Received 9 July 2024, accepted 22 July 2024, date of publication 29 July 2024, date of current version 7 August 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3434747

APPLIED RESEARCH

Environment Monitoring Subsystem Based on Recursive InterNetwork Architecture

DAVID SARABIA-JÁCOME[1], FRANÇOIS-DENIS GONTHIER[2], STEVE BUNCH[2], GEORGIOS SIACHAMIS[3], EDUARD GRASA[1], MARISA CATALÁN[1], AND GEORGIOS STAVROPOULOS[3]

[1]Internet of Things Research Group, Fundació i2CAT, 08034 Barcelona, Spain
[2]TRIA Network Systems, LLC, Orlando, FL 32804, USA
[3]Information Technologies Institute, Centre for Research and Technology Hellas (CERTH), 570 01 Thessaloniki, Greece

Corresponding author: David Sarabia-Jácome (david.sarabia@i2cat.net)

This work was supported in part by Departament de Recerca de Universitats de la Generalitat de Catalunya under Grant 2021 SGR 01524, and in part by European Union's "Horizon 2020" Research and Innovation Program as a part of the TERMINET Project under Grant 957406.

ABSTRACT The Internet of Things (IoT) facilitates intelligent building management by deploying IoT devices as crucial components of building subsystems. However, the ongoing evolution of intelligent buildings presents challenges in network management, scalability, and security. In this regard, smart buildings require innovative network architectures capable of adapting to meet future requirements. The Recursive InterNetwork Architecture (RINA) emerges as a clean slate network architecture aiming to overcome various current network limitations and simplify network complexity. While RINA has demonstrated several advantages in multi-homing, scalability, and security, these benefits must be analyzed in realistic conditions. This paper introduces a RINA-based environment monitoring subsystem that implements and deploys RINA components within specific IoT hardware and software in a relevant environment. The subsystem comprises several RINA sensors, RINA-based IoT gateways, and an edge node. The subsystem's design and implementation details are provided for efficient and secure communication between RINA sensors and the edge node. The RINAsense architecture was used to implement the sensors, with additional enhancements for resource management and energy efficiency. Also, the IRATI open-source software facilitates the implementation of the IoT gateway and edge node to ensure smooth RINA communications within the subsystem. The subsystem was evaluated in terms of latency, goodput, and energy consumption, and it achieved a 2.05-millisecond delay, 0.86 Mbps as goodput, and reduced 82% energy consumption. Finally, the feasibility of using RINA in smart buildings was confirmed by integrating the proposed subsystem into an operative intelligent building system.

INDEX TERMS Future networks, Internet of Things (IoT), recursive InterNetwork Architecture (RINA), smart buildings systems, smart sensors.

I. INTRODUCTION

Smart buildings are designed to maximize efficiency, reduce operational costs, and improve the comfort and safety of their occupants by optimizing available resources. To accomplish this purpose, smart buildings rely on various technologies such as sensors, networks, data analytics, and AI to collect and analyze valuable data [1]. Automation in smart buildings includes subsystems such as lighting, Heating Ventilation Air Conditioning (HVAC), security, and energy management, all equipped with sensors and devices to monitor and control the building environment. The real-time data generated by the sensors and devices are analyzed to make automatic adjustments that ensure optimal performance and energy efficiency [2]. However, smart buildings constantly evolve by integrating new heterogeneous technologies, communication protocols, and automation systems. The expansion of the smart buildings concept introduces challenges such

The associate editor coordinating the review of this manuscript and approving it for publication was Maurizio Casoni.

as network management complexity, scalability, interoperability, and security. Consequently, smart buildings require innovative network architectures capable of accommodating future requirements while providing flexibility [3].

Future network architectures have been proposed to solve traditional networking issues. Software-defined Networking (SDN) and virtual network function (VNF) are fundamental technologies for modeling the next generation of Internet of Things (IoT) network architectures. Separating the control and data plane has resulted in more efficient networks [4]. SDN technology has enhanced smart buildings' network management, network performance, bandwidth efficiency, and quality of service (QoS) [5]. Additionally, smart building energy management can be achieved by adequately provisioning VNFs to support distributed automation and orchestration on IoT devices [6]. Although SDN technology has provided new ways to simplify network management complexity and improve network flexibility, some open challenges in current literature (e.g., scalability, security, or reliability) require attention [4], [7].

A few years ago, Recursive InterNetwork Architecture (RINA) emerged as a revolutionary paradigm to overcome the limitations of traditional networks based on the Transmission Control Protocol (TCP)/Internet Protocol (IP) stack. RINA provides a robust, scalable, and flexible framework for interconnecting computer networks, as evidenced by several studies [8], [9], [10], [11]. RINA is based on a unique programmable layer, which is called Distributed Inter-Process Communication Facility (DIF). The DIF layer is simpler and more affordable than comparable IP solutions [12]. The most relevant aspect of RINA is its recursion, allowing for the repetition of the DIF layer to meet the evolving requirements of networks and using a uniform interface model [13]. This recursive nature ensures that the network can scale up or down based on the building's evolving needs and available communication facilities and networks. In addition, each layer's properties (e.g., routing, QoS schemes, and security, among others) can be programmed by defining policies specific to the requirements of the DIF and potentially different from other DIFs. As a result, each layer can have appropriate security mechanisms, and communication between layers can be controlled and authenticated. This is particularly crucial in smart buildings, where security and privacy are essential to protect sensitive data and ensure the integrity of critical systems, as well as the integration with pre-existing security mechanisms. RINA can potentially reduce the network complexity of the smart building network and provide a flexible framework to improve scalability, security, traffic differentiation, and performance. Although these benefits have been evaluated and validated in experimental and generic application environments, they have yet to be analyzed in a specific context.

The main motivation of this work is to apply RINA principles in a specific use case and implement the RINA components within specific hardware (e.g., sensors and IoT platforms). Toward this goal, this work proposes an innovative environment monitoring subsystem based on RINA for improving scalability within smart building environments. The system comprises several RINA-based sensors, an RINA-based IoT gateway, and an edge node. The RINA sensor has been implemented using the ESP-32 System on a Chip (SoC) IoT platform and the RINAsense software developed previously in [14]. Additionally, the RINA-based IoT gateway has been deployed on a Raspberry Pi 4 using the IRATI software for prototyping Linux-based RINA nodes [15]. The system has been deployed and seamlessly integrated into Centre for Research and Technology Hellas (CERTH)'s smart building infrastructure as part of the TERMINET project [16]. Following this implementation, an evaluation of the performance of RINA devices has been conducted, along with an analysis of their respective advantages and limitations.

To sum up, our contributions are:

- Design and implementation details of the RINA network for smart buildings (sections III, IV).
- Smart building's network efficient communications regarding latency management, data transmission effectiveness, and protocol efficiency through the RINA-based resource optimization, protocol optimization, and minimizing overhead (subsection III-C).
- Smart building's network secure communications through the RINA-based secure policies and its principle of secure by design architecture in where a self-contained security layer operates independently, eliminating the need for additional, specialized protocols or equipment (subsubsection III-B2).
- Programmable layers and flexible protocol design to accommodate the growing network requirements (subsection III-B).

The remainder of this document is organized as follows: Section II reviews the current smart building application and presents the RINA concept and its developments. Section III describes the proposed smart building climate monitoring system based on RINA. Section IV presents the system implementation and deployment. Section V discusses the results. Section VI compiles the lessons learned, and Section VII presents proposals for future work.

II. STATE OF THE ART

In our initial phase, we thoroughly examined the relevant literature concerning Smart Buildings, emphasizing environmental subsystem architectures, including communication protocols, sensors, security, scalability, and communication efficiency. Regarding theoretical fundamentals, we comprehensively examined the basic principles of RINA, which include an in-depth analysis of its advantages, as well as recent studies and advances aimed at overcoming security and scalability issues. Additionally, we delved into recent studies and advances in RINA applied to the IoT ecosystem. This study served as a basis for proposing potential solutions to transition the RINA experimental study into more mature and applied technology. This section presents the state of the art

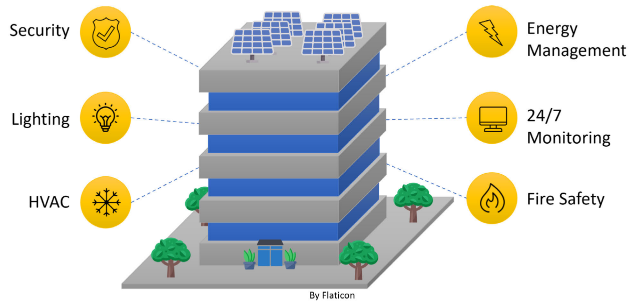


FIGURE 1. Smart building concept.

of the smart buildings concept, related works, and the main features of RINA and its recent advances in IoT domains.

A. SMART BUILDING SYSTEMS

A smart building encompasses any structure that manages its resources using automatic processes to control the building operations [17]. These buildings contribute to overall environmental sustainability goals by reducing greenhouse gas emissions, minimizing waste, and fostering efficient resource utilization [2]. A smart building relies on several subsystems to control the building's resources, such as lighting, HVAC, security, and energy management, among others, to achieve the goals above. Figure 1 depicts a high-level smart building concept and its subsystems.

Smart building subsystems continually advance by implementing diverse technologies, communication protocols, and automation systems. This expansion of the smart building concept is introducing network management complexity, scalability, and security challenges. Sensors and devices deployed in smart buildings utilize a wide range of application protocols and wireless technologies (e.g., Wireless Fidelity (Wi-Fi), Bluetooth Low Energy (BLE), and ZigBee) for communication [18]. In current practice, the Message Queuing Telemetry Transport (MQTT) and the Constrained Application Protocol (CoAP) serve as IoT application protocols to facilitate communication between sensor devices and the cloud. While the Datagram Transport Layer Security (DTLS) protocol is employed to ensure secure communications in cloud-based IoT architectures, it has been observed to degrade communication performance in terms of latency and throughput [19]. Additionally, an evaluation of energy consumption for DTLS and CoAP revealed opportunities for minimizing energy consumption. Consequently, this technological heterogeneity inherent in smart building deployments exacerbates network management complexity.

IoT communications also impact smart building efficiency. IoT poses specific challenges, such as hardware malfunctions, battery depletion, or wireless interference. As a result, communication within IoT-enabled smart buildings must be reliable and efficient. Al-Kadhim and Al-Raweshidy studied the potential impact of IoT communications in smart buildings [20]. Their work proposed several schemes for optimized standby route selection, desired reliability levels, reliability-based sub-channels, and data compression based on reliability. Their QoS schemes reduced the negative effect of reliability on traffic power consumption. Their contributions not only focused on improving QoS but also on efficient communications and scalability. Sauer et al. proposed a reliable system to collect data from IoT devices based on Transport Layer Security (TLS) and Hypertext Transfer Protocol (HTTP) [21]. Additionally, that work provided several examples of data treatment and visualization to ease decision-making. Ferrández-Pastor et al. studied the implications of edge computing in scalability and interoperability to improve IoT communications. Their work proposed a model based on edge and fog computing for designing and developing new services in smart building scenarios [22]. Their system provides scalability based on the edge-fog-cloud continuum architecture and QoS based on the MQTT QoS levels.

One of the most critical aspects of every smart building is to sustain a favorable and comfortable climate for residents, while being energy-efficient. Such a task is complicated, as the design must consider many factors, ranging from area climate to building characteristics [2]. Kim et al. proposed a system to monitor the building environment and control seasonal energy consumption [23]. The authors estimated the energy consumption by tracking the residents' behaviour (e.g., room occupancy, occupancy time), enabling the proposed system to take adaptive action to ensure thermal comfort. Additionally, their research explored the use of Secure Sockets Layer (SSL) for secure communications. Rico et al. proposed an adaptive sensor fusion and hybrid machine learning architecture to classify room activity states [24]. Their advancements contributed to understand human behaviour and spatial utilization patterns. Furthermore, their system improves scalability issues by providing adaptability to accommodate various rooms with minimal hardware and software requirements.

As a building infrastructure evolves, integrating new devices and sensors while ensuring seamless communication and interoperability becomes challenging. A smart building's network must maintain reliable connectivity, low latency, and high bandwidth effectively as it grows in size or complexity. Due to the potential for security vulnerabilities introduced by each new device, the sheer number of devices in smart buildings directly impacts their security. Smart building networks must be able to protect the ecosystem without compromising their efficiency or being prohibitively expensive or complex to manage. Consequently, smart buildings require innovative network architectures for addressing present challenges and anticipating future ones. Our proposed subsystem achieves these objectives based on the advantages of RINA. RINA achieves scalability through recursion [8], security by policies without adding protocols or special equipment in the network [10], efficient communication by reducing the protocol overhead [14], and QoS by providing

TABLE 1. Similar subsystems comparative.

System	Scalability	Sensors	Security	Protocol	Efficiency	QoS
[19]	-	-	DTLS	CoAP	x	x
[20]	x	-	-	-	x	x
[21]	x	DHT22, RC-5, ERS	TLS	HTTP	-	-
[22]	x	BCM4332	-	MQTT	-	x
[23]	-	DHT22, PMS7003	SSL	-	-	-
[24]	x	AM312	-	-	-	-
This work	x	DHT22, MQ7	x	CDAP	x	x

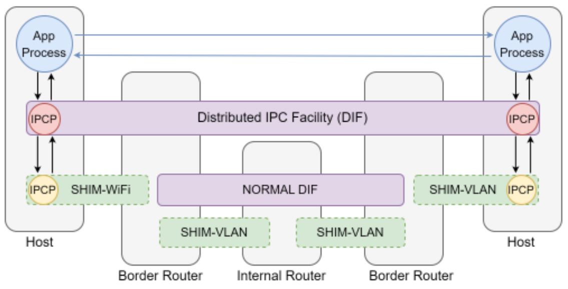


FIGURE 2. Example of a RINA network architecture.

customized QoS Cubes [25]. Table1 compares state-of-art systems and our proposal.

B. RINA-BASED NETWORKS

RINA networks rely on the idea that a computer network is embodied by a distributed application that provides distributed Inter-Process Communication (IPC) services to other distributed applications [13]. However, this networking concept is not new; it dates back to the CYCLADES project in 1970 and has its roots in RFC 61-62 [26], [27]. The aggregation of two or more communicating Inter Process Communication Processs (IPCPs) forms the basis of a DIF layer. This DIF layer can be repeated as many times as required by the network designer. For example, to isolate applications from underlying communication service specifics (e.g., protocol), abstract dissimilar transports into a unified one to present to applications, to group and protect nodes within the same security domain, or to tune policies (e.g., QoS, address length) to the needs of a group of nodes. This re-utilization of the DIF layer is the most essential aspect of RINA networks [13]. The DIF layer provides IPC services to upper DIFs or applications through a well-defined and standard Application Programming Interface (API). Compared with traditional network architectures, RINA avoids implementing redundant mechanisms and extracts as much commonality as possible (patterns) [28]. RINA facilitates network customization through policies thanks to its separation of transfer mechanisms from policies [28].

Figure 2 illustrates an example of a RINA architecture composed of a normal and shim DIFs. The shim DIF, abstracted and implemented at each node using a shim IPCP, can utilize any legacy or wire protocol available to connect to other shim IPCPs and provide the standard RINA DIF API to any DIF or application above it. This adaptability allows RINA to seamlessly operate over any legacy network or communication medium, freeing applications to use the RINA API for communication without needing to be aware of the specific legacy communication networks being used in traditional networks to RINA. Currently, several shim DIF adapters have been developed to overlay RINA on top of traditional network protocols (e.g., Wi-Fi [15], Virtual Local Area Network (VLAN) [29], and TCP/User Datagram Protocol (UDP)), further demonstrating its compatibility with legacy systems [30]. Meanwhile, the normal DIFs provide network connectivity through the well-defined RINA API [28]. The RINA API allows instances of applications or IPCPs implementing higher DIFs to request flows to other application instances or IPCPs. A normal DIF uses the services of underlying DIFs (shims or normal) to transfer data across the network.

Each DIF provides a set of two protocol frameworks: (i) data transfer through the Error Flow Control Protocol (EFCP) and (ii) layer management through the Common Distributed Application Protocol (CDAP). The components of the IPCP are grouped into three categories: (i) data transfer, (ii) data transfer control, and (iii) layer management, as shown in Figure 3.

The data transfer functionalities (sequencing, loss, and duplication detection) and data transfer control functionalities (transmission control, retransmission control, and flow control) oversee the EFCP single data transport protocol framework. To correctly manage these functionalities, the EFCP has been divided into two parts: (i) the Data Transfer Protocol (DTP) and (ii) the Data Transfer Control Protocol (DTCP). The DTP implements tightly coupled mechanisms to data transfer protocol data units (PDUs), and the DTCP implements loosely coupled mechanisms [28]. The DTP is decoupled from the DTCP through a vector state component, which decouples the high-performance DTP actions from the less-urgent DTCP actions; this decoupling enables optimizing the hardware/software implementation split if desired. DTP and DTCP functionalities are based on the Delta-t protocol proposed by Watson [31]. Upon transmission of an Service Data Unit (SDU), EFCP creates one or more data transfer PDUs and hands them to the Relaying and Multiplexing Task (RMT). The RMT is responsible for forwarding decisions to transfer data using any or all available N-1 DIF [28]. EFCP is complemented with SDU delimiting, which provides SDU fragmentation, reassembly, concatenation, separation, and SDU protection for SDU integrity and error detection [28].

The layer management tracks and models the IPCP's state and provides functions and services to execute management procedures. The IPCPs share states and layer management information with peers over the network by communicating with one another using the CDAP application protocol over a RINA flow [28]. The Resource Information Base (RIB) stores

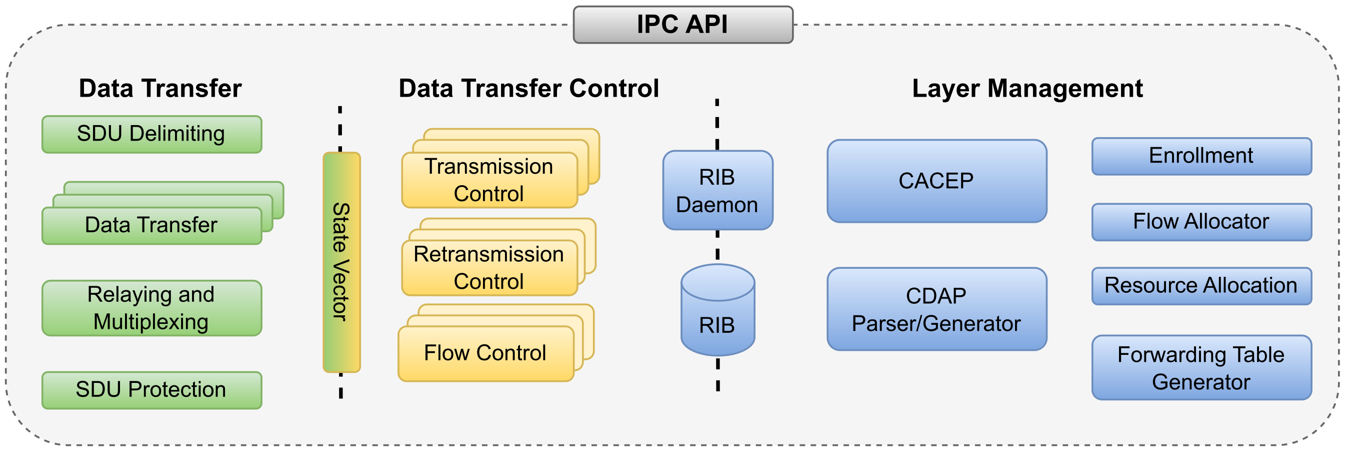


FIGURE 3. **Components of the IPC API.**

the IPCP state as objects. The RIB objects model includes object naming, relationships between objects (inheritance, containment, among others), object attributes, and the CDAP operations that can be applied to them [28]. The RIB Daemon manages access to the RIB and exchanges CDAP PDUs with the RIB Daemons of neighboring IPCPs. The only operations that can be performed on objects are create/delete, read/write, and start/stop. The IPCP implements the Common Application Connection Establishment Phase (CACEP), which exchanges naming information with peers. CDAP implements layer management functionalities such as the Enrollment Task, Flow Allocator, Resource Allocation, and Forwarding Tables Generator. The Enrollment Task is the procedure to join an IPCP to an existing DIF or another unenrolled IPCP. This enrollment process ensures the IPCP receives enough information to become a fully operational DIF member [28]. The Flow Allocator is responsible for creating and managing the flow life cycle. The Resource Allocation oversees the management of network resources (buffer space and scheduling capacity) inside the IPCP to allocate resources to flows. The Forwarding Table Generator maps the next-hop address and the IPCP port identification (portId) to facilitate the routing process. The policy implemented by the routing algorithm, which can be customized in any DIF, computes these tables. The described services and components briefly defined here are relevant to understanding the content of this work. Detailed descriptions and specifications are provided in the RINA report [28].

RINA has been widely studied, analyzed, and evaluated in several research projects. The research efforts focused on demonstrating how RINA solves the current networking challenges. Mainly, RINA has shown considerable advantages in terms of security, mobility, multi-homing, and scalability. RINA provides scalability through its recursion. From this perspective, network designers can scale their network both horizontally and vertically by creating larger DIFs or stacking DIFs on top of each other [28]. Extensive scalability tests on topological forwarding and routing policies were conducted by Gaixas et al., and that work achieved a reduction in routing communication and computational cost [8]. RINA's security approach has been analyzed and tested in the PRISTINE project, demonstrating the benefits of protecting a complete layer instead of individual protocols [32]. RINA minimizes security costs and provides flexibility and reusability of security policies [10]. Similarly, an extensive assessment of threats and security risks was conducted to identify risks in RINA components [10]. This security evaluation demonstrated that RINA provides an inherently secure environment. Additionally, RINA's mobility and multi-homing properties have been analyzed in the ARCFIRE project [33]. Mobility management in RINA is an inherent advantage derived from the complete naming scheme without specific protocols or mechanisms to support mobility. In other words, RINA does not require setting up tunnels or rewriting packet headers to support mobility; instead, it combines routing updates, changing the address of IPCP, and a proper design of DIF layers [34]. Multi-homing can be considered a particular case of mobility, as nodes can unsubscribe from one network and subscribe to another [35]. Finally, RINA reduces network complexity by providing only two protocols for data transfer and management [12] while offering flexibility to adapt the DIF layer to specific requirements through policies. This flexibility empowers network designers to tailor the DIF layer to their specific requirements, thereby simplifying network design and management [12]. Thus, RINA networks can adapt to growing network requirements and facilitate network management through policy-based dynamic service management [36].

Another remarkable project was IRATI, which aimed to implement the RINA architecture [15] for GNU/Linux-based systems. This project resulted in an open source software to implement RINA networks for experimental environments, available on the GitHub repository.[1] Rlite is a lightweight open-source implementation of RINA components focused

[1]https://github.com/IRATI/stack

on robustness and performance [37]. Moreover, ProtoRINA proposed in [38] aims to facilitate the prototyping of RINA networks in academic environments. Along with ProtoRINA, RINAsim was implemented as a framework to simulate RINA networks for educational and research purposes.

From the perspective of RINA in the IoT area, some academic research has mapped the potential advantages of implementing RINA in IoT environments. For example, Ramezanifarkhani and Teymoori analyzed the security challenges of IoT and how RINA can address them [39]. Recently, and Teymoori and Ramezanifarkhani studied the efficiency and security of IoT using RINA [40], and Neelam and Shimray demonstrated its applicability [41]. Additionally, a RINA-based architecture for IoT was designed and studied in [42]. The authors of this RINA-based architecture discussed how commonality in IoT can be achieved by introducing a RINA DIF. Recently, the first RINA implementation for embedded devices was presented in RINAsense [14]. This RINAsense implementation showed several advantages in the network regarding latency and goodput. Additionally, this work studied the overhead of IoT protocols and how RINA reduces this overhead. The application of RINA's QoS in IoT environments has recently received significant attention [43]. In that work, four RINA QoS Cubes were designed and tested to support tactile Internet, virtual reality (VR), augmented reality (AR), and IoT applications [43]. Thus, the RINAsense implementation opens the opportunity to enable testbeds and IoT devices to test RINA in IoT environments.

III. PROPOSED SUBSYSTEM ARCHITECTURE

The main objective of this work is to propose a RINA-based environmental subsystem capable of supporting multi-platform, energy- and resource-efficient RINA sensors, providing secure and protocol-efficient communications between all subsystem devices, and offering flexible and programmable layers to accommodate future smart building requirements. The adaptability of this subsystem is a key feature, ensuring it can monitor specific parameters such as temperature, humidity, and air quality and can interoperate smoothly with the current smart building subsystems. At this stage of our research methodology, we designed the architecture of the RINA-based environmental subsystem to achieve the mentioned objective. The design process defined the components, interfaces, modules, RINA protocol, and policies needed to facilitate communication and data exchange within the subsystem. The RINA principles of recursion, single programmable and isolated layer (DIF), and simple network design were mainly leveraged to achieve flexibility, scalability, and security.

The smart building subsystem comprises several RINA-based sensors, RINA-based IoT gateways, and an edge node. Figure 4 shows a high-level architecture of the smart building subsystem following the edge-fog-cloud continuum IoT architecture. The RINA sensors are based on the RINAsense architecture and its implementation described in [14]. The first version of the RINAsense architecture was developed as a proof of concept to rapidly prototype RINA-based sensors and test their performance in a controlled environment. However, this version needed to be more stable to operate in a production environment, thus requiring some improvements, detailed in subsection III-A. These improvements include resource allocator, battery manager and portability. The RINA sensors are connected to the RINA IoT Gateway using Wi-Fi and are distributed on each floor. The RINA-based IoT Gateways are built upon the IRATI open-source implementation. These RINA-based IoT gateways are connected to the edge node using an Ethernet switch. In this case, the subsystem uses a shim VLAN Ethernet DIF to facilitate the communication. The shim DIFs are connected using normal DIFs to enable scalable and secure communication between RINA nodes. More details about the design are explained in subsection III-B. The design of RINA policies to support the routing and security requirements is also detailed in subsection III-B. Finally, the subsystem's RINA network requirements guided the design of the RINA protocol to enable seamless communication between the RINA-based sensors, the RINA-based IoT gateways, and the edge node. The RINA protocol design is detailed in subsection III-C.

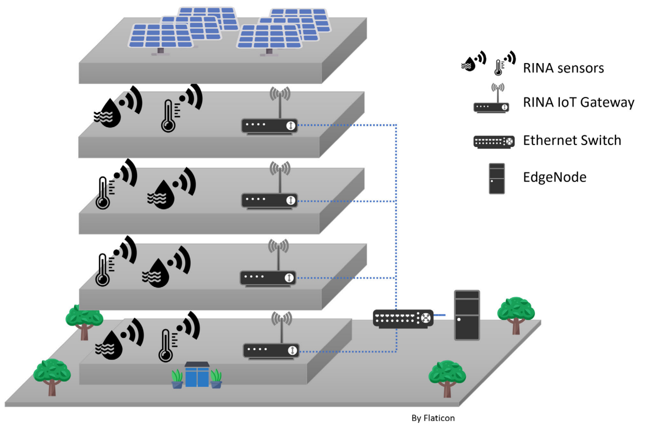


FIGURE 4. Proposed high-level architecture for environmental monitoring subsystem.

A. RINA SENSOR ARCHITECTURE

The RINAsense architecture aims to enable RINA networks to operate in embedded systems. The first instance of RINAsense architecture focused on implementing the essential IPCP components and testing them as a proof of concept. This first version was composed of the following components: EFCP, RMT, RIB, RIB Daemon, Enrollment, Flow Allocator, RINA task, shim Wi-Fi, RINA API, Identification (IDs) Manager, and the RINA timers. However, several tests in a controlled environment evidenced the necessity of a resource allocation scheme and battery management policy. These requirements were addressed in the RINAsense implementation version proposed in this work. Also, the

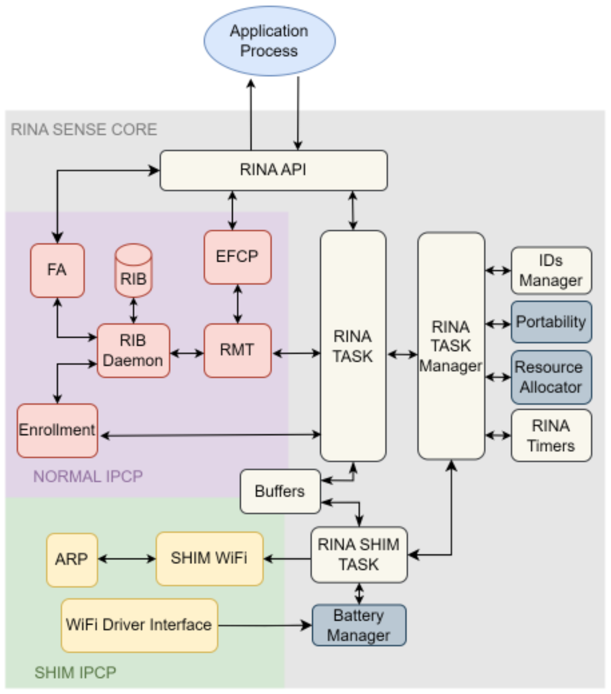


FIGURE 5. **RINAsense stack implementation.**

new version of RINAsense supports the Arduino framework thanks to its Portability layer. Figure 5 shows the components developed to overcome the limitations of the first version in dark grey. These new components are the Resource Allocator, the Battery Manager, and the Portability layer. Each component is described in the following subsubsections.

In addition to the aforementioned enhancements, we have improved the initial version by isolating the normal DIF and the shim DIF. This isolation was achieved by separating the RINA task into two tasks dedicated to each DIF. Furthermore, we have introduced the RINA task manager, which oversees both the RINA task and RINA shim task. Data buffers were also incorporated to facilitate communication between the RINA tasks. Figure 5 also includes these components. By implementing these changes, the updated version adheres to the crucial concept of DIF isolation in RINA.

1) RESOURCE ALLOCATOR

Memory allocation is a critical aspect in the operation of embedded devices, which often operate with limited memory. The careful allocation of memory ensures that applications have the necessary resources for specific uses when needed. This is particularly crucial for temporary buffers, connection state, application state, or various temporal data structures. Some usage is fixed and predictable, providing a comforting known factor in your work; some, such as memory for connection state and buffers, can be dynamic, with the instantaneous need based on the use case or activity level.

Typical heap memory allocation, such as malloc/free, allocates anonymous memory blocks. Identifying where all the memory is used when memory runs short, or performance problems appear is often complex. Managing dynamic memory usage by its purpose can be an invaluable aid to debugging (e.g., finding a memory leak), preventing over-commitment of memory to less-important usages, and tuning. RINAsense provides an allocator that allows dynamic allocation of objects of a specific type and keeping track of them. We call each such object an "rsrc", short for "resource", and the allocator the "rsrc allocator". The allocator manages this with collections, or "pools", of objects of the same type, as described below.

Using a straightforward API, each "pool" can be created as a statically allocated array of same-size objects or a pool of dynamically allocated same-size or different-size objects (different-sized objects are allocated from and returned to the heap). A pool can be fixed in size or defined to be extensible in increments up to a limit as needed. For example, if based on testing, a developer determines that a pool should be static instead of dynamic or vice versa, a one-line change to the definition makes that change without affecting any uses elsewhere in the system of that pool. Every pool knows about every object of that type that it is managing, making it possible to collect statistics per object type. If compiled in, the object maintains each resource's "user" –typically a calling function or allocation purpose– as a debugging tool.

The rsrc pool mechanism provides a per-object-type print function to simplify printing objects. The rsrc pool structures themselves are rsrc's. When the generic print functions are called on a pool's rsrc, the default short print routine prints the statistics for the pool (allocated, free, total usages, high and low counts). Especially during testing and tuning, being able to sample the instantaneous resource usage of the entire system by object type is very valuable. In addition, runaway allocation can be immediately identified, and the "user" of the resources can often point the developer directly to the malfunctioning code.

2) BATTERY MANAGER

IoT devices sometimes require a power source to operate and cannot always be connected to a power grid. In such scenarios, these IoT devices rely on alkaline and lithium-ion batteries. However, these batteries have limited capacity, and their duration depends on the demands of the use case. To address this, the battery manager was integrated into the RINAsense implementation. Achieving a modest power consumption is challenging as it relies on the behavior of IoT hardware and RINA components. To tackle this, we conducted a thorough analysis of the communication components, ensuring a robust and effective communication strategy.

Communication components such as Wi-Fi drivers or BLE drivers consume the most energy. The battery manager module aims to optimize the use of these communication components. In a sensor node, the battery manager powers off the drivers when they are not required and powers them on only when the sensor needs to send data. The battery manager was configured to take advantage of the long periods between packets in some IoT applications. DIF isolation, achieved by adding a new task for the shim DIF, facilitates the activation and deactivation of communication drivers. This ensures that the battery optimization phase, which may involve imminent destruction of the shim Wi-Fi flows, does not affect the normal DIF. The RINA shim task calls the battery manager API when the normal IPCP adds a new PDU to the buffer. Consequently, the battery manager activates the communication driver and empties the buffer by sending the PDUs. When the buffer is empty and the waiting time is over, the battery manager powers down the drivers.

3) PORTABILITY

IoT devices can be implemented using various hardware and software platforms. The choice of a specific hardware platform for a product can be constrained by cost, physical size, interfaces, power consumption, size and type(s) of memory available, supply chain issues, etc., leading to a bewildering variety of potential hardware/software combinations. These include different Central Processing Unit (CPU) architectures, vendor-supported software tools, supported standards, executive/Operative System (OS) choices, etc. In this environment, software portability techniques can help to minimize the disruption of moving to a different underlying platform or supporting multiple platforms with a common code base.

RINAsense was initially designed to work on an ESP32 board IoT platform using the Free Real-Time Operative System (FreeRTOS). It was subsequently ported to run on Linux on larger computers to facilitate application validation during application development stages. This Linux support was accomplished using a ‘‘portability’’ layer of functions that take advantage of Portable Operating System Interface (POSIX) APIs to emulate hardware control primitives on a target that supports POSIX, an interface to Linux networking to enable access to Wi-Fi, and additional support functions to eliminate some dependencies on the FreeRTOS environment such as queues and ‘‘console’’ printing. The previous cmake-based build code was enhanced to allow target selection for a build, with the target selecting the correct set of libraries and commands.

The Linux port serves several purposes. First, it enables RINAsense development to take place in a large-machine environment with powerful tools such as debuggers, Valgrind, testing frameworks, and CPU profiling available to aid development. Second, it also opens up a potential new target for RINAsense: RINA-enabled applications capable of supporting a network of RINA IoT devices on any OS that supports POSIX APIs but does not necessarily require OS-level RINA support. The hardware-platform independence layer can be extended to other Real-Time Operative System (RTOS) or hardware targets.

In addition to RTOS and CPU dependencies, IoT application developers need to manage the underlying hardware's peripheral and connectivity interfaces, as well as the devices

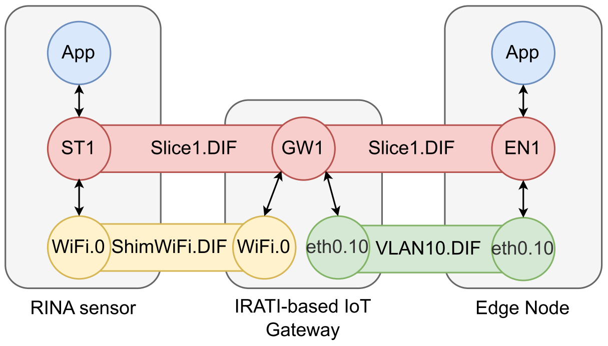


FIGURE 6. RINA standard graph of the proposed subsystem

connected to them, such as sensors and communication links. The widely-used Arduino environment provides APIs that supports various CPU and hardware targets, along with many standard hardware interfaces. Additionally, Arduino serves as a demonstration and reference target for numerous sensors and other devices commonly used in the IoT environment, making sample code readily available. To leverage this widely-used environment, RINAsense and its build process were enhanced to build RINAsense as a library and enable building Arduino ‘‘sketches’’ within the Arduino environment. Currently, this extension provides access to the existing Arduino environment for multiple ESP-32-based targets, but it could be extended to any other underlying hardware platform that supports Arduino and FreeRTOS. This improvement allows an Arduino IoT ‘‘sketch’’ to incorporate the RINA protocol using RINAsense, providing access to an extensive suite of example codes for many devices, such as sensors that are well-supported in the Arduino space. This capability is in its early stages of development —it does not yet fully support all of the RINA API or the Arduino Integrated Development Environment (IDE) (sketches are built using the cmake environment that underlies it).

B. RINA DIF DESIGN

The proposed RINA network subsystem comprises two shim DIFs and one normal DIF, as shown in Figure 6. The employed shim DIFs are the shim Wi-Fi DIF for underlying the Wi-Fi interface and the shim Ethernet VLAN 10 for the Ethernet interface. These DIFs use a switch Ethernet with support for VLANs to connect the IRATI-based IoT gateways with the Edge Node (more details about shim Wi-Fi and shim Ethernet VLANs can be found in [15] and [29]). Over these shim DIF layers is the Slice1 normal DIF. This Slice1 DIF provides the communication for the devices in the RINA network (sensors, gateways, and the edge node). The Slice1 DIF's QoS, security, and routing policies were designed to enable a scalable and secure network. These policies are described below.

1) QoS DESIGN

RINA defines a set of supported QoS models through the QoS Cubes. A QoS cube is an N-dimensional performance space with N-independent parameters (such as latency, loss, and others) [28]. A DIF can support one or more QoS cubes, allocating its resources (communication, buffering, etc.) to provide a range of QoS capability to flows; each flow is assigned to a QoS cube. Each QoS Cube has a set of associated policies designed to comply with the range supported by the cube. Such policies mainly deal with flow control, forwarding, scheduling, and congestion management [28]. The latency and loss parameters are crucial in the design of the QoS Cubes.

In the current design, the subsystem's sensors are not time-sensitive and are tolerant of packet loss. These sensors send packets with low data sizes at frequent intervals, and data (such as temperature or humidity) do not change significantly in short intervals. As a result, for example, if a packet containing temperature data is lost, it would not significantly affect the subsystem. In this view, unreliable QoS services were selected to offer a service that is not guaranteed.

Since retransmission is not required, this QoS cube does not use the DTCP component, which oversees the transmission and retransmission control. Since this simple QoS cube does not require prioritization of traffic, the RMT was configured using the First Input First Output (FIFO) scheme as a relaying PDUs policy. Since all the Normal DIF traffic has these simple requirements, one QoS Cube was configured with unreliable QoS service and without retransmission control.

2) SECURITY DESIGN

The security manager policy defines the security configuration for authentication and SDU protection profiles, known as "authSDUProtProfiles". The authentication profiles aim to establish secure parameters for communication with all neighbor IPCPs or with the N-1 DIF. RINA provides the "PSOC-authentication-password" policy to enable device authentication using a password. This authentication profile hashes a random string with a shared secret (password). This policy ensures that only the RINA nodes with the correct password can enroll in the Slice1 DIF. Two "PSOC-authentication-password" profiles were employed to provide security in the DIF. The first was used for secure communication between IPCPs in the same DIF, and the other for secure communication with the N-1 DIF. Each profile was configured with a different shared secret (password). Additionally, this policy includes an error check (Protection) policy based on Cyclic Redundancy Check (CRC) with 32 bits of polynomial length, known as CRC32. This policy ensures the integrity of the PDU.

3) ROUTING DESIGN

The Routing policy defines how the next hop is generated and maintained. A static routing policy was configured in this case because the current version of RINAsense only supports static routing. The subsystem requires unidirectional flows to enable communication between sensors and the edge node since the objective of the subsystem is to monitor the climate and air quality inside the building in this initial implementation phase. In a second phase, other kind of sensors (motion sensors, smoke sensors, water leakage sensors, light sensors, and contact sensors) will be added to the subsystem to monitor other smart building parameters. In this context, the RINA-based IoT gateway oversees the route of the packet received through the shim Wi-Fi to the edge node via the shim Ethernet VLAN. Consequently, the RINA-based IoT gateway must know all the IPCP's addresses (sensors and node edge) to route PDUs to the correct address.

C. RINA PROTOCOL DESIGN

The RINA protocol design involves the protocol logic for exchanging state information between IPCPs. Each IPCP inserts Protocol Control Information (PCI) into each PDU on one side (sender) and strips it on the other (receiver) [28]. The following subsubsections detail all these policies and the PDU format. The PDU consists of two parts: (i) the PCI, which is understood and interpreted by the DIF, and (ii) the User-Data, which is incomprehensible to this Protocol Machine (PM) and is passed to its user.

The RINA PCI format is defined by policy, and may be different for different DIFS. The format defines a fixed set of fields, but each field can be of any length appropriate for the DIF requirements —perhaps even zero in some cases. A DIF-wide agreement on the PDU format and other constants is necessary for IPCPs to understand each other, and all IPCPs learn these constants upon enrollment. The DIF-wide protocol constants are the QoS identifier length, the port identifier length, the connection endpoint (CEP) identifier length, the sequence number length, the address length, the PCI length (lengthLength), the data rate length, the DIF Integrity, and the Maximum Packet Lifetime (MPL). All length-based constants are expected to be defined in bits. The DIF integrity defines if the PDU is encrypted, and the MPL is an integer number to define the maximum PDU lifetime. Meanwhile, the PCI structure, shown graphically in Figure 16, illustrates how these constants determine the format of the PCI.

In RINA, the data transfer constants can be set on a per-DIF basis, allowing the network designer to tune the PCI format for variables like DIF size (number of IPCPs participating), number of outstanding flows per IPCP pair, etc. This allows the designer to minimize PCI length and simplify its processing, to adopt large values for a large network, or to adopt a wider public standard set of values if such per-DIF tuning is unnecessary. For this project, the data transfer constants chosen were:

- **AddressLength**: refers to the IPCPaddress. The value assigned is 1 byte because the RINA network will not be extensive. There are no more than 256 devices in the RINA network.
- **CepIdLength**: is a data transfer instance identifier unique within the application entity. The value assigned

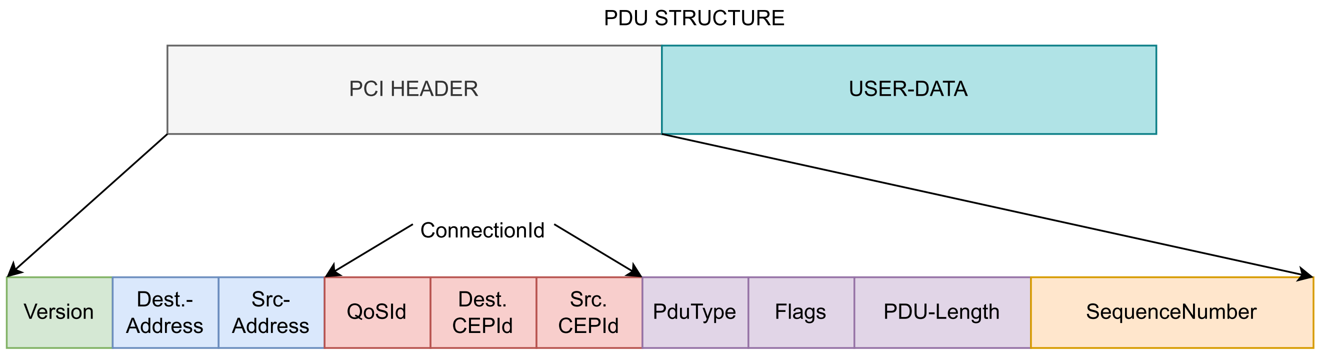


FIGURE 7. **PDU structure and protocol header PCI.**

is 1 byte because only a few flows between pairs of IPCPs will be needed.

- **PortIdLength**: Port identifier to bind the application process and the IPC flow. The PortID is internal to an IPCP. The value assigned is 1 byte as each IPCP will only support a few flows at once.
- **QoSIdLength**: identifies a QoS cube unique within the DIF. The value assigned is 1 byte because this implementation is anticipated to eventually need four QoS cubes.
- **RateLength**: refers to the maximum number of PDUs sent in a time unit. By default, 4 bytes are required to identify the maximum rate.
- **SequenceNumberLength**: PDU sequences number on a flow. By default, 4 bytes.
- **LenghtLength**: total PCI length. In this case, 4 bytes encode the total PCI length.

The PDU format indicates the standard PCI fields or PDU's headers. It is relevant to mention that the User-Data is formatted according to PDU-Type (data PDU or management PDU). Figure 16 shows the PDU structure and its standard PCI fields.

- **Version**: An identifier indicating the version of the protocol.
- **Destination-Address**: A synonym for the remote Application-Process-Name.
- **Source-Address**: A synonym for the local Application-Process-Name.
- **Connection Id**: A three-part identifier unambiguously used to identify the connection between IPCPs.
  - *QoS-Id*: The QoS Cube assigned for this connection.
  - *Destination-CEP-id*: Identifier unique within the system in which the destination IPCP resides.
  - *Source-CEP-id*: identifier unique within the system in which the source IPCP resides.
- **PDU-Type**: Identifies the type of PDU, Data Transfer PDU, DTCP PDU, or management PDU.
- **Flags**: indicates conditions that affect the specific properties of a PDU.
- **PDU-Length**: indicates the total length of the PDU in bytes.
- **Sequence-Number**: indicates the sequence number of the PDU.

IV. IMPLEMENTATION

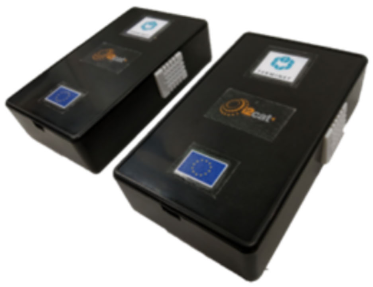
During this stage, we developed a prototype implementation of the proposed RINA-based environmental subsystem. This involved selecting appropriate hardware and software components, configuring network elements, and implementing communication protocols based on RINA principles. Additionally, we deployed the developed prototype in a relevant and controlled environment (an operational smart building). The main challenge at this stage was to integrate the RINA-based environmental subsystem with the existing infrastructure of the smart building. The proposed subsystem needed to ensure compatibility and interoperability with other building systems, such as HVAC and management and control systems.

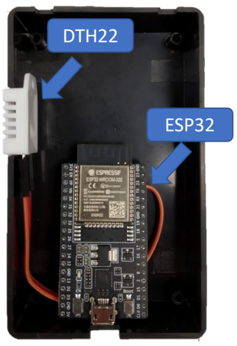
The proposed subsystem was implemented in CERTH's smart house, situated in Thessaloniki, Greece. CERTH's smart house provides an Information and Communication Technologies (ICT) infrastructure for rapid prototyping and demonstration of novel technologies [44]. CERTH's smart house features embedded energy, lighting, HVAC, flooring, and motion monitoring systems. The proposed environmental monitoring subsystem was integrated with the CERTH's systems as part of the TERMINET project pilot. The following subsections describe the software and hardware employed to implement the proposed subsystem, the deployment of the proposed subsystem in CERTH's smart house, and the integration with the existing infrastructure.

A. SOFTWARE AND HARDWARE

The RINA sensors were implemented using the RINAsense architecture, incorporating the features described in subsection III-A. The RINAsense implementation is available on GitHub[2] as open-source. Two kinds of sensors were implemented with RINAsense. The first was a temperature

[2]https://github.com/Fundacio-i2CAT/rinasense


(a) External view



(b) Internal view

FIGURE 8. Temperature and humidity RINA-based sensor.

and humidity sensor using the Espressif IoT Development Framework (ESP-IDF). The second type was a Carbon Monoxide (CO) using the Arduino framework. These sensors were designed to utilize two frameworks to evaluate the portability layer described in subsection III-A and introduced in this new RINAsense version.

The hardware components to implement the RINA sensors consisted of:

- **ESP32-WROOM-32 development board kit**: The ESP32 is a low-cost SoC system that integrates Wi-Fi and dual-mode Bluetooth. The ESP32-WROOM-32 features a Tensilica Xtensa dual-core 32-bits LX6 microprocessor, 320 KiloBytes (KB) Random Access Memory (RAM), and 4 MegaBytes (MB) flash memory. This IoT platform was selected due to its popularity among micro-controller units (MCUs). Additionally, ESP32 supports IEEE 802.15.4 in its C6 version, providing potential for the future development of additional RINA features to support IEEE 802.15.4.
- **DHT22 sensor**: a low-cost sensor to measure relative humidity and temperature.
- **MQ-7 gas sensor**: low-cost high sensitive detector of CO.

Figure 8a shows the RINA-based temperature and humidity sensor in a black box (external view), and Figure 8b shows the sensor components inside the black box (internal view).

The RINA-based IoT gateways were implemented using the IRATI open-source software [15] and using as hardware

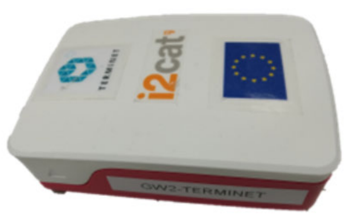


FIGURE 9. RINA-based IoT gateway.

a Raspberry Pi 4 with Debian Buster OS, Quad-core Cortex-A72 (ARM v8) 64-bit SoC @1.8GHz and 4GB SDRAM. IRATI was developed to integrate with any GNU/Linux-based OS, and this software can be deployed on a Raspberry Pi. A few minor changes to IRATI were made to support new kernel and Debian OS versions, and it is available in the IRATI Github repository. Figure 9 shows the deployed RINA-based IoT gateway.

The Edge Node is deployed as a Virtual Machine (VM), using VirtualBox as the supervisor/virtualization software. It is Ubuntu-based and supports docker and docker-compose. Its specs are the following: 1GB RAM, 150GB Disk Space, and 1-core CPU. The specs are relatively minimal but deemed sufficient to demonstrate the use case. Two VMs were deployed: the edge node core and the IRATI-based. The edge node core VM had the EdgeX platform in charge of managing the sensor data. The separate IRATI-based VM implements the RINA edge node.

B. DEPLOYMENT

Many different subsystems are utilized in the CERTH's smart house, such as heating/cooling and lighting, as well as an automation and control system. The HVAC system installed in CERTH's smart house is a Variable Refrigerant Flow (VRF) air conditioning system used for both heating and cooling of the occupied spaces. Two identical heat pump units are installed in the building, with the first being responsible for heating the ground floor, while the second is for the floor above it.

Three temperature sensors and one air quality sensor were deployed on the CERTH's smart house. The sensors were distributed in the CERTH smart house's rooms. Figure 10 shows the sensors placed in the master room 10a, the nursery room 10b, and the living room 10c. The air quality sensor was installed in the living room. Finally, the RINA-based IoT gateway was installed in the most centralized room (nursery room) to cover the whole CERTH's smart house.

The IRATI module is essential for managing the RINA IPCPs in the edge node, installed and running in its separate VM. Once all necessary components are operational, the sensor should be capable of transmitting data to the EdgeX component, which should then be visible on the CERTH smart house's dashboard. A RINA application was developed to establish a connection between the RINA sensors and the MQTT-based EdgeX Broker. This RINA application

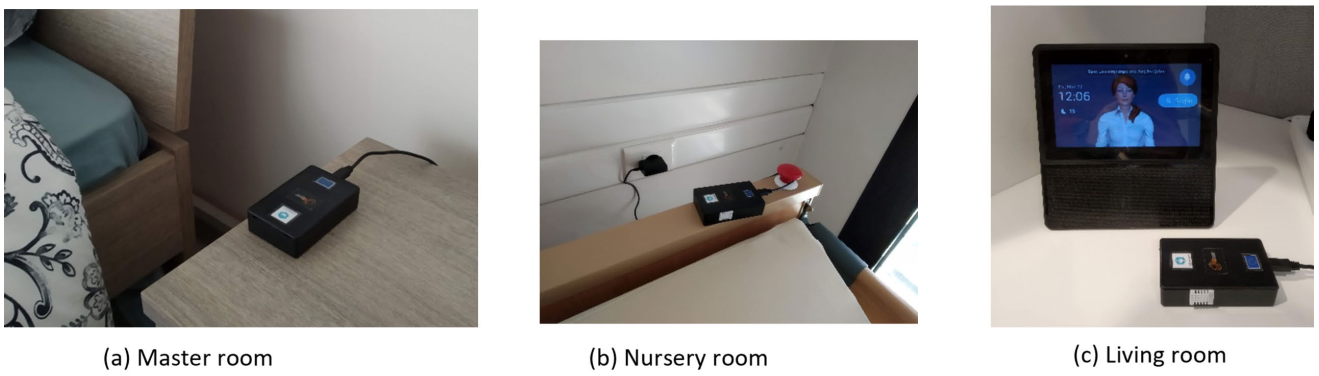


FIGURE 10. Deployment of sensors in the CERTH's smart house.

translates the CDAP protocol used by the sensors into the MQTT protocol employed by the EdgeX Broker. This translation is imperative because the EdgeX platform cannot utilize the RINA API. The Python-based RINA application utilizes the rinacat application, which is a client/server application designed to bidirectionally copy data between a RINA flow and its own standard input/output files or a forked command.[3] The Python application receives data from sensors, translates them, and publishes them into the EdgeX MQTT broker using the paho-mqtt library.

V. PERFORMANCE EVALUATION

At this stage of the research methodology, we conducted thorough testing and evaluation of the implemented environmental subsystem. As part of the performance evaluation, we assessed the proposed RINA subsystem in terms of communication and energy efficiency, scalability, and security. Our methodology involved comparing our subsystem to existing solutions, focusing on parameters such as goodput, delay, and packet losses while contrasting the proposed RINA-based architecture with similar communication protocols and services. Additionally, we evaluated the subsystem's effectiveness in monitoring and controlling environmental conditions within the smart building and validated its deployment to identify any issues or areas for improvement. The results are presented and analyzed in the following subsections. The result analysis helps identify the specific elements of the solution that are performing optimally and the modules that may require redesigning or alteration, contributing to our continuous improvement process.

A. RINA SENSOR PERFORMANCE

The RINA sensor performance was evaluated regarding delay, packet losses, and power consumption. The main goal was to evaluate the delay and packet losses in flows allocated between the RINA sensor and the edge Node — RINA network end-to-end delay—and to evaluate the power consumption of the RINA sensor.

[3]https://github.com/IRATI/

1) DELAY AND PACKET LOSSES

The delay and packet losses were measured using the rinaperf application in RINA networks. Rinaperf is a multi-threaded client/server application that measures network throughput and latency performance [37]. In this scenario, the edge node hosted the rinaperf server application, while the RINA sensor ran a rinaperf client application variation specifically adapted for embedded RTOS. The rinaperf client requested two flows, one for sending control information and the other for sending user data. The rinaperf client required parameters such as the number of SDUs to send, the SDU size in bytes, and the test time interval during which the delay was measured.

Once the flow was allocated, the application sent an SDU to the rinaperf server, activated a timer, and waited for a response for 2 seconds. On the other side, the rinaperf server returned the SDU as soon as possible. Upon reading the SDU using the `RINA_read()` API, the rinaperf client halted the timer, calculated the round-trip delay, and stored it. This process was repeated for the configured number of SDUs in the application.

At the end of the test, the rinaperf server transmitted a message containing the test results, including the number of packets received and the average number of packets per second. These results were employed to calculate the packet loss rate.

During the test, the rinaperf client was configured to send 1000 SDUs with an SDU size of 64 bytes, which is the maximum SDU size for RINA sensors in this use case). Figure 11 illustrates the Cumulative Distribution Function (CDF) of round-trip delay experienced by flows allocated between RINA sensors and the Edge Node. The flows experienced an average round-trip delay of 2.0550 $\pm$ with a standard deviation of 0.1674 milliseconds. This delay is deemed acceptable for Wi-Fi connections. Comparing it with the delay observed in MQTT, which is a protocol commonly used in similar proposed systems, we observed that the delay in RINA is lower than the 4.38 milliseconds analyzed in [45] and the time to completion (min = 10.47 milliseconds, max = 118.33 milliseconds) analyzed in [46]. As Table 2 summarizes, the proposed subsystem provided lower delay

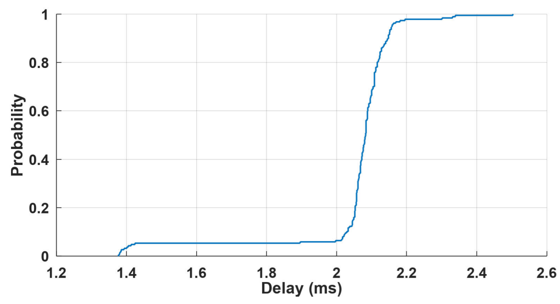


FIGURE 11. Round-trip delay perceived by flows allocated between RINA sensor and edge node.

TABLE 2. Delay and packet losses comparative.

System	Delay	Packet Loss	Protocol
[19]	Min: 18.3 ms, Max: 20.9 ms	30%	MQTT
[45]	Avg: 4.38 ms	-	MQTT
[46]	Min: 10.47 ms, Max: 118,33 ms	0.3%	CoAP
This work	Avg: 2.055 ms	0.01%	CDAP

Avg = average, Min = minimum, Max = maximum, ms = milliseconds.

than similar works based on MQTT and CoAP protocols. Additionally, since the application in this use case is not time-sensitive, the round-trip delay experienced by flows is suitable for transporting sensor data. Moreover, the packet loss rate was calculated during the test using the rinaperf server results (average number of packets received per second), which was found to be under 0.01%. As Table 2 summarizes, the proposed subsystem achieved a lower packet loss percentage than similar works based on MQTT and CoAP protocols.

2) ENERGY CONSUMPTION

On the other hand, the energy consumption was measured to evaluate the sensor's performance when utilizing the battery manager module. The measurements were conducted using the Otti-ARC power analyzer by Qoitech.[4] This analyzer allows users to obtain a power profile by measuring current and voltage in real-time. Additionally, the Otti-ARC facilitates the quickly identification of energy drains, enabling optimization of battery life.

Figure 12 shows the sensor's current profile in milliamperes (mA) when the sensor was implemented using RINAsense without the energy manager activated, serving as our baseline. Figure 13 displays the sensor's current profile in mA when the sensor was implemented using RINAsense and the battery manager is activated. For comparison, Figure 14 illustrates the sensor's current profile in mA when it was implemented using the MQTT protocol and the TCP/IP stack protocol. The observed peaks in Figure 12 and Figure 14 are quite similar, indicating various activities such as the Wi-Fi initial handshake to associate and establish the connection, data transmission, and Wi-Fi beacons for maintaining the connection. In Figure 13, a few peaks are associated with data

[4]https://www.qoitech.com/

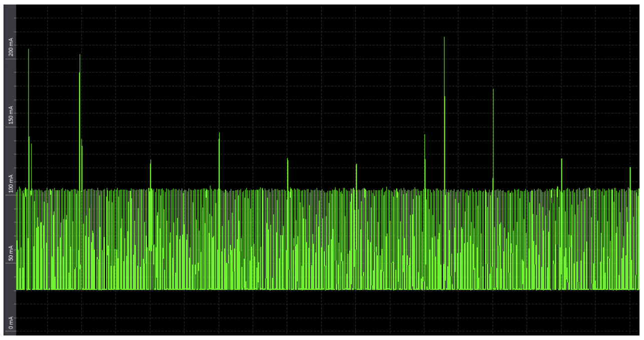


FIGURE 12. Sensor current (mA) profile baseline.

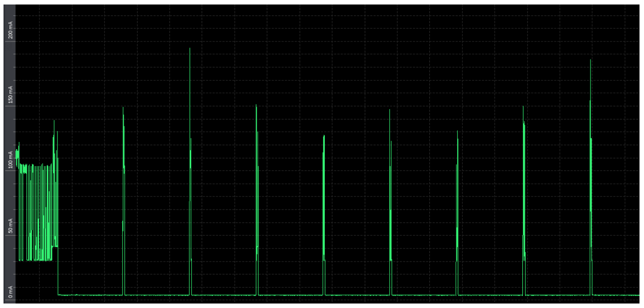


FIGURE 13. Sensor current (mA) profile when the battery manager is activated.

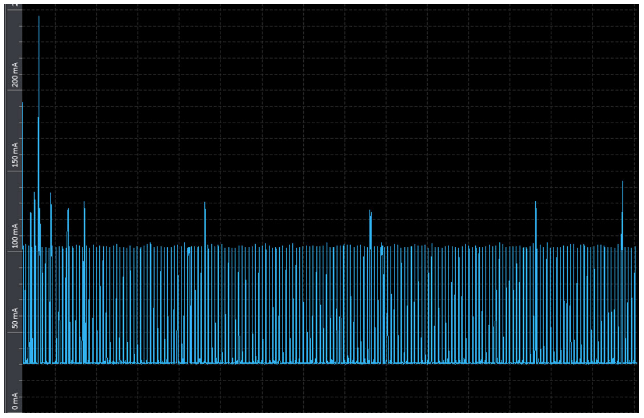


FIGURE 14. Sensor current (mA) profile when the sensors is implemented using the MQTT protocol and the TCP/IP stack.

transmission and some Wi-Fi beacons. The energy manager effectively reduced the Wi-Fi driver's energy consumption by activating the driver only when data is available to be sent and managing the Wi-Fi beacon interval, thereby reducing Wi-Fi energy consumption.

The sensor current profiles were sampled using a ten-second sliding window to estimate the energy consumption in microwatts. We used the Ottii tools to estimate the average energy consumption for each window. Fifty samples were captured using this method. Figure 15 compares the three sensor current profiles (with and without activating the battery manager functionality, and when the sensors

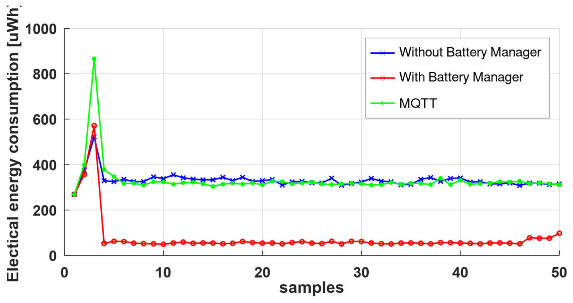


FIGURE 15. Electrical energy consumption comparative.

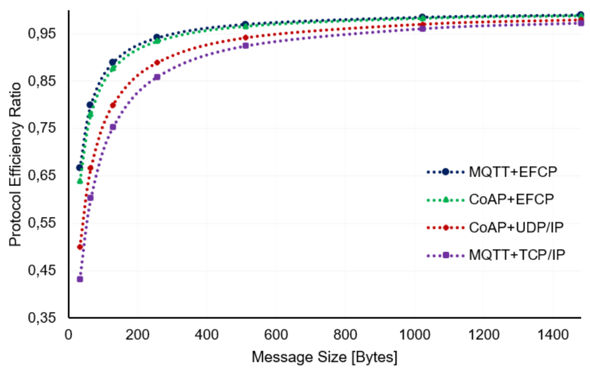


FIGURE 16. Protocol efficiency ratio in function of message size.

were implemented using the TCP/IP stack and the MQTT protocol). For the two RINA cases, the analysis initially observed an average peak of 523 microwatts per hour (μWh) in both scenarios. This peak was produced by the initial sensor processes such as the shim Wi-Fi DIF enrollment (association with the RINA-based IoT gateway), the normal DIF enrollment (finding peers in the same DIF and interchanging CACEP messages), allocating the data flow with the edge node application, and sending the first SDU. After that, the energy consumption was stable, though at different levels, because at this stage, only the sensors were sending data, and this process did not produce significant consumption peaks. Similarly, the MQTT consumption presented an average peak of 843 (μWh), which was produced by the initial messages interchange using TCP/IP and MQTT. Figure 15 shows that the battery manager module significantly reduced the electrical energy consumption at this stage. The sensor required 327.282 $\pm$ 11.43 μWh on average when the battery manager was not activated, but the power was reduced as the RINA sensor device required an average of 57.52 $\pm$ 8.77 μWh when the battery manager was activated. This represents a significant reduction in energy consumption. As a result, the battery manager transformed the RINAsense implementation into an energy-efficient one. This energy efficiency will be helpful for sensor devices that need to be powered by batteries.

3) PROTOCOL EFFICIENCY

The efficiency of a protocol is a crucial factor in IoT performance, as it determines how effectively data is transferred between nodes. Each transport protocol uses headers to assist in data transfer, but if the header size exceeds the data size, efficiency decreases, directly affecting the goodput. To calculate the protocol efficiency ratio, we compared the data size to the total message size (data size + header size). In our analysis, we compared the performance of two well-known IoT application protocols, MQTT and CoAP, with the TCP/IP stack and the RINA EFCP protocol. We considered header sizes of 40 Bytes for TCP/IP, 28 Bytes for UDP/IP, and 14 Bytes for RINA EFCP. Figure16 provides a visual comparison of the protocol efficiency ratio. The EFCP protocol exhibited greater efficiency than the TCP/IP stack due to its smaller header size. This efficiency is particularly noticeable when transferring small messages (less than 300 Bytes), while it is similar for larger messages (more than 1000 Bytes). As IoT typically involves transferring small messages, the EFCP protocol is more efficient for IoT applications compared to the TCP/IP stack. These findings have practical implications, highlighting the relevance and applicability of our research in smart building's network efficient communications.

B. RINA QoS CUBE PERFORMANCE

The main goal of this evaluation is to validate the RINA network capability to accommodate the growing IoT devices of the slice1 DIF QoS Cube in terms of goodput (quantity of useful information) and latency, based on the number of RINA sensor application flows.

The first part of the analysis aimed to investigate the impact of sensor quantity on the goodput of the RINA sensor application within the unreliable QoS Cube(subsection III-B1) over the slice1 DIF. The test consisted of deploying ten RINA sensors in the subsystem, allocating unreliable flows — similar to a UDP connection— and sending as many packets of 32 bytes SDU size as possible within a 10-second interval. Using the rinaperf application [37], it was possible to evaluate the goodput of each flow allocated by each sensor. Figure 17 illustrates the variation in goodput experienced by each RINA sensor application as the number of sensors increased. As anticipated, the goodput diminished as the number of sensors increased due to heightened congestion and resource utilization in the Wi-Fi channel resulting from the increased transmission of flows and packets. Additionally, each RINA sensor experienced different goodput, but their difference was insignificant until reaching eight sensors. Subsequently, with nine and ten sensors, some flows experienced an increment in goodput, but others experienced a significant decrease. For this quantity of flows, the range increased significantly. For example, the range with nine flows was 0.166 Mbps (max 0.435 Mbps and min 0.269 Mbps), and with ten flows was 0.197 Mbps (max 0.484 Mbps and min 0.287 Mbps). This variation in range, when the number of flows increased, was caused by the limitations of the IRATI version implementation, which was later addressed

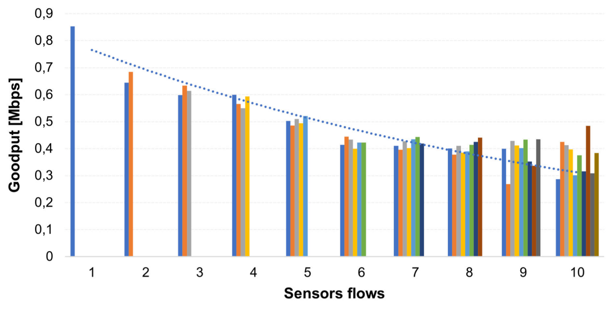


FIGURE 17. Goodput depending on number of sensors.

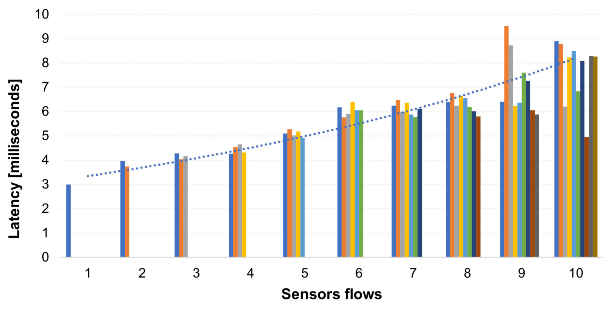


FIGURE 18. Latency depending on number of sensors.

through investigation and resolution with the rlite open-source software [37]. In summary, the findings indicate a tendency for goodput to decrease as the number of sensors increases.

The second part of the analysis aimed to determine the latency experienced by each sensor application as the flows within the slice1 DIF increased. Similar to the prior analysis, the test involved deploying ten RINA sensors in the subsystem, allocating unreliable flows, transmitting packets of 32 Bytes SDU size, and assessing the latency experienced by each flow. The rinaperf application measured the latency by employing its round-trip delay operation mode. Figure 18 illustrates the latency experienced by each RINA sensor flow as the number of RINA sensor flows increases. This behavior was expected as a larger number of packets queued in the RMT buffer introduces delays. Additionally, the allocation of more sensor flows introduced greater variance. In other words, the RINA QoS Cube did not guarantee a delay nor prioritize traffic flows, but it could be configured by specifying the delay during the QoS Cube set-up. This aspect will be analyzed in future research experiments. The minimum delay recorded was three milliseconds with one sensor flow, and the maximum was 9.5 milliseconds with nine sensor flows. In summary, the latency increases as the number of sensors increases.

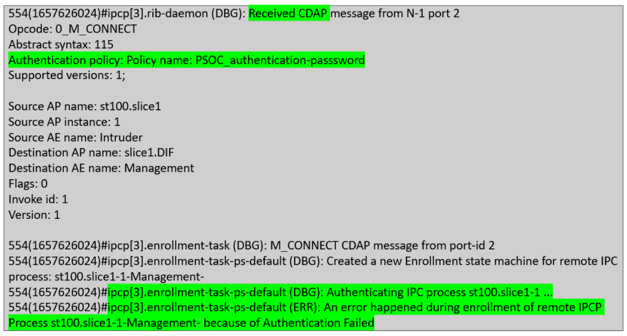
According to the findings, the performance of RINA sensors only experienced a slight decrease, which is acceptable considering that the sensors used in smart buildings for these purposes do not require low latency or high goodput. Therefore, the RINA-based IoT gateway ensures that the system can effectively handle an increasing number of sensors and provide scalable communications. However, for more demanding smart building applications such as video security, it is recommended to add additional RINA-based IoT gateways (horizontal scalability) to ensure that the system can effectively manage larger amounts of data. This horizontal scalability is achievable due to RINA's recursive nature and its flexible programmability layer (DIF). A detailed assessment of RINA's scalability for this IoT scenario was studied in [43] and [47].

C. RINA SECURITY POLICY EVALUATION

RINA offers a key advantage in its provision of secure layers rather than protocols. This feature allows only the IPCPs within the same DIF to examine packets. RINA ensures that only authenticated IPCPs can join the DIF through authentication during enrollment, minimizing the risk of unauthorized access or impersonation attacks. The main goal of this evaluation is to validate the resiliency of the DIF subsystem to avoid authentication attacks.

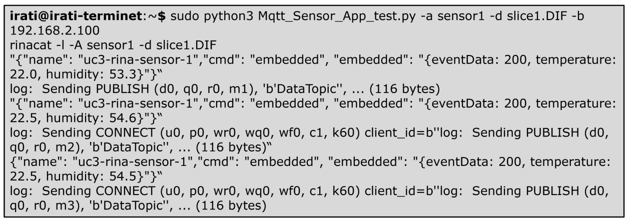
To achieve this, a RINA sensor (intruder) attempted to spoof the data by becoming part of the DIF. The intruder IPCP tried to enroll in the DIF using the CACEP. During the CACEP, the IPCP must authenticate to be accepted into the DIF. In this case, the RINA-based IoT gateway received the CDAP authentication message, as shown in Figure 19, with the authentication policy (PSOC_authentication_password) highlighted. The RINA-based IoT gateway validated the username and password, and it denied the intruder's authentication request because the credentials did not match, as highlighted. An IPCP must complete the CACEP in RINA before exchanging data messages. Since the intruder IPCP did not successfully authenticate the CACEP, it could not access data. RINA provides a high degree of flexibility in implementing and updating security measures, allowing the CACEP authentication policy to be tailored to the needs of a DIF. For example, public key and other robust methods can be appropriate, depending on the threat environment. As a result, eavesdropping or spoofing attacks are avoided in the proposed subsystem. Using a policy to manage security in the proposed subsystem provided effective resiliency against device authentication threats.

The proposed configuration ensures secure communication within the smart building environment monitoring subsystem without the need for additional protocols or equipment. By requiring the only trusted entities can inspect the packets on the DIF, the impact of an attack is limited to a single DIF, thereby preserving the integrity of the smart building network and reducing the attack surface. Furthermore, RINA's recursive nature allows for the reuse of security mechanisms at various scales, minimizing smart building network management and enhancing scalability. With strict authentication and access control, unauthorized entities are prevented from accessing or manipulating sensitive data within the network, a critical aspect in IoT networks where substantial amounts of potentially sensitive data are regularly transmitted.



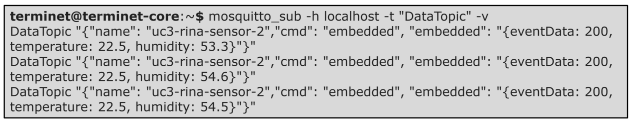
```
554(1657626024)#ipcp[3].rib-daemon (DBG): Received CDAP message from N-1 port 2
Opcode: 0_M_CONNECT
Abstract syntax: 115
Authentication policy: Policy name: PSOC_authentication-passsword
Supported versions: 1;

Source AP name: st100.slice1
Source AP instance: 1
Source AE name: Intruder
Destination AP name: slice1.DIF
Destination AE name: Management
Flags: 0
Invoke id: 1
Version: 1

554(1657626024)#ipcp[3].enrollment-task (DBG): M_CONNECT CDAP message from port-id 2
554(1657626024)#ipcp[3].enrollment-task-ps-default (DBG): Created a new Enrollment state machine for remote IPC
process: st100.slice1-1-Management-
554(1657626024)#ipcp[3].enrollment-task-ps-default (DBG): Authenticating IPC process st100.slice1-1 ...
554(1657626024)#ipcp[3].enrollment-task-ps-default (ERR): An error happened during enrollment of remote IPCP
Process st100.slice1-1-Management- because of Authentication Failed
```

FIGURE 19. **RINA-based IoT gateway log security policy validation.**



```
irati@irati-terminet:~$ sudo python3 Mqtt_Sensor_App_test.py -a sensor1 -d slice1.DIF -b
192.168.2.100
rinacat -l -A sensor1 -d slice1.DIF
"{"name": "uc3-rina-sensor-1","cmd": "embedded", "embedded": "{eventData: 200, temperature:
22.0, humidity: 53.3}"}"
log:  Sending PUBLISH (d0, q0, r0, m1), 'b'DataTopic'', ... (116 bytes)
"{"name": "uc3-rina-sensor-1","cmd": "embedded", "embedded": "{eventData: 200, temperature:
22.5, humidity: 54.6}"}"
log:  Sending CONNECT (u0, p0, wr0, wq0, wf0, c1, k60) client_id=b''log:  Sending PUBLISH (d0,
q0, r0, m2), 'b'DataTopic'', ... (116 bytes)"
{"name": "uc3-rina-sensor-1","cmd": "embedded", "embedded": "{eventData: 200, temperature:
22.5, humidity: 54.5}"}"
log:  Sending CONNECT (u0, p0, wr0, wq0, wf0, c1, k60) client_id=b''log:  Sending PUBLISH (d0,
q0, r0, m3), 'b'DataTopic'', ... (116 bytes)
```

FIGURE 20. **RINA application receiving data from CDAP and publising in MQTT broker.**



```
terminet@terminet-core:~$ mosquitto_sub -h localhost -t "DataTopic" -v
DataTopic "{"name": "uc3-rina-sensor-2","cmd": "embedded", "embedded": "{eventData: 200,
temperature: 22.5, humidity: 53.3}"}"
DataTopic "{"name": "uc3-rina-sensor-2","cmd": "embedded", "embedded": "{eventData: 200,
temperature: 22.5, humidity: 54.6}"}"
DataTopic "{"name": "uc3-rina-sensor-2","cmd": "embedded", "embedded": "{eventData: 200,
temperature: 22.5, humidity: 54.5}"}"
```

FIGURE 21. **mosquitto subscription to sensor topic.**

D. SUBSYSTEM INTEGRATION

CERTH's smart house systems are IP-based and cannot use the RINA API. For this reason, the Python-based RINA application played a crucial role in translating the CDAP protocol into the MQTT protocol. The integration tests of the subsystem aimed to validate the accuracy of protocol translation. On the RINA side, it was confirmed that the rinacat application, using RINA protocol flows, successfully received data from RINA sensors. Meanwhile, the data collected by the RINA sensors from the EdgeX broker was ingested into the EdgeX MQTT broker.

Figure 20 depicts the Python-based RINA application operating in the IRATI VM at the edge node. This application received data from the RINA *sensor1*, translated them and published them in the EdgeX MQTT broker. Figure 21 shows the Mosquitto client subscription results in the EdgeX VM. As shown, the EdgeX MQTT broker successfully received data from the RINA sensors. Internally, the EdgeX platforms connected the MQTT broker and the core data storage. Subsequently, the CERTH smart house system retrieved and processed these data for extensive analysis.

Finally, the system integration was validated by displaying the data in a user interface. The dashboard was generated using Grafana software. This dashboard shows a summary

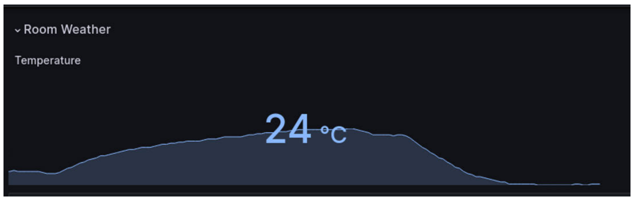


FIGURE 22. **Humidity at living room.**

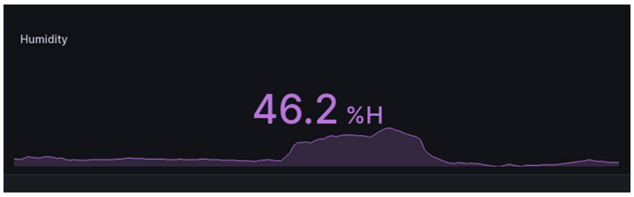


FIGURE 23. **Humidity at living room.**

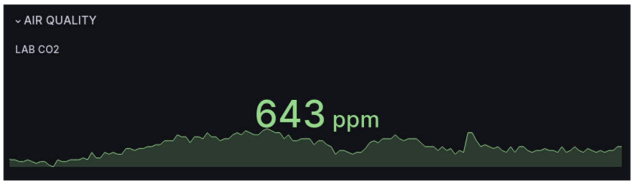


FIGURE 24. **air quality at living room.**

of data RINA data sensors and facilitates quick decision-making. Figures 22, 23, 24 provide an example of the temperature, humidity, and CO levels in the living room gathered by RINA sensors.

VI. CONCLUSION

This paper has presented an innovative environment monitoring subsystem based on RINA. The environment monitoring subsystem comprised several RINA sensors deployed in the CERTH's smart house, a RINA-based IoT gateway, and the CERTH's edge node. We have improved the first RINAsense architecture version, initially designed in [14], by changing its operational structure, adding a battery and a resource manager, and implementing a portability layer. As a result, the RINAsense prototype has demonstrated significant improvement in energy consumption. The battery manager has met the specific requirements of sensors, and has maintained the RINA DIF isolation principle.

Additionally, the RINAsense architecture maturity has reached Technology Readiness Level (TRL) 5 (technology validated in relevant environment) from its initial TRL 3 (experimental proof of concept). Furthermore, we have designed the RINA protocol to facilitate seamless and secure communication between sensors and control systems. The RINA protocol design details we selected can guide IoT practitioners in replicating or adapting it in other IoT application domains.

The feasibility of our proposal was validated by integrating the environment monitoring subsystem into an operative smart building system. A protocol translation was required to ingest the RINA sensor data into the EdgeX MQTT broker since current applications (TCP/IP-based) cannot use the RINA API. This integration will reduce the barrier to using RINA and open new opportunities for experimentation and adoption of RINA in IoT domains.

VII. FUTURE WORK

RINA is still evolving, and widespread adoption requires further development, standardization, and real-world testing. Ongoing research and testing suggest that RINA has the potential to become a feasible alternative to the current Internet architecture in the future. Since the RINA protocol is customized, further research will be conducted to study the effects of changing the sizes of RINA constants (e.g., SequenceNumberLength and RateLength) on the RINAsense prototype. Additionally, the proposed subsystem will be evaluated in more complex building systems and environment configurations. The present RINAsense prototype currently accommodates IEEE 802.11 as its physical layer. Therefore, it will be imperative to broaden its support to include additional physical layers, such as IEEE 802.15.4, and conduct further analysis of its implications. Furthermore, recent advances in lightweight cryptography algorithms will enable efficient mechanisms to ensure privacy in the RINAsense prototype. Though not measured in this experiment, using CDAP messages as an application protocol in IoT environments requires special attention because it promises to be more efficient than current protocols. Further research will be conducted to understand how IoT applications can be modeled to fit the CDAP protocol. Thus, the RINAsense architecture and prototype will achieve further TRLs with the vision of leveraging them as a commercial product in the future.

ACKNOWLEDGMENT

David Sarabia-Jácome thanks John Day with RINA, a concept initiator and a developer–for reviewing the article and providing valuable insight to enhance it and to keep it aligned with the RINA principles; and acknowledge the Secretaría Nacional de Educación Superior, Ciencia, Tecnología e Innovación (SENESCYT), Ecuador. Also, G. Siachimis thanks Dr. Dimitrios Tzovaras, is the President of the CERTH's Board–for its support.

REFERENCES

[1] H. Farzaneh, L. Malehmirchegini, A. Bejan, T. Afolabi, A. Mulumba, and P. P. Daka, "Artificial intelligence evolution in smart buildings for energy efficiency," *Appl. Sci.*, vol. 11, no. 2, p. 763, Jan. 2021, doi: 10.3390/app11020763.

[2] R. Eini, L. Linkous, N. Zohrabi, and S. Abdelwahed, "Smart building management system: Performance specifications and design requirements," *J. Building Eng.*, vol. 39, Jul. 2021, Art. no. 102222, doi: 10.1016/j.jobe.2021.102222.

[3] C. K. Metallidou, K. E. Psannis, and E. A. Egyptiadou, "Energy efficiency in smart buildings: IoT approaches," *IEEE Access*, vol. 8, pp. 63679–63699, 2020, doi: 10.1109/ACCESS.2020.2984461.

[4] A. Abuarqoub, "A review of the control plane scalability approaches in software defined networking," *Future Internet*, vol. 12, no. 3, p. 49, Mar. 2020, doi: 10.3390/fi12030049.

[5] M. U. Younus, S. U. Islam, I. Ali, S. Khan, and M. K. Khan, "A survey on software defined networking enabled smart buildings: Architecture, challenges and use cases," *J. Netw. Comput. Appl.*, vol. 137, pp. 62–77, Jul. 2019, doi: 10.1016/j.jnca.2019.04.002.

[6] A. Hakiri, B. Sellami, S. B. Yahia, and P. Berthou, "A SDN-based IoT architecture framework for efficient energy management in smart buildings," in *Proc. Global Inf. Infrastructure Netw. Symp. (GIIS)*, Oct. 2020, pp. 1–6, doi: 10.1109/GIIS50753.2020.9248495.

[7] S. Ahmad and A. H. Mir, "Scalability, consistency, reliability and security in SDN controllers: A survey of diverse SDN controllers," *J. Netw. Syst. Manage.*, vol. 29, no. 1, Nov. 2020, doi: 10.1007/s10922-020-09575-4.

[8] S. Leon Gaixas, J. Perelló, D. Careglio, E. Grasa, D. R. López, and P. A. Aranda, "Scalable topological forwarding and routing policies in RINA-enabled programmable data centers," *Trans. Emerg. Telecommun. Technol.*, vol. 28, no. 12, p. e3256, Nov. 2017, doi: 10.1002/ett.3256.

[9] E. Grasa, M. P. de Leon, S. van der Meer, D. Lopez, and M. Tarzan, "Open multi-access edge computing and distributed mobility management with RINA," in *Proc. IEEE Conf. Netw. Function Virtualization Softw. Defined Netw. (NFV-SDN)*, Nov. 2017, pp. 1–2, doi: 10.1109/NFV-SDN.2017.8169850.

[10] E. Grasa, O. Rysavy, O. Lichtner, H. Asgari, J. Day, and L. Chitkushev, "From protecting protocols to layers: Designing, implementing and experimenting with security policies in RINA," in *Proc. IEEE Int. Conf. Commun. (ICC)*, May 2016, pp. 1–7, doi: 10.1109/ICC.2016.7510780.

[11] E. Trouva, E. Grasa, J. Day, I. Matta, L. T. Chitkushev, S. Bunch, M. P. de Leon, P. Phelan, and X. Hesselbach-Serra, "Transport over heterogeneous networks using the RINA architecture," in *Proc. Wireless Internet Commun.*, 2011, pp. 297–308, doi: 10.1007/978-3-642-21560-5_25.

[12] E. Grasa, B. Gastón, S. van der Meer, M. Crotty, and M. A. Puente, "Simplifying multi-layer network management with rina," in *Proc. TERENA Net. Conf. (TNC)*, Czech Republic, 2016.

[13] J. Day, I. Matta, and K. Mattar, "Networking is IPC," in *Proc. ACM CoNEXT Conf. (CONEXT)*, 2008, pp. 1–6, doi: 10.1145/1544012.1544079.

[14] D. Sarabia-Jácome, E. Grasa, and M. Catalán, "RINAsense: A prototype for implementing RINA networks in IoT environments," in *Proc. 6th Conf. Cloud Internet Things (CIoT)*, Mar. 2023, pp. 70–76, doi: 10.1109/CIoT57267.2023.10084905.

[15] S. Vrijders, D. Staessens, D. Colle, F. Salvestrini, E. Grasa, M. Tarzan, and L. Bergesio, "Prototyping the recursive Internet architecture: The IRATI project approach," *IEEE Netw.*, vol. 28, no. 2, pp. 20–25, Mar. 2014, doi: 10.1109/MNET.2014.6786609.

[16] TERMINET H2020 Project. *Next Generation of Smart Interconnected IoT*. Accessed: Sep. 10, 2023. [Online]. Available: https://terminet-h2020.eu/

[17] ThoughtWire. *Glossary of Terms That Define the Smart Building*. Accessed: May 5, 2023. [Online]. Available: https://info.thoughtwire.com/smart-building-definitions

[18] A. Verma, S. Prakash, V. Srivastava, A. Kumar, and S. C. Mukhopadhyay, "Sensing, controlling, and IoT infrastructure in smart building: A review," *IEEE Sensors J.*, vol. 19, no. 20, pp. 9036–9046, Oct. 2019.

[19] A. Kumar, S. Sharma, N. Goyal, A. Singh, X. Cheng, and P. Singh, "Secure and energy-efficient smart building architecture with emerging technology IoT," *Comput. Commun.*, vol. 176, pp. 207–217, Aug. 2021.

[20] H. M. Al-Kadhim and H. S. Al-Raweshidy, "Energy efficient and reliable transport of data in cloud-based IoT," *IEEE Access*, vol. 7, pp. 64641–64650, 2019, doi: 10.1109/ACCESS.2019.2917387.

[21] D. Sauer, R. Jumar, R. Lutz, T. Schlachter, C. Schmitt, and V. Hagenmeyer, "Towards smart buildings: A versatile acquisition setup for indoor climate data," in *Proc. IEEE PES Innov. Smart Grid Technol. Eur. (ISGT-Europe)*, Oct. 2020, pp. 1196–1200, doi: 10.1109/ISGT-Europe47291.2020.9248925.

[22] F.-J. Ferrández-Pastor, H. Mora, A. Jimeno-Morenilla, and B. Volckaert, "Deployment of IoT edge and fog computing technologies to develop smart building services," *Sustainability*, vol. 10, no. 11, p. 3832, Oct. 2018, doi: 10.3390/su10113832.

[23] J. Kim, S. Kim, S. Bae, M. Kim, Y. Cho, and K.-I. Lee, "Indoor environment monitoring system tested in a living lab," *Building Environ.*, vol. 214, Apr. 2022, Art. no. 108879, doi: 10.1016/j.buildenv.2022.108879.

[24] A. Rico, C. Smuts, and K. Larson, "Chameleon: Adaptive sensor intelligence for smart buildings," *IEEE Internet Things J.*, vol. 9, no. 19, pp. 19362–19372, Oct. 2022, doi: 10.1109/JIOT.2022.3165349.

[25] S. Leon, J. Perelló, D. Careglio, and M. Tarzan, "Guaranteeing QoS requirements in long-haul RINA networks," in *Proc. 19th Int. Conf. Transparent Opt. Netw. (ICTON)*, Jul. 2017, pp. 1–4, doi: 10.1109/ICTON.2017.8025071.

[26] D. C. Walden, *Note on Interprocess Communication in a Resource Sharing Computer Network*, document RFC 61, Jul. 1970. Accessed: Jun. 28, 2024. [Online]. Available: https://www.rfc-editor.org/info/rfc61

[27] D. C. Walden, *Systems for Interprocess Communication in a Resource Sharing Computer Network*, document RFC 62, Aug. 1970. Accessed: Jun. 28, 2024. [Online]. Available: https://www.rfc-editor.org/info/rfc62

[28] European Telecommunications Standards Institute (ETSI). (2019). *Next Generation Protocols (NGP): An Example of a Non-IP Network Protocol Architecture Based on RINA Design Principles*. Accessed: Jun. 27, 2024. [Online]. Available: http://www.etsi.org

[29] S. Vrijders, E. Trouva, J. Day, E. Grasa, D. Staessens, D. Colle, M. Pickavet, and L. Chitkushev, "Unreliable inter process communication in Ethernet: Migrating to RINA with the shim DIF," in *Proc. 5th Int. Congr. Ultra Modern Telecommun. Control Syst. Workshops (ICUMT)*, Sep. 2013, pp. 215–221, doi: 10.1109/ICUMT.2013.6798429.

[30] E. Grasa, E. Trouva, S. Bunch, P. DeWolf, and J. Day, "Developing a RINA prototype over UDP/IP using TINOS," in *Proc. 7th Int. Conf. Fut. Int. Tech.*, Sep. 2012, pp. 31–36, doi: 10.1145/2377310.2377321.

[31] R. W. Watson, "Timer-based mechanisms in reliable transport protocol connection management," *Comput. Netw.*, vol. 5, no. 1, pp. 47–56, Feb. 1981, doi: 10.1016/0376-5075(81)90031-3.

[32] Eur. Commission. *Programmability In RINA for European Supremacy of VirTualised NEtworks (PRISTINE)*. Accessed: Jun. 13, 2023. [Online]. Available: https://cordis.europa.eu/project/id/619305/es

[33] ARCFIRE Project. *Large-scale RINA Experimentations on FIRE+ Infrastructure*. Accessed: Jun. 13, 2023. [Online]. Available: https://ict-arcfire.eu/

[34] E. Grasa, L. Bergesio, M. Tarzan, D. Lopez, S. van der Meer, J. Day, and L. Chitkushev, "Mobility management in RINA networks: Experimental validation of architectural properties," in *Proc. IEEE Wireless Commun. Netw. Conf. (WCNC)*, Apr. 2018, pp. 1–6, doi: 10.1109/WCNC.2018.8377265.

[35] V. Ishakian, I. Matta, and J. Akinwumi, "On the cost of supporting mobility and multihoming," in *Proc. IEEE Globecom Workshops*, Dec. 2010, pp. 310–314, doi: 10.1109/GLOCOMW.2010.5700332.

[36] P. Thompson and N. Davies, "Towards a RINA-based architecture for performance management of large-scale distributed systems," *Computers*, vol. 9, no. 2, p. 53, Jun. 2020, doi: 10.3390/computers9020053.

[37] V. Maffione. (2017). *A Light RINA Implementation*. [Online]. Available: https://github.com/rlite/rlite

[38] Y. Wang, I. Matta, F. Esposito, and J. Day, "Introducing ProtoRINA," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 3, pp. 129–131, Jul. 2014, doi: 10.1145/2656877.2656897.

[39] T. Ramezanifarkhani and P. Teymoori, "Securing the Internet of Things with recursive InterNetwork architecture (RINA)," in *Proc. Int. Conf. Comput., Netw. Commun. (ICNC)*, Mar. 2018, pp. 188–194, doi: 10.1109/ICCNC.2018.8390263.

[40] P. Teymoori and T. Ramezanifarkhani, "Reconciling efficiency and security of the Internet of Things: A recursive InterNetwork architecture (RINA) approach," *Res. Square*, Aug. 2023, doi: 10.21203/rs.3.rs-3234524/v1.

[41] B. S. Neelam and B. A. Shimray, "Applicability of RINA in IoT communication for acceptable latency and resiliency against device authentication attacks," in *Proc. 6th Int. Conf. for Converg. Technol. (I2CT)*, Apr. 2021, pp. 1–7, doi: 10.1109/I2CT51068.2021.9417992.

[42] M. Rizinski, J. Day, and L. Chitkushev, "IoT architecture based on RINA," in *Proc. 23rd Conf. Innov. Clouds, Internet Netw. Workshops (ICIN)*, Feb. 2020, pp. 41–45, doi: 10.1109/ICIN48450.2020.9059367.

[43] D. Sarabia-Jácome, E. Grasa, and M. Catalán, "Fine-grained traffic differentiation in tactile Internet scenarios using RINA," in *Proc. IEEE 9th World Forum Internet Things (WF-IoT)*, Aveiro, Portugal, Oct. 2023, pp. 1–6, doi: 10.1109/wf-iot58464.2023.10539444.

[44] Next Gener. IoT Initiatve. (2020). *The Near Zero Energy Building (nZEB) Smart House—An Innovation Hub, IoT Testbed and Ecosystem*. [Online]. Available: https//www.ngiot.eu/certh/#1638802538755-3d54710d-736f

[45] M. H. Amaran, N. A. M. Noh, M. S. Rohmad, and H. Hashim, "A comparison of lightweight communication protocols in robotic applications," *Proc. Comput. Sci.*, vol. 76, pp. 400–405, Jan. 2015, doi: 10.1016/j.procs.2015.12.318.

[46] D. Silva, L. I. Carvalho, J. Soares, and R. C. Sofia, "A performance analysis of Internet of Things networking protocols: Evaluating MQTT, CoAP, OPC UA," *Appl. Sci.*, vol. 11, no. 11, p. 4879, May 2021, doi: 10.3390/app11114879.

[47] D. Sarabia-Jácome, S. Giménez-Antón, A. Liatifis, E. Grasa, M. Catalán, and D. Pliatsios, "Progressive adoption of RINA in IoT networks: Enhancing scalability and network management via SDN integration," *Appl. Sci.*, vol. 14, no. 6, p. 2300, Mar. 2024, doi: 10.3390/app14062300.

DAVID SARABIA-JÁCOME received the B.S. degree in electronics and computer networks engineering from Escuela Politécnica Nacional, Ecuador, the M.S. degree in communications technologies, systems, and networks from Universitat Politècnica de València, in 2016, and the Ph.D. degree in telecommunication engineering from Universitat Politècnica de València, in 2020. Since 2021, he has been a Postdoctoral Researcher with Fundació i2CAT, with a focus on RINA network architecture for the IoT environment. He is actively contributing to RINAsense open-source software. His research activities and interests include the IoT, future networks, edge-fog-cloud continuum, and artificial intelligence applied to IoT environments.

FRANÇOIS-DENIS GONTHIER received the B.S. degree in computer science from Univeristé de Sherbrooke, QC, Canada. He has worked with computer networks through the M.Sc. project that was spent working on P2P network and network simulation with the DOMUS Smart Home and Health Telematics Laboratory. He has since worked outside the academic world, as a Software Developer, developing projects with open-source tools and Linux. He also worked professionally on AOSP-based projects, with modest contributions to things, such as Google Project Ara. He became involved with RINA in a project performed under the auspices of the Government of Quebec and a commercial company and contributed to the RINAsense open-source software.

STEVE BUNCH received the B.S. degree in engineering, major in computer science from the University of Oklahoma, in 1972, and the M.S. degree in computer science/electrical engineering from the University of Illinois at Urbana–Champaign, in 1976. He has been involved in early Arpanet development, UNIX networking and security, secure networking and operating systems, processor architecture, software architecture for cellular telephones, and embedded device hardware and software. He is currently involved with the development of the RINA networking architecture.

GEORGIOS SIACHAMIS received the B.S.-integrated M.S. degree from the Electrical Engineering and Computer Science Department, Democritus University of Thrace. He has been a Research Assistant with the Informatics and Telematics Institute, Centre for Research and Technology Hellas, Thessaloniki, since April 2021. His main research interests include blockchain technologies, open-source blockchain frameworks, smart contract development, and secure software development.

EDUARD GRASA received the degree in telecommunication engineering with the Technical University of Catalonia, in 2004, and the Ph.D. degree, in 2009. He has participated in several national and international research projects, including UCLP, HULP, FP6 PHOSPHORUS, FP7 FEDERICA, FP7 OFELIA, DREAMS, HPDMnet, and laaS Framework. He was the Technical Lead of the FP7 IRATI Project and its follow-up, FP7 PRISTINE. His current interest includes RINA, an internetwork architecture proposed by John Day.

MARISA CATALÁN received the B.S. degree in telecommunications engineering and the Ph.D. degree in telematics from the Universitat Politècnica de Catalunya, Barcelona, Spain, in 2003 and 2009, respectively. She leads the Internet of Things Group, i2CAT. She has received a Pre-Doctoral FI Grant (Catalan Government). She is currently the Local Technical Manager of The Thinx 5GBarcelona Laboratory, a joint initiative with Telefonica, the Mobile World Capital, and Fira Barcelona. She has an expertise of more than 19 years in wireless networks, IoT technologies, communication and protocols, and their application to real use cases. She is the author of 45 national and international publications (one book, three book chapters, five journals, and 36 conferences). She has been a personal investigator of the research coordinated project 5GCity (TEC2016-76795-C6-2-R) and the Fem-IoT initiative (https://femiot.cat/) that coordinated leading Catalan research centers and universities in the creation of IoT solutions and technologies. She is a supervisor of a Ph.D. thesis under the MSCA-ITN 5GSmartfact Project. She has supervised ten M.Sc. and bachelor's thesis at the UPC (ETSETB, EETAC).

GEORGIOS STAVROPOULOS has been a Research Assistant with the Informatics and Telematics Institute, Centre for Research and Technology Hellas, Thessaloniki, since October 2006. His main research interests include computer vision, 3D search, and soft biometrics K@L gait analysis.