and SPI protocols. *Instrumentation & Measurement Magazine, IEEE 12*(1), 8–13.

Light, R. A. (2017). Mosquitto: server and client implementation of the MQTT protocol. *Journal of Open Source Software 2*(13), 265.

Maier, A., A. Sharp, and Y. Vagapov (2017, 09). Comparative analysis and practical implementation of the ESP32 microcontroller module for the Internet of Things. In *2017 Internet Technologies and Applications (ITA)*, pp. 143–148. IEEE.

Rahiem, M. D. H. (2020, June). The Emergency Remote Learning Experience of University Students in Indonesia amidst the COVID-19 Crisis. *International Journal of Learning, Teaching and Educational Research 19*(6), 1–26.

Yassein, M. B., M. Q. Shatnawi, S. Aljwarneh, and R. Al-Hatmi (2017, May). Internet of Things: Survey and open issues of MQTT protocol. In *2017 International Conference on Engineering MIS (ICEMIS)*, pp. 1–6. IEEE.

Yokotani, T. and Y. Sasaki (2016, 09). Comparison with HTTP and MQTT on required network resources for IoT. In *2016 International Conference on Control, Electronics, Renewable Energy and Communications (ICCEREC)*, pp. 1–6.