applied sciences

MDPI

Article

Progressive Adoption of RINA in IoT Networks: Enhancing Scalability and Network Management via SDN Integration

David Sarabia-Jácome [1,*], Sergio Giménez-Antón [2], Athanasios Liatifis [3], Eduard Grasa [4], Marisa Catalán [1] and Dimitrios Pliatsios [3]

[1] IoT Research Group, Fundació i2CAT, 08034 Barcelona, Spain; marisa.catalan@i2cat.net
[2] Software Networks Research Group, Fundació i2CAT, 08034 Barcelona, Spain; sergio.gimenez@i2cat.net
[3] Department of Electrical and Computer Engineering, University of Western Macedonia, 50100 Kozani, Greece; aliatifis@uowm.gr (A.L.); dpliatsios@uowm.gr (D.P.)
[4] Operations and Digital Infrastructure and Services, Fundació i2CAT, 08034 Barcelona, Spain; eduard.grasa@i2cat.net
* Correspondence: david.sarabia@i2cat.net; Tel.: +34-93-55-325-10

Abstract: Thousands of devices are connected to the Internet as part of the Internet of Things (IoT) ecosystems. The next generation of IoT networks is expected to support this growing number of Intelligent IoT devices and tactile Internet solutions to provide real-time applications. In view of this, IoT networks require innovative network architectures that offer scalability, security, and adaptability. The Recursive InterNetwork Architecture (RINA) is a clean slate network architecture that provides a scalable, secure, and flexible framework for interconnecting computers. SDN technology is becoming a de facto solution to overcome network requirements, making RINA adoption difficult. This paper presents an architecture for integrating RINA with SDN technologies to lower the barriers of adopting RINA in IoT environments. The architecture relies on a RINA-based distributed application facility (DAF), a RINA southbound driver (SBI), and the RINA L2VPN. The RINA-based DAF manages RINA nodes along the edge–fog–cloud continuum. The SBI driver SDN enables the hybrid centralized management of SDN switches and RINA nodes. Meanwhile, the RINA L2VPN allows seamless communication between edge nodes and the cloud to facilitate the data exchange between network functions (NFs). Such integration has enabled a progressive deployment of RINA in current IoT networks without affecting their operations and performance.

Keywords: edge computing; IoT; future networks; L2VPN; NFV; SDN; RINA

Citation: Sarabia-Jácome, D.; Giménez-Antón, S.; Liatifis, A.; Grasa, E.; Catalán, M.; Pliatsios, D. Progressive Adoption of RINA in IoT Networks: Enhancing Scalability and Network Management via SDN Integration. *Appl. Sci.* **2024**, *14*, 2300. https://doi.org/10.3390/app14062300

Academic Editor: Subhas Mukhopadhyay

Received: 7 February 2024
Revised: 29 February 2024
Accepted: 5 March 2024
Published: 9 March 2024

1. Introduction

Future networks envision the evolution of traditional network architectures into more advanced and efficient ones. Along with this vision, several research efforts focus on enhancing the network capacity and data transmission rates to provide low latency and massive connectivity [1]. These new goals are modeling the future network architectures based on new networking paradigms and key enabler technologies such as Software-Defined Networking (SDN), Network Function Virtualization (NFV), and edge computing. Internet of Things (IoT) architectures are taking advantage of the emerging technologies to improve their current architectures [2]. For example, SDN technologies enable centralized IoT network management and programmability. As result, SDN has improved network visibility and dynamic configuration. Furthermore, NFV has facilitated the virtualization of network functions to provide greater flexibility, scalability, and cost-efficiency. Meanwhile, edge computing has enabled data analytics closer to the data source, reducing latency and reducing the large amount of sensor data sent to the cloud [2].

A few years ago, the Recursive InterNetwork Architecture (RINA) was proposed as an alternative to traditional networking architectures mainly based on the Transmission Control Protocol (TCP)/Internet Protocol (IP) stack. RINA is a novel network architecture

paradigm that provides a robust, scalable, and flexible framework for interconnecting computer networks. The foundation of RINA is that a computer network is a distributed application that provides distributed Inter-Process Communication (IPC) services to other distributed applications [3]. The set of distributed IPCs forms a single layer known as the Distributed Inter-Process Communication Facility (DIF). The most important aspect of RINA is its recursion so that the DIF layer can be repeated depending on the network's requirements. RINA avoids redundant mechanism implementation in the current network architectures and extracts as much commonality as possible [4]. As a result, RINA facilitates network customization through policies.

RINA has demonstrated several advantages in terms of scalability [5], flexibility [6], mobility [7], security [8], and performance [9] that are desired by IoT environments. However, RINA has not been studied in IoT environments yet. The current IoT networks rely on SDN technologies to fulfill the IoT requirements, making the adoption of RINA difficult. For this reason, the integration of RINA with SDN technologies aims to lower the barriers of adopting RINA in IoT environments. Such integration enables a progressive deployment of RINA in current IoT networks without affecting their operations and performance. Consequently, this strategical integration will open new areas of ongoing exploration and development for modeling the next generation of IoT network architectures.

However, achieving such integration is challenging because RINA must guarantee seamless interoperability along the edge–fog–cloud continuum. On this account, the IoT devices and the IoT gateways must be able to implement the RINA protocols and layer management functionalities to enable a RINA network at the edge and fog computing levels. On the other hand, the current SDN controllers cannot support the RINA network model. For example, SDN controllers have their own data models to manage their SDN switches, and these data models are quite different from the RINA management layer and transmission protocols. Furthermore, the current IoT applications and services cannot seamlessly use the RINA Application Programming Interfaces (APIs). IoT applications and services must be able to communicate transparently with each other using RINA protocols in the fog–cloud computing levels.

This work addresses the challenges mentioned earlier to enable the integration of SDN technologies and RINA by providing a RINA-based management Distributed Application Facility (DAF), a RINA SouthBound Interface (SBI) driver, and a RINA-Layer 2 Virtual Private Network (L2VPN). The architecture proposed in this work enables the management of RINA nodes and the management of SDN switches. The DAF management enables a DIF layer to establish management connections between the RINA nodes and the SDN controller. A RINA SBI driver is proposed to translate the SDN data models into RINA data models. The RINA SBI driver is a Ryu framework [10] component with a well-defined API to enable the development of control and management applications. Finally, the RINA L2VPN provides a transparent connection between applications that cannot use the RINA API to allocate flows. These developments have been accomplished within the context of the TERMINET [11] project.

In summary, the main contributions and novelties of the proposed work are the following:

- A RINA-based DAF to manage RINA nodes along the edge–fog–cloud continuum. A detailed description of the Management DAF can be found in Section 3.1; moreover, a validation can be found in Section 5.1.1.
- The SBI driver SDN to enable the hybrid centralized management of SDN switches and RINA nodes. A detailed description of its design and implementation be found in Section 3.2; moreover, a performance evaluation can be found in Section 5.2.1.
- The RINA L2VPN to enable seamless communication between edge nodes and the cloud by using RINA to facilitate the data exchange between Network Functions (NFs). A detailed description of RINA L2VPN's design and implementation can be found in Section 3.3; moreover, a performance evaluation can be found in Section 5.2.2.

The remainder of this document is organized as follows: Section 2 reviews the current SDN and NFV technologies with a special focus on IoT environments, and presents the

RINA concept and its current developments. Section 3 describes the proposed architecture to integrate SDN technologies with RINA. Section 4 presents the smart building use case implementation. Section 5 provides information on the obtained results and, lastly, Section 6 compiles the lessons learned and presents the future work.

2. State of the Art

This section presents the state of the art on the main topics related to this work. Mainly, this section describes SDN and NFV technologies in general, and the main features of RINA and its recent advances.

2.1. SDN and NFV in IoT Environments

SDN has demonstrated its benefits widely in managing complex networks. The separation of the data plane and data control has brought about highly dynamic and scalable network operations [12]. The SDN controller centralizes the data control and maintains global visibility of the network state [13]. The SDN controller can monitor, prioritize, and de-prioritize network traffic through APIs [14].

The generic SDN architecture is composed of three layers: (i) the application layer, (ii) the control layer, and (iii) the infrastructure layer, as is shown in Figure 1. The SDN controller is part of the control layer and keeps communication with the entities of the application layer and infrastructure layer through APIs [15]. Furthermore, the SDN controller is responsible for providing security, routing, and network management [16]. The infrastructure layer comprises virtual (e.g., OpenvSwitch or OpenFlow) and physical network devices. The SDN controller communicates with the infrastructure layer through southbound APIs and the application layer through northbound APIs.

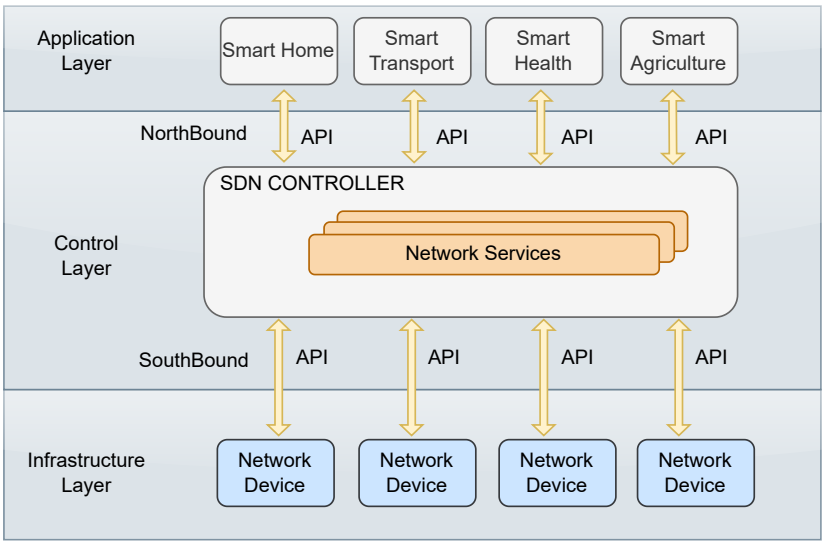

Figure 1. Generic SDN architecture in IoT environments.

The southbound APIs facilitate the SDN controller's ability to control heterogeneous networks. By doing so, the SDN controller can dynamically change forwarding rules installed in switches and routers, among others [17]. OpenFlow is the most used southbound API [18], but it is not the only one available. For instance, the following are implemented southbound APIs: NETCONF, OF-Config, Opflex, as well as some routing protocols such as IS-IS, OSPF, and BGP [17]. OpenFlow controls network devices' routing tables and other hardware using an SDN controller. The SDN controller can remotely configure the flow tables of OpenFlow switches. Each open-flow table consists of the packet header, action, and statistics [13]. In this way, the user can program the behavior of the network.

On the other hand, the northbound interfaces support and enable innovative applications. Currently, most controllers implement the northbound API through the Representa-

tional State Transfer (REST) web architecture, but it can offer other more specialized APIs such as NVP NBAPI and SDMN API [19]. There are other northbound interfaces to abstract the controller functionalities and facilitate the data plane behavior from the application developers [19], but these are more related to programming and query-based languages, such as Frenetic, Procera, and Netcore, among others [13].

Several controller implementations have been proposed to manage, mainly, OpenFlow-based switches. The NOX and the Python-based open source (POX) were the first open-source controller implementation of the OpenFlow protocol based on the C++ and Python programming languages [17]. Other controller implementations have since been proposed for providing a centralized and distributed control based on the C, Java, JavaScript, or Ruby programming languages. These controller implementations include Beacon, Helios, ONIX, ONOX, OpenDaylingt, and the Ryu framework [13]. Furthermore, some implementations aim to manage wireless networks and provide programmable and flexible stack layers for wireless networks, such as openRadio and MobileFlow [19]. This wireless network controller implementation focuses on solving mobility routing issues.

The SDN approach has been used in the design of IoT architectures [20]. SDN demonstrates that it can reconfigure the IoT network according to the environment changes [14]. In addition, SDN technologies can handle IoT requirements in terms of Quality of Service (QoS) guarantee, traffic engineering, and security services. Furthermore, the virtualization of the network functions has demonstrated an enhancement of IoT networks' scalability and flexibility [21].

2.2. RINA-Based Networks

RINA networks are built upon the use of a single type of layer that is being reused. That layer is called DIF. A DIF is no more than a distributed application composed of several Inter Process Communication Processs (IPCPs) that cooperate with each other to provide IPC services to applications of upper DIFs. There are two types of DIFs: shim DIFs and normal DIFs. The shim DIFs are adaptations of existing protocols such as Wi-Fi [22], VLAN [23], and TCP/UDP [24]. The shims facilitate the underlying protocol layer with the IPC service interface. Meanwhile, a normal DIF provides fixed mechanisms for data transfer, data transfer control, routing, access control, and Service Data Unit (SDU) protection, among others. A normal DIF uses the services of underlying DIFs (shims or normal) to transfer data across the network. Figure 2 shows an example of a generic RINA architecture that is composed of normal DIFs and shim DIFs.

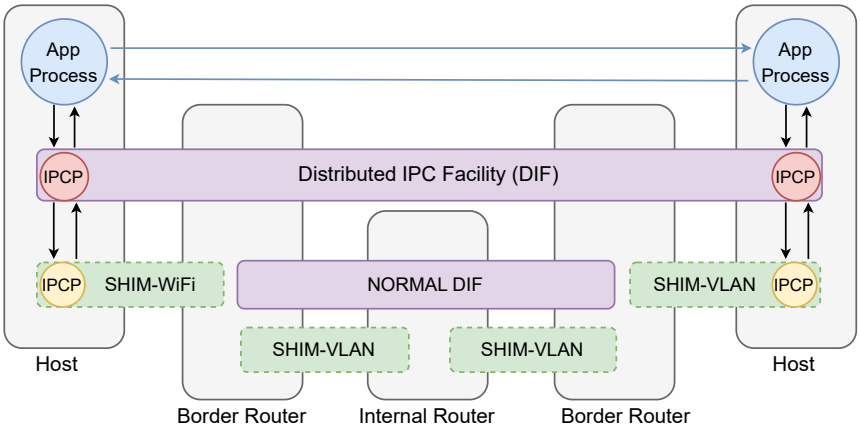

Figure 2. Example of a RINA network architecture.

RINA-based networks are only built using two generic protocols in every DIF: the Error Flow Control Protocol (EFCP) and the Common Distributed Application Protocol (CDAP). No more protocols are required because DIF functionalities (e.g., security, QoS, or routing) are programmed using policies without implementing dedicated protocols.

EFCP oversees the data transfer and data transfer control. EFCP provides tightly coupled mechanisms for transferring SDUs, such as flow addressing or sequencing. Furthermore, EFCP provides loosely coupled mechanisms, such as flow control or re-transmission control, for data transfer control. Meanwhile, CDAP oversees layer management functions such as enrollment, namespace management, flow allocation, resource allocation, routing, and security coordination [4]. CDAP operates on remote objects used by all the layer management functions. All the objects and states are modeled and stored on the Resource Information Base (RIB). The RIB objects model includes the object naming, relationships between objects—inheritance and containment, among others—the object attributes, and the CDAP operations that can be applied to them [4]. The RIB daemon manages access to the RIB and exchanges CDAP Protocol Data Units (PDUs) with RIB daemons of neighbor IPCPs. The only operations that can be performed on objects are: create/delete, read/write, and start/stop. Table 1 details the CDAP messages and their purposes to operate over RIB objects. The RIB daemon implements the Common Application Connection Establishment Phase (CACEP), which exchanges naming information with peers. CDAP helps implement layer management functionalities such as enrollment, flow allocation, and routing.

Table 1. CDAP messages and purpose.

Message Name	Purpose
M_CREATE	Create a RIB object
M_CREATE_R	Response to M_CREATE, carries the result of a create request
M_DELETE	Delete a specified object
M_DELETE_R	Response to M_DELETE, carries the result of a deletion attempt
M_READ_R	Response to M_READ, returns the result and (possibly incomplete) value of the object
M_WRITE	Writes a specified value to a specified object
M_WRITE_R	Response to M_WRITE, carries the result of a write operation
M_START	Start the operation of a specified application object, typically used when the object has operational and non-operational states
M_START_R	Response to M_START, indicates the result of the start operation
M_STOP	Stops the operation of a specified application object, typically used when the object has operational and non-operational states
M_STOP_R	Response to M_STOP, indicates the result of the stop operation

RINA-based networks have demonstrated advantages regarding network management compared to traditional networks. A RINA-based network reduces the number of specialized protocols to perform some network features. For instance, RINA has demonstrated that it does not require particular protocols or mechanisms to support mobility by providing a complete naming scheme [25]. Naming the interface instead of the node raises a relevant issue in enabling mobility, and it is solved by using specialized protocols for setting up tunnels [4]. Furthermore, RINA has demonstrated its capacity to support transport over heterogeneous networks [9] as well as its simplicity in managing multiple layers in the multi-tenant data center network scenario [6].

The use of RINA in large-scale networks, such as the ones found in NFV-based environments, is a topic that has been widely discussed. Previous research showcases the advantages of RINA networks in large-scale networks. On the one hand, ref. [6] discusses the advantages of managing a large data center RINA network in comparison with managing a similar IP-based network. On the other hand, ref. [24] presents and discusses several Key Performance Indicators (KPIs) regarding configuration management, software complexity, and automation in large-scale networks.

These large-scale scenarios bring diverse and challenging requirements, such as a broader range of performance, cost, security protection, and mobility management. Network slicing has emerged as a potential approach to fulfill such requirements [26]. In [27] is presented what the authors called IPC Virtual Private Network (VPN) (Inter-Process Communication Virtual Private Network) slicing. IPC VPN is a RINA-based network slicing solution that focuses on connecting groups of applications instead of connecting endpoints with IP or Media Access Control (MAC) addresses. The goal is no longer focused on devices and physical location but on providing an isolated and fit-for-purpose connectivity domain to instances of distributed applications, regardless of their physical location. Following this IPC VPN slicing approach, recent research has been conducted in the framework of the Open-VERSO project [28], proposing RINA L2VPN, a virtual networking solution to provide tailored connectivity to virtualized NFs. In [26], RINA L2VPN is briefly presented as a novel network slicing solution, and it is compared with other network slicing solutions.

Currently, a RINA L2VPN slicing solution is being implemented, and its preliminary results were presented in [29]. In such an implementation, performance is one of the key features we have in mind to support the transmission of high loads, e.g., for media use cases running in 5G and beyond 5G infrastructures. In this regard, the current implementation, as proved in [29], overcomes by 30–50 times the performance provided by IRATI [22]—the most generic and mature RINA implementation to date—in terms of speed—packets processed per second—and throughput. The high throughput result was achieved by implementing the data path with netmap [30] instead of the Unix kernel stack. The advantages of a netmap-based implementation are twofold. First, it provides relatively high throughput with a significant amount of flexibility; since the implementation of the software program is only bound by the constraints of compute hardware instruction sets. Second, the diversity of supported available hardware platforms—Linux and FreeBSD systems.

3. Architecture Proposed

This section describes the architecture proposed to manage a RINA network for IoT environments by using an SDN controller. Figure 3 presents the layered RINA and SDN integration architecture proposed in this work.

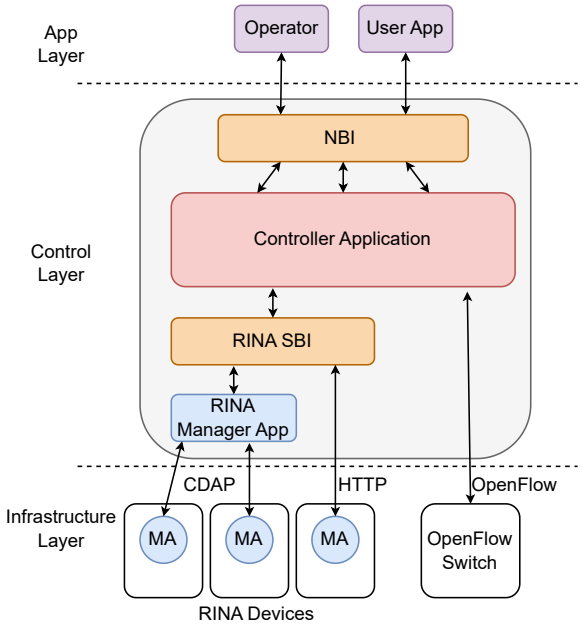

Figure 3. RINA and SDN integration architecture proposed.

The proposed architecture follows the SDN generic architecture that is based on three layers: the application layer, control layer and infrastructure layer, to facilitate the adoption of RINA in the current SDN networks. The architecture presents three key components in

the control layer and the infrastructure layer to enable the management of RINA devices (RINA-based IoT devices, RINA-based IoT gateways, RINA-based EdgeNodes, and the RINA-based core). These components are the RINA Management Agent (MA), the RINA manager application, and the RINA SBI.

The RINA MA enables the control of RINA devices. There are two kinds of MAs: CDAP-based and the Hypertext Transfer Protocol (HTTP)-based. The CDAP-based MA follows the RINA principles of design and implementation detailed in [4]. Meanwhile, the HTTP-based MA is a particular solution for enabling the progressive adoption of RINA in cloud-based scenarios. Specifically, the HTTP-based MA facilitates the deployment of L2VPN to enable seamless communication between edge nodes and cloud application by using RINA capacities. Section 3.3 presents the L2VPN functionalities and explains the HTTP-based MA functionalities.

The RINA manager application facilitates the control of RINA devices under the same management DAF by using CDAP messages. A DAF is the collection of two or more cooperating application processes that exchange information using DIF's IPC services. In this case, the RINA manager and the CDAP-based MA make up the management DAF. Section 3.1 explains the management DAF's main characteristics and functionalities.

The RINA SBI driver abstracts the RINA manager and HTTP-based MA functionalities to enable a seamless integration with current controller applications. Section 3.2 details the RINA SBI driver design and implementation.

3.1. Management DAF

The network management DAF is composed of several application processes that enable management characteristics in RINA nodes or RINA systems. In this case, the application processes are the MAs implemented in each RINA system and the RINA manager implemented in the SDN controller. Figure 4 shows the management DAF composed of the MAs and the RINA manager.

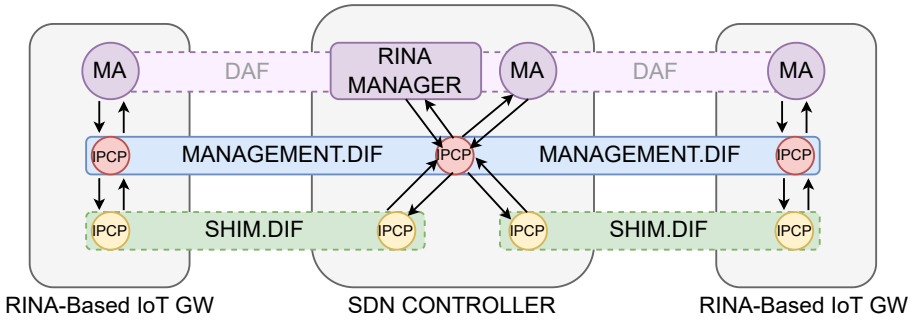

Figure 4. Network management DAF.

The MAs are in charge of running one or more IPCPs for implementing one or more DIFs. To do so, the MAs have read and write permissions to operate over the IPCPs' RIB. In this case, the MAs can operate over the management DIF IPCPs, as shown in Figure 4. This capability allows for creating and deleting RIB objects and setting up DIF's configurations—e.g., routing table, and security policy, among others. The RINA manager controls the MAs through the interchange of CDAP messages. In that way, the RINA manager can send commands to operate over the IPCPs' RIB. In the architecture proposed, there are some MAs deployed on cloud infrastructures that are TCP/IP based and do not support the RINA protocols (EFCP and CDAP). As result, the MA deployed on cloud infrastructures cannot interchange CDAP messages to operate over the IPCPs' RIB. To overcome this limitation, MAs based on HTTP were designed and implemented to be deployed in these kinds of scenarios (cloud-based scenarios). In these cases, the HTTP-based MAs expose a well-defined REST API to facilitate the management from the SDN controller. These HTTP-based MAs do not follow the RINA MA principles of design detailed in [4] because they are considered a particular solution for enabling the progressive

adoption of RINA in the current networks' architectures. More details about the HTTP-based MA's design and implementation are explained in Section 3.3.

On the other hand, the RINA manager is the core of this centralized network management DAF and must be implemented in the SDN controller. The RINA manager oversees the communication with the agents in each system of its management domain to provide central configuration, fault, security, and performance management. There is the possibility of implementing more than one RINA manager and reduce the area each RINA manager covers. In other words, the IoT network can be split into several management subareas, with an SDN controller for each subarea. This approach reduces the IoT network management complexity.

The application processes require a single DIF dedicated to interconnecting the RINA manager with each MA. Figure 4 shows how the management DIF is implemented for communicating the RINA manager with the MAs of each system to be controlled. The RINA manager comprises the configuration manager module and the events manager. The configuration manager is responsible for sending a request to the agent management to create or delete IPCPs by sending CDAP messages. Meanwhile, the events manager receives and handles the events sent by the agent managements [31].

Once the single dedicatedDIF has been established, the MAs intend to communicate with the RINA manager. To do so, the MAs and the RINA manager change several CDAP messages in the first stage of shaping the DAF management. Figure 5 shows this initial process. The MAs intend to send an `M_CONNECT`—CDAP message—to the RINA manager. This message contains information about the source application process, destination application process, and authentication mechanism employed. The application process will change more messages depending on the authentication mechanisms selected in each case. Once the authentication mechanisms have finished, the RINA manager responds with an `M_CONNECT_R`. Immediately, the RINA manager sends an `M_READ` to request a reading of the remote RIB objects. The MAs react with as many `M_READ_R` messages as the MA has RIB objects. Once this initial stage is finished, the RINA manager is ready to start operations.

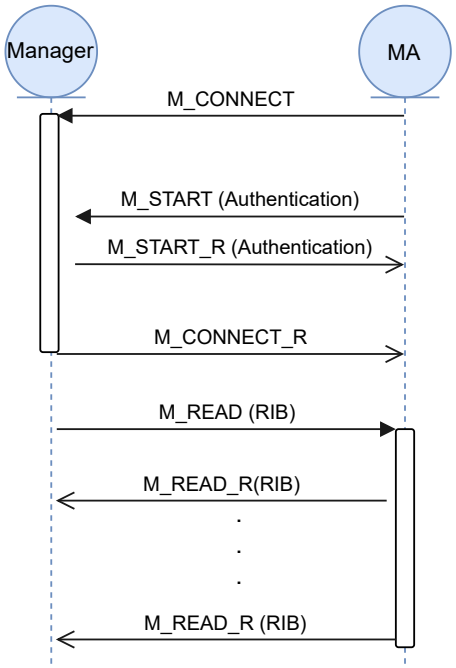

Figure 5. Initial process of CDAP message sharing.

3.2. RINA SBI Driver

The RINA SBI driver serves as the API to communicate between the SDN controller, the RINA manager application process, and the HTTP-based MA. The RINA SBI driver is

responsible for translating the application request into the RINA data model descriptor file (DIF configuration templates), managing the descriptor files (creating or deleting DIF configuration templates), managing the connection with the RINA manager (connecting or disconnecting), managing the connection with the HTTP-based MAs in the cloud, and sending an operational request so that the RINA manager executes it.

The driver abstracts the current functionalities of the RINA manager application process and HTTP-based MAs. The driver was developed as a library in Python to facilitate the development of management network applications based on the Ryu Framework SDN controller (https://github.com/faucetsdn/ryu) (accessed on 15 June 2023). The library is composed of two main classes: the `RinaManager()` that makes an abstraction of the RINA manager, and the `RinaCoreManager()` that makes an abstraction of the HTTP-based MAs. An instance of these classes allows making a connection with the desired MA (CDAP-based and HTTP-based) to be controlled, as is shown in Figure 3. This RINA manager-instanced object requires the following initialized parameters: the RINA manager socket path and the path to store the description files. The description files contain the RINA nodes that will take part of the DIF and the specific DIF configuration options, such as routing algorithm, QoS supported in that DIF, the security schemes used, and the RINA nodes' names and addresses that are part of the DIF. More details about the DIF configuration options are found in the IRATI GitHub repository (https://github.com/IRATI) (accessed on 10 May 2023). Meanwhile, the RINA core manager-instanced object requires the following initialized parameters: the MA Uniform Resource Locator (URL), a system identifier (ID), and a short description of the MA. In this case, the DIF specifications are not supported because the HTTP-based MA has as only functionality to create and delete L2VPN.

Moreover, the `RinaManager()` and the `RinaCoreManager()` objects offer a set of methods related to the DIF manager functionalities that each MA supports. Each method creates the operation request, arranges it in a fixed format, and sends it to the DIF manager. Meanwhile, the DIF manager receives the requested operation and operates the management of DAF members. The current methods supported are described as follows:

- `list_systems()`: Makes a list of systems that the RINA SBI driver can control. Each RINA node is considered a system. Each system has its ID, the MA name, and the RINA port's ID exposed in the DAF.
- `create_ipcp(systemId, IpcpDescription)`: Creates an IPCP in the requested system using the IPCP template file. It returns the remote IPCP's created ID.
- `create_dif(DifDescription)`: Creates a DIF following the DIF template file. The DIF template file must indicate the systems' ids that will participate in the DIF.
- `destroy_ipcp(systemId, IpcpId)`: Destroys a specific remote IPCP by using as parameters the remote system's ID and the IPCP's ID.
- `destroy_dif(difName)`: Destroys the group of IPCPs that take part of the requested DIF name.
- `create_L2VPN(systemId)`: Creates an IPCP to enable an L2VPN in the indicated system. This operation is supported only by the HTTP-based MAs.
- `delete_L2VPN(systemId)`: Destroys a specific remote IPCP in the indicated system. This operation is supported only by the HTTP-based MAs.

The above operations are currently available because they are required for the context of the TERMINET project and its use case validation. However, the object-oriented programming approach chosen for the SBI driver development ensures ease and extensibility in the future to improve the current version and support more operations. The RINA SBI driver GitHub repository (https://github.com/esmaxness/RinaDriver) (accessed on 15 June 2023) provides some examples for an easy understanding of the SBI execution and more details.

3.3. RINA L2VPN

The proposed RINA L2VPN networking solution provides a framework to deploy a network overlay on top of a RINA network in the form of Layer two (L2) networks. The L2VPN

overlay allows non-RINA network services made up of distributed NFs to allocate flows in the underlying RINA network, without making use of the RINA API. In this way, the network service takes advantages of RINA network capabilities without the need to use the RINA APIs. This approach is useful to enable seamless communication between edge nodes and cloud applications or NFs, either in the form of Container Network Functions (CNFs) or Virtual Network Functions (VNFs). Figure 6 shows a network service formed by two NFs.

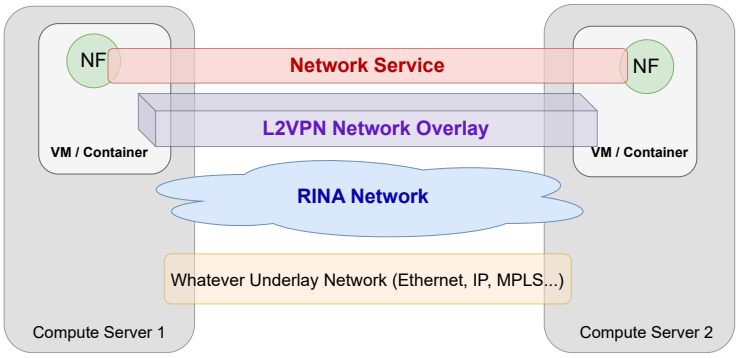

Figure 6. Generic diagram of the relevant layers involved in the RINA L2VPN solution (front view).

RINA L2VPN offers horizontal scalability to support several network overlays on top of a RINA network, i.e., the last DIF of the network. In this way, several L2VPNs overlays can be deployed on top of the same DIF network depending on the network service demands. Furthermore, the solution can group several network services into a network overlay to isolate the traffic generated by these NFs. Figure 7 shows a high level representation of several independent network overlays supported in the same RINA network, as well as network services running on top of these overlays.

It is important to note that RINA L2VPN is implemented on top of netmap [30], a fast packet processing framework that enables high performance packet switching, programmable via a set of APIs at user space. That way, RINA L2VPN is able to provide higher packet data rates than traditional Linux networking mechanisms.

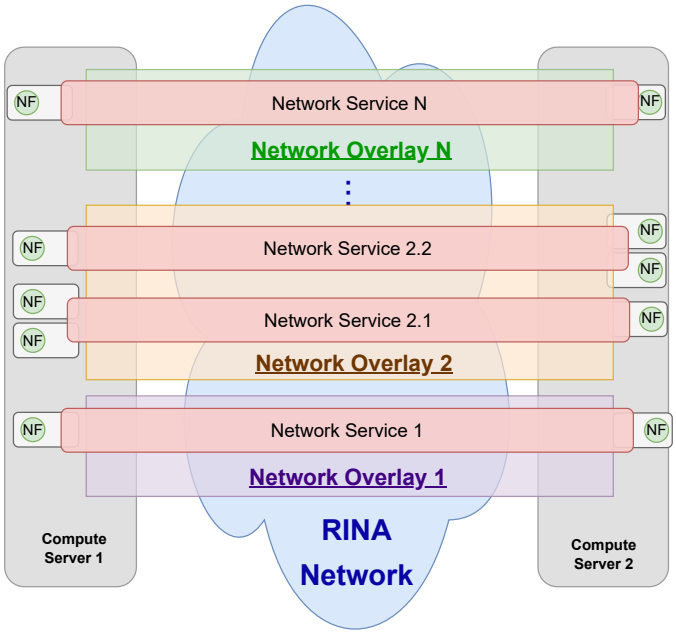

Figure 7. Generic diagram of the relevant layers involved in the RINA L2VPN solution (zenithal view).

The HTTP-based MAs take a key role in the deployment of the RINAL2VPN solution on cloud-based applications. These HTTP-based MAs are in charge of (i) slice allocation

and de-allocation requests, (ii) managing the creation of L2 switches using the L2 Factory Module—see Figure 8—and (iii) exposing an interface to interact with upper management layers. There is one HTTP-based MA per compute node, and its interfaces are depicted in Figure 8. The HTTP-based MA exposes a well-defined northbound interface (NBI) to the upper management layers, so the RINA SBI can interact with it. On the other hand, the HTTP-based MA exposes an SBI—the IPCP Connector in Figure 8—directly attached to the IPCP, in order to ask to allocate or de-allocate the necessary RINA flows needed to deploy a slice. This SBI is implemented using the netmap API, the endpoints exchange a domain-specific language codified using protocol buffers. The domain-specific language contains the following fields:

1. `opcode`. There are six operational codes to identify which operation the remote MA should execute:
   - for creating an L2 overlay: `M_ACK`, `M_ALLOCATION_REQ`, `M_ALLOCATION_RSP`.
   - for deleting an overlay: `M_NACK`, `M_DELETE_REQ`, `M_DELETE_RSP`.
2. `invoke_id`. Sequence number of the message. Every pair of request–response shares the same invoke ID; hence, it is used to handle asynchronous requests and responses.
3. `slice_id`. Network overlay unique identifier. In RINA terminology, the slice identifier maps to the application entity, which refers to a task within an application process directly involved with exchanging application information with other application processes. In other words, it is used to identify the instance of the network overlay that the message refers to.
4. `serialised_flow_msg`. Serialized object value that is directly passed without any processing to the IPCP. It contains information that the IPCP needs to allocate the flow within the neighbor IPCP.

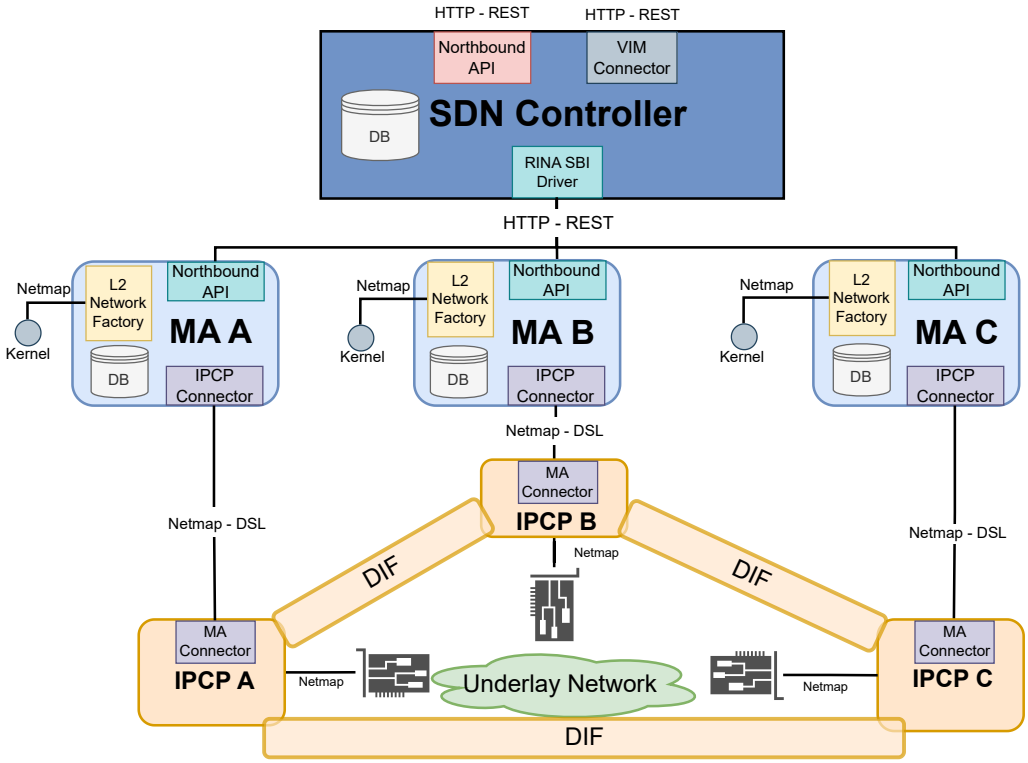

Figure 8. RINA L2VPN's overall architecture and interfaces.

The SDN controller can control the HTTP-based MAs by using the RINA SBI. The RINA SBI is responsible for requesting the needed resources to every HTTP-based MA involved in the slice. Every slice creation or deletion request is processed with the reception of an HTTP request on the REST endpoint `notify` of the NBI. Figure 9 shows the message

interchanged between the RINA SBI driver and the HTTP-based MAs to create the slice. In order to create an L2 network overlay between two MAs, one takes the role of *master* and the other takes the role of *slave*. The *master* will start the flow allocation process and the *slave* will be awaiting the allocation request from the other MA. By looking at Figure 9, it can be seen that the SBI driver starts the L2VPN overlay creation by sending a message to `MA_A`, stating that the remote MA involved is `MA_B`. In this L2VPN creation, `MA_A` will perform the role of *slave*, i.e., it will be awaiting `MA_B`'s flow allocation request. On the other hand, the SBI driver sends a message to `MA_B` stating that the remote MA is `MA_A` and its role in the overlay creation is *master*. Immediately, `MA_B` starts the L2VPN creation by sending a RINA flow allocation request to `MA_A`.

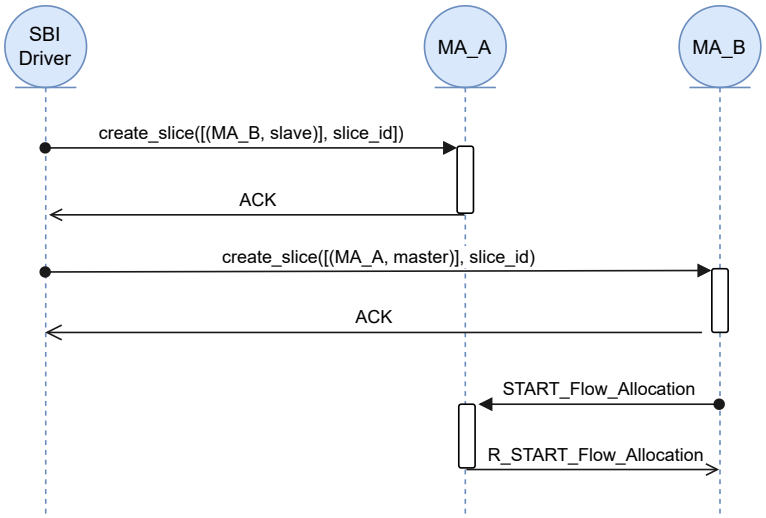

Figure 9. Message exchange between RINA SBI Driver and two MAs for the creation of a network slice.

4. Use Case: Smart Buildings

Smart buildings' concept aims to manage properly building resources and optimizing them by using advanced technology and automation for enhancing the comfort, safety, and productivity of its occupants [32]. To achieve this objective, smart buildings leverage a range of technologies, including sensors, data analytics, machine learning, and Artificial Intelligence (AI), to collect and analyze data on various aspects of building performance, such as energy consumption, indoor air quality, occupancy, and security [16]. A smart building is a complex system that is composed of several subsystems such as lighting, Heating Ventilation Air Condition (HVAC), security, and energy management, among others. Smart buildings' complexity directly affects network performance and management. Consequently, smart buildings are facing network management complexity, scalability, and security issues.

RINA has the potential to offer several benefits to smart building scenarios such as hierarchical management, scalability, service differentiation, and security, among others. The RINA recursion enables the network designer to re-use a single DIF as many times as it is required. In smart buildings, each DIF can be responsible for a particular function and communicate with adjacent DIFs in a hierarchical manner. Furthermore, as the building's needs change or new devices and sensors are added, additional DIFs can be dynamically created to accommodate the growing network requirements. In fact, each DIF can be designed to allow the prioritization and management of different services with the smart building network. In addition, RINA is secure by design, so a DIF is a secure layer with highly customizable privacy and security levels based on policies.

This section details the proposed architecture implementation in a smart building scenario to enhance the network scalability and management by using the RINA approach.

4.1. Indoor Environmental Subsystem Scenario

A smart building has already implemented several subsystems such as energy, lighting, HVAC, flooring, and motion sensors. The current smart building subsystems rely on technologies such as Wi-Fi, Bluetooth, ZigBee, Message Queuing Telemetry Transport (MQTT) and SDN as communication technologies. The smart building architecture follows an edge–fog–cloud continuum approach, that was demonstrated to be suitable for reducing the large amount of data generated by sensors [33]. In other words, the smart building has an edge node that can process data, analyze data, and make decisions. By monitoring and analyzing data in real-time, smart buildings can make automatic adjustments to various subsystems, such as heating, cooling, lighting, and ventilation, to ensure optimal performance and energy efficiency. The smart building system follows a centralized learning approach. Typically, the centralized learning approach centralizes the data from sensors in the cloud and trains AI models that can be distributed and applied to all devices [34]. Consequently, historical data is stored in the cloud infrastructure, to be used to train AI models to optimize the smart building resources. The trained models are deployed to the edge node for making useful predictions to automate decision making. Thus, the smart building has distributed application and services along the edge–fog–cloud continuum that require a continuous communication between the edge node and the cloud infrastructure.

An indoor environmental subsystem based on RINA was designed, implemented, and deployed at a smart building following the architecture proposed to integrate this subsystem with the current SDN-based system. Figure 10 shows the subsystem architecture for the analyzed scenario. The subsystem is composed of several RINA-based sensors to measure the temperature, humidity, and carbon dioxide (CO_2). These sensors communicate to the edge node RINA-based application by using a RINA-based IoT gateway (more details are provided in Section 4.2). The RINA-based application receives data that come from the sensors and inserts them into an InfluxDB database system. The InfluxDB database system contains the data from all smart building subsystems. In other words, this database receives data through a MQTT broker from the lighting, energy management, security, and fire subsystems that are already working on the building with SDN OpenFlow switches and other IoT wireless technologies (Wi-Fi, Bluetooth, and ZigBee) and communication protocols.

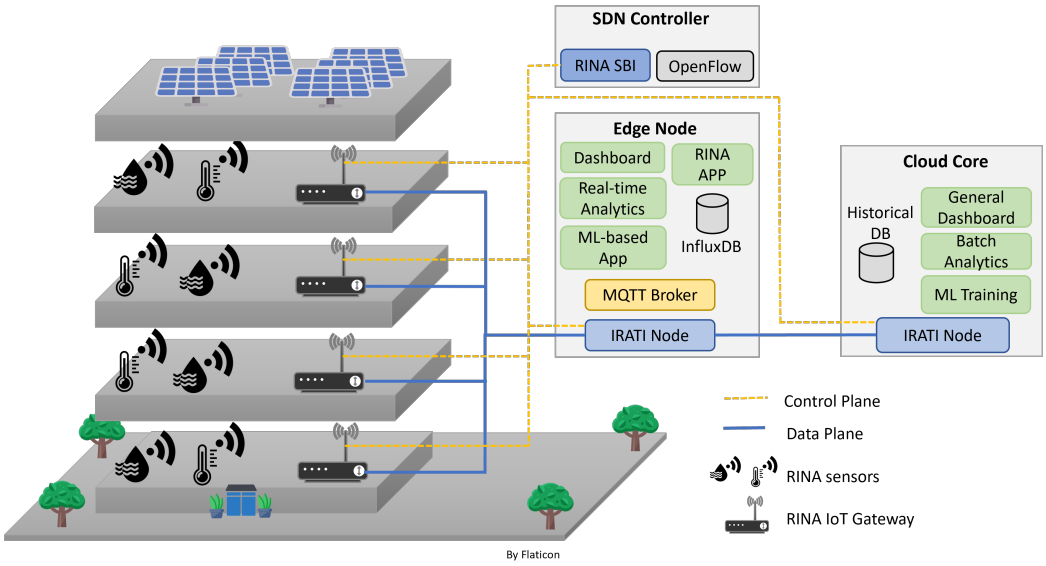

Figure 10. Indoor Environmental subsystem high level architecture.

The real-time analytics application and predictive services are fed by using the data within InfluxDB. The data process application is in charge of three main tasks: (i) aggregating the sensors' data to reduce the large amount of data to be transferred to the cloud, (ii) filtering

and preparing data for predictive analytics carried on by the AI models, and (iii) calculating descriptive analytics (e.g., mean, maximum, and minimum values) to be published on a dashboard. Meanwhile, the predictive services are AI-based applications that make predictions to make automatic adjustments in the smart building subsystems. The current distributed data process application and predictive services were designed for receiving support from the TCP/IP stack. This fact complicates the use of the RINA network to enable communication between the application and predictive services deployed on the edge node and the cloud core. To overcome this limitation, the RINA L2VPN provides an innovative solution to TCP/IP-based applications for using a RINA network without changing the current applications and services. Consequently, end-to-end RINA flows are smoothly allocated to take advantage of the RINA performance derived from the protocols' headers reduction, previously analyzed in [35,36].

4.2. Implementation

The scenario described was implemented in a controlled environment. The following subsubsections describe the hardware and software employed to build the RINA network implementation and describe the integration of the RINA SBI with an SDN controller.

4.2.1. Rina Network Implementation

The RINA network comprises five types of nodes (IoT devices, RINA-based IoT Gateway, EdgeNode, Core, and SDN controller), as Figure 11 shows. The DAF management was implemented using the aforementioned nodes configured with the latest version of IRATI open-source software v2.3.0 (https://github.com/IRATI/stack) (accessed on 10 May 2023) [31]. IRATI enables a fast prototyping of Linux-based RINA nodes.

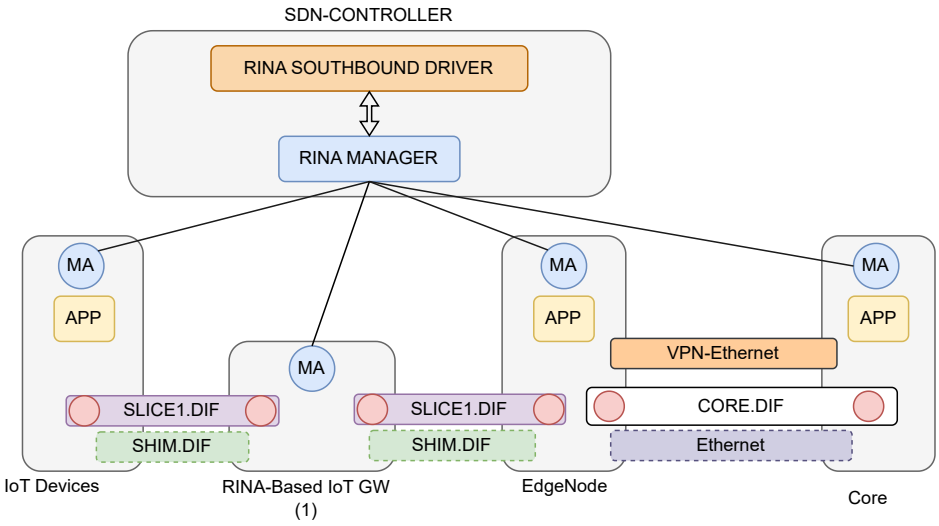

Figure 11. RINA architecture for the use case.

The IoT devices are RINA sensors that are in charge of collecting temperature and humidity data. These sensors have been implemented using the ESP32 IoT platform and the DHT22 sensor. Furthermore, the RINAsense open-source software was employed to develop the temperature and humidity application. The RINAsense (https://github.com/Fundacio-i2CAT/rinasense) (accessed on 15 June 2023) is based on the FreeRTOS real-time embedded Operative System (OS) that allows a rapid prototyping of RINA sensors for RINA networks [36].

The RINA-based IoT gateway was implemented using the IRATI open-source software [31], and using as hardware a Raspberry Pi 4 with Debian Buster OS, Quad-core Cortex-A72 (ARM v8) 64-bit SoC @1.8GHz, and 4GB SDRAM. IRATI has been developed with a GNU/Linux based OS, and this software can be deployed on a Raspberry Pi.

The EdgeNode and the Core were implemented using two Intel® NUC9VXQNX with Intel® Xeon® E-2286M 2.40GHz with 32 GB of RAM, running Ubuntu 22.04. Furthermore, the Docker platform was employed in the EdgeNode to ease the deployment of applications in Docker containers. The applications deployed in Docker were the InfluxDB, Mosquitto MQTT Broker, Graphana dashboard, Python-based ML application, and Apache Kafka stream analytics. In addition, a RINA application was deployed on the edge node responsible for collecting data from RINA sensors (temperature and humidity) and ingesting them into the InfluxDB. It is relevant to highlight that only this RINA application used the RINA APIs to write into RINA flows. The other applications did not use RINA APIs because they are based on TCP/IP stack. However, these applications would communicate with their peers in the core using the L2VPN to enable the progressive adoption of RINA in current TCP/IP networks.

The SDN controller was implemented using as hardware an Intel® NUC 11 Performance Core i5-1135G7 4.2 GHzwith Ubuntu 20.04 LTS OS. Furthermore, it required the IRATI open-source software for deploying the RINA manager and the UoWM SDN Controller software. More details about the SDN controller will be detailed in Section 4.2.2.

4.2.2. SDN Controller Integration

The Ryu SDN framework is a widely acclaimed open-source SDN controller, implemented in the Python programming language, known for its user-friendly and highly extensible nature [37]. Leveraging Ryu's Software Development Kit (SDK), developers can craft sophisticated SDN applications with ease, enabling network monitoring, data plane statistics collection, and the enforcement of intricate network policies [38]. Figure 12 illustrates the core modules of the Ryu framework structured in a layered fashion. The lower layers house basic protocols and message handlers, while the middle layer consists of core modules like event dispatcher, responsible for delivering events to registered applications, core libraries such as topology, and basic classes offering a unified API. At the higher levels, Ryu built-in applications and user-defined applications (RyuApps) reside. RyuApps are SDN applications that perform routing, load balancing, security, or expose a REST API for third-party tools and services. Finally third-party SDN applications can also interact with the Ryu SDN controller using the REST API.

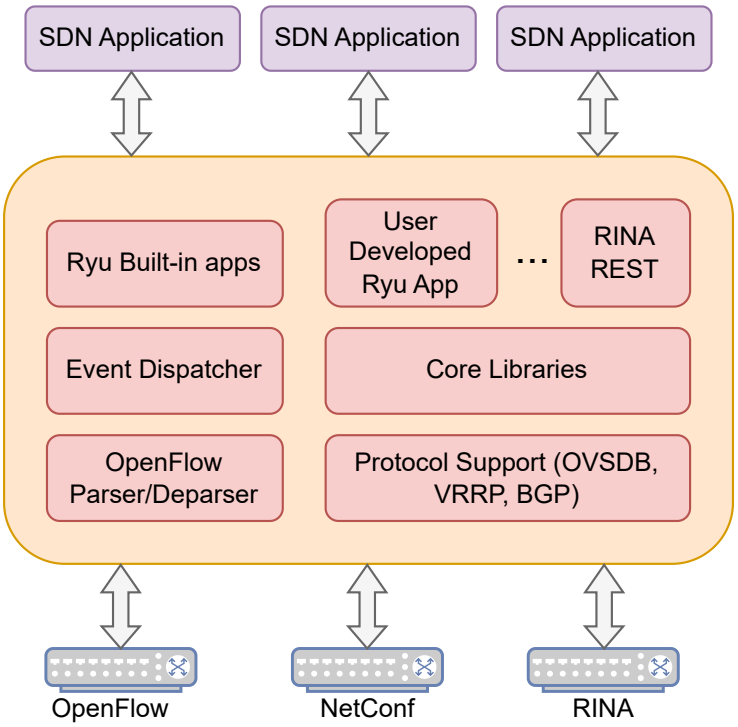

Figure 12. Ryu-based SDN controller integrating RINA capabilities.

The RyuApp developed in this work extends the existing REST API framework to support RINA commands. The RyuApp communicates with the RINA SBI driver using a UNIX socket and exposes common commands to create and destroy RINA objects, and obtain information from the RINA-based data plane using the commands of the SBI driver detailed in Section 3.2 RINA SBI Driver.

5. Evaluation

The architecture instance implementation was evaluated and validated to determine its performance and limitations. The SDN controller demonstration is shown in the first part of this evaluation. Then, the architecture components (RINA IoT gateways and RINA L2VPN) are evaluated in terms of the quantity of useful information (goodput).

5.1. SDN Controller Evaluation

The SDN controller and the RINA SBI driver were evaluated in terms of their ability to interact with the RINA nodes—functional testing. In addition, the Appendix A shows the SDN controller dashboard functionalities.

5.1.1. Funtional Testing

The main objective was to validate the SDN controller's ability to control RINA-based network devices. The functional testing is divided into two parts: (i) validation of DAF management with RINA manager and MAs, and (ii) validation of RINA SBI to translate requests from the SDN controller.

The first validation consisted of verifying the IRATI console of the edge node to check if the IPCP was enrolled into the management DAF. During enrollment, the RINA manager interchanges CDAP messages with the MA to establish the DAF management, as described in Section 3.1. Figure 13 shows the edge node console, and we can notice in the last line that the *edgeNode1.management* was assigned to DIF management. In this case, the *edgeNode1* MA was accepted as part of the management DAF and interchanged CDAP messages with the *terminet.manager-1*. Furthermore, the RINA SBI registered MA as a new system and assigned a system ID. This register is relevant to keep track of the MA connected into the DAF and to return as systems information when the SDN controller requests the list_systems API.

```
Management Agent name: edgeNode1-1—
Management Agent active connections  ( Manager name | via DIF )
     terminet.manager-1-- | management

Current IPC processes (id | name | type | state | Registered applications | Port-ids of flows provided)
1 | edgeNode1.10:1:: | shim-eth-vlan | ASSIGNED TO DIF 10 | - | -
2 | edgeNode1.100:1:: | shim-eth-vlan | ASSIGNED TO DIF 100 | edgeNode1.management-1-- | -
3 | edgeNode1.management:1:: | normal-ipc | ASSIGNED TO DIF management | - | -
```

Figure 13. *EdgeNode1* IRATI management console.

The second validation consisted of creating a slice (RINA DIF) using the SDN controller. The RINA SB driver translates the SDN controller data model into the RINA data model to create DIFs. The DIFs' creation request is sent to the *edgeNode1*. The RINA SB driver interchanges CDAP messages with the *edgeNode1*, as part of the creation process. Figure 14 shows the CDAP messages interchanged with the *edgeNode1* (`M_CREATE` and `M_CREATE_R`). The first CDAP message (1740 Bytes) contained the IPCP configuration description (IPCP name, N-1 DIF, and name of the DIF to be enrolled) and the DIF configurations (QoS Cube, routing, namespace directory, enrollment task, resource allocator, and EFCP configurations). Once the IPCP was created and enrolled successfully, the *edgeNode1* agent sent the result of the request (result:0 means success) using an `M_CREATE_R`. Figure 15 shows the console of

the *edgeNode1*, and we can notice in the last line that there is a new IPCP (*edge1.slice1*) and it was assigned into the *slice1* DIF.

The other functionalities (destroy DIF, destroy IPCP, create IPCP) were tested following the same procedure as the create DIF showed. In those cases, we validated that the RINA SBI drive translated the SDN controller requests into CDAP messages and sent them to the *edgeNode1* using the CDAP messages (`M_CREATE` and `M_CREATE_R`).

5.2. Performance Analysis

The performance analysis intended to show the performance of the RINA network and it was split into RINA IoT gateway performance and RINA L2VPN performance.

```
2852(1665493886)#cdap (DBG): Sent CDAP message through port 7 of 1740 bytes:
Opcode: 4_M_CREATE
Flags: 0
Invoke id: 1
Object class: IPCProcess
Object name: /csid=1/psid=1/kernelap/osap/ipcps/ipcpid
Scope: 0

2852(1665493886)#net-manager (DBG): Read 104 bytes
2852(1665493886)#cdap (DBG): Received CDAP message from port 7
Opcode: 5_M_CREATE_R
Flags: 0
Invoke id: 1
Object class: IPCProcess
Object name: /csid=1/psid=1/kernelap/osap/ipcps/ipcpid=4
Object instance: 4
Result: 0
```

Figure 14. *EdgeNode1* log during CDAP messages interchanging.

```
Management Agent name: edge1-1—
Management Agent active connections  ( Manager name | via DIF )
      terminet.manager-1-- | management

Current IPC processes (id | name | type | state | Registered applications | Port-ids of flows provided)
1 | edge1.10:1:: | shim-eth-vlan | ASSIGNED TO DIF 10 | edge1.slice1-1-- | -
2 | edge1.100:1:: | shim-eth-vlan | ASSIGNED TO DIF 100 | edge1.management-1-- | -
3 | edge1.management:1:: | normal-ipc | ASSIGNED TO DIF management | terminet.manager-1-- | 104, 103
4 | edge1.slice1:1:: | normal-ipc | ASSIGNED TO DIF slice1 | - |
```

Figure 15. *EdgeNode1* IRATI management console after CDAP messages.

5.2.1. RINA IoT Gateway Performance

The main goal of this performance evaluation was to measure the quantity of useful information (goodput) transmitted by the RINA IoT gateway when the number of RINA sensors connected to the RINA IoT gateway increases. Then, the RINA IoT gateway performance was measured on the shim Wi-Fi DIF and shim eth-VLAN DIF.

The first evaluation was focused on determining how the number of flows affects the goodput of unreliable with re-transmission control flows over shim Ethernet VLAN DIF by using the rinaperf application [39]. The test consisted of allocating unreliable flows with the re-transmission control policy activated, and sending 10,000 packets of different sizes. This rinaperf configuration follows the recommendations detailed on the rlite open-source repository [39]. Figure 16 shows the goodput of unreliable flows with re-transmission in the function of the number of flows for different SDU sizes and the test for diverse SDU sizes. The curves

in the figure show the expected goodput degradation as far as the increasing number of flows that the RINA-based IoT gateway supports. Since the allocated flows are unreliable, the goodput achieved is lower than the theoretical throughput in a fast Ethernet network card and higher than the unreliable flows with transmission control. The maximum goodput achieved with 1400 bytes of SDU size is 34.5 Mbps. This is 1/3 of the theoretical throughput 100 Mbps, and this significant degradation is caused by the transmission control overhead.

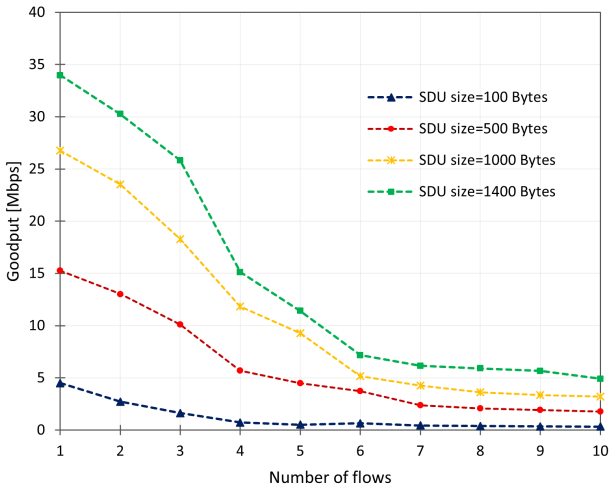

Figure 16. Goodput of unreliable flows with re-transmission control.

In contrast to the first test, this test allocated unreliable flows without re-transmission control, so the allocated flows are similar to a User Datagram Protocol (UDP) connection. The measurement was performed similarly to the first experiment. Figure 17 shows the goodput of unreliable flows in the function of the number of flows for different SDU sizes and the test for diverse SDU sizes. Similar to the first experiment, the curves show a degradation of goodput as far as the increasing number of flows that the RINA-based IoT gateway supports. Since the allocated flows are unreliable, the goodput achieved is lower than the theoretical throughput in a fast Ethernet network card and higher than the unreliable flows with transmission control. This variability is the expected behavior because this case does not use the transmission control mechanism that affects the goodput, as was observed in the unreliable flows with the transmission control case. The maximum goodput achieved was 72.2 Mbps with SDUs of 1400 bytes. This goodput is more than 2/3 of the theoretical throughput (100 Mbps).

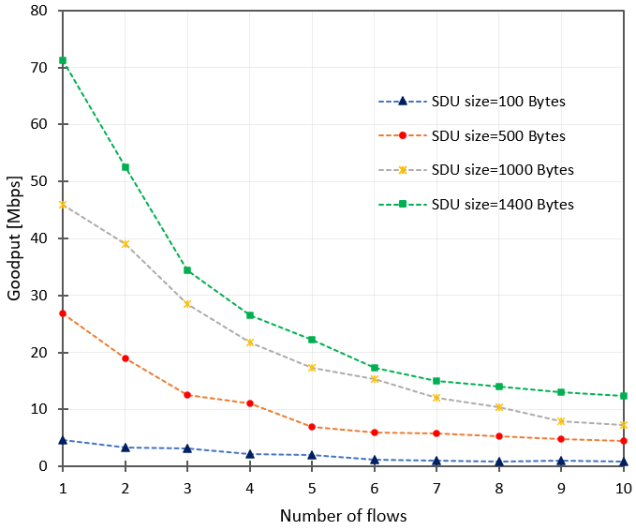

Figure 17. Goodput of unreliable flows without re-transmission control.

5.2.2. RINA L2VPN Performance

In order to evaluate the performance of RINA L2VPN, two different tests have been performed. In the initial test depicted in Figure 18, a traffic generator was employed to transmit L2 packets across a single overlay. This test scenario is illustrated in Figure 6, where the network service took on the role of the traffic generator, while the receiver and sender sides functioned as dedicated network functions (NFs). For the traffic generator, we opted for pkt-gen v11.3 [40], a traffic generator leveraging the netmap API. This choice was motivated by the netmap API's capacity to attain higher data rates. It is important to note that all experiments were conducted within a fully virtualized environment, thus avoiding hardware limitations. Consequently, these controlled experiments serve as a benchmark.

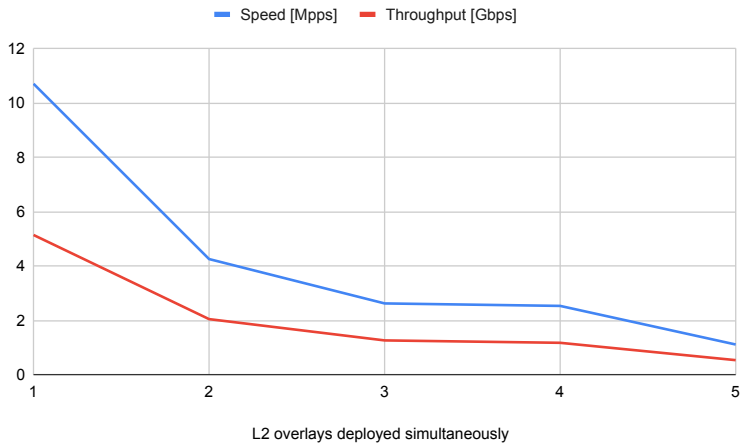

Figure 18. Performance degradation of several L2 VPNs running at the same times. Packet size is set to 60 bytes.

The second test shown in Figure 19 validates the viability of deploying multiple network overlays within a single RINA network, as well as characterizing how performance degrades as more network overlays are added in parallel. An idea of that experiment is illustrated in Figure 7. This characterization was accomplished through the stacking of L2 overlays, coupled with the deployment of the traffic generator tasked with transmitting traffic at its maximum attainable capacity. This test served as a pivotal assessment of the system's scalability and adaptability under conditions of heightened demand.

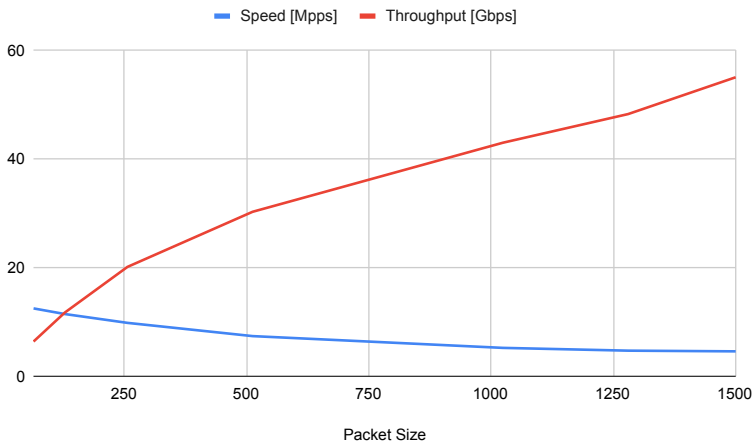

Figure 19. Evolution of speed and throughput of a flow running on single L2 VPN overlay.

6. Conclusions and Future Work

This article has presented a methodology for enabling a progressive adoption of RINA in IoT networks and overcoming the challenges of current IoT networks. The proposed architecture has tackled the current limitation on seamless integration of RINA and SDN

technologies by providing a RINA SBI driver to control RINA nodes using an SDN controller. The proposed methodology has highlighted three components: (i) RINA-based DAF, (ii) RINA SBI driver, and (iii) RINA L2VPN.

The RINA-based DAF has provided a secure management framework based on the RINA manager, the SDN controller, and the CDAP-based and HTTP-based MAs. Using this secure management framework, the SDN controller through the RINA SBI could request to allocate resources along the edge–fog–cloud continuum. Functional testing has demonstrated that the DAF management helped enable a secure connection between MAs and the RINA SBI for interchanging crucial CDAP messages to create and destroy slices (DIFs). Consequently, all the RINA nodes deployed along the edge–fog–cloud continuum can be managed using a centralized approach.

Furthermore, the RINA SBI driver has powered the current OpenFlow-based SDN controller software to enable the hybrid centralized management of SDN switches and RINA nodes. The SDN controller and RINA SBI driver integration will open opportunities for RINA in some challenging IoT scenarios with emerging technologies such as the tactile Internet. Furthermore, the RINA L2VPN was demonstrated to be an efficient solution enabling seamless communication between edge nodes and the cloud. By doing so, the no-RINA applications could benefit from RINA networks without any changes. All these features will enable a progressive adoption of RINA in IoT environments and bridge the gap between future network developments and current network deployments. As an alternative Internet stack, RINA may introduce benefits in terms of scalability and flexibility in an environment where devices are highly mobile. The RINA nodes can be managed from an SDN controller by using the acRINA SBI driver. As a result, the SDN controller can manage heterogeneous network devices (OpenFlow-based and RINA-based), allowing the opportunity to progressively adopt RINA in IoT environments.

The proposed architecture integration is the starting point for establishing RINA as a network solution option in high–low latency applications. Further research will be conducted to analyze and experiment with RINA on the Radio Access Network (RAN). Furthermore, the current implementation will be used to conduct experiments to evaluate the QoS in IoT and tactile Internet scenarios and assess scenarios centered on mobility applications. In addition, the RINA SBI driver will be adapted to support to other controllers (ONOS, μONOS).

Author Contributions: Conceptualization, D.S.-J. and E.G.; methodology, D.S.-J. and S.G.-A.; software, D.S.-J., S.G.-A. and A.L.; validation, D.S.-J., S.G.-A. and A.L.; formal analysis, D.S.-J. and S.G.-A.; investigation, D.S.-J. and S.G.-A.; resources, D.S.-J.; data curation, D.S.-J. and S.G.-A.; writing—original draft preparation, D.S.-J., S.G.-A. and A.L.; writing—review and editing, D.S.-J. and M.C.; visualization, D.S.-J., S.G.-A. and A.L.; supervision, E.G., M.C. and D.P.; project administration, E.G., M.C. and D.P.; funding acquisition, E.G., D.P. and M.C. All authors have read and agreed to the published version of the manuscript.

Funding: This research received funding from the Departament de Recerca de Universitats de la Generalitat de Catalunya under grant 2021 SGR 01524 and from the European Union's "Horizon 2020" research and innovation program as a part of the TERMINET project under Grant Agreement nº 957406.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Upon reasonable request, the underlying data for this article will be made available by the corresponding author.

Acknowledgments: David Sarabia-Jácome acknowledges the support provided by the Secretaría Nacional de Educación Superior, Ciencia, Tecnología e Innovación (SENESCYT), Ecuador.

Conflicts of Interest: The authors declare no conflicts of interest.

Appendix A. SDN Controller Dashboard

The SDN dashboard is a Python3 Django-based app that interacts with the SDN controller using its REST API. The SDN dashboard can operate on OpenFlow-based and RINA-based environments simultaneously, offering a simple centralized point of interaction for both protocol stacks. A dedicated menu is developed to visualize RINA information to ensure backwards compatibility and mitigate any interoperability challenges caused by the differences between the traditional TCP/IP stack and the RINA stack, as show Figure A1.

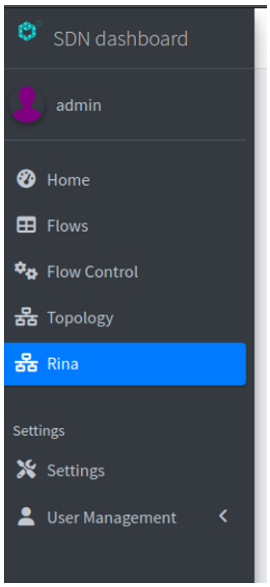

Figure A1. RINA topology section in the SDN controller.

As illustrated in Figure A2, the RINA menu offers a clear visualization of the parsed REST replies, offering the user a valuable resource of compact information related to MAs and the IPCP layers of the registered systems.

Figure A2. RINA systems visualisation in the SDN controller.

Furthermore, the SDN controller dashboard shows the RINA topology and the switch port summary, as illustrated in Figure A3.

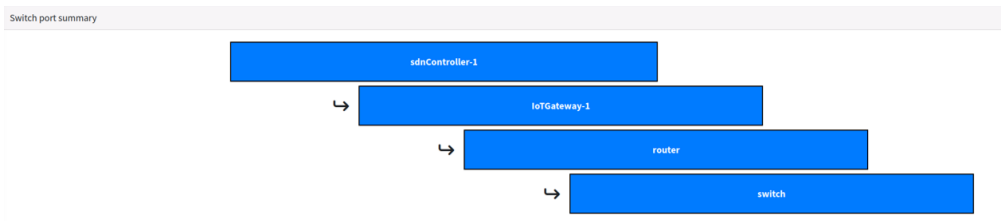

Figure A3. RINA topology and switch por summary in the SDN controller.

References

1. Yan, B.; Liu, Q.; Shen, J.; Liang, D.; Zhao, B.; Ouyang, L. A survey of low-latency transmission strategies in software defined networking. *Comput. Sci. Rev.* **2021**, *40*, 100386. [CrossRef]
2. Salman, O.; Elhajj, I.; Chehab, A.; Kayssi, A. IoT survey: An SDN and fog computing perspective. *Comput. Netw.* **2018**, *143*, 221–246. [CrossRef]
3. Day, J.; Matta, I.; Mattar, K. Networking is IPC: A Guiding Principle to a Better Internet. In Proceedings of the 2008 ACM CoNEXT Conference, Madrid, Spain, 9–12 December 2008; Association for Computing Machinery: New York, NY, USA, 2008. [CrossRef]
4. *ETSI GR NGP 009 V1.1.1:2019-02*; Next Generation Protocols (NGP): An Example of a Non-IP Network Protocol Architecture Based on RINA Design Principles. European Telecommunications Standards Institute (ETSI): Sophia Antipolis, France, 2019.
5. Gaixas, S.L.; Perelló, J.; Careglio, D.; Grasa, E.; López, D.R.; Aranda, P.A. Scalable topological forwarding and routing policies in RINA-enabled programmable data centers. *Trans. Emerg. Telecommun. Technol.* **2017**, *28*, e3256. [CrossRef]
6. Grasa, E.; Gastón, B.; van der Meer, S.; Crotty, M.; Puente, M.A. Simplifying multi-layer network management with RINA. In Proceedings of the TERENA Networking Conference (TNC), Prague, Czech, 12–16 June 2016.
7. Grasa, E.; de Leon, M.P.; van der Meer, S.; Lopez, D.; Tarzan, M. Open multi-access edge computing and distributed mobility management with RINA. In Proceedings of the 2017 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN), Berlin, Germany, 6–8 November 2017; IEEE: Piscataway, NJ, USA, 2017. [CrossRef]
8. Boddapati, G.; Day, J.; Matta, I.; Chitkushev, L.T. Assessing the security of a clean-slate Internet architecture. In Proceedings of the 20th IEEE International Conference on Network Protocols, ICNP 2012, Austin, TX, USA, 30 October–2 November 2012; Computer Society; IEEE: Piscataway, NJ, USA, 2012; pp. 1–6. [CrossRef]
9. Trouva, E.; Grasa, E.; Day, J.; Matta, I.; Chitkushev, L.T.; Bunch, S.; de Leon, M.P.; Phelan, P.; Hesselbach-Serra, X. Transport over Heterogeneous Networks Using the RINA Architecture. In *Lecture Notes in Computer Science*; Springer: Berlin/Heidelberg, Germany, 2011; pp. 297–308. [CrossRef]
10. Tomonori, F. Introduction to Ryu SDN Framework. In *Open Networking Summit*; NTT: Tokyo, Japan, 2013; pp. 1–14.
11. TERMINET H2020 Project. Next Generation of Smart Interconnected IoT. Available online: https://terminet-h2020.eu/ (accessed on 30 September 2022).
12. Feamster, N.; Rexford, J.; Zegura, E. The Road to SDN: An Intellectual History of Programmable Networks. *Queue* **2013**, *11*, 20–40. [CrossRef]
13. Caraguay, Á.L.V.; Peral, A.B.; López, L.I.B.; Villalba, L.J.G. SDN: Evolution and Opportunities in the Development IoT Applications. *Int. J. Distrib. Sens. Netw.* **2014**, *10*, 735142. [CrossRef]
14. Siddiqui, S.; Hameed, S.; Shah, S.A.; Ahmad, I.; Aneiba, A.; Draheim, D.; Dustdar, S. Toward Software-Defined Networking-Based IoT Frameworks: A Systematic Literature Review, Taxonomy, Open Challenges and Prospects. *IEEE Access* **2022**, *10*, 70850–70901. [CrossRef]
15. Rawat, D.B.; Reddy, S.R. Software Defined Networking Architecture, Security and Energy Efficiency: A Survey. *IEEE Commun. Surv. Tutorials* **2017**, *19*, 325–346. [CrossRef]
16. Younus, M.U.; ul Islam, S.; Ali, I.; Khan, S.; Khan, M.K. A survey on software defined networking enabled smart buildings: Architecture, challenges and use cases. *J. Netw. Comput. Appl.* **2019**, *137*, 62–77. [CrossRef]
17. Salman, O.; Elhajj, I.H.; Kayssi, A.; Chehab, A. SDN controllers: A comparative study. In Proceedings of the 2016 18th Mediterranean Electrotechnical Conference (MELECON), Lemesos, Cyprus, 18–20 April 2016; IEEE: Piscataway, NJ, USA, 2016. [CrossRef]
18. Singh, S.; Jha, R.K. A Survey on Software Defined Networking: Architecture for Next Generation Network. *J. Netw. Syst. Manag.* **2016**, *25*, 321–374. [CrossRef]
19. Kreutz, D.; Ramos, F.M.V.; Verissimo, P.E.; Rothenberg, C.E.; Azodolmolky, S.; Uhlig, S. Software-Defined Networking: A Comprehensive Survey. *Proc. IEEE* **2015**, *103*, 14–76. [CrossRef]
20. Abid, M.A.; Afaqui, N.; Khan, M.A.; Akhtar, M.W.; Malik, A.W.; Munir, A.; Ahmad, J.; Shabir, B. Evolution towards Smart and Software-Defined Internet of Things. *AI* **2022**, *3*, 100–123. [CrossRef]
21. Ojo, M.; Adami, D.; Giordano, S. A SDN-IoT Architecture with NFV Implementation. In Proceedings of the 2016 IEEE Globecom Workshops (GC Wkshps), Washington, DC, USA, 4–8 December 2016; IEEE: Piscataway, NJ, USA, 2016. [CrossRef]
22. Grasa, E.; Tarzan, M.; Bergesio, L.; Gastón, B.; Maffione, V.; Salvestrini, F.; Vrijders, S.; Staessens, D. IRATI: Open Source RINA Implementation for Linux. *Softw. Impacts* **2019**, *1*, 100003. [CrossRef]
23. Vrijders, S.; Trouva, E.; Day, J.; Grasa, E.; Staessens, D.; Colle, D.; Pickavet, M.; Chitkushev, L. Unreliable inter process communication in Ethernet: Migrating to RINA with the shim DIF. In Proceedings of the 2013 5th International Congress on Ultra Modern Telecommunications and Control Systems and Workshops (ICUMT), Almaty, Kazakhstan, 10–13 September 2013; pp. 215–221. [CrossRef]
24. van der Meer, S.; Keeney, J.; Fallon, L.; Feghhi, S.; de Buitleir, A. Large-scale Experimentation with Network Abstraction for Network Configuration Management. In Proceedings of the 2019 22nd Conference on Innovation in Clouds, Internet and Networks and Workshops (ICIN), Paris, France, 19–21 February 2019; IEEE: Piscataway, NJ, USA, 2019. [CrossRef]
25. Grasa, E.; Bergesio, L.; Tarzan, M.; Lopez, D.; van der Meer, S.; Day, J.; Chitkushev, L. Mobility management in RINA networks: Experimental validation of architectural properties. In Proceedings of the 2018 IEEE Wireless Communications and Networking Conference (WCNC), Barcelona, Spain, 15–18 April 2018; pp. 1–6. [CrossRef]

26. Fernández, Z.; Gabilondo, Á.; Vázquez-Rodríguez, Á.; Giraldo-Rodríguez, C.; Escudero-Garzás, J.; Giménez, S.; Cárdenas, A.; Herranz, C. Solutions for Traffic Isolation in 5G Infrastructures Using Network Slicing Techniques. In Proceedings of the 2022 32nd International Telecommunication Networks and Applications Conference (ITNAC), Wellington, New Zealand, 30 November–2 December 2022; pp. 64–69. [CrossRef]
27. Ponce de Leon, M.; Ranganathan, R.; Bainbridge, D.; Ramanarayanan, K.; Corston-Petrie, A.; Fundacio, E.G. Multi-operator IPC VPN slices: Applying RINA to overlay networking. In Proceedings of the 2019 22nd Conference on Innovation in Clouds, Internet and Networks and Workshops (ICIN), Paris, France, 19–21 February 2019; IEEE: Piscataway, NJ, USA, 2019. [CrossRef]
28. Open Verso. OPEN VERSO | Excellence National Network on 5G Technology. Available online: https://www.openverso.org/en/ (accessed on 13 June 2023).
29. Antón, S.G.; Grasa, E.; Fernández, C.; Siddiqui, M.S. RINA-based Virtual Networking Solution for Distributed VNFs: Prototype and Benchmarking. In Proceedings of the 2022 Joint European Conference on Networks and Communications & 6G Summit (EuCNC/6G Summit), Grenoble, France, 7–10 June 2022; pp. 369–374. [CrossRef]
30. Rizzo, L. Netmap: A Novel Framework for Fast Packet I/O. In Proceedings of the 21st USENIX Security Symposium (USENIX Security 12), Grenoble, France, 7–10 June 2012; pp. 101–112.
31. Vrijders, S.; Staessens, D.; Colle, D.; Salvestrini, F.; Grasa, E.; Tarzan, M.; Bergesio, L. Prototyping the recursive internet architecture: The IRATI project approach. *IEEE Netw.* **2014**, *28*, 20–25. [CrossRef]
32. King, J.; Perry, C. *Smart Buildings: Using Smart Technology to Save Energy in Existing Buildings*; Amercian Council for an Energy-Efficient Economy: Washington, DC, USA, 2017.
33. Cao, H.; Wachowicz, M. An Edge-Fog-Cloud Architecture of Streaming Analytics for Internet of Things Applications. *Sensors* **2019**, *19*, 3594. [CrossRef] [PubMed]
34. Abdulrahman, S.; Tout, H.; Ould-Slimane, H.; Mourad, A.; Talhi, C.; Guizani, M. A Survey on Federated Learning: The Journey From Centralized to Distributed On-Site Learning and Beyond. *IEEE Internet Things J.* **2021**, *8*, 5476–5497. [CrossRef]
35. Smith, L.; Cokely, D.; Bell, H.; Day, J.; Chitkushev, L. Unifying Wi-Fi® and VLANs with the RINA Model. In Proceedings of the 22nd Conference on Innovation in Clouds, Internet and Networks and Workshops, ICIN 2019, Paris, France, 19–21 February 2019; Galis, A., Guillemin, F., Noldus, R., Secci, S., Idzikowski, F., Sayit, M., Eds.; IEEE: Piscataway, NJ, USA, 2019; pp. 43–48. [CrossRef]
36. Sarabia-Jácome, D.; Grasa, E.; Catalán, M. RINAsense: A prototype for implementing RINA networks in IoT environments. In Proceedings of the 2023 6th Conference on Cloud and Internet of Things (CIoT), Lisbon, Portugal, 20–22 March 2023; IEEE: Piscataway, NJ, USA, 2023. [CrossRef]
37. Ali, M.; Jehangiri, A.I.; Alramli, O.I.; Ahmad, Z.; Ghoniem, R.M.; Ala'anzy, M.A.; Saleem, R. Performance and Scalability Analysis of SDN-Based Large-Scale Wi-Fi Networks. *Appl. Sci.* **2023**, *13*, 4170. [CrossRef]
38. Radoglou–Grammatikis, P.; Sarigiannidis, P.; Efstathopoulos, G.; Karypidis, P.A.; Sarigiannidis, A. DIDEROT. In Proceedings of the Proceedings of the 15th International Conference on Availability, Reliability and Security, Virtual, 25–28 August 2020; ACM: New York, NY, USA, 2020. [CrossRef]
39. Maffione, V. A Light RINA Implementation. 2017. Available online: https://github.com/rlite/rlite (accessed on 23 October 2023).
40. Università di Pisa. Packet Generator for Use with Netmap. Available online: https://github.com/luigirizzo/netmap/blob/master/apps/pkt-gen/pkt-gen.c (accessed on 18 September 2023).