Hermes: a Wireless Communication Interface for Edge Computing

Davide Carnemolla[1], Fabrizio Messina[1], Corrado Santoro[1] and Federico Fausto Santoro[1]

[1]*Department of Mathematics and Computer Science, University of Catania*

Abstract

Using wireless communication technologies and protocols in embedded systems is not a trivial undertaking, since it frequently requires an exhaustive understanding of protocols and APIs. The goal of this work is to address the issue by providing a unified interface (API) for developers to utilise various communication protocols in a black-box manner. This interface is provided by Hermes, the library we developed. Hermes uses the object-oriented programming paradigm and it is written entirely in C++ using the Arduino libraries. The library supports ESP-NOW for short-range wireless communications, as well as WiFi Mesh and LoRa Mesh for mesh network construction, with the first one supporting point-to-point communications over small distances and the second one supporting long-range communications. Although this library is useful in any context, it has been specifically designed to enable distributed communication between the agents of Democle, a declarative multi-agent platform for agent-based edge computing, which we will briefly describe. This paper presents an overview of Hermes, including its architecture, capabilities, and utility inside the Democle platform.

Keywords

multi-agents, edge computing, internet of things, networks

1. Introduction

The Internet of Things (IoT) has transformed the digital world, introducing novel concepts and technologies into our daily lives. Such innovations have become an integral part of our everyday lives, both at home and at work, and have contributed to the improvement of the quality of life for many people. One of the most fascinating aspects of such technologies is network communication since these devices are primarily used to gather data using suitable sensors or to give services through novel modes of engagement. Given the nature of the applications, research in the field focuses on reducing energy consumption for transmissions, as well as developing self-managed networks and attaining long-range communication.

In light of these considerations, several network technologies and protocols have been developed to address these issues to the specific requirements of the intended usage scenario. Unfortunately, developers are often required to possess a comprehensive understanding of such protocols to utilise them effectively. Furthermore, if it is determined through empirical testing that the technology is unsuitable for a particular purpose, it is necessary to rewrite the code.

WOA 2024: 25th Workshop "From Objects to Agents", July 8-10, 2024, Forte di Bard (AO), Italy

davide.carnemolla@unict.it (D. Carnemolla); fabrizio.messina@unict.it (F. Messina); corrado.santoro@unict.it (C. Santoro); federico.santoro@unict.it (F. F. Santoro)

0009-0001-2575-0874 (D. Carnemolla); 0000-0002-3685-3879 (F. Messina); 0000-0003-1430-5676 (F. F. Santoro)

In this paper, we put forth the concept of a library, called Hermes, which is designed to provide a unified application programming interface (API), thereby resolving the problem in question. The protocols supported by Hermes include ESP NOW, WiFi Mesh and LoRa Mesh, which collectively permit the library to address the majority of potential usage scenarios. Later, in Section 4, we will give a summary of the features of these protocols and their operational context.

Another outcome that we will briefly discuss is the integration of Hermes into software agents and multi-agent systems (MAS) to develop autonomous distributed systems. In particular, we integrated our library within the Democle [1] platform, a multi-agent system for embedded systems written in C++ that employs a logic/declarative programming approach.

2. Related work

In [2] authors discuss the development of an open-source gateway that supports multiple protocols for the IoT. The gateway is designed to facilitate seamless communication between devices and applications in IoT systems, which often involve heterogeneous devices and protocols. The article highlights the challenges of integrating devices with different communication protocols in IoT systems. It emphasizes the need for a gateway that can convert between different protocols, ensuring interoperability and facilitating the exchange of data between devices and applications. The authors of the article propose an open-source framework for the development of such a gateway. The framework is designed to be highly customisable, allowing developers to adapt it to specific IoT applications and use cases. The framework includes a set of APIs and tools that enable developers to integrate different protocols and devices into the gateway. The article also discusses the benefits of using an open-source gateway in IoT systems. These benefits include increased flexibility, reduced costs, and improved scalability. The authors argue that an open-source gateway can be easily modified and extended to meet the specific needs of different IoT applications, making it a valuable tool for developers and researchers working in the field.

The authors of [3] present MINOS, a multi-protocol SDN platform designed to address the challenges of IoT networks such as elasticity, heterogeneity, and mobility. MINOS is introduced as an SDN platform that integrates multiple protocols for IoT environments, enabling service awareness and logically centralized network control. It provides a programmable interface for configuring protocols on demand and a GUI for real-time visualization. MINOS supports experimentation with network control features and protocols, optimizing routing and adapting to dynamic network conditions. MINOS implements an SDN-based architecture that decouples the data and control planes, thus simplifying network management and enhancing performance. Experimental results show that MINOS improves packet delivery ratios with minimal control overhead compared to standard IoT protocols, demonstrating its effectiveness in diverse and dynamic IoT environments.

In [4], authors introduce a system designed to optimize data transmission for IoT healthcare devices by balancing data rate, transmission range, and power consumption. he system integrates WiFi, Bluetooth, and LoRaWAN protocols, offering configurable connectivity for personal and wearable healthcare devices. It includes an optimized protocol (E-LoRaWAN) to extend network

range through multi-hop communication. This system aims to provide continuous operation for devices that track users indoors and outdoors, significantly outperforming similar systems in energy efficiency and communication range.

A similar approach is proposed in [5], where the proposed system addresses the conflict of wireless signals in environments with multiple medical IoT devices. It prioritizes communication based on device necessity and separates processes to enhance the efficiency of ISM band usage. The system operates without altering the PHY layer, focusing on the MAC and upper layers.

3. Overview of Democle

This section reports a short overview of the Democle framework.

Democle is a C++ multi-agent framework that allows a developer to write multi-agent systems using a declarative approach. Democle is based on the BDI paradigm and lets developer to write agent's behaviour by means of *plans* triggered by events and subject to a condition based on the agent's knowledge. An agent in Democle is an instance of a subclass of the basic **Agent** class whose **run()** method is overridden to implement the specific agent's behaviour. Moreover, the agent has a knowledge-based (KB) which stores agent's knowledge in terms of *beliefs*, which are defined using atomic formulas with zero or more ground terms. *Plans* in Democle are expressed using a specific syntax and are made of the following parts:

- **triggering event:** it is the assertion or retraction of a belief with a given pattern, or a definition of a procedure that can be explicitly called;
- **condition:** it is a predicate checking the presence in the KB of one or more beliefs with given parameters; here variable unification is also possible;
- **action:** it is the code that is executed when the triggering event occurs and the condition is met.

Figure 1 reports the code of an agent running on an embedded device that samples data from an analog to digital converter and applies a low-pass filter according to a parameter (i.e. alpha); data is then processed by another piece of code (not shown in the example).

The main reason tied to the development of Democle is that, since it uses C++, it can run not only standard PCs but also in boards equippend with MCUs (in particular it has been tested in ESP32 and STM32 environments), making it perfect for IoT agent-based application; its memory footprint is small and, when it runs in an MCUs, additional features are enabled such as the interface with the peripherals (GPIO, timers, etc.) thus making it possible to trigger plans following the occurrence of events in a specific peripheral.

The only feature missing in the original design of Democle is the possibility of exchanging messages among agents running in different devices, a feature that is now added by Hermes that is described in the paper.

For a comprehensive description of the features and the architecture of Democle, we refer the reader to the original paper[1].

```
#include "democle.h"

belief(data);
singleton(alpha);
reactor(adc);

class ADCSampler : public Agent {
public:
    ADCSampler() : Agent("adc_sampler") {};
    void run() {
        var(T); var(H); var(A);
        AnalogInputSampler * as = new AnalogInputSampler("A0", A0, -55.0, 150.0, 3600);
        attach(as);
        +adc("A0", T) / (data(X) & alpha(A)) >> [T,X,A](Context & c) {
            float current = c[T];
            float prev = c[X];
            float alph = c[A];
            current = prev * alph + current * (1. - alph);
            c - data(X);
            c + data(current);
        };
        +adc("A0", T) >> [T](Context & c) {
            float current = c[T];
            c + data(current);
        };
        +data(X) >> [X](Context & c) {
            // process data
            ...
        };
    };
};
```

Figure 1: Example of an Agent in Democle

4. Architecture and Features of Hermes

Hermes is a library for embedded systems whose purpose is to provide a single interface for network communication supporting different transmission protocols. The main objective of our project is to provide a straightforward interface for wireless network protocols, allowing the developers to utilise them in a black-box mode. Hermes supports ESP NOW[6], WiFI Mesh[7, 8] and LoRa Mesh protocols[9, 10], which allow us to cover the most common scenarios of use.

Furthermore, to facilitate the communication between the nodes in the network, we have included a name management service in Hermes. During the initialization phase, a name is associated with each node of the network. As we discuss later in this paper, this service simply allows each node to send data to another node without knowing its network address.

In the following subsection, we will first describe the software architecture of Hermes and then its integration with the Democle platform.

4.1. Hermes Architecture

The Hermes library is written in an object-oriented way using the C++ language[11] and the Arduino libraries[1]. The design of Hermes is made by a few classes which, in fact, represent its object-based architecture, which is shown in Figure 2.

The main class of the project is called **Hermes**. This class models a generic network protocol and, for this reason, it is implemented as an abstract class. This class defines the attributes and methods to be implemented by the subclasses. Each instance of **Hermes** has a name, which represents the name of the running node, a **NameService**, which is used to store and retrieve the network addresses associated with a node name, and a **MessageQueue**, which is used to store the messages that cannot be sent at that time. Finally, there is a **hermes_recv_cb_t** within the class that allows us to define the desired behaviour of our device in response to a received message.

Each subclass of Hermes represents the implementation of a specific network protocol. At present, as shown in Figure 2, we have implemented three subclasses: HermesEspNow, HermesWiFiMesh and HermesLora, which are the supported protocols by Hermes. In the following paragraphs, we explain the main features of the protocols and other details on Hermes architecture.

ESP NOW It is a connectionless wireless communication protocol designed by Espressif[6]. The maximum payload size is 250 bytes and the default bit-rate is 1Mbps. ESP-NOW has been designed for point-to-point and point-to-multipoint communication; however, it cannot be used to build complex network topologies. Due to the characteristics described above and its simplicity, this protocol is mainly used in smart lights, remote controls and sensors. We used the official library in the Espressif IoT Development Framework (ESP-IDF) [12] to implement this protocol.

WiFi Mesh [8] It is a WiFi networking protocol designed to support the construction of a Wireless Local-Area Network (WLAN) having a large number of nodes across a wide physical area. A WiFi mesh network holds a few important properties, such as self-organization and self-healing, in other words, it is capable of tuning up its configuration over time. Furthermore, this protocol allows nodes in the network to communicate with the Wide Area Network (WAN) through a router node. Since we intend to implement a library for internal communication between nodes we have chosen to use a straightforward topology with a fixed root node and without a router. Here too, we used the Espressif IoT Development Framework (ESP-IDF) [12] to implement this protocol. The reader may refer to the official Espressif documentation for further details.

LoRa Mesh [9, 10] It is a widely used technology for low-power and long-range communications. LoRa's communication range can be up to 5 km in urban areas and 15 km in rural areas. In our work, we used this technology to allow mesh network realization through the usage of the LoRa Mesher library [9] which, in turn, makes use of the well-known RadioLib library [13]

[1]https://www.arduino.cc/reference/en/libraries/

to interact with the LoRa radio chip. The list of supported radio modules is available on github[2]. This library uses a proactive distance-vector routing protocol as proposed in [14].

Message It is the class that represents messages in Hermes. The attributes of the class are the following:

- `type`: the type of message which is set to `DEFAULT_MESSAGE` in the case of ordinary messages, `WHO_IS_REQUEST` for node name resolution requests;
- `buffer`: a buffer of bytes containing the message data;
- `size`: the buffer size in bytes.

In addition, there is a simple data serialisation function, which is necessary for sending and receiving methods.

MessageQueue This class is used to store messages that cannot be sent at a particular time because the NameService does not know the address of the receiving node. These messages are placed in a queue associated with the recipient's name and are processed by the NameServer when it receives the address associated with that node.

NameService It is the class which manages node names in the Hermes network, regardless of the chosen protocol. This class has a name-address map, which records the known address by the node. This map is updated whenever we send a message to an unknown node. In particular, when we execute the send function to an unknown node (i.e., not existing in the map), Hermes broadcasts a `WHOIS_REQUEST_MESSAGE` containing the node's name and waits asynchronously, with an *ad hoc* task, to receive a `WHOIS_REPLY_MESSAGE` from the target node. This message is sent in response to the `WHOIS_REQUEST_MESSAGE` directly from the node with that name. Furthermore, once the node address has been saved, the NameService takes care of sending the messages in the message queue for that node, emptying it as it goes.

Obviously, this approach works in a legitimate context, i.e. one that has no malicious nodes. Without this assumption, we would have to consider the usage of some authentication mechanism.

Tests Hermes has been tested on several devices including ESP32-S3, ESP32-C6 and TTGO-LoRa32-V2.1 (with a SX1276 LoRa module) using the PlatformIO tool.

4.2. Democle Integration

As mentioned in Section 1, Hermes has been implemented as a stand-alone library, but this was primarily designed to provide a communication module for the Democle platform introduced in Section 3. In the same section, we discussed how the platform implements the agent communication feature and stated that communication is constrained to the agents present on the running device. We solved this limitation by integrating Hermes into Democle and, as a

[2]https://github.com/jgromes/RadioLib.

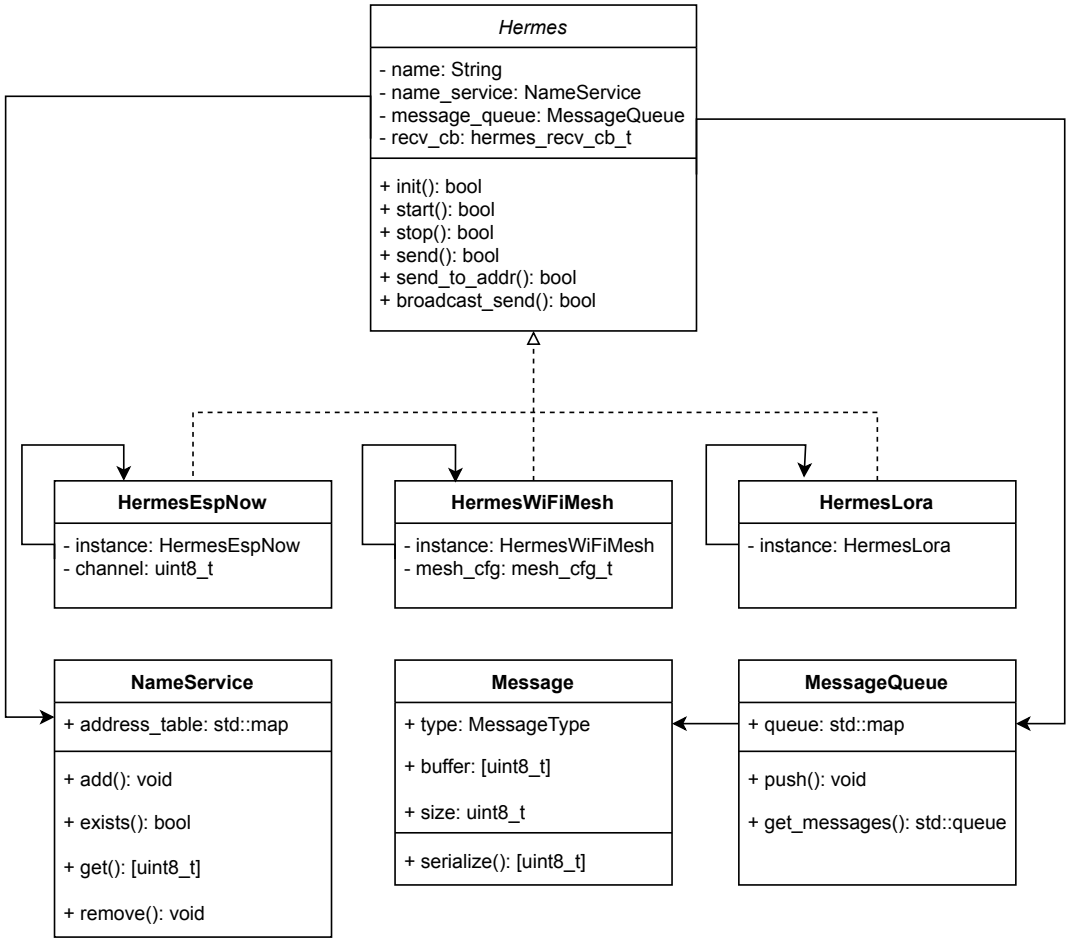

Figure 2: Hermes Architecture

result, expanded the platform's usage scenarios. We can imagine a large number of nodes with multiple running agents cooperating over short or long distances to achieve a shared goal.

To accomplish this, we had to slightly modify the Democle syntax for sending messages. Specifically, it is now required to indicate the protocol to be utilized, the name of the node on which the agent is running and the agent's name in the destination variable using the following conventions:

- `local@agent_name` when the destination agent is running on the same node;
- `espnow:node_name@agent_name` to send a message to an agent running on another node, using EspNow;
- `wifimesh:node_name@agent_name` to send a message to an agent running on another node, using WiFi Mesh;
- `loramesh:node_name@agent_name` to send a message to an agent running on another node, using LoRa Mesh.

It is important to note that Hermes must be properly initialized before being used on the Democle platform by calling the init function for the chosen communication protocol.

5. Conclusions

This work has introduced Hermes, a C++ library which allows developers to send and receive messages using different wireless network protocols through a single library interface.

The project is currently still under development and the introduction of some new features is planned, e.g. encryption, authentication and compression of the messages.

Hermes is an open-source project released under the GNU-GPLv3 licence and available on Github at https://github.com/Herbrant/Hermes/.

Acknowledgments

*(i)*This work is supported by the MIUR project "T-LADIES" under grant PRIN 2020TL3X8X. *(ii)* The contribution of Federico Fausto Santoro was supported by MUR under Mission 4, Component 2, Investment 1.4 under the project HPC. *(iii)* "PIACERI", funded by the University of Catania;

References

[1] F. Messina, C. Santoro, F. F. Santoro, A declarative C++ agent platform for agent-based edge computing, in: WOA, volume 3579 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2023, pp. 206–215.

[2] B. Oniga, A. Munteanu, V. Dadarlat, Open-source multi-protocol gateway for internet of things, in: 2018 17th RoEduNet Conference: Networking in Education and Research (RoEduNet), 2018, pp. 1–6. doi:10.1109/ROEDUNET.2018.8514136.

[3] T. Theodorou, G. Violettas, P. Valsamas, S. Petridou, L. Mamatas, A multi-protocol software-defined networking solution for the internet of things, IEEE Communications Magazine 57 (2019) 42–48. doi:10.1109/MCOM.001.1900056.

[4] T. Polonelli, D. Brunelli, A. Girolami, G. N. Demmi, L. Benini, A multi-protocol system for configurable data streaming on iot healthcare devices, in: 2019 IEEE 8th International Workshop on Advances in Sensors and Interfaces (IWASI), 2019, pp. 112–117. doi:10.1109/IWASI.2019.8791381.

[5] X. Wang, J. T. Wang, X. Zhang, J. Song, A multiple communication standards compatible iot system for medical usage, in: 2013 IEEE Faible Tension Faible Consommation, 2013, pp. 1–4. doi:10.1109/FTFC.2013.6577775.

[6] R. Pasic, I. Kuzmanov, K. Atanasovski, Esp-now communication protocol with esp32, Journal of Universal Excellence 6 (2021) 53–60.

[7] M. Buffa, F. Messina, C. Santoro, F. F. Santoro, Design of self-organizing protocol for lowpan networks, in: Internet and Distributed Computing Systems, Cham, 2019, pp. 424–433.

[8] C. K. Toh, Ad hoc mobile wireless networks: protocols and systems, Pearson Education, 2001.

[9] J. M. Solé, R. P. Centelles, F. Freitag, R. Meseguer, Implementation of a lora mesh library, IEEE Access 10 (2022) 113158–113171. doi:10.1109/ACCESS.2022.3217215.
[10] S. Devalal, A. Karthikeyan, Lora technology-an overview, in: 2018 second international conference on electronics, communication and aerospace technology (ICECA), IEEE, 2018, pp. 284–290.
[11] B. Stroustrup, The C++ programming language, Pearson Education, 2013.
[12] espressif, esp-idf: Espressif IoT Development Framework. Official development framework for Espressif SoCs., https://github.com/espressif/esp-idf, 2016.
[13] jgromes, RadioLib: Universal wireless communication library for embedded devices, https://github.com/jgromes/RadioLib, 2018.
[14] R. Pueyo Centelles, Towards LoRa mesh networks for the IoT, Ph.D. thesis, Universitat Politècnica de Catalunya, 2021.