16

Recursive InterNetwork Architecture, Investigating RINA as an Alternative to TCP/IP (IRATI)

Eduard Grasa[1], Leonardo Bergesio[1], Miquel Tarzan[1], Eleni Trouva[2], Bernat Gaston[1], Francesco Salvestrini[3], Vincenzo Maffione[3], Gino Carrozzo[3], Dimitri Staessens[4], Sander Vrijders[4], Didier Colle[4], Adam Chappel[5], John Day[6] and Lou Chitkushev[6]

[1]Fundacio 12CAT, Spain
[2]University of Patras, Greece
[3]Nextworks, Italy
[4]University of Ghent, Belgium
[5]Interoute, UK
[6]Boston University, USA

16.1 Introduction

Driven by the requirements of the emerging applications and networks, the Internet has become an architectural patchwork of growing complexity which strains to cope with the changes. Moore's law prevented us from recognising that the problem does not hide in the high demands of today's applications but lies in the flaws of the Internet's original design. The Internet needs to move beyond TCP/IP to prosper in the long term, TCP/IP has outlived its usefulness.

The Recursive InterNetwork Architecture (RINA) is a new Internetwork architecture whose fundamental principle is that networking is only interprocess communication (IPC). RINA reconstructs the overall structure of the Internet, forming a model that comprises a single repeating layer, the DIF (Distributed IPC Facility), which is the minimal set of components required to allow distributed IPC between application processes. RINA supports inherently and without the need of extra mechanisms mobility, multi-homing and Quality of Service, provides a secure and configurable environment, motivates for a more competitive marketplace and allows for a seamless adoption.

RINA is the best choice for the next generation networks due to its sound theory, simplicity and the features it enables. IRATI's goal is to achieve further exploration of this new architecture. IRATI will advance the state of the art of RINA towards an architecture reference model and specifcations that are closer to enable implementations deployable in production scenarios. The design and implemention of a RINA prototype on top of Ethernet will permit the experimentation and evaluation of RINA in comparison to TCP/IP. IRATI will use the OFELIA testbed to carry on its experimental activities. Both projects will benefit from the collaboration. IRATI will gain access to a large-scale testbed with a controlled network while OFELIA will get a unique use-case to validate the facility: experimentation of a non-IP based Internet.

16.1.1 RINA Overview

RINA is the result of an effort that tries to work out the general principles in computer networking that applies to everything. RINA is the specific architecture, implementation, testing platform and ultimately deployment of the theory. This theory is informally known as the Inter-Process Communication "IPC model" [1, 2] although it also deals with concepts and results that are generic for any distributed application and not just for networking. RINA is structured around a single type of layer – called Distributed IPC Facility or DIF – that repeats as many times as needed by the network designer (Figure 16.1). In RINA all layers are distributed applications that provide the same service (communication flows between distributed applications) and have the same internal structure. The instantiation of a layer in a computing system is an application process called IPC Process (IPCP). All IPCPs have the same functions, divided into data transfer (delimiting, addressing, sequencing, relaying, multiplexing, lifetime termination, error check, encryption), data transfer control (flow and retransmission control) and layer management (enrollment, routing, flow allocation, namespace management, resource allocation, security management). The functions of an IPCP are programmable via policies, so that each DIF can adapt to its operational environment and to different application requirements.

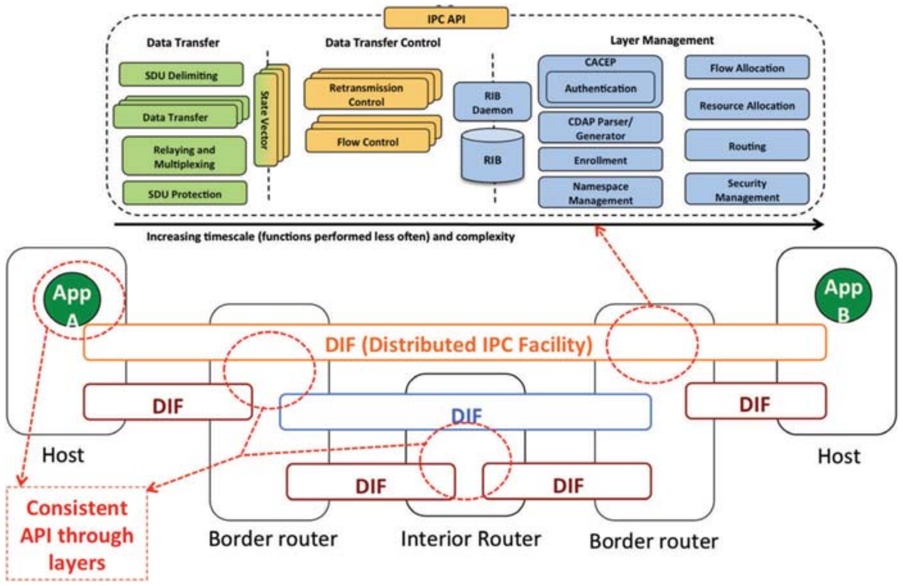


Figure 16.1 Illustration of the RINA structure: DIFs and internal organisation of IPC Processes (IPCPs).

The DIF service definition provides the *abstract description* of an API as seen by an Application Process using a DIF (specific APIs are system-dependant and may take into account local constraints; in some cases there may not be an API at all, but an equivalent way to have equivalent interactions). The Application Process might be an IPC Process, reflecting the recursive nature of RINA (a DIF can be used by any distributed application, including other DIFs). All DIFs provide the same service, called *flows*. A flow is the instantiation of a communication service between two or more application process instances.

In contrast with traditional network architectures in which layers have been defined as units of modularity, in RINA layers (DIFs) are distributed resource allocators [3]. It isn't that layers perform different functions; they all perform the same functions at different scopes. They are doing these functions for the different ranges of the environments the network is targeted at (a single link, a backbone network, an access network, an internet, a VPN, etc.). The scope of each layer is configured to handle a given range of bandwidth, QoS, and scale: a classic case of divide and conquer. Layers manage resources over a given range. The policies of each layer will be selected to optimize that range, bringing programmability to every relevant function within the layer [4]. How many layers are needed? It depends on the range of bandwidth, QoS, and scale: simple networks have two layers, simple internetworks, 3; more complex networks may have more. *This is a network design question, not an architecture question*.

One of the key RINA design principles has been to maximize invariance and minimize discontinuities. In other words, extract as much commonality as possible without creating special cases. Applying the concept from operating systems of separating mechanism and policy, first to the data transfer protocols and then to the layer management machinery (usually referred to the control plane), it turns out that only two protocols are required within a layer [1]:

- A single data transport protocol that supports multiple policies and that allows for different concrete syntaxes (length of fields in the protocol PDUs). This protocol is called EFCP – the *Error and Flow Control Protocol* – and is further explained in Section 4.2.
- A common application protocol that operates on remote objects used by all the layer management functions. This protocol is called CDAP – the Common Distributed Application Protocol.

Separation of mechanism and policy also provided new insights about the structure of those functions within the layer, depicted in Figure 16.1. The primary components of an IPC Process are shown in Figure 16.1 and can be divided into three categories: a) Data Transfer, decouple through a state vector from b) Data Transfer Control, decoupled through a Resource Information Base from c) Layer Management. These three loci of processing are characterized by decreasing cycle time and increasing computational complexity (simpler functions execute more often than complex ones).

- *SDU Delimiting*. The integrity of the SDU written to the flow is preserved by the DIF via a delimiting function. Delimiting also adapts the SDU to the maximum PDU size. To do so, delimiting comprises the mechanisms of fragmentation, reassembly, concatenation and separation.
- *EFCP, the Error and Flow Control Protocol*. This protocol is based on Richard Watson's work [5] and separates mechanism and policy. There is one instance of the protocol state for each flow originating or terminating at this IPC Process. The protocol naturally cleaves into Data Transfer (sequencing, lost and duplicate detection, identification of parallel connections), which updates a state vector; and Data Transfer Control, consisting of retransmission control (ack) and flow control.

- *RMT, the Relaying and Multiplexing Task*. It makes forwarding decision on incoming PDUs and multiplexes multiple flows of outgoing PDUs onto one or more (N – 1) flows. There is one RMT per IPC Process.
- *SDU Protection*. It does integrity/error detection, e.g. CRC, encryption, compression, etc. Potentially there can be a different SDU Protection policy for each (N – 1) flow.

The state of the IPC Process is modelled as a set of objects stored in the Resource Information Base (RIB) and accessed via the RIB Daemon. The RIB imposes a schema over the objects modelling the IPCP state, defining what CDAP operations are available on each object and what will be their effects. The RIB Daemon provides all the layer management functions (enrolment, namespace management, flow allocation, resource allocation, security coordination, etc) with the means to interact with the RIBs of peer IPCPs. Coordination within the layer uses the Common Distributed Application Protocol (CDAP).

16.2 IRATI Goals

The overarching goal of ARCFIRE is to contribute to the experimental research and development of RINA, investigating it as an alternative technology to functional layering and TCP/IP. This objective is divided into the following four goals:

1. **Enhancement of the RINA architecture reference model and specifications, focusing on DIFs over Ethernet**. The enhancement of the RINA specifications carried out within IRATI will be driven by three main forces: i) the specification of a DIF over Ethernet as the underlying physical media; ii) the completion of the specifications that enable RINA to provide a level of service similar to the current Internet (low security, best-effort) and iii) the project use cases targeting ambitious scenarios that are challenging for current TCP/IP networks (targeting features like multi-homing, security or quality of service). The industrial partners in the consortium will be leading the elaboration of the use cases, with the input of the External Advisory Board.

2. **RINA open source prototype over Ethernet for a UNIX-like OS**. This is the goal that can better contribute to IRATI's impact and the dissemination

of RINA. Besides being the main experimentation vehicle of the project, the prototype will provide a solid baseline for further RINA work after the project. By the end of the project the IRATI partners plan to setup an open source community in order to attract external interest and involve other organizations in RINA R&D.

3. **Experimental validation of RINA and comparison against TCP/IP**. This objective is enabled due to the availability of the FIRE facilities, which provide the experimentation environment for a meaningful comparison between RINA and TCP/IP. IRATI will follow iterative cycles of research, design, implementation and experimentation, with the experimental results retrofitting the research of the next phase. Experiments will collect and analyse data to compare RINA and TCP/IP in various aspects like: application API, programmability, cost of supporting multi-homing, simplicity, vulnerability against attacks, hardware resource utilization (proportional to energy consumption). The industrial partners in the consortium will be leading the choice of benchmarking parameters, with the input of the External Advisory Board.

4. **Provide feedback to OFELIA in regards to the prototyping of a clean slate architecture**. Apart from the feedback to the OFELIA [6] facility in terms of bug reports and suggestions of improvements, IRATI will contribute an OpenFlow controller capable of dynamically setting up Ethernet topologies to the project. IRATI will be using this controller in order to setup different topologies for the various experiments conducted during the project. Moreover, experimentation with a non-IP based solution is an interesting use case for the OFELIA facility, since IRATI will be the first to conduct these type of experiments in the OFELIA testbed.

16.3 Approach

The technical work of the IRATI Project comprises requirements analysis, design, implementation, validation, deployment and experimentation activities organized in three iterations. Such activities have been broken down in three technical work packages.

WP2 is the overarching work package that will define the scope of the use cases to be validated, propose a set of refinements and enhancements to the RINA architecture reference model and specifications, and elaborate a high level software architecture for the implementation. For each phase of the project, WP2 will:

- Elaborate the use cases to be showcased during the experimentation phases, analyze them and extract requirements. Use cases will try to focus at first on the availability/integration of core RINA functionalities in basic experimental setups; then, more complex scenarios that are challenging with the current Internet will be targeted to explore the full RINA functionalities and thus meet the expectations/take-up strategies of network operators and cloud service providers (like Interoute). The use cases will drive the experiment design and provide requirements for the completion/validation of the RINA architecture reference model and specifications.
- Analyse the RINA architecture reference model and specifications, identify holes in the mechanisms or missing policies, and propose enhancements/refinements.
- Based on the RINA architecture reference model and specifications on one side and the phase scenario and targeted platform on the other side, provide a high-level software architecture for the design and implementation of the prototype. This high-level software architecture will be the unifying document for the WP3 implementation tasks.

WP3 is the development work package of the project. Its overall objective is to translate the WP2 specifications and high-level software design into a set of prototypes that will be used by WP4 for its test-bed activities and experimentation. The main objectives of this WP are:

- to provide a common development environment
- to implement a RINA prototype over Ethernet for Linux/OS
- to integrate the various functionalities and components into a demon strable system (at node-level)

The architecture releases at the various project phases and the related functional decompositions delivered by WP2 are the starting point of work for WP3. Software prototypes are the major WP3 outcomes to be delivered to WP4. Moreover, it is expected that WP3 will produce a number of feedbacks on previous or concurrent activities, both internally (i.e. among tasks) and externally (i.e. towards other WPs). The feedbacks produced by WP3 to either internal or external tasks will have eventually an impact on the work produced by the target task, i.e. its deliverable. As a general rule, it is expected that major feedbacks on a task could lead to fix and reissue the deliverable(s) produced by that task previously.

WP4 is the experimentation and validation work package, responsible of the following goals:

- Design the experiments required to validate the use cases and deploy WP3 prototypes into the OFELIA facility for experimentation.
- Validate the correctness of the prototype with respect to its compliance with the use cases through experimentation.
- Compare and document RINA benefits against TCP/IP in different areas: application interface, multi-homing, support of heterogeneous applications, security and others identified by WP2.
- Based on the experiments result analysis, provide feedback to the RINA specifications enhancement and high-level software architecture design activities in WP2.

16.4 Discussion of Technical Work and Achievements

16.4.1 Enhancements of the RINA Specifications and Reference Model

16.4.1.1 Shim DIF over 802.1Q layers

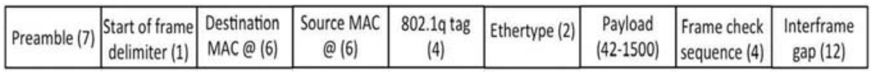
This specification defines the shim IPC process for the Ethernet (IEEE 802.3) layer using IEEE 802.1Q [7]. Other Shim DIFs specifications will cover the use of Ethernet with other constraints. This type of IPC process is not fully functional. It presents an Ethernet layer as if it was a regular DIF. The task of a shim DIF is to put as small as possible a veneer over a legacy protocol to allow a RINA DIF to use it unchanged. In other words, because the DIF assumes it has a RINA API below, the Shim DIF allows a DIF to operate over Ethernet without change. The shim IPC process wraps the Ethernet layer with the IPC process interface. The goal is not to make legacy protocols provide full support for RINA and so the shim DIF should provide no more service or capability than the legacy protocol provides.

An Ethernet shim DIF spans a single Ethernet “segment”. This means relaying is done only on 1-DIF addresses. Each shim DIF is identified by a VLAN (IEEE 802.1Q) id, which is in fact the shim DIF name. Each VLAN is a separate Ethernet Shim DIF. All the traffic in the VLAN is assumed to be shim DIF traffic.

Ethernet comes with the following limitations, which are reflected by the capabilities provided by the Ethernet shim DIF:

- It assumes that there is a single “network layer” protocol machine instance that is the only user of the Ethernet protocol machine at each system. The Ethertype field on the Ethernet header just identifies the syntax of the “network layer” protocol.
- Because it is only possible to distinguish one flow between each pair of MAC addresses, there is no explicit flow allocation.
- There are no guarantees on reliability.

These limitations impact the usability of the Ethernet shim DIF: it only provides enough capabilities for another DIF to be the single user of the Ethernet shim DIF. Therefore, the only applications that can register in an Ethernet shim DIF are IPC Processes. Moreover, since Ethernet doesn’t provide the means to distinguish different flows nor explicit flow allocation, there can only be one instance of an IPC Process registered at each Ethernet shim IPC Process. The following figure illustrates an Ethernet frame as used by the Ethernet shim DIF.



Preamble (7)	Start of frame delimiter (1)	Destination MAC @ (6)	Source MAC @ (6)	802.1q tag (4)	Ethertype (2)	Payload (42-1500)	Frame check sequence (4)	Interframe gap (12)

- *Destination MAC address*: The MAC address assigned to the Ethernet interface the destination shim IPC Process is bound to.
- *Source MAC address*: The MAC address assigned to the Ethernet interface the source shim IPC Process is bound to.
- *802.1Q tag*: The DIF name.
- *Ethertype*: Although it is not strictly required to have a special Ethertype for the correct operation of the shim DIF (since all the traffic in the VLAN is assumed to be shim DIF traffic), it is handy to define an Ethertype for RINA (if, for no other reasons, to facilitate debugging). Therefore the Ethernet frames used within the shim Ethernet DIF will use the 0xD1F0 value for the Ethertype field.
- *Payload*: Carries the upper DIF SDUs. The maximum length of the SDU must be enforced by the upper DIF, since the Ethernet shim DIF doesn’t perform fragmentation and reassembly functions. The only delimiting supported by this Shim DIF is “1 for 1,” i.e. it is assumed the entire Payload is a single SDU.

Instead of using the RINA Flow Allocator, the Ethernet shim DIF reuses ARP in request/response mode to perform this function. ARP resolves a network layer address into a link layer address instantiating the state for a flow to this

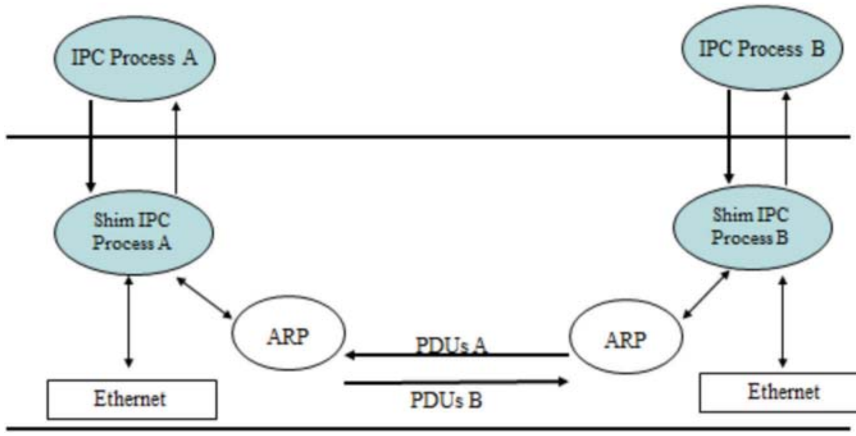


Figure 16.2 Relation between protocol machines.

protocol. In effect, in the context of the shim DIF, mapping the application process name to a shim IPC Process address and instantiating a MAC protocol machine equivalent of DTP, exactly the function that the Flow Allocator provides. The relation between these different protocol machines can be seen in Figure 16.2.

16.4.1.2 Shim DIF for hypervisors

In order to cope with the vast amount of virtualization technologies and hardware (HW) architectures, this specification aims at providing a generic schema in order to ease defining shim IPC Processes for Hypervisors (HV) [8]. Therefore, it does not target any particular HV/HW technology but describes basic mechanisms that shim IPC Process for specific virtualization technologies could inherit. It is expected that other specifications, focusing on specific HV/HW solutions, will be deriving from this one and will be defining the necessary low-level details.

Shim IPC Processes for HV are not fully functional processes, they just present their mechanisms for VM-to-VM intercommunication as if they were a regular DIF. The task of this shim DIF is to put as small as possible layering overhead on the VM-to-VM intercommunication mechanisms, wrapping them with the RINA API.

The scope of the communication mechanisms described is point-to-point and HV local, i.e. between guest and host VMs in the same HV system. Therefore, the host must rely on other shims IPC Processes to allow for inter-communications with other systems than the ones managed by the Hypervisor,

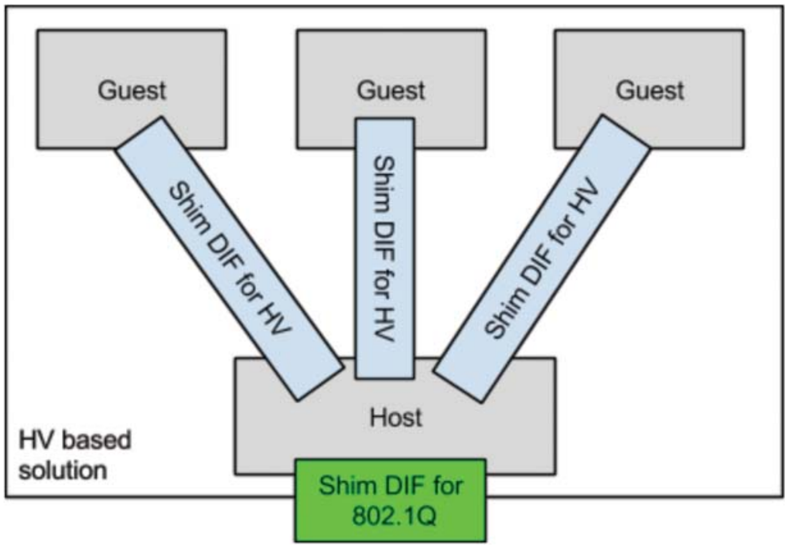


Figure 16.3 Environment of the shim DIF for Hypervisors.

for instance the shim IPC Process over Ethernet 802.1Q as depicted in Figure 16.3.

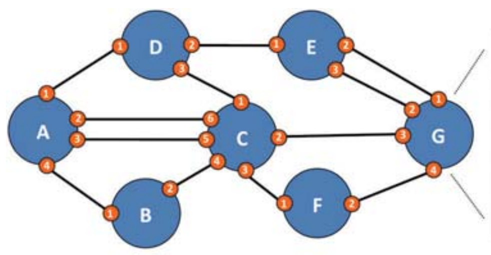
16.4.1.3 Link state routing policy

This specification describes a link-state routing based approach to generate the PDU forwarding table [9]. The PDU forwarding table is a component of an IPC Process. Routing in RINA is all policy; that is, each DIF is free to decide on the best approach to generate the PDU forwarding table for its environment. Link-state routing is just one of the possible policies. The ultimate goal of routing research in RINA is to create a framework for investigating different routing schemes in a recursive model. Since most DIFs will be of moderate size, starting with link-state routing seems a reasonable approach.

In its simplest form, link-state routing is based on the dissemination of link-state information among all the nodes in a network. Every node constructs the network connectivity graph based on its current view of the state of the links in the network, and applies an algorithm to this graph in order to compute the routes to every other node in the network. The routing table contains entries for the next hop to the shortest route to each other node (destination address) in the network. The forwarding table is generated by recording the mapping between each destination addresses and the corresponding outbound interface towards the next hop. Traditional link-state routing is made more scalable by organizing the network into a hierarchy (for instance, prefix-based), where link-state routing is performed in each level of the hierarchy (and the dissemination of link-state information is limited to the nodes that belong to the same hierarchical level or prefix). Since in RINA routing table sizes can be bound by recursing, link-state routing can scale in most cases. Even with large DIFs it is possible to create subnets of nodes within a DIF where link state is used within each subnet and topological addresses across the subnets of the DIF; so that no routing calculation is required between subnets [10].

The functions of the PDU forwarding table generator component are to collaborate with the RIB daemon to disseminate and collect information about the status of the N – 1 flows in the DIF and to use this information to populate the PDU forwarding table used by the Relaying and Multiplexing Task (RMT) to forward PDUs. Current link-state routing protocols, such as Open Shortest Path First (OSPF) or Intermediate System to Intermediate System (IS–IS), perform additional functions such as automatic neighbour discovery, adjacency forming or link-state failure detection. In an IPC Process these functions are performed by other components such as the Resource Allocator, therefore the PDU Forwarding Table generator doesn't need to do them (Figure 16.4).

The PDU Forwarding Table generator is one of the layer management components of the IPC Process. These components interact, using the RIB daemon, with their counterparts in neighbouring IPC Processes by exchanging Common Distributed Application Protocol (CDAP) messages over N – 1 flows. The CDAP messages perform remote operations (create/delete/start/stop/read/write) on one or more objects of the targeted IPC Process RIB. The PDU forwarding table generator is the handler of the RIB operations



Destination @	Next hop @
A	C
B	C
C	C
D	E
E	E
F	F

IPC Process G routing table

Destination @	Forward to
A	3
B	3
C	3
D	2
E	1
F	4

IPC Process G forwarding table

Figure 16.4 A simple example of link-state routing.

targeting the objects related to link-state routing. To summarize, the important details for this specification are that:

- The Enrollment task explicitly notifies the PDU Forwarding Table Gene rator component when enrollment with a new neighbour IPC Process has completed successfully. Then the PDU Forwarding Table Generator can initiate the procedure to synchronize its knowledge about the state of N – 1 flows in the DIF with its new neighbour using the RIB daemon.
- The Resource Allocator component of the IPC Process will explicitly notify the PDU Forwarding table generator of events affecting the local N – 1 flows (local N – 1 flows are flows that have the IPC Process as source or target). Such events include the allocation, deallocation and changes in the status of these flows (N – 1 flow up / N – 1 flow down).
- The PDU Forwarding Table Generator component shares their view of the network using the RIB daemon to communicate with other IPC Processes. The RIB Daemon notifies the PDU Forwarding Table generator when it receives CDAP messages targeting these relevant objects (which are defined later in this document).
- The PDU Forwarding Table Generator propagates changes in the status of local or remote N – 1 flows by requesting the RIB Daemon to send CDAP messages to one or more neighbour IPC Processes. These CDAP messages cause operations on the relevant objects in the RIBs of neighbour IPC Processes.

Figure 16.5 illustrates these details, showing the inputs and outputs of the PDU Forwarding table component, and its relationship with the other components in the IPC Process.

16.4.2 RINA Implementation Activities

16.4.2.1 Implementation goals and major design choices

The IRATI implementation of RINA had to accomplish two main goals:

- **Become a platform for RINA experimentation**. The implementation has to be: i) flexible and adaptable (a RINA "node" can be configured as a host, border router or interior router); ii) the design must be modular so that it can be easily updated as new insights into RINA allow for simpler/better implementations; iii) it must be programmable so that researchers can experiment with different behaviours; iv) must be able to run over multiple lower layers (Ethernet, TCP, UDP, shared memory, USB, etc) and v) must be able to support native RINA applications.

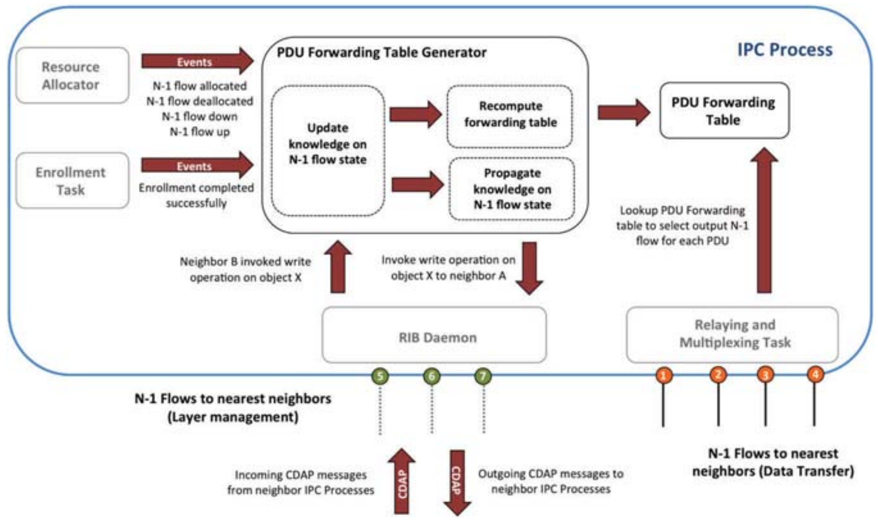
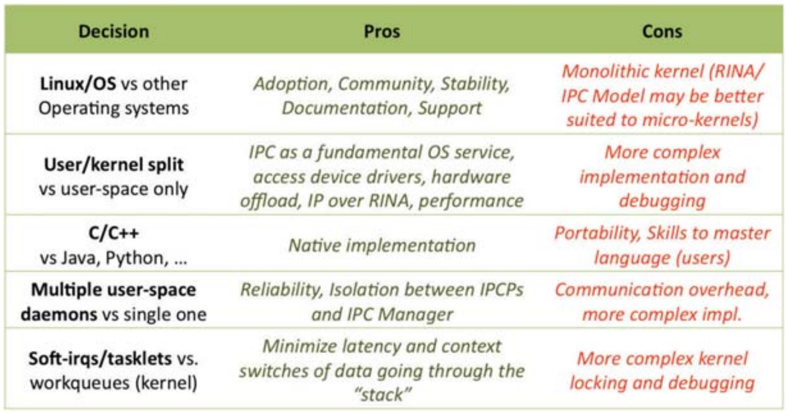

IPC Process
Resource Allocator
Events
N-1 flow allocated
N-1 flow deallocated
N-1 flow down
N-1 flow up
Enrollment Task
Events
Enrollment completed successfully
PDU Forwarding Table Generator
Update knowledge on N-1 flow state
Recompute forwarding table
Propagate knowledge on N-1 flow state
PDU Forwarding Table
Lookup PDU Forwarding table to select output N-1 flow for each PDU
Neighbor B invoked write operation on object X
Invoke write operation on object X to neighbor A
RIB Daemon
Relaying and Multiplexing Task
N-1 Flows to nearest neighbors (Layer management)
N-1 Flows to nearest neighbors (Data Transfer)
Incoming CDAP messages from neighbor IPC Processes
CDAP
CDAP
Outgoing CDAP messages to neighbor IPC Processes

Figure 16.5 Inputs and outputs of the Link-state based PDU Forwarding Table generator policy.

- **Become the basis of future RINA-based products**. In order to achieve this goal, the implementation should: i) tightly integrate with the Opera ting System; ii) allow for performance-related optimizations; iii) enable hardware offload of some functions in the future; iv) seamlessly support existing applications and v) enable RINA to carry IP traffic – RINA as a transport of the IP layer (Figure 16.6).



Decision	Pros	Cons
Linux/OS vs other Operating systems	*Adoption, Community, Stability, Documentation, Support*	*Monolithic kernel (RINA/ IPC Model may be better suited to micro-kernels)*
User/kernel split vs user-space only	*IPC as a fundamental OS service, access device drivers, hardware offload, IP over RINA, performance*	*More complex implementation and debugging*
C/C++ vs Java, Python, …	*Native implementation*	*Portability, Skills to master language (users)*
Multiple user-space daemons vs single one	*Reliability, Isolation between IPCPs and IPC Manager*	*Communication overhead, more complex impl.*
Soft-irqs/tasklets vs. workqueues (kernel)	*Minimize latency and context switches of data going through the "stack"*	*More complex kernel locking and debugging*

Figure 16.6 Major design choices of the IRATI implementation.

With that goals and requirements in mind, the table displayed above shows a summary of the pros and cons behind the major design decisions taken in the IRATI implementation of the RINA architecture: Linux/OS was the chosen operating system; RINA functionalities were spread between the user-space and the kernel; C and C++ were chosen as the programming languages (however, bindings for other languages of the application API can be generated); the implementation contains multiple user-space daemons and the kernel infrastructure exploits the use of soft-irqs and tasklets in order to minimize latency and context switches in the kernel data-path.

Figure 16.7 illustrates the user-kernel split in terms of RINA components implemented in user-space or the kernel. Looking at the different tasks that form an IPC Process, the data transfer and data-transfer control ones – which have stringent performance requirements and execute at every PDU or every few PDUs – were implemented in the kernel. Layer management functions, which do not execute so often and can be much more complex, have been implemented in user-space. Shim IPC Processes have also been implemented in the kernel, since i) they usually need to access device drivers or similar functions only available in kernel space; and ii) the complexity of the functions performed by shim IPC Processes is relatively low.

16.4.2.2 Software architecture overview

The software architecture of the IRATI implementation is show in Figure 16.8. A more in depth description has been published in [11]. The main components of IRATI have been divided into four packages:

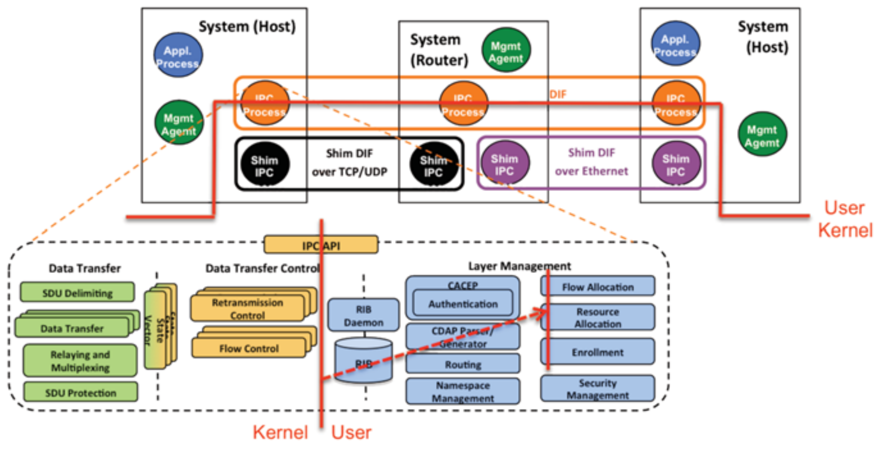


Figure 16.7 IPC Process split between the user-space and kernel.

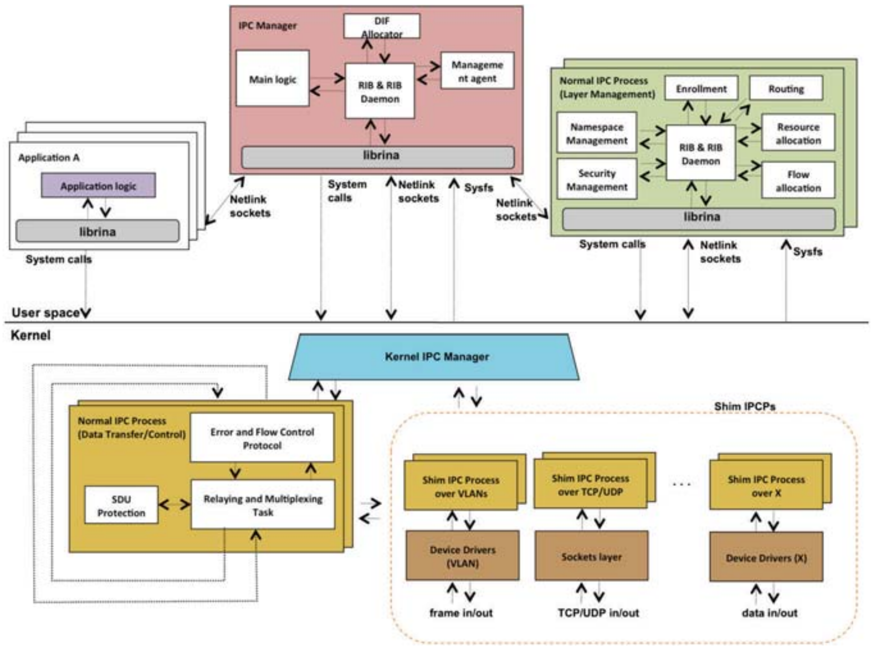


Figure 16.8 Software architecture of the IRATI RINA implementation.

1. **Daemons** (rinad). This package contains two types of daemons (OS Processes that run in the background), implemented in C++.
   - **IPC Manager Daemon** (rinad/src/ipcm). The IPC Manager Daemon is the core of IPC Management in the system, acting both as the manager of IPC Processes and a broker between applications and IPC Processes (enforcing access rights, mapping flow allocation or application registration requests to the right IPC Processes, etc.).
   - **IPC Process Daemon** (rinad/src/ipcp). The IPC Process Daemons (one per running IPC Process in the system) implement the layer management components of an IPC Process (enrollment, flow allocation, PDU Forwarding table generation or distributed resource allocation functions).
2. **Librina** (librina). The librina package contains all IRATI libraries that have been introduced to abstract from the user all the kernel interactions (such as syscalls and Netlink details). Librina provides its functionalities to user-space RINA programs via scripting language extensions or

statically/dynamically linkable libraries (i.e. for C/C++ programs). Librina is more a framework/middleware than a library: it has its own memory model (explicit, no garbage collection), its execution model is event-driven and it uses concurrency mechanics (its own threads) to do part of its work.

3. **Kernel components** (linux/net/rina). The kernel contains the implementation of the data transfer/data transfer control components of normal IPC Processes as well as the implementation of shim DIFs – which usually need to access functionality only available at the kernel. The Kernel IPC Manager (KIPCM) manages the lifetime (creation, destruction, monitoring) of the other component instances in the kernel, as well as its configuration. It also provides coordination at the boundary between the different IPC processes.
4. **Test applications and tools** (rina-tools). This package contains test applications and tools to test and debug the RINA Prototype. Right now the rina-tools package contains the rina-echo-time application, which can work both in "echo" (ping-like behavior between two application instances) or "performance" mode (iperf-like behaviour).

16.4.2.3 Open source

The IRATI software has been made available as open source. Interested users and developers can access the code at the IRATI github side [12], which includes documentation of the project, installation guides and tutorials on how to perform simple experiments. There is also a mailing list available to users and developers, in order to facilitate the interaction with the maintainers of the IRATI implementation.

16.4.3 Experimental evaluation of RINA on the FIRE infrastructure

16.4.3.1 Experimental evaluation of the shim DIF for hypervisors

In order to assess the possible gains from deploying the shim DIF for hypervisors in the DC, we measured the performance of the IRATI stack against the performance of the TCP/IP stack in Linux, when deployed to support VM networking. The full description of the experiment has been published in [13]. Note however up front that the IRATI stack is currently not optimized for performance yet. The tests reported in this section involve a single physical machine (the host) that acts as a hypervisor for one or two VMs. We performed two different test scenarios:

- Host-to-VM tests; where a benchmarking tool (rina-echo-time for IRATI tests and netperf for TCP/IP tests) is used to measure the goodput between a client running in the host and a server running on VM.
- VM-to-VM tests; where a benchmarking tool is used to measure the goodput between a client running on a VM and a server running on a different VM.

The measurements were taken on a processing system with two 8 core Intel E5-2650v2 (2.6 GHz) CPUs and 48 GB RAM. QEMU/KVM was chosen as the hypervisor, since it is one of the two hypervisors supported by the shim DIF for hypervisors provided by the IRATI prototype. For the host-to-VM scenario, three test sessions were executed. The first two tests sessions assess UDP goodput performance at variable packet size, therefore assessing the performance of traditional VM networking. The tap device corresponding to emulated NIC in the VM is bridged to the host stack through a Linux in-kernel software bridge.

The second test session makes use of a paravirtualized NIC model, the virtio-net device. Paravirtualized devices don't correspond to real hardware, instead they are explicitly designed to be used by virtual machines, in order to save the hypervisor from the burden of emulating real hardware. Paravirtualized devices allows for better performances and code reusability. The only difference between the first and the second test session is the model of the emulated NIC. In the first session, a NIC belonging to the Intel e1000 family is used, which is implemented in QEMU by emulating the hardware behaviour (full virtualization) – e.g. NIC PCI registers, DMA, packet rings, offloadings, etc. Despite being more virtualization-friendly than e1000 (or other emulated NICs like r8169 or pcnet2000), the guest OS still sees the virtio-net adapter like a normal ethernet interfaces, with all the complexities and details involved, e.g. MAC, MTU, TSO, checksum offloading, etc.

The third test session shows the performance of the shim DIF for hypervisors. A scenario comparable to the one deployed in the first and second test sessions involves a shim IPC process for hypervisors on the host and the corresponding one on the guest. No normal IPC processes are used, the applications can run directly over the shim DIF. This is a consequence of the flexibility of RINA, since the application can use the lowest level DIF whose scope is sufficient to support the intended communication (guest-to-host in this case) and that provides the required QoS.

The host runs our rina-echo-time application in server mode, while the guest runs rina-echo-time in client mode. Rina-echo-time is a simple RINA benchmarking application, which uses the IPC API to measure goodput. Each test run consists of the client sending to the server an unidirectional stream of PDUs of a specified size. Measurements have been taken varying the PDU size. The current maximum SDU size of the shim DIF for Hypervisors is the page size (4096 bytes on our machine). We repeated every measurement 20 times. The result of these goodput measurements for host-to-VM communication scenario are shown in Figure 16.9 – 95% confidence levels are also depicted, as well as a third degree polynomial regression line.

The shim DIF for hypervisors outperforms both e1000 and virtio-net NIC setups, which validates that a simpler and cleaner architecture allows for better performance, even with an unoptimized prototype. Next, similar goodput performance measurements were taken on the VM-to-VM scenario. Again, three test sessions were performed, the first two for traditional VM networking and the third one for IRATI stack. The setup of the first two sessions is very similar to corresponding one in the host-to-VM scenario. The VMs are given an emulated NIC, whose corresponding tap device is bridged to the host stack through a Linux in-kernel software bridge. Measurements are again performed with the netperf utlity, with the netperf server running on a VM and the netperf client running on the on the other VM.

In the case of the IRATI tests, point-to-point connectivity between host and VM is provided by the shim DIF for hypervisors. A normal DIF is

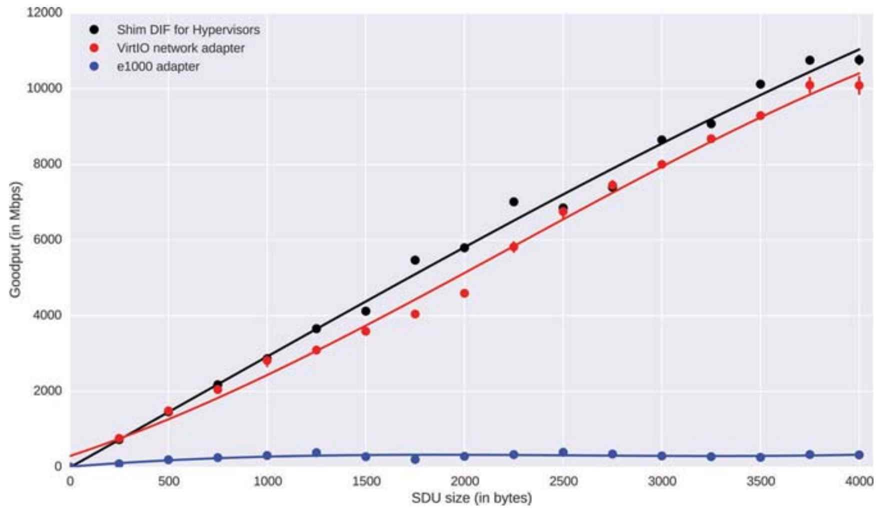


Figure 16.9 VM to Host goodput experiment results.

overlayed on these shim DIFs to provide connectivity between the two VMs. Tests are performed again with the rina-echo-time application, using a flow that provides flow control without retransmission control. Flow control is used so that receiver's resources are not abused. In TCP/IP, this kind of functionality – flow control without retransmission control – is not available. Hence we chose again UDP to perform the tests for the traditional networking solution, since its functionality is most similar. The result of these tests sessions are depicted in Figure 16.10. Full virtualization again performs poorly. The paravirtualized solution currently slightly outperforms the unoptimized IRATI stack. However, the IRATI prototype can still be optimized.

16.4.3.2 Evaluation of the link-state routing policy

The physical connectivity graph that we used for this experiment is shown in Figure 16.11. For each physical link, an instance of the shim DIF for 802.1Q was instantiated, after assigning a unique VLAN id to each link. A normal DIF was stacked on top of these shim DIFs. In this way, we show that routing works in a 1-DIF, a basic scenario. We performed tests with rina echo-time, an application that calculates the Round Trip Time (RTT) like the well-known ping tool. It calculates the time it takes for a client to send an SDU to the server and receive the same SDU back again. The server was running on node M. On every other node we ran the client and performed the test 50 times. The size of the SDU that was used was 64 bytes. We ran the application on top of a normal DIF, which runs on top of the shim DIF over 802.1Q.

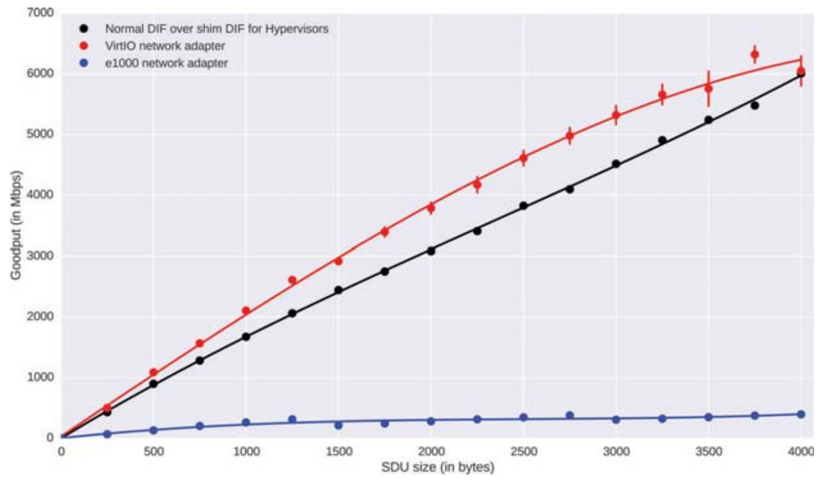


Figure 16.10 Host to Host goodput experiment results.

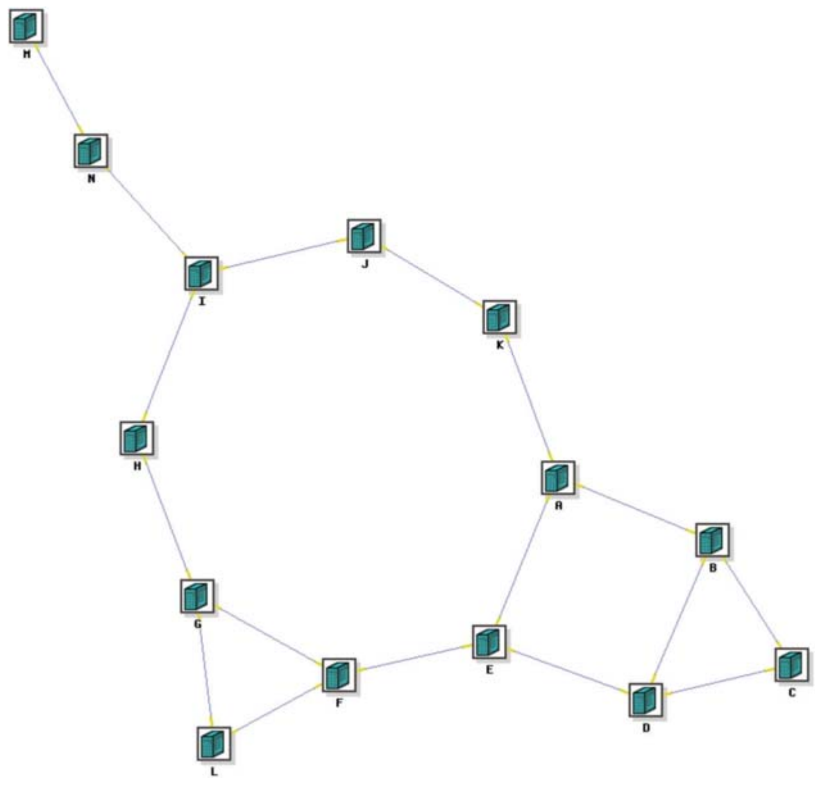


Figure 16.11 Physical connectivity graph for the routing experiments.

We did the experiment first with kernel-space debugging logs enabled, then we repeated the experiment with the debugging logs disabled (as these significantly stress the system) to get more accurate performance-oriented results. The results of the experiments can be seen in Figure 16.12. The data points on top represent the experiment with the kernel with logs enabled, while the data points at the bottom are the results of the experiment with the logs disabled. The further the distance from server M, e.g. the more hops needed to reach M, the longer the round trip time. In the case of a kernel with logs disabled, it takes 504.94 ± 68.10 µs to send and receive the same SDU again on node N, where no forwarding of SDUs was needed. Per extra node needed to forward the SDU, about 250 µs is added. Some nodes have the same distance to M, but their average round trip times differ somewhat from each other. In conclusion, the basic operation of the link-state routing policy has been verified.

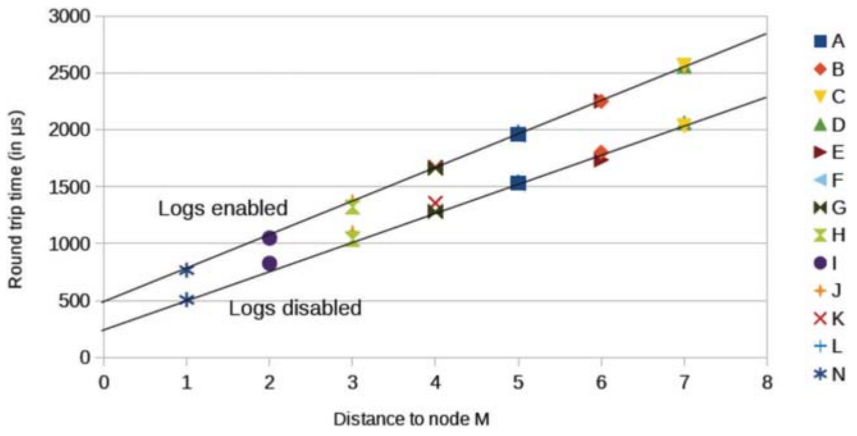


Figure 16.12 Results of the link-state routing experiments.

16.4.3.3 Performance evaluation on the iMinds OFELIA island

We carried out experiments to measure the performance of the phase 2 prototype. We executed them on the iLab.t Virtual wall, which is a controlled environment. The experiment depicted in Figure 16.13 was used to measure the performance. The complete results of these experiments have been published in [14].

The RINAperf client/server application is a RINA-native performance-measurement tool, measuring the available goodput between two application processes. We realised goodput measurements in the following 3 scenarios:

- The first scenario (Shim DIF + Application Process) runs the RINAperf application directly over the shim IPC process for 802.1Q. This scenario should be the fastest, but offers the least functionality.
- The second scenario (Shim DIF + Normal DIF A + Application Process) builds upon the first one by stacking a normal DIF on top of the shim IPC process for 802.1Q. A lot of functionality becomes immediately available since it is provided by the normal DIF (e.g. multihoming, QoS). This is a scenario that would be typically found in Local Area Networks (LANs), where the scope is the network.
- The third scenario (Shim DIF + Normal DIF A + Normal DIF B + Application Process) stacks another normal DIF on top of the previous. This scenario is added to show the influence of stacking multiple DIFs on top of each other and would the one be used in an internetwork: connecting together different networks.

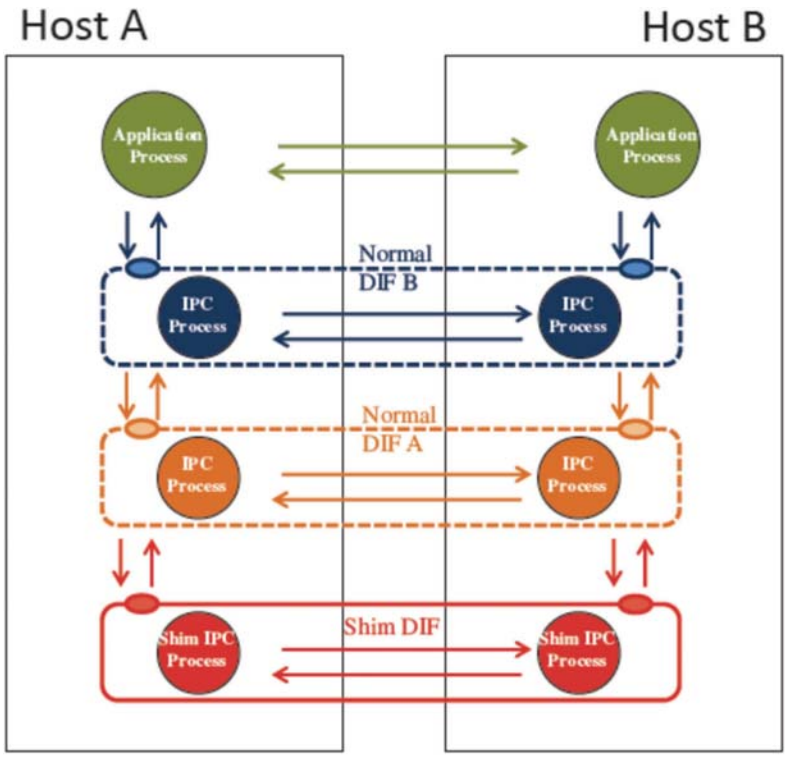


Figure 16.13 Scenario for the performance evaluation of prototype 2.

We executed experiments on 2 nodes from the iMinds OFELIA island iLab.t virtual wall aggregate. We used the RINAperf application to measure the maximum achievable goodput between two application processes given a certain SDU size. In all our experiments, we set the RINAperf timeout to 10 seconds and each one is repeated for different SDU sizes, ranging from 1 byte to the maximum SDU size for that scenario. The obtained goodput when these experiments were repeated in July can be seen in Figure 16.14. The values represent the mean of the goodputs obtained, together with their respective 95 percent confidence intervals (50 samples per interval).

As can be seen from Figure 16.14, the goodput increases as the SDU size increases, which is of course to be expected as the per-packet processing overhead due to the PCI headers is amortised over more bytes. The maximum throughput on the link when measured with iperf is 970 Mbit/s. Adding additional normal DIFs decreases the goodput, because of the extra processing

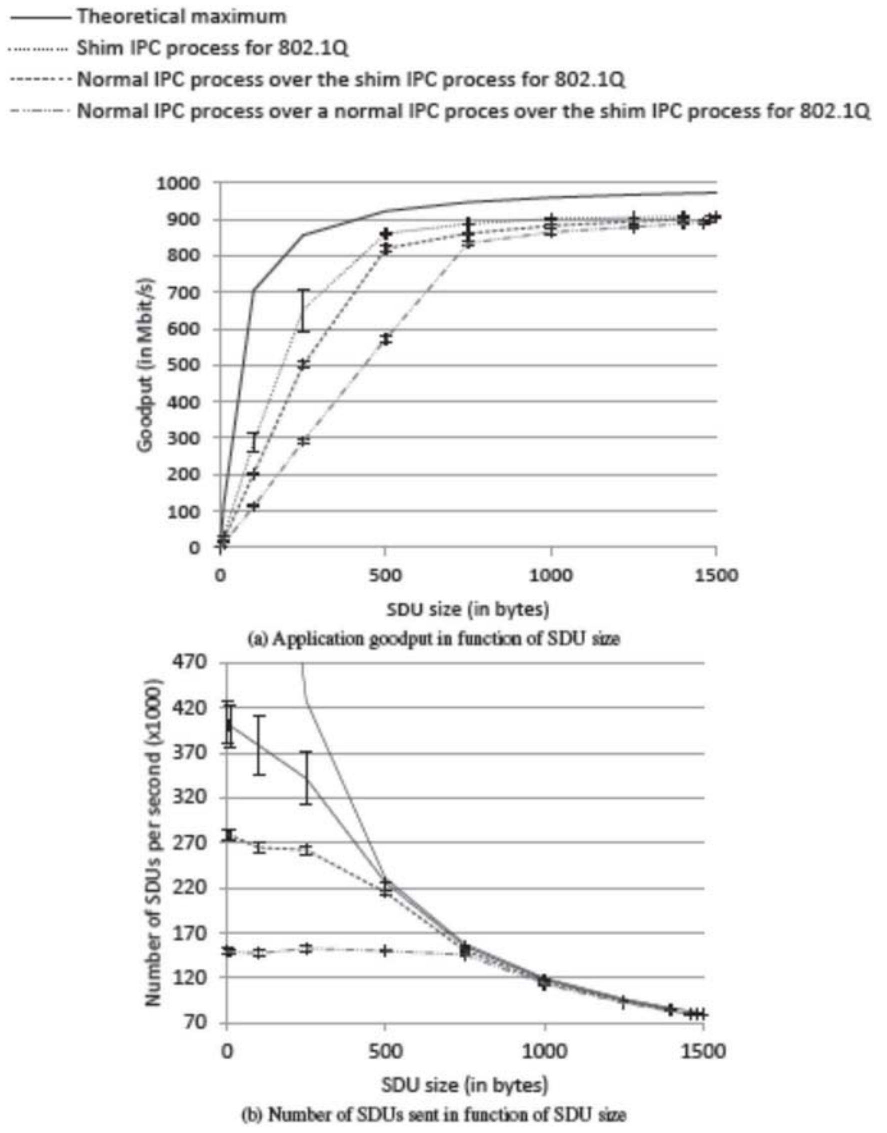


Figure 16.14 Performance evaluation results.

overhead and decreased maximum packet size. Tests executed with v0.8.0 at the maximum MTU allowed by the system recorded the mean goodput achieved as 907:67 ± 1:45 Mbit/s for scenario 1, 902:09 ± 9:32 Mbit/s for scenario 2, and 891:05 ± 6:91 Mbit/s for scenario 3. So each additional normal DIF incurred only a small performance penalty, and the overall goodput achieved was very close to line rate.

16.4.3.4 Validation of location-independence

First, the rina-echo-time server application registers with normal.DIF at System 3 (Figure 16.15). The "rina-echo-time" client application at System 1 allocates a flow to the server application, using the Application Name, sends a couple of SDUs and terminates. After that the rina-echo-time server at System 3 terminates, and an instance of the same application is registered at Sys tem 2. Then, the client application process at System 1 again allocates a flow to the server application process, sends a couple of SDUs and terminates again. Note that the application does not need any state updates. The DIF internally updates its Application-Name-to-IPC-Process address mappings in order to forward the flow allocation request to the right IPC process in both scenarios.

This section validated the location independence of Application Names and decoupling of connections and flows in the RINA architecture and the IRATI implementation. It shows correct operation when an application moves from on host (or, more accurately, from one IPCP process) to another. The directory of the underlying DIF is correctly updated, and the client application can reach the server application transparently without knowing it has moved location. This experiment illustrates one of the huge benefits RINA has with regards to application mobility due to it's location-independent naming scheme.

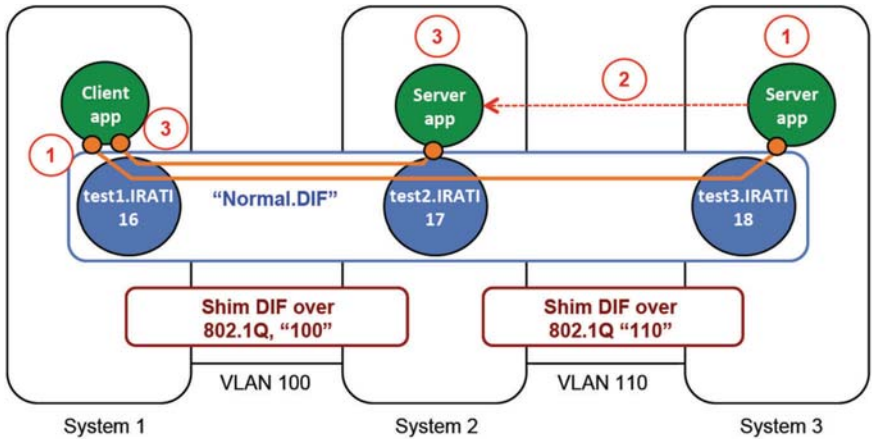


Figure 16.15 Experimental setup for application-location independence.

16.4.4 Feedback to the OFELIA Facility

16.4.4.1 IRATI VM image and XEN servers

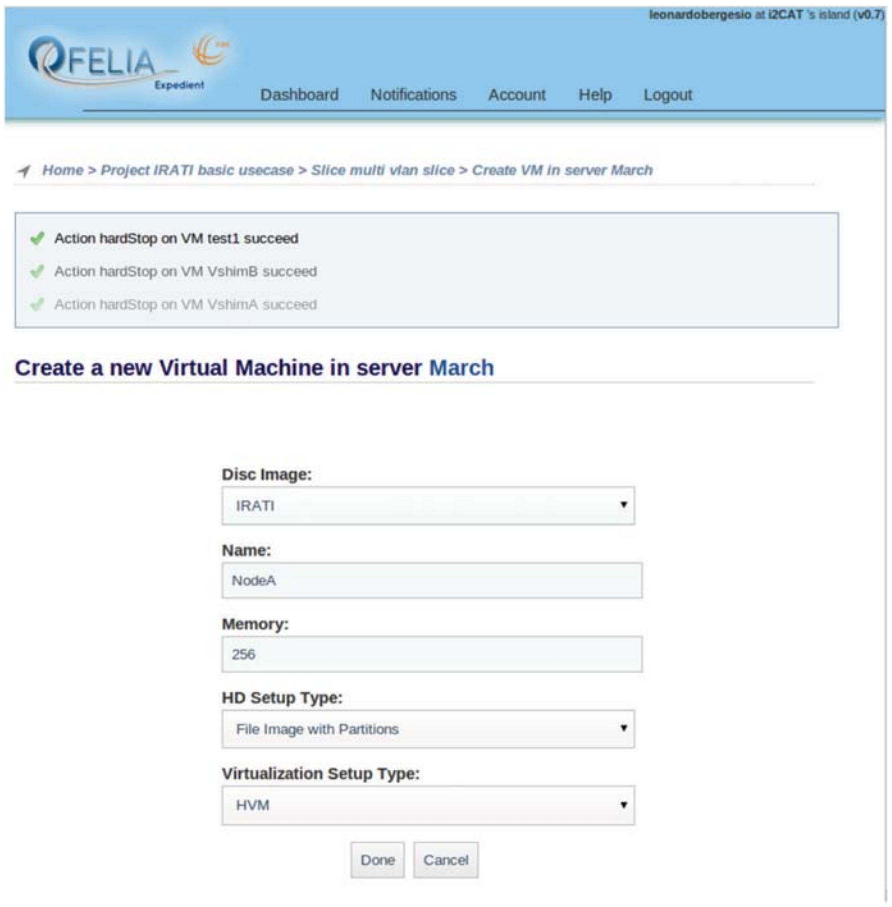
Many of the modules of the RINA stack are located in kernel space. Since the Linux kernel versions used by the IRATI software are not always the same available in OFELIA resources, the VMs used for IRATI experimentation need to be upgraded in order to support the deployment of the RINA prototype. The VMs in OFELIA use the same unmodified kernel that the host server uses. This constraint was completely blocking for IRATI purposes, as previously exposed. Nevertheless, OCF (OFELIA Control Framework) is capable of supporting almost any kind of images to create VMs, using any of the described virtualization mechanisms. In order to address IRATI's requirements it was decided to create a new VM image based on IRATI's reference machine. This image has been installed on the servers of the I2CAT Island, and in later phases of the project it will be spread over the rest of the islands for the inter-island experiments. The new image template was updated to support OFELIA's authentication and access control features: authentication modules that allow users whose credentials are stored at the OFELIA LDAP directory direct access to the VM using SSH were added to the VM.

Finally, to fully integrate the new image template in the OFELIA testbed, some additions in the OCF, were required. The development done was integrated in OCF v0.7 and can be visualized in Figure 16.16. The following paragraph describes the main additions performed to the OCF:

- The Expedient web UI was modified to allow the creation of IRATI VMs.
- The Virtualization Aggregate Manager (VT AM) was modified to support the new VM.
- The OFELIA XEN Agent (OXA) running in the XEN servers was modified in order to handle the automatic creation and modification of HVM virtualized machines to provide them with IP addresses and ssh credentials in order to be accessible by the testbed users.

16.4.4.2 VLAN translator box

The iLab.t virtual wall emulab infrastructure uses Virtual LANs (VLANs) in the central Force10 hub switch to separate traffic between different experi ments. The central switch does not support double tagging (802.1ad), so 802.1Q VLAN-tagged frames cannot be used inside an experiment in this testbed environment (all frames with Ethertype 0x8100 are dropped by the central switch). The shim DIF over Ethernet uses a VLAN tag as DIF name, so we patched the Linux kernel and Network Interface Controller (NIC)


leonardobergesio at i2CAT 's island (v0.7)
OFELIA Expedient
Dashboard
Notifications
Account
Help
Logout
Home > Project IRATI basic usecase > Slice multi vlan slice > Create VM in server March
Done
Cancel

Action hardStop on VM test1 succeed

Action hardStop on VM VshimB succeed

Action hardStop on VM VshimA succeed

Create a new Virtual Machine in server March

Disc Image: IRATI

Name: NodeA

Memory: 256

HD Setup Type: File Image with Partitions

Virtualization Setup Type: HVM

Figure 16.16 Creation of an IRATI VM with the OFELIA Control Framework.

device drivers of the physical machines to be used as RINA hosts. They use ethertype 0x7100 instead of 0x8100 for 802.1Q traffic, allowing transparent use of VLAN tags. In order to allow seamless operation within OFELIA, a translation needs to be done for traffic entering and leaving the iMinds virtual wall island. In order to translate the ethertype, we decided to implement a Linux Loadable Kernel Module (LKM) from scratch to do the translation. During this implementation it was found that the current kernel always untags incoming packets, stripping the VLAN header from the packet data and storing the VLAN header implicitly in a separate vlan tci field inside the socket buffer struct.

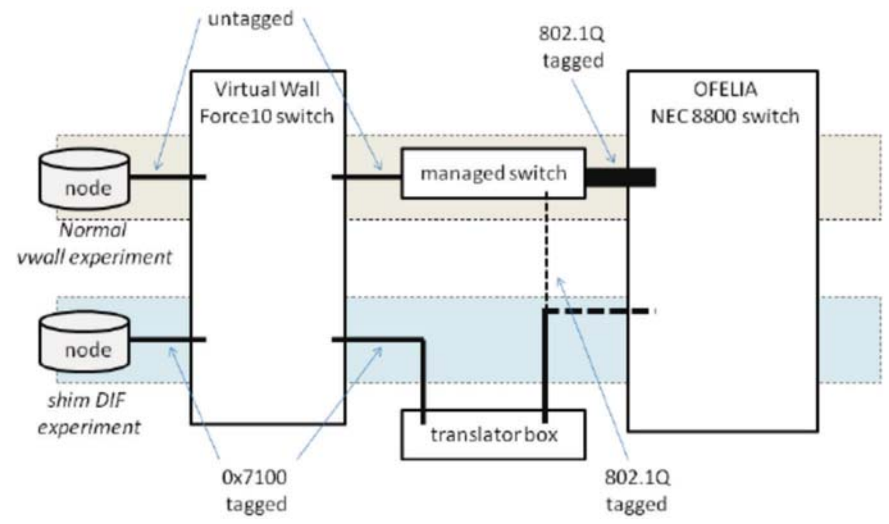


Figure 16.17 VLAN box in the OFELIA testbed.

The Linux LKM re-inserts the VLAN header (changing the ethertype to 0x7100) before sending the packet as “untagged” traffic. A server running this Linux kernel module is now integrated into the OFELIA infrastructure at iMinds. It may also be integrated into the infrastructure for Fed4FIRE. Some VMs in OFELIA do not support 1504 byte frames, so the Maximum Transmission Unit (MTU) will have to be reduced by 4 bytes to allow the VLAN header to be correctly inserted. This is not necessary between machines in the virtual wall.

Figure 16.17 shows (on top) the normal configuration for a virtual wall experiment. These use untagged traffic, and a managed switch is used to tag the traffic with the OFELIA-compatible VLAN tag, aggregate all traffic, and forward it over a 10 Gbit/s fiber to the central OFELIA switch (and vice versa for traffic coming from OFELIA). For traffic that is tagged (as explained, with a 0x7100 ethertype), the ethertype is translated to a normal 802.1Q 0x8100 one by the translator box, and then offered to either the managed switch for aggregation, or directly to a free port on the OFELIA switch.

16.5 Conclusions

In spite of being a small and relatively short research project, the impact of the work performed within the FP7 IRATI project is producing a long lasting impact. First of all IRATI produced the first kernel-based RINA implementation for the Linux/OS that can be overlaid on top of Ethernet, TCP/UDP, and shared memory mechanisms for Guest to Host communication in a Virtualized environment. Open sourcing this implementation has reduced the barriers of entry to researchers and innovators wanting to experiment with RINA. Second, initial experimental results of the RINA prototype over the FIRE testbeds have shown some of the RINA benefits in practice. Third, the IRATI project allowed the creation of a core group of European partners with expertise in RINA design and implementation. Last but not least, the dissemination and outreach activities of the project contributed to better position RINA in the radars of the industry, academia, funding bodies and Standard Development Organisations.

All these efforts have crystallized in the funding of three EC research projects which directly exploited FP7 IRATI results, in particular the open source RINA implementation: i) IRINA [15] studied the applicability of RINA in National Research Education Networks and GEANT; ii) PRISTINE [16] is working on bringing programmability to the IRATI implementation and studying/experimenting with policies in the areas of congestion control, resource allocation, routing, security and network management, and ARC-FIRE [17] is investigating the benefits of applying RINA to the design of converged operator networks and hardening the IRATI implementation to enable large-scale experiments on FIRE+ testbeds. Moreover, RINA is being considered for standardisation in the context of ETSI's Next Generation Protocols ISG [18] and ISO's SC6 Working Group 7 on Future Networks [19].

References

[1] J. Day, "*Patters in Network Architecture: A Return to Fundamentals*". Prentice Hall, 2008.

[2] J. Day, I. Matta, and K. Mattar. 2008. "*Networking is IPC: a guiding principle to a better Internet". In Proceedings of the 2008 ACM CoNEXT Conference (CoNEXT '08).*

[3] J. Day. "*About layers: more or less*". PSOC Tutorial, available online at http://pouzinsociety.org

[4] V. Maffione, F. Salvestrini, E. Grasa, L. Bergesio, and M. Tarzan, "A Software Development Kit to exploit RINA programmability". *IEEE ICC 2016, Next Generation Networking and Internet Symposium.*

[5] R. W. Watson. "Timer-based mechanisms in reliable transport protocol connection management". Book in innovations in Networking, pages 296–305. Artech House Inc.

[6] FP7 OFELIA Project website. Available online at http://www.fp7-ofelia.eu

[7] The IRATI consortium. "Specification of a Shim DIF over 802.1Q layers", available as part of IRATI's D2.3: http://irati.eu/deliverables-2/

[8] The IRATI consortium. "Specificafion of a Shim DIF for Hypervisors", available as part of IRATI's D2.3: http://irati.eu/deliverables-2/

[9] The IRATI consortium. "Specification of a Link-State based routing policy for the PDU Forwarding Table Generator"; available as part of IRATI's D2.4.: http://irati.eu/deliverables-2/

[10] J. Day, E. Trouva, E. Grasa, P. Phelan, M. P. de Leon, S. Bunch, I. Matta, L. T. Chitkushev, and L. Pouzin, "*Bounding the Router Table Size in an ISP Network Using RINA,"* Network of the Future (NOF), 2011.

[11] S. Vrijders, D. Staessens, D. Colle, F. Salvestrini, E. Grasa, M. Tarzan and L. Bergesio "Prototyping the Recursive Internetwork Architecture: The IRATI Project Approach", *IEEE Network,* Vol. 28, no. 2, March 2014.

[12] IRATI open source RINA implementation. Available online at https://github.com/IRATI/stack

[13] S. Vrijders, V. Maffione, D. Staessens, F. Salvestrini, M. Biancani, E. Grasa, D. Colle, M. Pickavet, J. Barron, J. Day, and L. Chitkushev 'Reducing the complexity of Virtual Machine Networking'. *IEEE Communications Magazine,* Vol. 54. No. 4, pp. 152–158, April 2016.

[14] S. Vrijders, D. Staessens, D. Colle, F. Salvestrini, V. Maffione, L. Bergesio, M. Tarzan, B. Gaston, E. Grasa; "Experimental evaluation of a Recursive InterNetwork Architecture prototype", *IEEE Globecom 2014,* Austin, Texas.

[15] IRINA Project website. Available online at http://www.geant.org/Projects/GEANT_Project_GN4/Pages/Home.aspx#IRINA

[16] FP7 PRISTINE Project website. Available online at http://ict-pristine.eu

[17] H2020 ARCFIRE Project website. Available online at http://ict-arcfire.eu

[18] ETSI NGP ISG website. Available online at http://www.etsi.org/technologies-clusters/technologies/next-generation-protocols

[19] ISO SC6 website. Available online at http://www.iso.org/iso/iso_technical_committee.html%3Fcommid%3D45072