Proyecto Final de Máster de Software libre

Especialidad Administración de redes y sistemas operativos

Prototipo de RINA sobre Ethernet

Autor:
Leonardo Bergesio

Consultor:
Miguel Martín Mateo

Tutor externo:
Eduard Grasa

28 de diciembre de 2013

Resumen

Debido a las exigentes aplicaciones y redes emergentes, Internet se ha convertido en una arquitectura repetidamente parcheada y de complejidad creciente con el objetivo de hacer frente a los cambios y requerimientos. Sin embargo, la base del problema no radica en las altas exigencias de las aplicaciones y servicios actuales, sinó en el diseño original de Internet, algo que la ley de Moore nos impedía reconocer al aumentar la capacidad del hardware. TCP/IP y el conjunto de protocolos sobre ellos que parchean Internet, no son capaces de hacer frente a los actuales requerimientos, o al menos no lo serán a largo término.

En este contexto, la Recursive InterNetwork Architecture (RINA) aparece como una arquitectura alternativa y novedosa cuyo principio básico es que networking es llanamente InterProcess Communication (IPC). Bajo este lema, RINA reconstruye la estructura general de Internet, creando un modelo formado únicamente por una capa que se repite recursivamente, la Distributed IPC Facility (DIF) Esta capa contiene la funcionalidad mínima e indispensable para proveer IPC entre aplicaciones y procesos. RINA soporta de forma inherente movilidad, multi-homing y Quality of Service (QoS) en un entorno seguro y configurable. Por otro lado, RINA es fácilmente adoptable sobre las tecnologías existentes debido a DIFs especiales cuyo cometido es esta adaptación.

Este PFM: "Prototipo de RINA sobre Ethernet" se enmarca dentro del trabajo a realizar por el proyecto IRATI [1]. IRATI es un proyecto STReP (Specific Targeted Research Project) financiado por la Unión Europea dentro del programa FP7 (Seventh Framework Programme for Research and Technological Development) [2]. El objetivo general de IRATI es conseguir una mayor compresión y exploración de RINA. El trabajo que se reportará en este PFM es (aproximadamente) la primera fase del diseño y desarrollo de un prototipo de RINA sobre Ethernet en el seno del kernel (Linux), basándose y generando software libre.

Lista de acrónimos

AP Application Process

ARM Address Resolution Module

CACEP Common Application Connection Establishment Protocol

CDAP Common Distributed Application Protocol

CRUD Create, Retrieve, Update, Delete

CLI Command Line Interface

COW Copy-On-Write

DIF Distributed IPC Facility

DMS DIF Management System

DTCP Data Transfer Control Protocol

DTP Data Transfer Protocol

DU Data Unit

DUT Device Under Test

EFCP Error and Flow Control Protocol

FA Flow Allocator

GPA Generic Protocol Address

GHA Generic Hardware Address

IDD Inter DIF Directory

IP Internet Protocol

IPC InterProcess Communication

IRM IPC Resource Manager

KIPCM Kernel IPC Manager

KFA Kernel Flow Allocator

MPL Maximum Packet Lifetime

OCF OFELIA Control Framework

OS Operating System

PCI Protocol Control Information

PDU Protocol Data Unit

PFTG PDU Forwarding Table Generator

PFT PDU Forwarding Table

PSOC Pouzin Society

QoS Quality of Service

RA Resource Allocator

RIB Resource Information Base

RINA Recursive InterNetwork Architecture

RMT Relaying and Multiplexing Task

RNL RINA Netlink Layer

SDU Service Data Unit

TCP Transmission Control Protocol

UDP User Datagram Protocol

VLAN Virtual Local Area Network

VPN Virtual Private Network

Índice de figuras

Índice de tablas

Índice general

1. Introducción

Este documento reporta el trabajo hecho para el PFM "Prototipo de RINA sobre Ethernet", realizado en la modalidad profesional. Las prácticas externas se desarrollaron en la Fundació i2CAT [3]. El trabajo se enmarca dentro del proyecto IRATI [1], abarcando un subconjunto de sus objetivos.

La estructura seguida en el documento comprende las siguientes secciones:

En la sección 1 se introduce el proyecto continuando el resumen presentado en la primera página. Además, se describen los objetivos del proyecto y se hace una breve introducción a los conceptos detrás de RINA. Dado que es imposible abarcar la definición de la arquitectura, y que por otro lado eso escapa a los objetivos del PFM, todo el texto se apoya en referencias de muy recomendable lectura de cara a entender los detalles de la información presentada en este documento. La sección 2 explica el análisis de requerimientos y produce una lista de éstos a partir de distintas fuentes como las especificaciones, casos de usos, etc. La sección 3 es la más extensa y se centra en el diseño de la arquitectura de sofware del prototipo RINA sobre Ethernet. La sección 4 se adentra en aspectos relacionados con el desarrollo y las herramientas utilizadas para la producción de código. El código producido corresponde con aproximadamente la primera fase de desarrollo del proyecto IRATI, y se nutre del diseño desarrollado en la sección 3. Cabe destacar que el diseño de la arquitectura de software y su desarrollo se retroalimentan cíclicamente. En la sección 5 se presenta la metodología de testing utilizada y los casos de uso para los experimentos de validación del prototipo mediante una batería de tests unitarios, de regresión y funcionales desplegados en dos entornos distintos. Finalmente, la sección 6 presenta los resultados conseguidos, así como las conclusiones obtenidas y el trabajo futuro.

1.1 Objetivos

Como ya se introdujo en secciones anteriores, este PFM se centra en un subconjunto de los objetivos del proyecto FP7 IRATI en el cual se realizaron las prácticas externas. IRATI trabaja en la mejora y desarrollo de un modelo de referencia y especificaciones de RINA que den lugar al desarrollo de un prototipo funcional eventualmente utilizable en entornos de producción. Dado que el periodo del PFM corresponde a menos de la primera fase de diseño e implementación de IRATI, no se espera que el prototipo implemente toda la funcionalidad RINA, sinó que el foco está en el desarrollo de un stack funcional en el kernel (Linux) y la adaptación a la capa Ethernet.

Cabe destacar que el desarrollo de este prototipo fue una actividad de investigación en sí misma, dado que las especificaciones de RINA no se encuentran en un estado de madurez que permita dicha implementación, con lo cual el desarrollo de éstas se antoja como una actividad paralela.

A continuación se lista y describen los objetivos de este PFM. La lista incluye tanto el objetivo principal correspondiente al desarrollo del prototipo, como aquellos objetivos secundarios que se ven afectados por el resultado final del prototipo o son inherentes a su desarrollo:

- **Prototipo open source de RINA sobre Ethernet para un Operating System (OS) UNIX.**
  El prototipo se presenta como el principal objetivo del PFM y el que mejor puede contribuir al impacto de IRATI. Además de ser la principal vía de experimentación, proveerá una base sólida para continuar el trabajo sobre RINA. Al final del proyecto, cuando el prototipo se encuentre en un estado más consolidado, se plantea la posibilidad de crear una comunidad open source alrededor de él. La funcionalidad mínima que se espera es la comunicación de dos aplicaciones en distintos sistemas mediante una DIF que adapte la capa Ethernet. mínimos que se reporta

- **Mejora del modelo de referencia de la arquitectura RINA y sus especificaciones, centrándose en la DIF sobre Ethernet.**
  Dado que las especificaciones de RINA no se encuentran en un estado consolidado, el propio diseño y desarrollo del prototipo necesitará completarla, centrándose en tres aspectos:

  - la especificación de la DIF sobre Ethernet como el medio físico.
  - la terminación de las especificaciones de RINA que permitan un nivel de servicio similar a la Internet de hoy (baja seguridad, best-effort).
  - los casos de uso de IRATI sobre aspectos donde TCP/IP flaquea (movilidad, multi-homing...)

- **Validación experimental del prototipo y de RINA y comparación con TCP/IP.**
  Mediante la utilización del testbed OFELIA[30], el desarrollo seguirá ciclos de investigación-diseño-implementación-experimentación, analizando el prototipo RINA contra TCP/IP en aspectos como APIs para las aplicaciones, coste para soportar multi-homing, vulnerabilidad, utilización de hardware, etc... Los resultados de la experimentación servirán como input para el siguiente ciclo.

- **Interoperabilidad con el prototipo RINA sobre UDP/IP de la Pouzin Society (PSOC)** [4]. Existe otro prototipo RINA sobre UDP/IP desarrollado en JAVA. La interoperabilidad de ambos, considerando que se encuentran implementados sobre distintas plataformas (middleware vs. OS kernel) y funcionan sobre distintas tecnologías (UDP/IP vs. Ethernet) contribuirá a la validación de las especificaciones.

1.2 Introducción a RINA

Los principios de RINA fueron presentados en el libro de John Day [5] "Patterns in Network Architecture: A return to Fundamentals" [6]. El trabajo presentado en dicho libro es un comienzo desde cero teniendo en cuenta las lecciones aprendidas en los 35 años de vida de los protocolos TCP/IP, los fallos de la OSI y las lecciones aprendidas de otras tecnologías de red de las últimas décadas como CYCLADES [7], DECNET [8] or XNS [9].

RINA toma como punto de partida la premisa básica de que networking es llanamente IPC y sólo IPC. La red provee los medios mediante los cuales los procesos en distintos sistemas se pueden comunicar, generalizando el modelo de IPC local en un único sistema. En un OS para permitir a dos procesos comunicarse, la funcionalidad de IPC requiere ciertas funciones como localización de los procesos afectados, determinación de permisos, transferencia de información, aprovisionamiento de recursos y administración de la memoria. El conjunto

de componentes del OS que realizan esta función podría ser consecuentemente llamado IPC Facility. Análogamente, dos aplicaciones (procesos) corriendo en sistemas distintos se comunican y comparten estado mediante la utilización de una Distributed IPC Facility (DIF). La Fig. 1.1 muestra distintos ejemplos de dos procesos de aplicación A y B comunicándose i)en un mismo sistema, ii)entre dos sistemas comunicados directamente y iii)entre dos sistemas conectados mediante un sistema dedicado a proveer servicios de IPC (por ejemplo un router). Cada escenario requiere una o más DIFs optimizadas para proveer IPC como un servicio sobre un cierto alcance o ámbito.

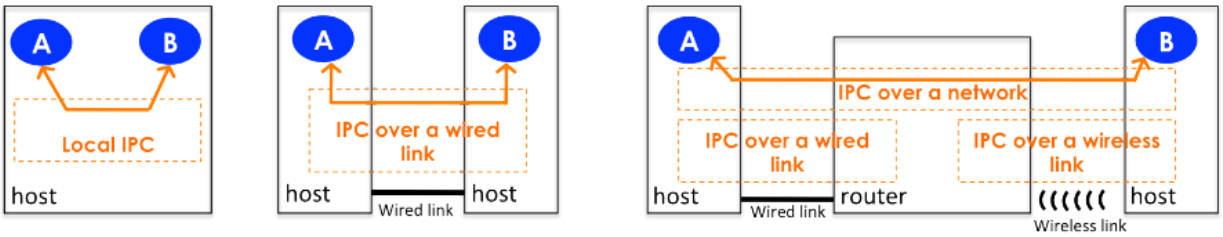


Figura 1.1: Ejemplos de DIFs

Una DIF es una estructura organizativa que agrupa procesos de aplicaciones que proveen servicios de IPC y está configurada mediante ciertas políticas específicas. Una DIF podría ser interpretada como lo que normalmente se llama "capa" o "layer". Según esta visión, networking no se trata de una serie de capas que agrupan distintas funcionalidades, sinó de una única capa de IPC distribuido que se repite con distintos alcances ofreciendo los mismos mecanismos o funcionalidades, pero especialmente configuradas mediante un conjunto de políticas propias para operar a distintos niveles de rendimiento (capacidad, pérdidas, retrasos, etc). La Fig. 1.2 muestra un ejemplo de esta arquitectura. Cada DIF ofrece servicios IPC con un determinado alcance. El primer nivel de DIFs operan sobre el medio físico, y sus políticas están optimizadas para ese entorno ofreciendo servicios a las DIFs superiores. A su vez, la segunda capa de DIFs son iguales a las primeras, pero con configuraciones (políticas) diferentes para cumplir con los requerimientos de la capa donde se encuentran. En esencia, una DIF es únicamente una aplicación distribuida, cuyos miembros, procesos de aplicaciones llamados procesos IPC, se especializan en ofrecer servicios IPC distribuidos. De esta manera, una DIF no es estructuralmente distinta a cualquier otra aplicación distribuida, únicamente se centra en realizar bien una única tarea: IPC.

A modo de resumen y presentación, RINA presenta las siguientes capacidades:

- Se construye sobre la premisa sencilla de que la red no es un conjunto de capas de distinta funcionalidad, sinó una única capa de IPC distribuida que se repite recursivamente ofreciendo las mismas funcionalidades (delay, capacidad, pérdidas) en un ámbito o con un alcance para el que están específicamente configuradas y optimizadas.

- Presenta una teoría completa de networking en lugar de presentar un parche para un problema específico (como la capa de sesión en OSI que ofrece una mínima funcionalidad extra para puentear redes de ISPs). El número de DIFs dependerá del rango en que debe operar la red, siendo éste un factor clave para diseñar la red dependiendo de los servicios que deba proveer.

- La estructura recursiva escala indefinidamente (obviando limitaciones físicas), evitando

problemas como el crecimiento excesivo de la tabla de rutas y soporta inherentemente movilidad y multi-homing con mínimo coste.

- Una aplicación que quiera hacer uso de la red sólo sabe el nombre del proceso de aplicación de destino (no conoce de direcciones ni "well-known ports"). Para conectarse a una DIF la aplicación debe autenticarse y cumplir con las políticas de seguridad específicamente configuradas en la DIF, ofreciendo así mayor seguridad.

- Apilar DIFs permite crear redes desde capas más pequeñas y manejables con menor alcance, de modo que se tiene un mejor control de los recursos evitando el sobreaprovisionamiento para mantener la calidad de servicio, permitiendo un mejor uso de las subredes (mayor que el 30 %-40 % de la actual Internet).

- RINA hace una clara distinción entre mecanismos: el mínimo set de funcionalidades que toda DIF debe proveer, y políticas: configuraciones específicas de los mecanismos (concepto bien conocido en los sistemas operativos).

- Las DIFs pueden configurarse para no sólo ofrecer los servicios típicos de las capas inferiores, sinó también aquellos servicios de aplicaciones como email, peer-to-peer, etc; removiendo la barrera de la capa de transporte de la actual Internet y abriendo nuevas posibilidades para los ISPs para proveer nuevos servicios.

- Las redes privadas (con direcciones privadas) son el caso normal. Los procesos IPC se identifican con direcciones internas a la DIF. De esta manera no existe una única red (direcciones públicas) donde todos deben conectarse, sinó que es posible elegir qué servicios y redes proveer y a qué red conectarse.

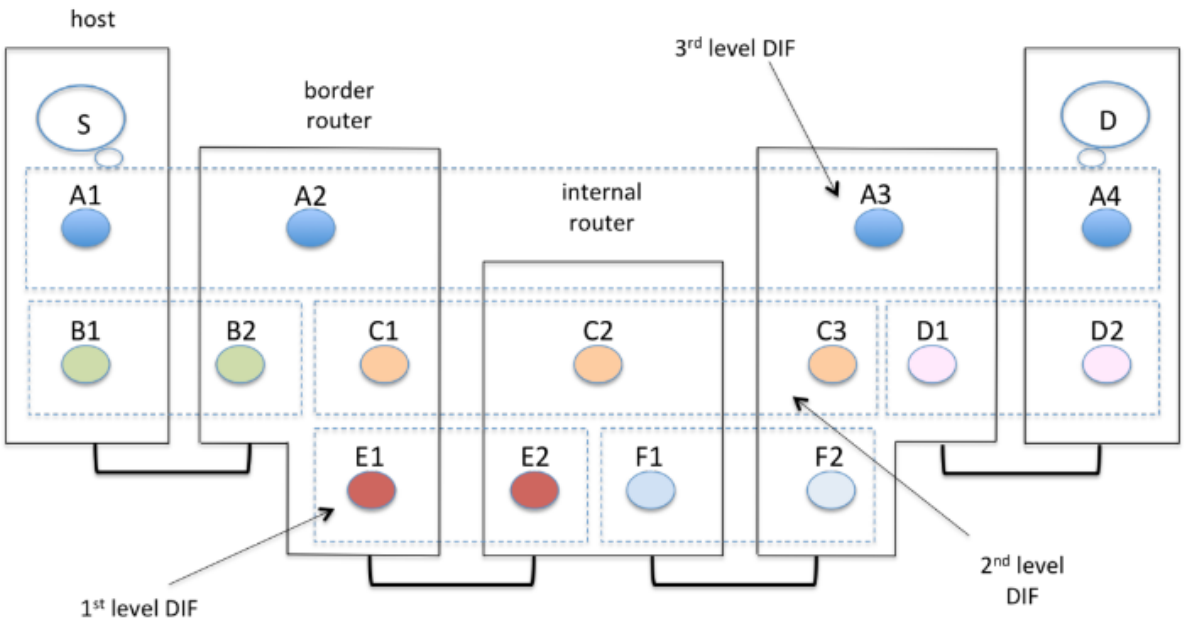


Figura 1.2: Ejemplo de la arquitectura de RINA. Los rectángulos verticales representan sistemas (máquinas), las líneas negras conexiones, los rectángulos horizontales DIFs y los círculos procesos IPC cuyo color indican la pertenencia a una DIF específica.

2. Análisis de requerimientos

Esta presentación presenta un listado de los requerimientos identificados para el desarrollo del prototipo. Los requerimientos se listan con un nombre identificativo que describe el contexto que lo origina, el cual da lugar a las distintas subsecciones.

2.1 Especificaciones

El sistema a diseñar y desarrollar debe cumplir todas las especificaciones impuestas por la arquitectura de RINA [10]. Éstas se presentan como un gran conjunto de requerimientos a cumplir por el prototipo que guiarán el diseño y desarrollo. Por otro lado, una de las tareas que también se realizarán en el proyecto es completar y mejorar estas especificaciones, en especial las de la shim DIF sobre Ethernet.

- **ESPEC-1.** Cumplir las especificaciones y el modelo funcional de RINA.
- **ESPEC-2.** Desarrollar las especificaciones de la shim DIF sobre Ethernet.

2.2 Interoperabilidad

La interoperabilidad entre distintas implementaciones de la arquitectura es muestra de unas buenas especificaciones y de un sistema maduro. De esta manera, la interoperabilidad del prototipo desarrollado y de los otros existentes se presenta como un test ha realizar, cuyo correcto resultado se presenta como un requerimiento:

- **INTEROP-1.** El prototipo debe poder comunicarse con otros prototipos existentes y al menos poder realizar el subconjunto de operaciones que ambos prototipos testeados soporten en común.

2.3 Diseño, implementación y guías de referencia

El desarrollo del prototipo se realizará sobre un sistema GNU/Linux contemplando el OS como un todo, en el cual los componentes de la arquitectura se distribuyen tanto en el espacio de usuario como en el kernel. Esto implica elegir adecuadamente los mecanismos de comunicación entre los distintos espacios prestando especial atención a los cambios de contexto que se producen más asiduamente con el objetivo de optimizar estos procesos (fast-path) lo máximo posible en busca de maximizar la latencia de las operaciones. La cooperación y comunicación entre los componentes también es clave, haciendo necesario mantener el nivel de concurrencia lo más bajo posible con el objetivo de no consumir recursos de forma exagerada.

- **DISEÑO-1.** Diseñar la distribución de los componentes entre user-space y el kernel identificando aquellos que pertencen al fast-path.
- **DISEÑO-2.** Diseñar las interfaces entre los distintos espacios.
- **DISEÑO-3.** Optimizar el nivel de concurrencia y uso de memoria.

Por otro lado, es necesario elegir una distribución madura y estable del OS. a la vez que es necesario continuamente sincronizar la versión del kernel con la última versión suministrada por la Linux Foundation con el objetivo de facilitar la integración del prototipo final. Este entorno da lugar a los siguientes requerimientos:

- **IMPLEMENTACION-1.** Elegir una distribución madura y estable: se trabajará sobre Debian 7 Wheezy.
- **IMPLEMENTACION-2.** Actualizar el código base del kernel continuamente con la última versión suministrada por la Linux Foundation.
- **IMPLEMENTACION-3.** Potenciar el uso de las estructuras y mecanismos ofrecidos por el kernel durante el desarrollo del prototipo con el objetivo de facilitar su adopción.
- **IMPLEMENTACION-4.** Desarrollar maximizando la reutilización del código o mediante la implementación de librerías.

2.4 Casos de uso

El objetivo de los casos de usos es delimitar el escenario para los tests de integración que verificarán el funcionamiento del prototipo. A su vez, estos escenarios determinarán una serie de requisitos adicionales a los impuestos por las propias especificaciones existentes. En esta sección no se describirán los casos de uso, sinó que se especificarán los requerimientos obtenidos. Para una descripción de los casos de uso se ofrecen las siguientes referencias: [11], [12] y la Sección 5 de este documento.

2.4.1 Shim DIF sobre Ethernet

- **SHIM-DIF-1.** El sistema debe proveer un fichero de configuración a través del cual el administrador de la red puede especificar qué shim IPC processes deben ser creados al arrancar el sistema.
- **SHIM-DIF-2.** El sistema debe proveer una Command Line Interface (CLI) para permitir al administrador de la red crear y destruir shim IPC processes, listar los procesos en el sistema junto con su estado (aplicaciones en el directorio, flows instanciados, etc).
- **SHIM-DIF-3.** El sistema debe proveer un agente de administración que permita las acciones antes descritas de forma remota a través de un DIF Management System (DMS).
- **SHIM-DIF-4.** Cada sistema debe soportar la existencia concurrente de más de un shim IPC process, con lo cual hace falta que el sistema cuente con varias tarjetas Ethernet, y éstas puedan ser miembros de varias Virtual Local Area Networks (VLANs). Cada shim IPC process debe ser debidamente aislado de los otros en el mismo sistema siendo imposible la comunicación directa entre ellos.
- **SHIM-DIF-5.** Cada shim IPC process debe permitir a una aplicación registrarse en cada shim IPC process (de la shim DIF). Dadas las restricciones de Ethernet, únicamente una aplicación por vez puede registrarse en el mismo shim IPC process. La Aplicación debe poder decidir cuando terminar el registro y dejar de estarlo.
- **SHIM-DIF-6.** Cada shim IPC process debe permitir a una aplicación reservar un flow hacia otra aplicación previamente registrada en la shim DIF mediante el nombre de la aplicación de destino. Si más de un flow es pedido entre el mismo par de aplicaciones, el shim IPC process devolverá un error o el indicador del flow ya reservado (handler, ID del puerto). Los shim IPC processes deben soportar la petición de una Aplicación para destruir un flow reservado y activo.
- **SHIM-DIF-7.** Las Aplicaciones deben poder enviar (escribir) datos en el flow y obtener (leer) información del mismo.

2.4.2 Caso de uso DIF normal

Los requerimientos para la DIF normal no son más que un superconjunto de los presentados en la sección anterior. De esta manera los requisitos de DIF-1 a DIF-4 se mapean a SHIM-DIF-1 a SHIM-DIF-4 respectivamente, a excepción de un añadido para DIF-2 y DIF-3:

- El sistema debe permitir al administrador poder hacer que un proceso IPC inicie la inscripción (a partir de ahora enrollment) con un proceso IPC remoto, o bien terminar la conexión de aplicación del proceso con otro miembro de la DIF.

Los requerimientos desde DIF-5 a DIF-8 son los mismos que desde SHIM-DIF-5 a SHIM-DIF-8 respectivamente, a excepción de que los procesos IPC deben permitir múltiples aplicaciones registradas y múltiples flows entre un mismo par de procesos IPC.

- **DIF-9.** El sistema debe proveer a las aplicaciones una forma de listar los nombres y propiedades de todas las DIFs a las que pueden acceder.

3. Diseño de la arquitectura

En esta sección se introducen los distintos componentes de la arquitectura, su función, sus interfaces y sus interacciones. Algunos de los componentes representa un módulo o tarea del modelo de referencia de RINA mientras que otros se presentan como utilitarios al servicio de la implementación. Este documento hace referencia a conceptos funcionales de la arquitectura RINA y al modelo de referencia [10], pero sin adentrarse en ellos en profundidad.

Los distintos componentes del stack se han distribuido entre el espacio de usuario (desde ahora user-space) o el kernel. Su ubicación final es motivada por cuestiones de rendimiento que se detallarán, o bien restricciones relacionadas con la propia programación. Las funcionalidades de bajo nivel se sitúan en el kernel apuntando a componentes de bajos requisitos de memoria y menor complejidad, pero que se ejecutan de forma intensa y repetida. Estos son los elementos de los IPC processes que se encargan de la transmisión de datos, moviendo estructuras de información entre las aplicaciones y el medio físico, el denominado "fast-path". Aquellos componentes que se ejecutan menor cantidad de veces (como la capa de configuración o administración) pueden soportar menor prioridad sin notable degradación del rendimiento, el denominado "slow-path", tiende a ubicarse en el user-space, siempre intentando minimizar el cambio de contexto al pasar de un espacio a otro.

Con tal de potenciar la reutilización de código, uno de los objetivo del diseño es la organización de un framework (Fig. 3.1) en librerías utilizadas por los distintos componentes, el cual alimenta a los componentes de la arquitectura resultante que puede verse en Fig. 3.2.

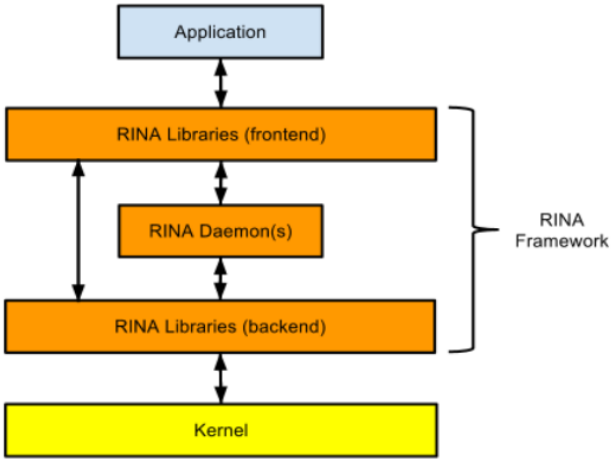


Figura 3.1: Framework del software RINA

A continuación se introducen los componentes más importantes del stack de RINA en cada espacio. En la siguientes secciones se analizará cada uno con mayor profundidad.

En user-space:

- **Application Process (AP):** Esta es una aplicación normal que utiliza los servicios de RINA para comunicarse con otras aplicaciones. Ejemplos podrían ser navegadores, clientes de email, etc.
- **IPC Manager** (implementado como un daemon del OS): Es único en el sistema y se encarga de administrar el stack de RINA en el sistema. Es el encargado de manejar

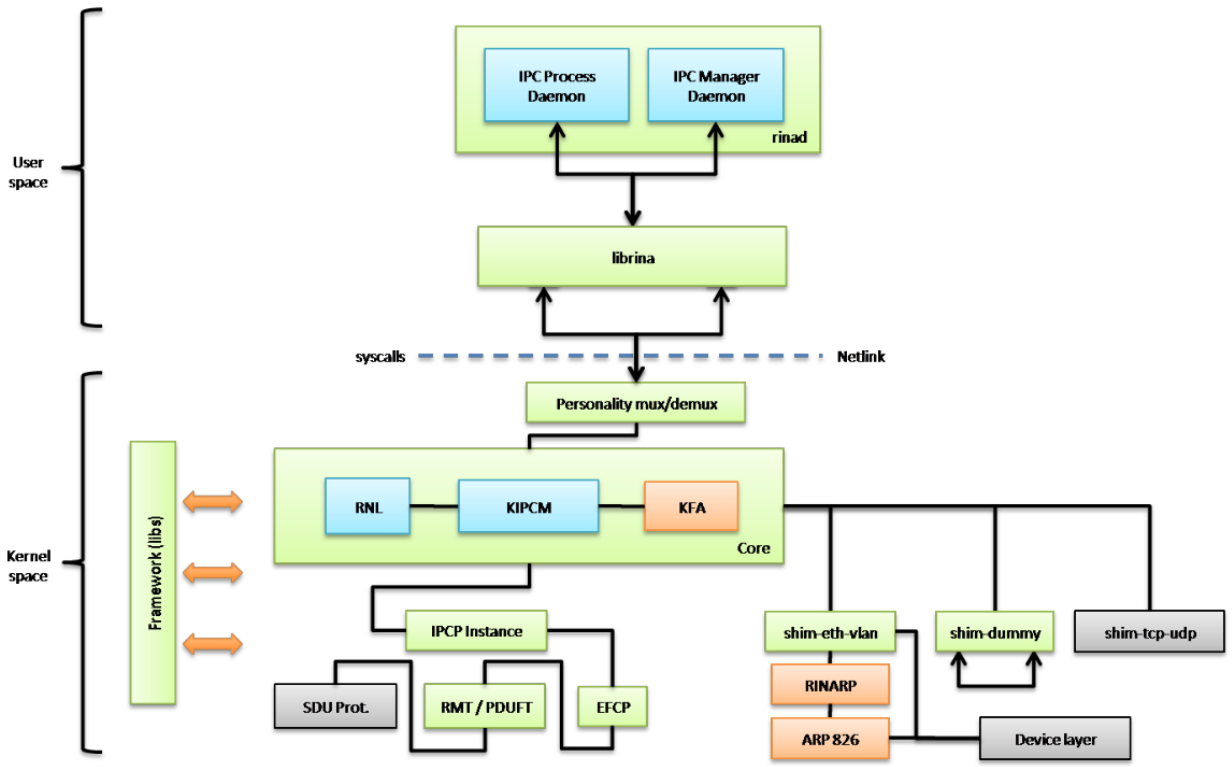


Figura 3.2: Componentes y arquitectura de la implementación de RINA

el ciclo de vida del resto de componentes, directamente en user-space y mediante el Kernel IPC Manager (KIPCM) en el kernel. Es también el punto de contacto para el administrador de la red, el cual puede acceder a él mediante ficheros de configuración, CLI o remotamente mediante un DMS. También se encarga de orquestar el registro de las aplicaciones y las peticiones de flows redirigiéndolas al proceso IPC más adecuado. Por último también implementa el Inter DIF Directory (IDD) que permite encontrar aplicaciones en DIFs a las que no pertenece el sistema.

- **IPC Process** (implementado como un daemon de OS): Cada daemon representa a un proceso IPC en el sistema e implementa la capa de administración de un proceso IPC (Flow Allocator (FA), Resource Information Base (RIB) daemon, Resource Allocator (RA), la tarea de Enrollment, PDU Forwarding Table (PFT) Generator).
- **Librerías RINA:** Las funcionalidades de RINA son ofrecidas a los componentes en user-space por el set de librerías librina, las cuales son compartidas por los componentes del user-space.

En el kernel:

- **Kernel IPC Manager (KIPCM):** Maneja el ciclo de vida y configuración de las instancias de los otros componentes en el kernel. A la vez tiene una función de coordinación entre:
  - Comunicación user-space - kernel.
  - Límite entre dos procesos IPC en el sistema: transmite datos al Error and Flow Control Protocol (EFCP) o los procesos IPC shim en un sentido; o a la Relaying and Multiplexing Task (RMT) o una aplicación (mediante el Kernel Flow Allocator (KFA)). en sentido contrario.

- **Kernel Flow Allocator (KFA):** Es el encargado de todas aquellas tareas relacionadas con la administración de flows en el kernel. Además se encarga de hacer el binding entre los IPC process y el KIPCM.
- **Error and Flow Control Protocol (EFCP):** Contenedor de las diferentes instancias de EFCP que imlplementan el Data Transfer Protocol (DTP) y el Data Transfer Control Protocol (DTCP) para cada conexión en cada IPC process. Pasa los datos entre la RMT en la dirección de salida, o al KFA en la dirección de entrada.
- **Relaying and Multiplexing Task (RMT):** Contenedor de las distintas instancias de RMT en el sistema. Cada RMT (una por proceso IPC) multiplexa los datos desde N conexiones EFCPs a M flows de la capa inferior pasándolos al KFA o al EFCP para la dirección de salida o entrada respectivamente.
- **SDU Protection:** Contenedor de distintos módulos para proteger o desproteger los datos a transmitir por la RMT.
- **Shim IPC Process sobre Ethernet:** Contenedor de los procesos IPC shim sobre Ethernet. Los procesos IPC shim son una versión simplificada del concepto de un proceso IPC normal de RINA, el cual para el caso de Ethernet, adapta una capa 802.1q ofreciendo una API de RINA para la posible interconexión de otros procesos IPC normales. Estos procesos pasan datos al KFA o al medio físico mediante el driver de la tarjeta Ethernet.
- **Shim IPC Process sobre TCP/UDP:** Mismo caso que el anterior pero para una capa TCP/IP adaptando la API de los sockets TCP/IP a RINA.
- **Shim IPC Process dummy:** El IPC Process shim dummy actúa como una interfaz de loopback permitiendo la comunicación de dos IPC process del mismo nivel dentro del mismo sistema.

3.1 Componentes de RINA en user-space

Como ya se ha introducido, los componentes en el user-space son más exigentes en cuanto a la computación y memoria necesaria. Sin embargo, su ejecución es menos frecuente. Estos componentes implementan la capa de administración de los procesos IPC, el IDD, IPC Resource Manager (IRM) y el Management Agent (dentro del IPC Manager) del modelo de referencia de RINA [10]. La estructura del user-space se divide entre procesos (daemons) y librerías. El modelo de intracomunicación (entre componentes del user-space) es Netlink, mientras que para la intercomunicación (con el kernel) es mediante Netlink o system calls.

3.1.1 Librerías RINA

Las librerías de RINA abstraen todas aquellas interacciones con el kernel proveyendo sus funcionalidades al user-space mediante extensiones de lenguajes de scripting o como librerías estática o dinámicamente linkables (como libc). El conjunto de sublibrerías se agrupa sobre el paquete librina, el cual funciona como un framework/middleware con sus propio modelo de memoria y mecánica mediante el uso de threads. Entre otras funcionalidades ofrecidas por librina, se encuentran:

- Capa intermedia entre espacios optimizando el cambio de contexto entre kernel y user-space.
- Implementa los daemons de RINA en user-space y los mecanismos de comunicación entre ellos.

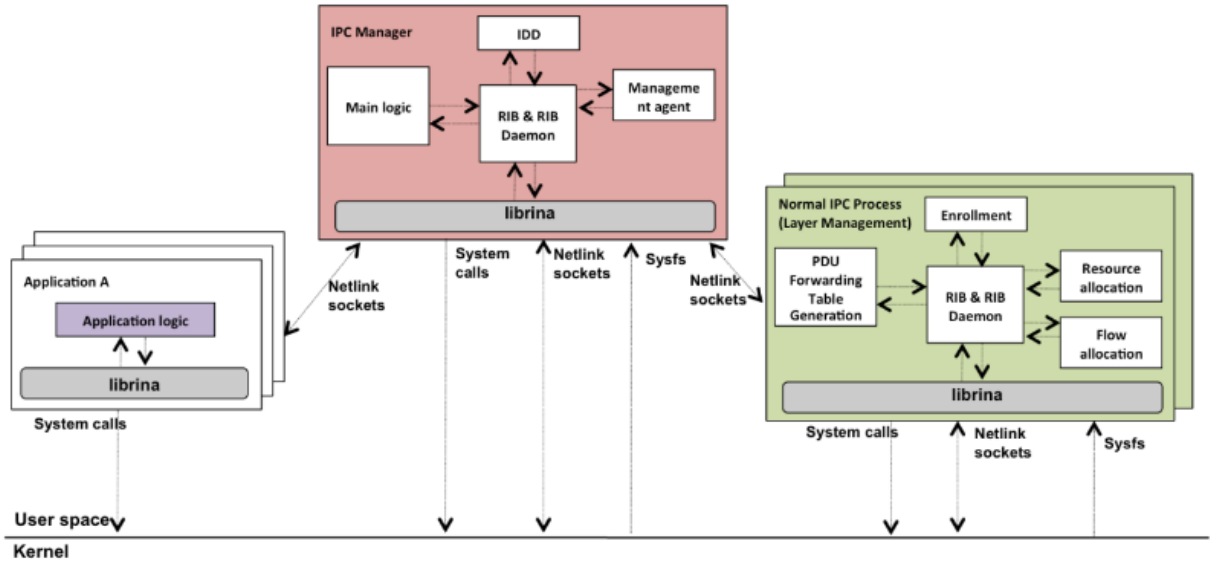


Figura 3.3: Arquitectura de alto nivel de los componentes del user-space

- Funciona como herramienta utilitaria que evita que la aplicación entre directamente al nivel del kernel (por ejemplo ofreciendo la funcionalidad de "malloc").
- Consigue mayor portabilidad de las aplicaciones ya que si éstas realizan llamadas a sistema directamente, quedan limitadas a una arquitectura, ya que las llamadas a sistemas suelen ser dependientes del O.S y la arquitectura.

La arquitectura de librina puede verse en la Fig. 3.4, donde además se aprecian las distintas sublibrerías por las que está compuesta. La comunicación entre componentes es user-space y con el kernel puede ser síncrona (llamadas a sistema) o asíncronas (Netlink), en cuyo caso un modelo basado en eventos activa las acciones a realizar.

3.1.1.1 librina-common

Esta librería provee todas aquellas definiciones y funcionalidades comunes al sistema y que son utilizadas por el resto de librerías. Ejemplos serían las definiciones de tipos propios o estructuras globales.

3.1.1.2 librina-sdu-protection

Las aplicaciones contarán con distintos niveles de confianza dentro de la DIF a la que se conecten. Cada DIF puede tener distintas probabilidades de corromper las Service Data Units (SDUs) transmitidas, o condiciones de seguridad que permitan interceptar el tráfico. Por estas razones, la aplicación puede elegir utilizar herramientas de protección de datos on-demand para adaptarse a las condiciones de la DIF. Algunos casos podrían ser:

- El IDD podría requerir proteger los mensajes Common Distributed Application Protocol (CDAP) que genera para comunicarse con otros IDDs.
- Las aplicaciones podrían requerir proteger las SDUs antes de enviarlas.
- Los procesos IPC podrían querer proteger las SDUs de management (este caso sería realmente el mismo que el anterior ya que el proceso IPC actuaría de la misma manera que una aplicación para la DIF inferior).

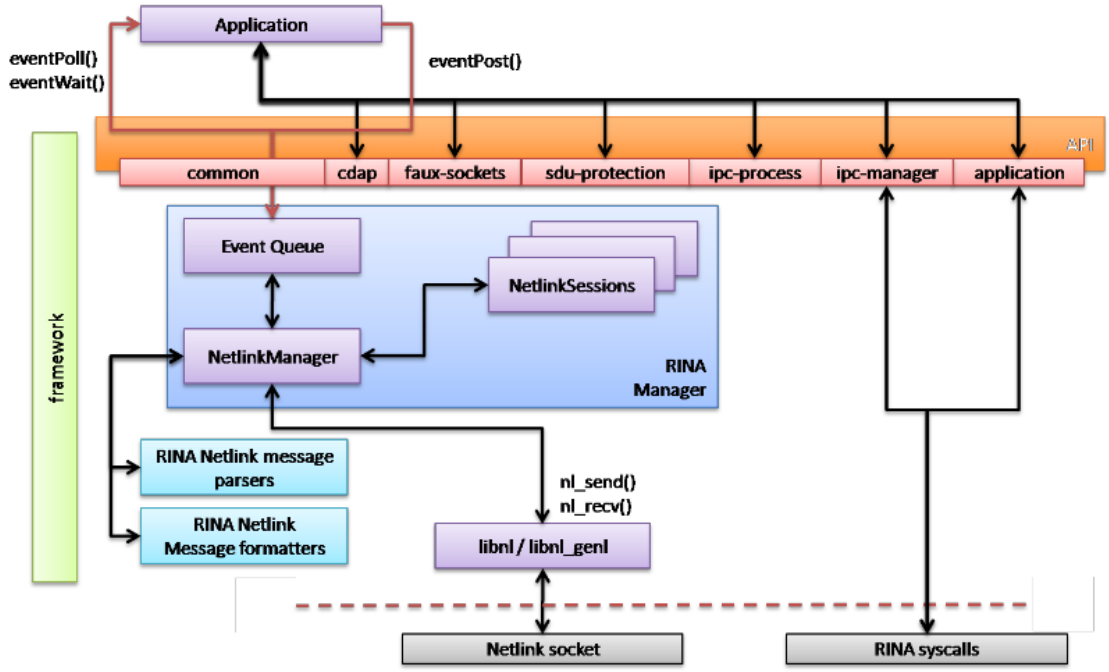


Figura 3.4: Arquitectura de alto nivel librina

La librería incluye mecanismos de detección de errores, encriptación, contadores de saltos, TTL, compresión, etc. Estas funcionalidades deben también estar disponibles en el kernel, quizá de forma más limitada, ya que la RMT también puede necesitar hacer uso de ellas. En ese caso, no sería librina-sdu-protection la encargada de ofrecer las funcionalidades sinó un modulo dedicado en el kernel.

3.1.1.3 librina-cdap

El CDAP [13] es el (único) protocolo de aplicación de RINA. CDAP permite a las aplicaciones comunicarse entre ellas remotamente mediante la realización de operaciones sobre la RIB de los procesos IPC. Antes de que dos aplicaciones puedan intercambiar mensajes CDAP tienen que establecer una conexión de aplicación para i) autenticarse una contra la otra si es requerido y ii) acordar la versión de la sintaxis y la codificación del protocolo de aplicación a utilizar durante la comunicación.

El proceso de conexión de aplicaciones en RINA se define con el Common Application Connection Establishment Protocol (CACEP), el cual provee hooks a plugins con soporte para distintos mecanismos de autenticación. librina-cdap implementa las máquinas de estado de ambos CDAP y CACEP. En esta etapa se considera Google Protocol Buffers [14] como la codificación a utilizar ya que es lo utilizado por otros prototipos existentes y de esta manera mantener la interoperabilidad.

Dadas las especificaciones de RINA, librina-cdap asume que solo una conexión de aplicación puede estar disponible por flow. De esta manera el ID del puerto de dicho flow sirve como identificador de las instancias de las máquinas de estado de CDAP y CACEP para dicha conexión. De esta manera, cuando una aplicación reciba un mensaje CDAP invocaría la librería, la cual procedería de la siguiente manera:

1. Comprobar la ya existencia de una máquina de estados para ese puerto, si no creará

una.
2. Decodificar el mensaje en una estructura utilizable.
3. Comprobar que el mensaje es válido para el estado en que se encuentra la máquina. Si es así, actualizar el estado y devolver el mensaje decodificado. Si no, devolverá un error.

Para enviar un mensaje, la aplicación debe:

1. Crear la estructura para un mensaje CDAP.
2. Codificar el mensaje comprobando que es coherente con el estado del sistema.
3. Enviar el mensaje por el flow ya reservado.
4. Actualizar el estado tras enviar.

3.1.1.4 librina-application

Esta librería ofrece a las aplicaciones una interfaz nativa RINA que les permite i) expresar la disponibilidad para ser accedidas vía una o más DIFs mediante la posibilidad de registrarse; ii) reservar y liberar flows a otras aplicaciones de destino (flow allocation y deallocation); iii) leer y escribir datos desde y hacia flows reservados (en forma de SDUs) y iv) preguntar al sistema por las DIFs disponibles y cuáles son sus características.

Para las operaciones del slow-path, librina-application interactúa con los daemons de RINA en user-space intercambiando mensajes de Netlink. Para el caso del fast-path (leer /escribir SDUs), librina-application utiliza llamadas a sistema para obtener los servicios ofrecidos por el kernel.

La API de la librería está basada en eventos. De esta manera, la API ofrece dos métodos para cada acción que se puede realizar (allocate_flow, register_application, etc), uno bloqueante y el otro no bloqueante. En el primer caso la aplicación y librina terminan la acción cuando envían el mensaje de petición y pueden seguir trabajando a la espera del evento de que la respuesta ha llegado. En el segundo caso, la aplicación se queda bloqueada esperando la respuesta hasta que ésta llegue.

3.1.1.4.1 Workflows

3.1.1.4.1.1 Flow Allocation

Los procesos de aplicaciones pueden jugar dos papeles en el proceso de petición de flow, ser origen o destino:

- En el primer caso, si una aplicación necesita un flow a otra aplicación, invoca la llamada `allocate_flow` de la librería librina-application pasando el nombre del proceso de aplicación de destino y los parámetros (QoS) del flow requerido. La librería envía un mensaje Netlink `allocate_request` al daemon del IPC Manager quien comprobará los permisos de la aplicación y decidirá qué DIF utilizar para llegar al proceso IPC necesario. Cuando la asignación se completa, el IPC Manager responde a la aplicación con un mensaje de Netlink `allocate_request_result`. El mensaje es recibido por la librería quien entonces termina la llamada original con el resultado de la operación.
- En el otro caso, el IPC Manager avisa a la aplicación destino de que se le está pidiendo ser el destino de un flow mediante un mensaje Netlink de `allocate_request_arrived`.

El mensaje es recibido por librina-application y guardado como un evento que podrá ser consultado por la aplicación mediante las llamadas `ev_wait` (bloqueante) o `ev_poll` (no bloqueante). Cuando la aplicación recibe el evento decidirá si acepta o no el flow y se lo comunicará a la aplicación de origen mediante otro mensaje `allocate_flow_response` utilizando la librería. Éste llegará a la aplicación mediante el IPC Manager.

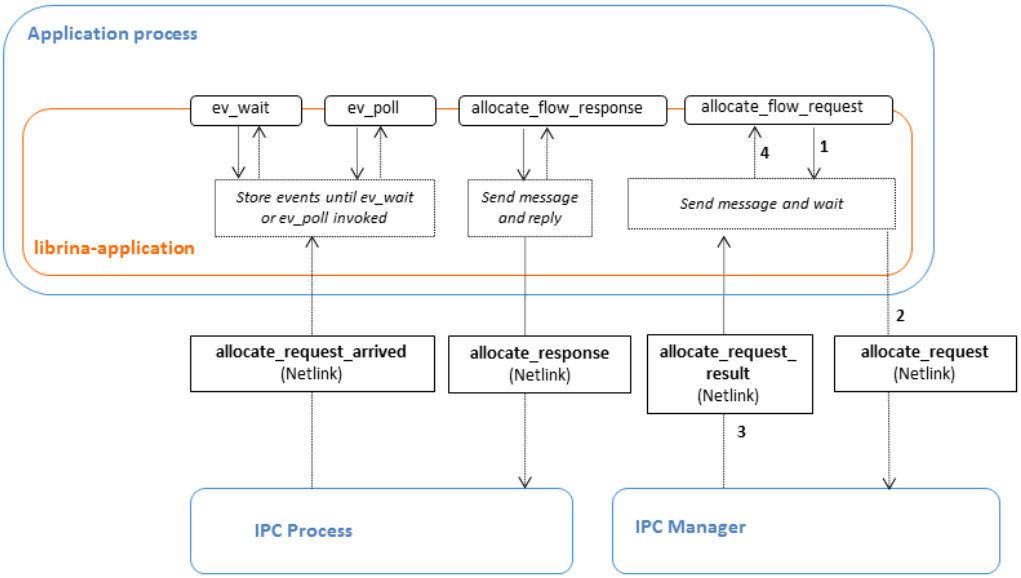


Figura 3.5: Comportamiento de librina-application en la reserva de flows

3.1.1.4.1.2 Flow Deallocation

Como en el caso anterior la aplicación puede jugar dos roles: como el solicitante de la destrucción del flow, o como el receptor pasivo de la notificación de la eliminación del flow.

- Como parte activa, la aplicación invoca `deallocate_flow` de librina-application. Entonces la librería envía un mensaje `deallocate_flow_request` al IPC process daemon de la DIF que soporta el flow y espera. Cuando el proceso termina, el IPC process le envía a la aplicación un mensaje `deallocate_flow_response` que es recibido por la librería y entonces vuelve de la operación de deallocation. Esta operación es local, con lo cual solo atañe al proceso (aplicación) que requiere la acción. El otro extremo del flow debe realizar la misma acción de manera coordinada. Sin embargo hay casos donde el flow debe ser destruido por otras razones como que uno de los IPC proceses cae. En ese caso, es el IPC Manager quien envía un mensaje al otro IPC process afectado para que inicie la operación de deallocate.
- Cuando una aplicación inicia el proceso de deallocation, enviará un mensaje `flow_deallocated` via la librería al proceso IPC destino quien tomará las acciones adecuadas.

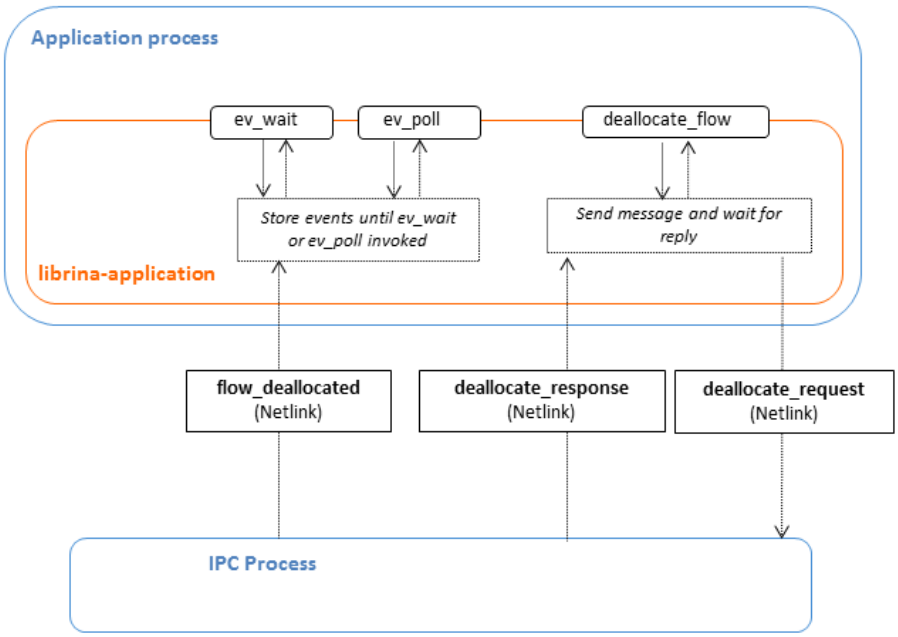


Figura 3.6: Comportamiento de librina-application en la destrucción de flows

3.1.1.4.1.3 Registro y desregistro de Aplicaciones

Los procesos de aplicación invocan la operación `register_application` pasando información del nombre de la aplicación y de la DIF a la que se quieren registrar. librina-application envía un mensaje `register_application` al IPC Manager, quien comprobará que la aplicación tenga los permisos suficientes para registrarse en esa DIF, y contactará al IPC process daemon (y quizás al IDD) para llevar a cabo el proceso de registro. Al finalizar, el IPC Manager enviará un mensaje `register_application_response` al proceso de la aplicación informando sobre el resultado de la operación. La cancelación del registro funciona de la misma manera mandando los mensajes simétricos.

Puede ocurrir que el IPC process de la DIF donde la aplicación está registrada decida cancelar el registro. En ese caso, el IPC process daemon enviará un mensaje `application_registration_canceled`, el cual será recibido por librina-application y guardará un evento para ser consumido por la aplicación con los mecanismos `wait` o `poll` ya mencionados.

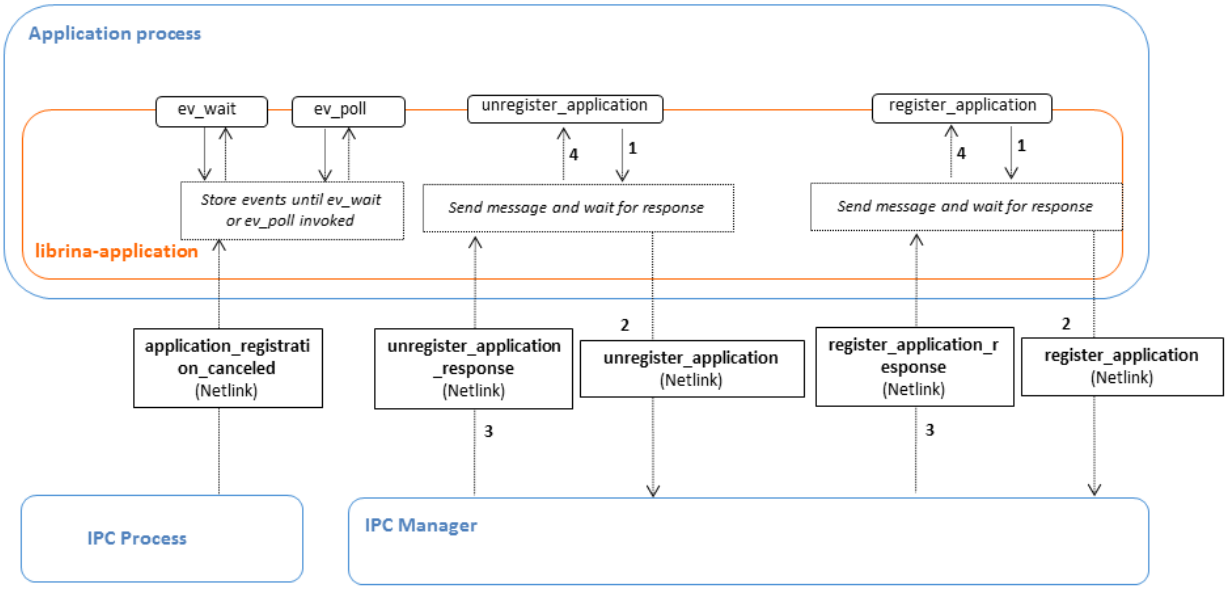


Figura 3.7: Comportamiento de librina-application en el registro/desregistro de aplicaciones

3.1.1.4.1.4 Leer/escribir Service Data Units (SDUs)

Cuando una aplicación quiere escribir SDUs en un flow invoca la primitiva de librina-application `write_sdu` pasando la SDU y el portID (identificador) del flow. La librería invocará la llamada a sistema `write_sdu` con los parámetros necesarios. Para leer, será la librería la que invoque la llamada a sistema `read_sdu` y almacenará las SDUs disponibles como eventos listos para ser consumidos por la aplicación. Cuando la aplicación utilice `ev_wait` o `ev_poll` recibirá las SDUs.

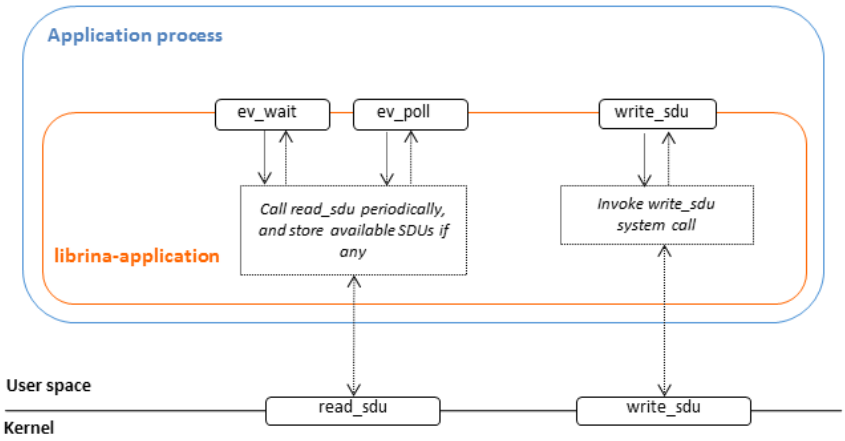


Figura 3.8: Comportamiento de librina-application en la escritura/lectura de SDUs

3.1.1.4.1.5 Obtener las propiedades de una DIF

Cuando una aplicación quiere obtener las propiedades de una DIF determinada, invoca `get_dif_properties` de librina-application. La librería envía el correspondiente mensaje `get_dif_properties_request` al IPC Manager y espera por la respuesta. Cuando recibe `get_dif_properties_response` vuelve de la llamada y entrega el resultado.

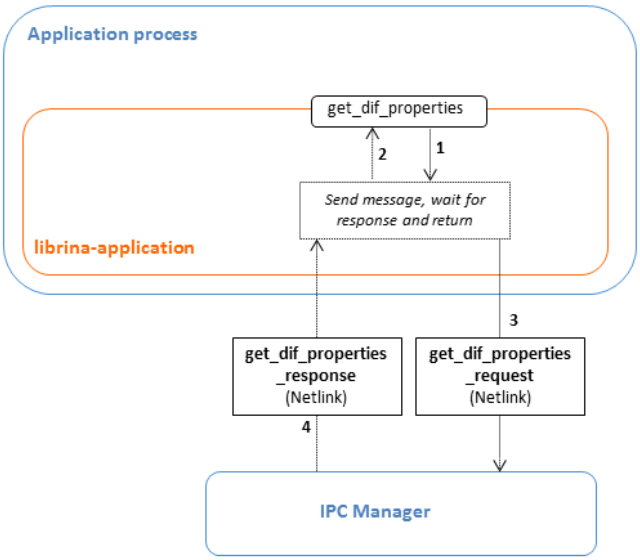


Figura 3.9: Comportamiento de librina-application para obtener la info de una DIF

3.1.1.5 librina-faux-sockets

El objetivo de esta librería es soportar las aplicaciones ya existentes sin necesidad de migrarlas para soportar la API nativa de RINA. librina-faux-sockets envuelve las operaciones de RINA con la API de los sockets TCP/IP de forma que las DIFs pueden soportar aplicaciones heredadas sin que éstas sean conscientes que operan sobre una estructura RINA. Este mismo hecho implica que al no ser conscientes de estar usando RINA, las aplicaciones heredadas no podrán utilizar todos los beneficios de la arquitectura.

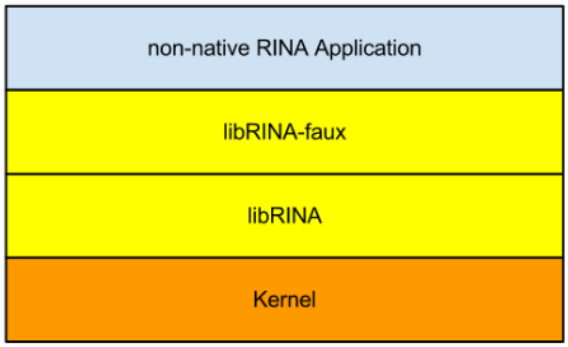


Figura 3.10: Posicionamiento de la librina-faux-sockets

3.1.1.6 librina-ipc-manager

Esta librería provee las funcionalidades referentes a la administración de procesos IPC realizadas por el IPC Manager daemon. Recibe los mensajes enviados por las aplicaciones al IPC Manager y maneja el IDD. También abstrae los mecanismos de comunicación utilizados por el IPC Manager para comunicarse con los procesos IPC (Netlink) o con el kernel (Netlink, system call, procfs/sysfs).

Las operaciones realizadas por el IPC Manager incluyen Create, Retrieve, Update, Delete (CRUD) y asignaciones de DIFs, registros de N-1 DIFs y lanzamiento y enrollment en DIFs. Las peticiones de las aplicaciones incluyen registro y desregistro en DIFs y reserva y destrucción de flows. La administración del IDD incluye la coordinación con otros IPC Managers para mantener la información de los distintos IDDs distribuidos actualizada.

3.1.1.7 librina-ipc-process

Esta librería encapsula las operaciones que permiten a los daemons de los procesos IPC servir las peticiones de las aplicaciones, obtener las configuraciones llevadas a cabo por el IPC Manager y configurar los componentes de los IPC processes en el kernel. Las peticiones de las aplicaciones incluyen reserva y destrucción de flows y registro y desregistro de aplicaciones en el IPC process. Las configuraciones del IPC Manager incluyen las peticiones de asignación de un IPC process a una DIF, el registro de uno o más procesos IPC de nivel N-1 o el lanzamiento de una operación de enrollment. Las acciones de configuración del IPC process en el kernel son CRUD de instancias de EFCP, actualización de la PFT o la modificación de las políticas de protección de SDUs. Las acciones de configuración del kernel se ejecutan mediante llamadas a sistema y procfs/sysfs mientras que para comunicarse con el IPC Manager y otros procesos IPC se utiliza Netlink.

Los eventos generados por librina-ipc-process son procesados por las distintas tareas del IPC process (FA, Enrollment, PFT Generator, RA). Dado que muchas veces estas tareas deberán interactuar con su contraparte en otro IPC process, los procesos IPC utilizan la API del RIB Daemon que permite a los procesos IPC realizar las operaciones de CDAP (create/delete/start/stop /read/write) en uno o más objetos remotos de un proceso IPC vecino. Cuando la operación es invocada, el RIB Daemon utiliza librina-cdap para generar un mensaje CDAP para realizar la acción correspondiente. El RIB daemon envía hacia el IPC process destino el mensaje mediante un flow de administración de la capa N-1.

3.1.2 rinad

rinad es el paquete de software que contiene los distintos daemons del stack que corren en user-space, así como también dos aplicaciones de testing desarrolladas para comprobar la funcionalidad de testing y que se describirán en siguientes secciones.

3.1.2.1 Proceso de Aplicación (Application Process)

Los procesos de aplicación son procesos del OS en user-space que realizan alguna funcionalidad. Si necesitan comunicarse con otros procesos de aplicación para realizar su trabajo, pueden utilizar las librerías provistas por librina para hacer uso de los servicios IPC de RINA presentes en el sistema. Ejemplos de aplicaciones podrían ser navegadores, servidores, email, etc; pero también daemons que implementan alguna funcionalidad de RINA en user-space como el IPC Manager.

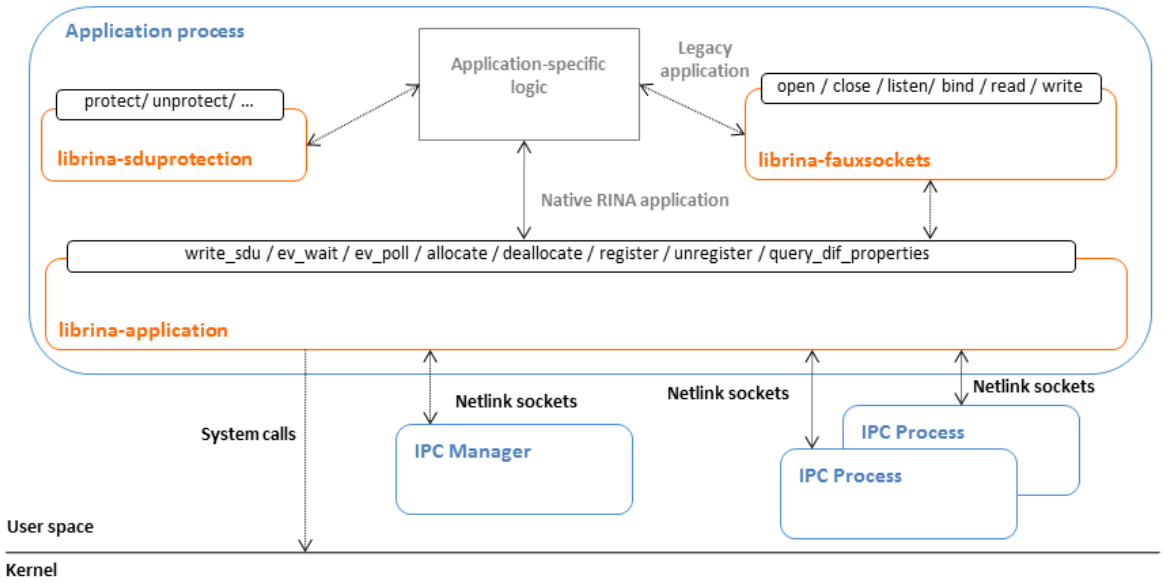


Figura 3.11: Estructura interna del Application Process y su relación con el resto de componentes de RINA

3.1.2.2 IPC Manager (Daemon)

El IPC Manager es el coordinador y administrador de los diferentes componentes de RINA, tanto en user-space como en el kernel. Su funcionalidad es accesible de distintas maneras: i) localmente a través de una CLI y/o archivo de configuración; o bien ii) remotamente a través de un DMS que utiliza CDAP para interactuar con el IPC Manager (a través del Management agent en el IPC Manager). El acceso remoto al DMS se efectúa de forma segura garantizando acceso sólo a los administradores del sistema. El IPC Manager implementa el IIRM, el Management Agent y el IDD descritos en el modelo de referencia de RINA[10].

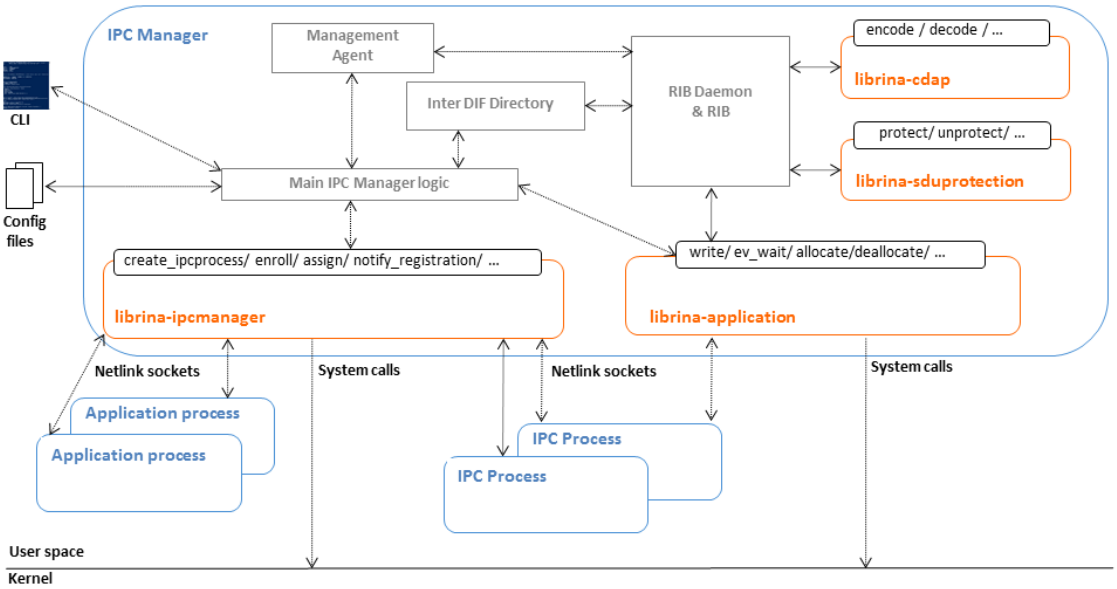


Figura 3.12: Estructura interna del IPC Manger Daemon y su relación con el resto de componentes de RINA

El IPC Manager utiliza librina-application como cualquier otro proceso de aplicación. De esta manera establece flows al DMS o a otro IPC Manager para realizar la tarea del IDD. Las funcionalidades del IPC Manager están delegadas en librina-ipc-manager. El DMS interactúa con el IPC Manager intercambiando mensajes CDAP que operan sobre los objetos del RIB del IPC Manager, el cual es una estructura lógica que organiza toda la información conocida por una aplicación (como el IPC Manager). Cuando una aplicación recibe un mensaje CDAP que actúa sobre uno o más objetos de la RIB, el RIB daemon se encarga de realizar dicha acción.

La librería librina-ipc-manager encapsula todas las operaciones de administración locales disponibles para el IPC Manager. Estas operaciones se pueden dividir en:

- Administración de procesos IPC locales
  - Creación/destrucción de procesos IPC.
  - Asignación de proceso IPC a DIFs dando información suficiente al IPC process para que pueda empezar a operar como miembro de la DIF.
  - Registro de procesos IPC a DIFs N-1 permitiendo a un IPC process de nivel N ser accesible vía una o más DIFs de N-1.
  - Forzar un proceso IPC a hacer enrollment con un proceso IPC remoto. El proceso IPC local puede ser o no ser parte de la DIF.
  - Inspeccionar el RIB de un proceso IPC.
- Servicios a procesos de aplicaciones
  - Redireccionamiento de peticiones de flows al daemon del proceso IPC más adecuado. En esta situación, o bien el IPC Manager cuenta con información suficiente para realizar esta acción, o bien debe consultar al IDD para identificar la DIF correcta para usar.
  - Redireccionamiento de peticiones de registro de las aplicaciones al proceso IPC que es miembro de la DIF objetivo.
  - Listado de DIFs en el sistema disponible para una cierta aplicación junto con sus características.

3.1.2.2.1 Comportamiento detalldo del IPC Manager

3.1.2.2.1.1 Iniciación

Cuando el IPC Manager daemon arranca lo primero que hace es un binding con un socket Netlink y empieza a escuchar en él. Seguidamente recupera la configuración del sistema desde alguna forma de almacenamiento permanente (fichero de configuración en este caso) y realiza las tareas necesarias para llevar al sistema al estado indicado por la configuración. Estas operaciones pueden suponer creación de procesos IPC, asignación a DIFs, registro de estos procesos en N-1 DIFs o forzarlos a hacer enrollment un proceso IPC remoto. La configuración debe incluir:

- DIFs conocidas por el sistema. Para cada DIF:
  - Nombre
  - Tipo (por ejemplo shim Ethernet, shim TCP/UDP, normal)
  - Cubos de QoS soportados (nombres y rangos de los atributos)
  - Tamaño de SDU máximo
  - La lista de políticas que los miembros de la DIF tienen que implementar.

- Procesos IPC a crear. Para cada proceso:
  - Nombre
  - Tipo (por ejemplo shim Ethernet, shim TCP/UDP, normal)
  - (Opcional) El nombre de la DIF a la que se lo debe asignar
  - (Opcional) Nombres de las N-1 DIF a las que registrarse
  - (Opcional) Nombre de los procesos IPC a los cuales hacer enrollment.
  - En caso de procesos IPC shim, información específica (por ejemplo VLAN id, nombre de la interfaz Ethernet, etc)
- (Opcional) Nombre del proceso DMS a contactar y N-1 DIF a utilizar
- Nombres de IPC Manager vecinos a los cuales contactar

3.1.2.2.1.2 Creación de IPC process

Cuando el IPC Manager daemon necesita crear un nuevo proceso IPC utiliza la operación `ipcm_create` de librina-ipc-manager. Esta operación supone la invocación de dos funciones: i)la syscall `system()` para instanciar un nuevo proceso en user-space; y ii) `ipc_process_create`, una syscall que será atendida en el kernel por el KIPCM y creará las estructuras necesarias en el kernel para el nuevo proceso IPC.

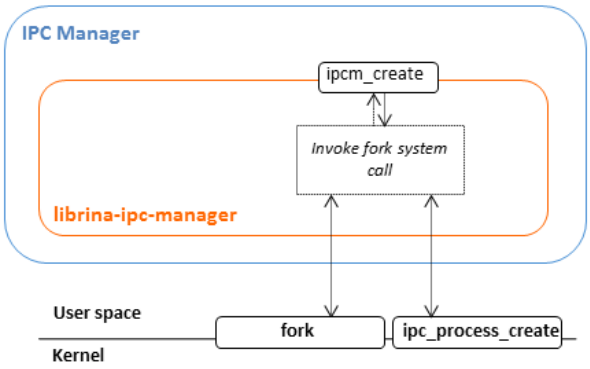


Figura 3.13: Comportamiento del IPC Manager daemon al crear un proceso IPC

3.1.2.2.1.3 Destrucción de IPC process Para destruir un proceso IPC, el IPC Manager daemon invoca `ipcm_destroy` de librina-ipc-manager. Esta operación supone la invocación de varias funciones: la syscall `kill` para enviar la correspondiente señal al proceso en user-space, y la syscall `ipc_process_destroy`, que será atendida en el kernel por el KIPCM quien destruirá las estructuras en el kernel del proceso IPC.

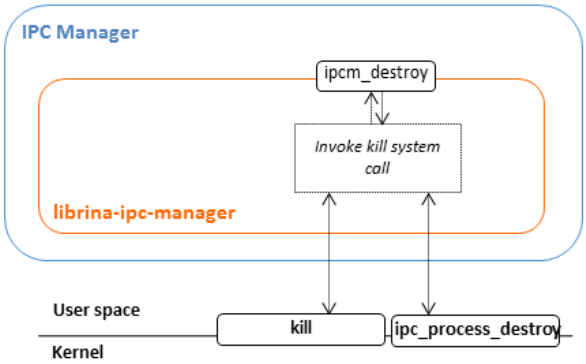


Figura 3.14: Comportamiento del IPC Manager daemon al destruir un proceso IPC

3.1.2.2.1.4 Enrollment de un IPC process a una DIF

Este proceso hace que un IPC process anteriormente creado se enliste como miembro de una DIF, aportando al nuevo miembro toda la información necesaria para operar en dicha DIF. El IPC Manager invoca `ipcm_assign` de la librería pasando el ID del IPC process y la información de la DIF. Esto genera un mensaje `assign_request` hacia el IPC process daemon. El IPC process daemon rellena su RIB con la información y devuelve un mensaje `assign_response`.

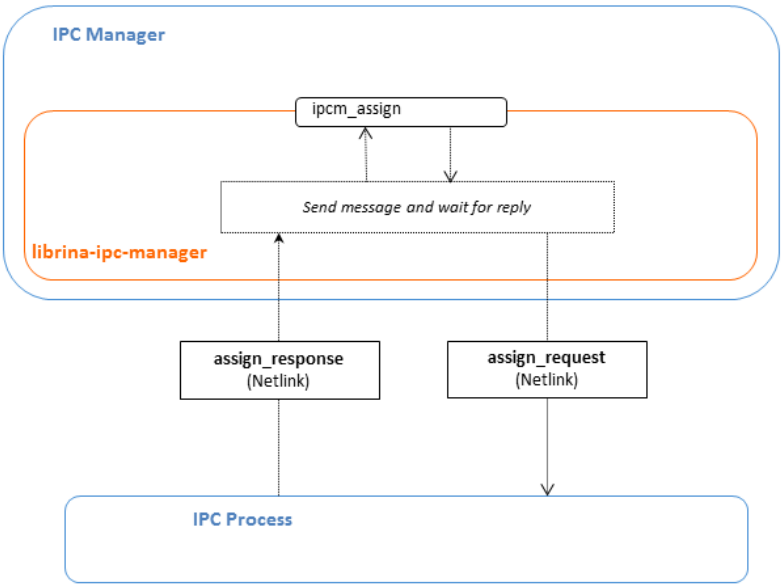


Figura 3.15: Comportamiento del IPC Manager daemon al asignar un IPC process a una DIF

3.1.2.2.1.5 Registro de un IPC process a una N-1 DIF

Después de crear un proceso IPC y registrarlo en una DIF, el siguiente paso es registrarlo en una N-1 DIF para que éste pueda ser accedido (igual que los procesos de aplicaciones). La acción la realilza el IPC Manager en nombre del IPC process a registrarse, hablando con aquellos IPC processes de las N-1 DIF a las que el IPC process necesita registrarse. Lo hace mediante la llamada `ipcm_register_app` que desencadena una serie de mensajes que se pueden ver en Fig. 3.16.La operación de desregistro es simétrica a la anterior y puede verse en Fig. 3.17.

3.1.2.2.1.6 Enrollment (alistamiento) de un IPC process en una DIF

El IPC Manager puede instruir a un IPC process a iniciar el enrollment con un IPC process remoto en otro sistema. Esta operación es lanzada por el IPC Manager mediante `ipcm_enroll` aportando el ID del IPC process, la DIF a la cual alistarse y la N-1 DIF a través de la cual se irá. Fig. 3.18 muestra este proceso.

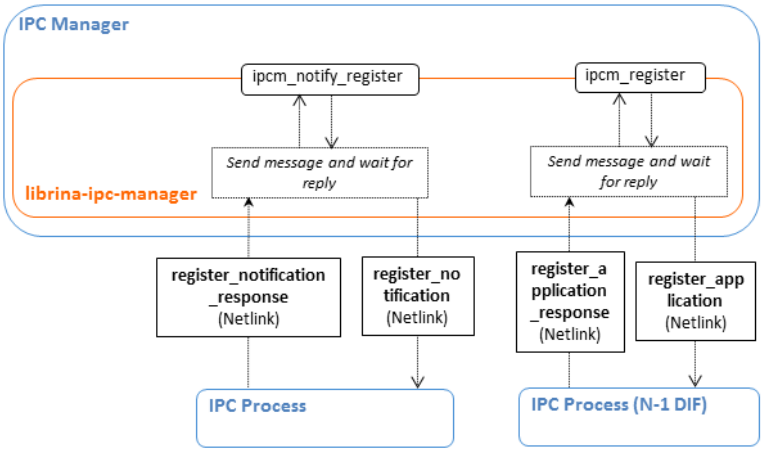


Figura 3.16: Comportamiento del IPC Manager daemon al registrar un IPC process en una N-1 DIF

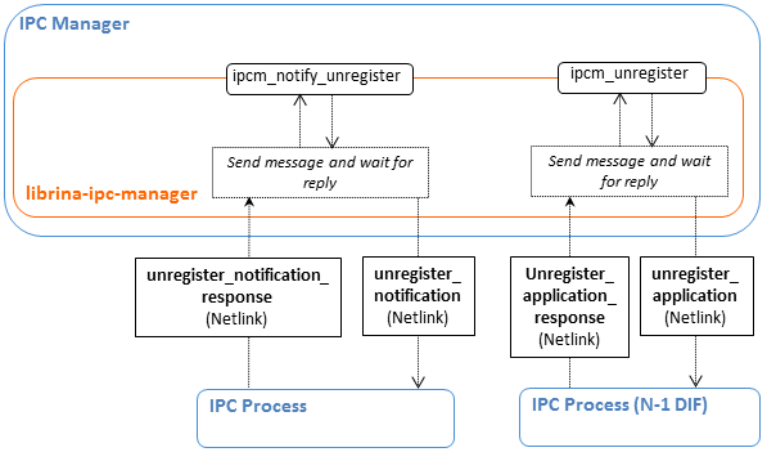


Figura 3.17: Comportamiento del IPC Manager daemon al desregistrar un IPC process de una N-1 DIF

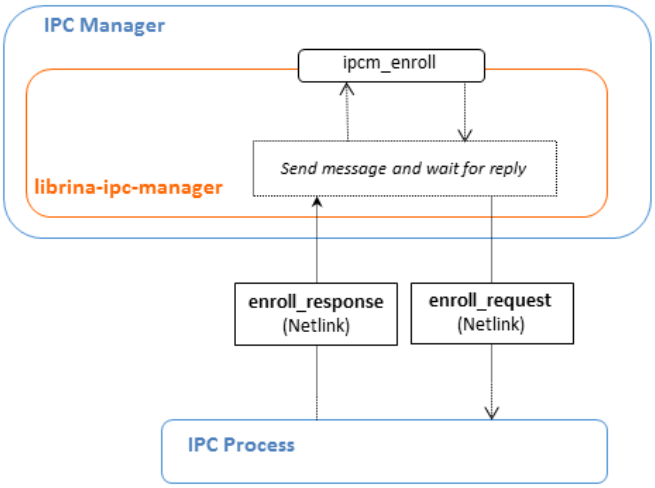


Figura 3.18: Comportamiento del IPC Manager daemon durante el enrollment de un IPC process

3.1.2.2.1.7 Desconexión de un IPC process de un vecino

Esta operación provoca que un IPC process daemon libere todos los N-1 flows que comunican con un IPC process vecino. El IPC Manager llama a `ipcm_disconnect` de librina-ipc-manager la cual genera el correspondiente mensaje Netlink y espera a la respuesta del IPC process. Finalmente la librería enviará el resultado de la operación al IPC Manager.

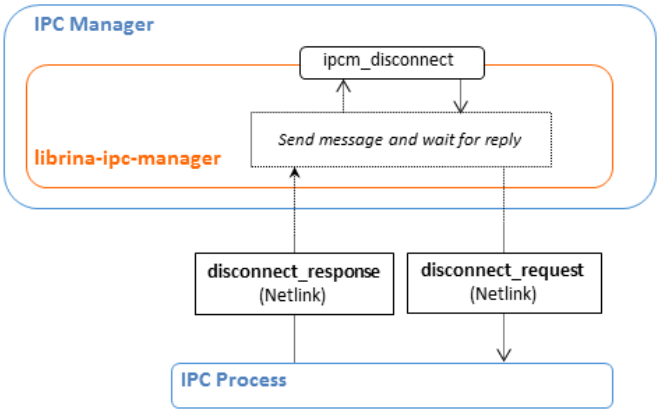


Figura 3.19: Comportamiento del IPC Manager daemon durante la desconexión de un IPC process de un vecino

3.1.2.2.1.8 Inspección de la RIB de un IPC process

Cuando el IPC Manager necesita obtener información de la RIB de un IPC process utiliza `ipcm_query_rib` de librina-ipc-manager, detallando el nombre y opcionalmente el abasto y un filtro para realizar la búsqueda. La operación bloquea hasta que la respuesta se produce con 0 o más resultados o un error ocurre. Los mensajes Netlink implicados pueden verse en Fig. 3.20

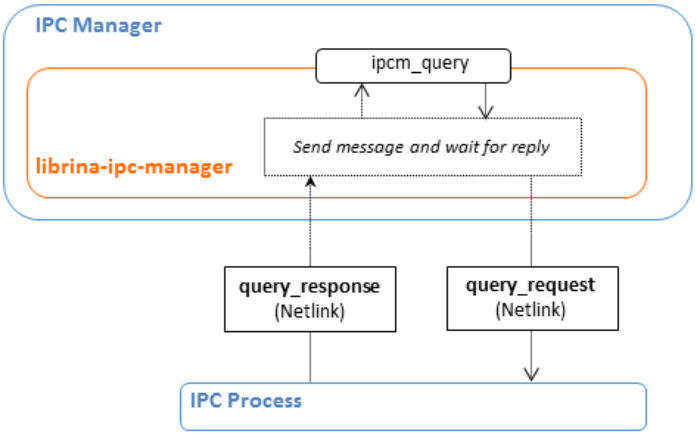


Figura 3.20: Comportamiento del IPC Manager daemon cuando pide información de la RIB de un IPC process

3.1.2.2.1.9 Registro/desregistro de aplicaciones

La librería librina-ipc-manager guarda la petición de registro hasta que el IPC Manager consume el evento. Entonces, el IPC Manager comprobará que la aplicación tiene suficientes permisos para registrarse en la DIF, en ese caso, invoca `register_app` que genera un

mensaje `register_app` y lo envía al IPC process representante de la DIF. Éste responde con un mensaje de respuesta con el resultado de la operación. Entonces el IPC Manager actualiza el IDD con la nueva aplicación registrada e invoca `ipcm_app_registered` para enviar el mensaje de resultado a la aplicación que inició el proceso.

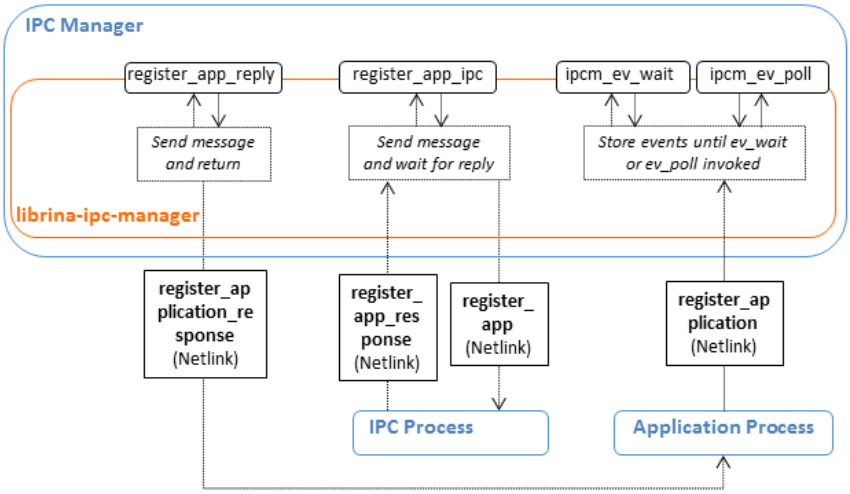


Figura 3.21: Comportamiento del IPC Manager daemon durante el registro de una aplicación en una DIF

El proceso de desregistro es igual pero utilizando las llamadas y mensajes contrarios. La única diferencia es que la llamada `ipcm_unregister_app` bloquea hasta obtener el resultado.

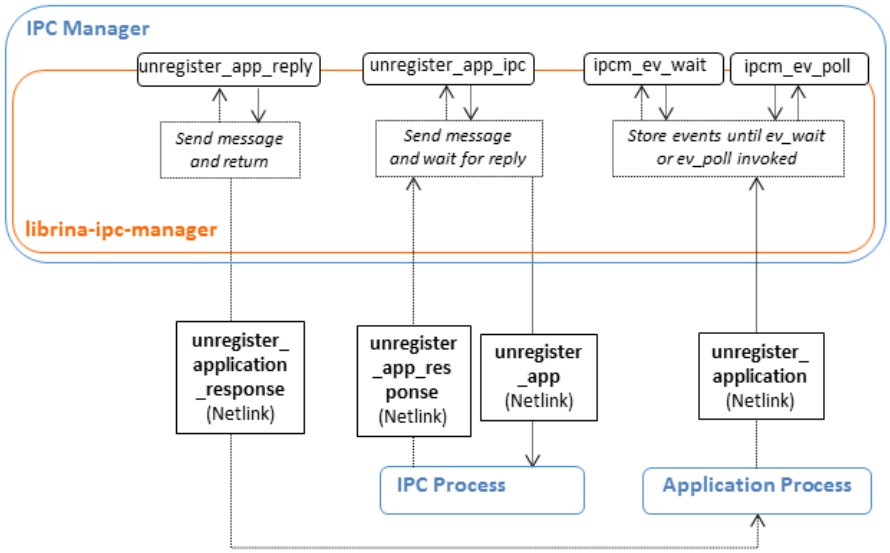


Figura 3.22: Comportamiento del IPC Manager daemon durante el desregistro de una aplicación en una DIF

3.1.2.2.1.10 Reserva de flow

La librería librina-ipc-manager guarda la request de flow y espera a que el IPC Manager consuma el evento. En ese momento el IPC Manager decide qué DIF debe ser usada para instanciar el flow, posiblemente utilizando el IDD. Una vez identificada, invoca

`ipcm_allocate_flow`, lo que produce un mensaje `allocate_flow_request` hacia el proceso IPC de la DIF elegida. Cuando recibe el mensaje de respuesta, generá otro mensaje de respuesta al proceso de aplicación que pidió el flow.

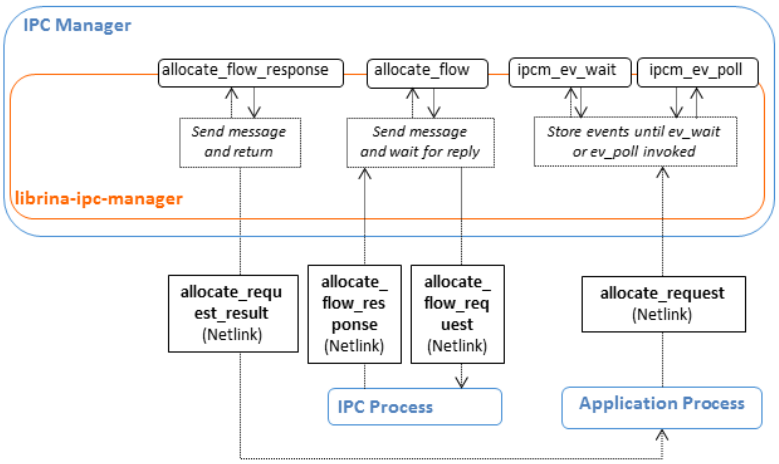


Figura 3.23: Comportamiento del IPC Manager daemon durante la reserva de un flow

3.1.2.2.1.11 Inspección de las propiedades de una DIF

El mensaje `get_dif_properties` es recibido por la librería librina-ipc-manager que guarda el evento hasta que lo consuma el IPC Manager. En ese momento recupera la información de la DIF pedida e invoca a `ipcm_dif_properties` en librina-ipc-manager para enviar la info al IPC process mendiante un mensaje de Netlink.

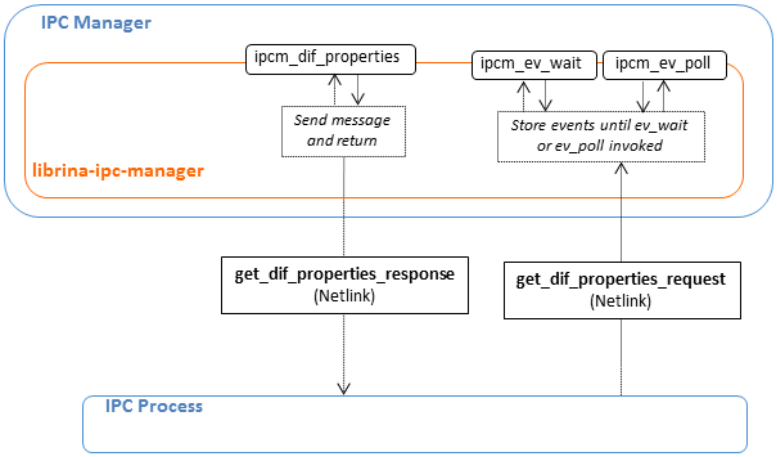


Figura 3.24: Comportamiento del IPC Manager daemon durante la petición de información sobre una DIF

3.1.2.3 IPC Process (Daemon)

El IPC process daemon imlpementa la capa de administración de un proceso IPC: el RIB, Enrollment, FA, PFT Generator y RA del modelo de referencia de RINA [10]. Como ya se explicó anteriormente, los componentes del IPC process restantes residen en el kernel que se explicará más adelante.

Los flows utilizados por el IPC process pueden ser de dos tipos: i) de la capa de administración utilizados entre procesos IPC "peer" intercambiando mensajes CDAP o ii) de transferencia de datos creados por el RMT y utilizados para multiplexar/desmultiplexar las Protocol Data Units (PDUs) de los N flows entre los N-1 flows.

La estructura y las librerías utilizadas por el IPC process daemon pueden verse en Fig. 3.25.

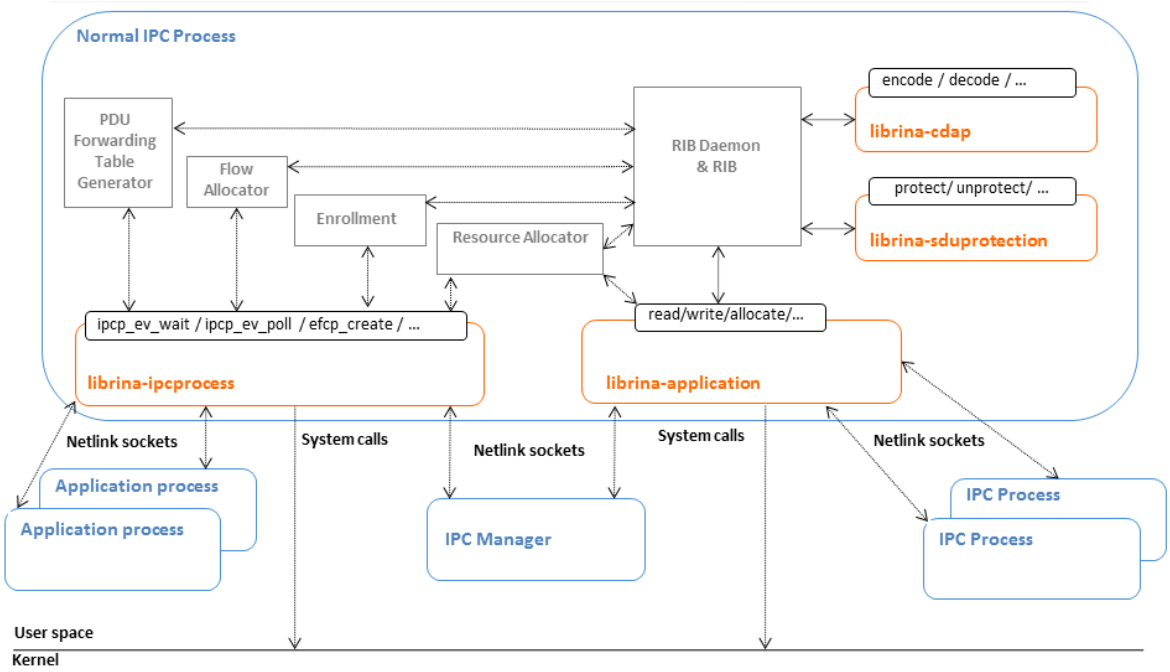


Figura 3.25: Estructura interna del IPC Process Daemon y su relación con el resto de componentes de RINA

3.1.2.3.1 La Resource Information Base

Como ya se explicó, la RIB es la estructura lógica que almacena los objetos que representan el estado distribuido de la aplicación. Para el IPC process esta información incluye el mapeo de direcciones, aplicaciones disponibles, credenciales de seguridad, etc. La RIB del IPC process está organizada en forma de árbol y acepta las 6 acciones provistas por CDAP: create, delete, read, write, start y stop.

3.1.2.3.2 El RIB daemon

El RIB daemon es la entidad que lee los N-1 flows desprotegiendo las SDUs entrantes. Utiliza librina-cdap para decodificar el mensaje CDAP, extrae la información de qué acción, sobre qué objeto de la RIB aplica y qué información debe ser recuperada. Después, delega la acción al componente adecuado para llevar a cabo la tarea (FA, Enrollment, etc).

3.1.2.3.3 La tarea de enrollment

Todas las comunicaciones experimentan tres fases: enrollment, reserva (establecimiento) y transferencia. RINA también. Enrollment es el proceso vía el cual un proceso IPC se conecta a una DIF y recibe información suficiente para empezar a operar como miembro. Comienza cuando el proceso IPC establece una conexión de aplicación con otro proceso IPC que ya es parte de la DIF. Durante la fase de establecimiento de la conexión, puede pasar que el proceso

IPC que se conecta sea requerido a autenticarse si las políticas de seguridad de la DIF así lo establecen. Una vez establecida la conexión se le pasa al proceso IPC la información de la DIF que necesita (asignación de dirección, políticas, etc).

3.1.2.3.4 PDU Forwarding Table Generator (PFTG)

El PDU FTG (o routing) es el componente del IPC process que intercambia información sobre la conectividad con otros procesos IPC de la DIF y aplica algoritmos para generar la tabla de forwarding utilizada por la RMT. Los algoritmos y la información requerida para generar dicha tabla puede ser múltiple, dependiendo de las clases de QoS soportadas por la DIF.

3.1.2.3.5 El Flow Allocator (FA)

El FA es el componente responsable de administrar los flows y su ciclo de vida: reserva, monitoreo, y destrucción. Las tareas del FA son:

- encontrar el proceso IPC vía el cual la aplicación de destino es accesible.
- mapear las políticas de QoS requeridas con el correspondiente flow.
- negociar la reserva del flow con el FA del IPC process de destino (control de acceso, políticas, etc).
- crear una o más instancias de DTP y opcionalmente DTCP para dar soporte al flow.
- Monitorizar las instancias de DTP/DTCP para asegurar el QoS durante la transferencia y lanzar cualquier cambio requerido para esto.
- Liberar los recursos reservados para el flow cuando éste es terminado o la aplicación termina inesperadamente.

A diferencia de TCP, en RINA, los flows y las conexiones son independientes. El FA reserva puertos y crea flows. Los port-IDs son los extremos del flow, únicos en el sistema. Un flow consiste en: i)el binding local entre el port-ID de origen y destino y el connection-endpoint-ID (CEP-ID) de origen y destino respectivamente; y ii) la conexión o el potencial de conexión entre instancias de EFCP (DTP/DTCP).

3.1.2.3.6 El Resource Allocator (RA)

Es el componente que decide cómo se reservan los recursos en el IPC Process (dimensionado de las colas, creación/suspensión/destrucción de colas, creación/destrucción de N-1 flows, etc). También actualiza la RIB dependiendo de las acciones realizadas. Se comunica con el KIPCM con tal de realizar las acciones que implican uso de los componentes en el kernel.

3.1.2.3.7 Comportamiento detallado

3.1.2.3.7.1 Inicialización

Cuando el IPC process daemon se inicia lo primero que hace es el binding a un socket de Netlink donde comienza a escuchar. Hasta que el IPC process no es asignado a una DIF o ordenado a hacer enrollment con una DIF por el IPC Manager no puede procesar ninguna reserva de flow o registro de aplicaciones ya que no tiene comunicación con ningún otro IPC process.

3.1.2.3.7.2 Asignación a DIF

El IPC Manager envía un mensaje de Netlink `assign_request` al IPC Process daemon. librina-ipc-process almacena el evento que es en algún momento consumido por el IPC process daemon. En ese momento el IPC process se actualiza con la información de la DIF e invoca `ipcp_assign_response` enviando un mensaje `assign_response` al IPC Manager.

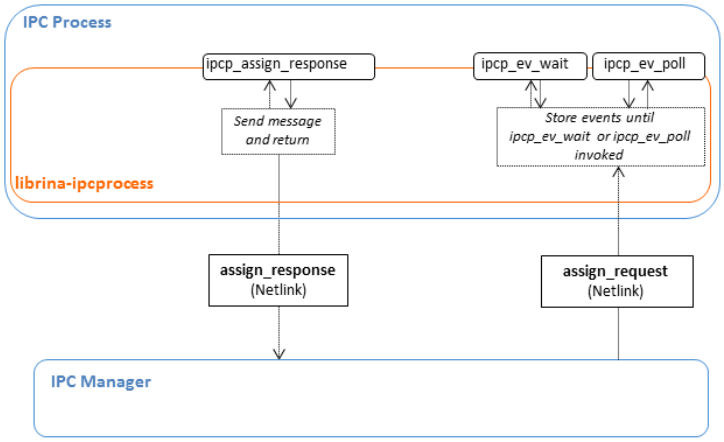


Figura 3.26: Comportamiento del IPC process daemon cuando es asignado a una DIF

3.1.2.3.7.3 Registro/desregistro de aplicaciones

La petición de la aplicación es redirigida por el IPC Manager al IPC process apropiado. El mensaje es consumido como un evento siguiendo el funcionamiento de librina. Las respuestas enviadas por el IPC process son las mismas explicadas en el IPC Manager para esta tarea y la figura descriptiva puede verse en Fig. 3.27.

Cuando el IPC process decide cancelar el registro de una aplicación por alguna razón, envía dos mensajes `application_registration_canceled` al proceso de aplicación y `ipcp_register_cancel` al IPC Manager utilizando las APIs de librina-ipc-process y librina-application.

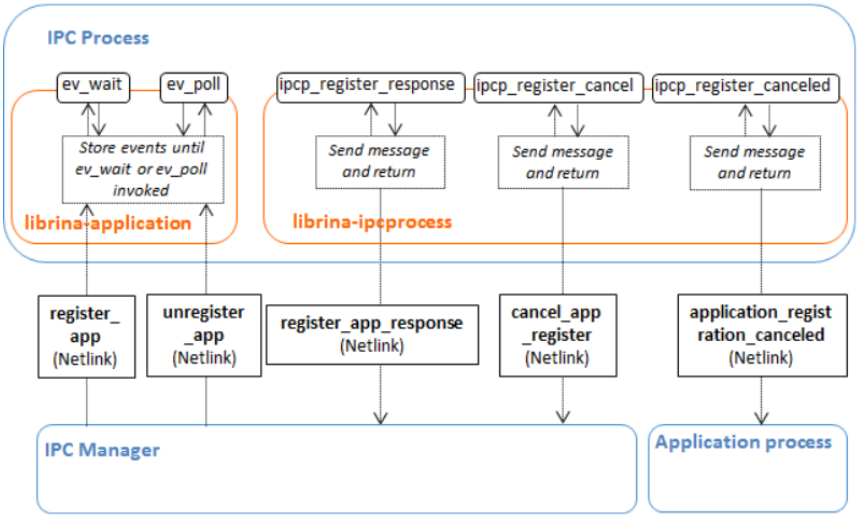


Figura 3.27: Comportamiento del IPC process daemon en el registro/desregistro de una aplicación

3.1.2.3.7.4 Reserva de flow

Tras recibir el correspondiente mensaje del IPC Manager, el IPC process redirige la petición al FA. Éste primero comprueba si la petición puede ser atendida dependiendo de sus especificaciones. Si es posible, invoca `efcp_create` para crear las estructuras necesarias en el kernel. Seguidamente llama al RIB daemon para comunicarse mediante CDAP con el FA del IPC process desde el cual la aplicación destino puede ser alcanzada.

Cuando la aplicación de destino acepta el flow, el RIB daemon comunica al FA los parámetros asignados al flow, en especial el CEP-id de destino. En ese momento el FA invoca `efcp_update` que a su vez invoca la syscall del mismo nombre para pasar los parámetros recibidos a las estructuras del kernel (donde reside la instancia de EFCP creada). Una vez finalizado el IPC process envía el mensaje `ipcp_allocate_response` al IPC Manager con el resultado.

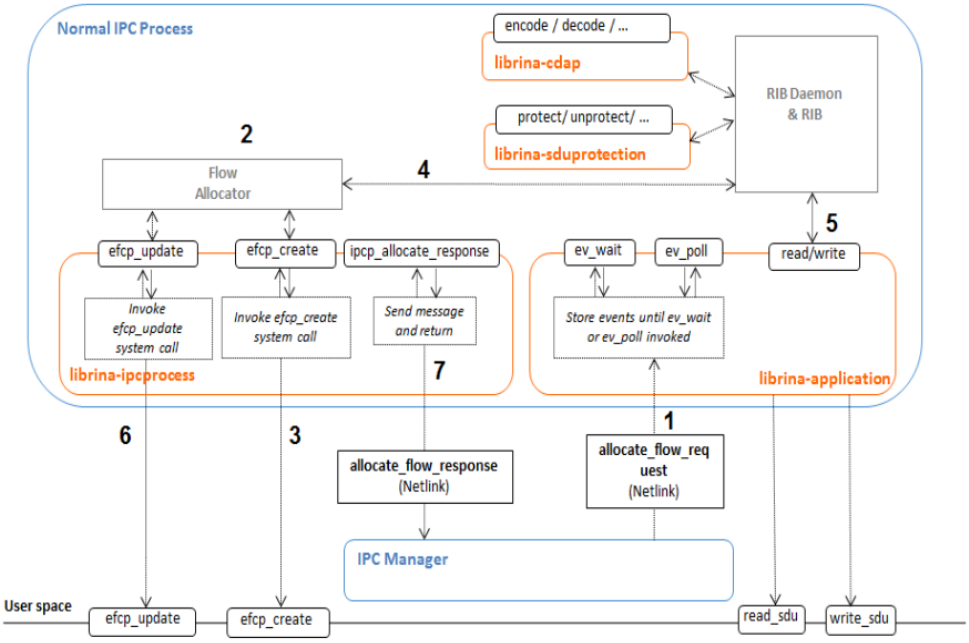


Figura 3.28: Comportamiento del IPC process daemon origen en la reserva de un flow

Las acciones realizadas en el IPC process de destino son: el RIB daemon recibe el mensaje CDAP del IPC process de origen a través de la operación `read`. Le envía el mensaje al FA el cual envía un mensaje `allocate_request_arrived` a la aplicación destino. Ésta reponde con un `allocate_response`. El IPC process daemon recibe el mensaje, lo reenvía al FA y si es positivo, el FA invoca `efcp_create` que deriva en una syscall para crear las estructuras del EFCP en el kernel. Finalmente llama al RIB daemon para que comunique el resultado al IPC process de origen.

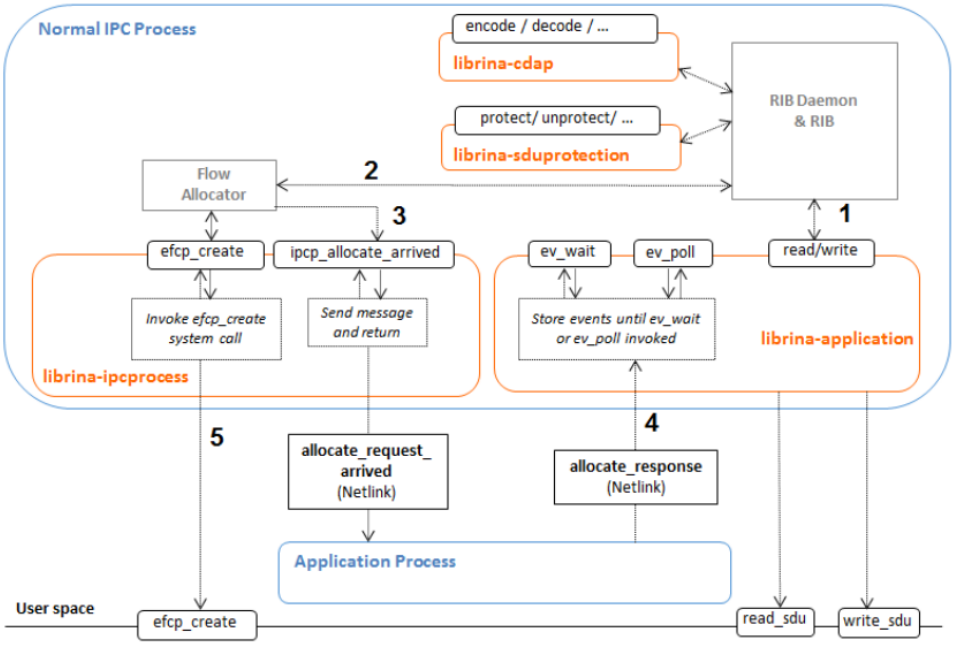


Figura 3.29: Comportamiento del IPC process daemon destino en la destrucción de un flow

3.1.2.3.7.5 Destrucción del flow

La aplicación envía el mensaje `deallocate_request` al IPC process, quien llama al FA. Éste inoca `efcp_destroy` para destruir las estructuras del kernel. La información del flow es removida de la RIB. El IPC process envía el resultado a la aplicación.

El IPC process remoto, destino del flow, cuando recibe el evento producido en el origen, realizará las mismas acciones con las llamadas y mensajes correspondientes que pueden verse en Fig. 3.30.

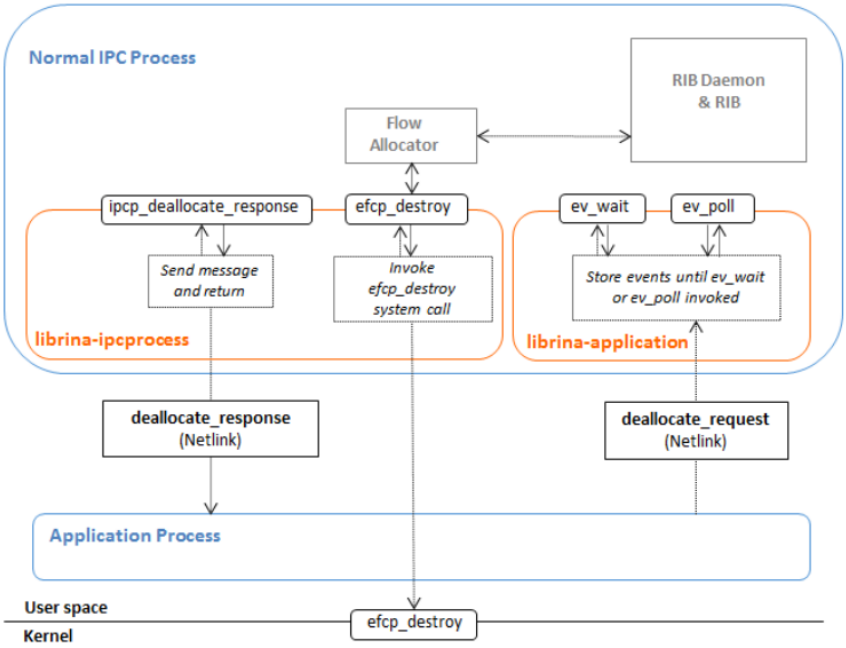


Figura 3.30: Comportamiento del IPC process daemon durante la cancelación de un flow

3.1.2.3.7.6 Notificación de registro/desregistro de aplicación

El IPC Process recibe un mensaje `register_notification` del IPC Manager informando que ha sido registrado en una N-1 DIF. El IPC process responde con un `register_notification_response`. El procedimiento simétrico es seguido en el desregistro.

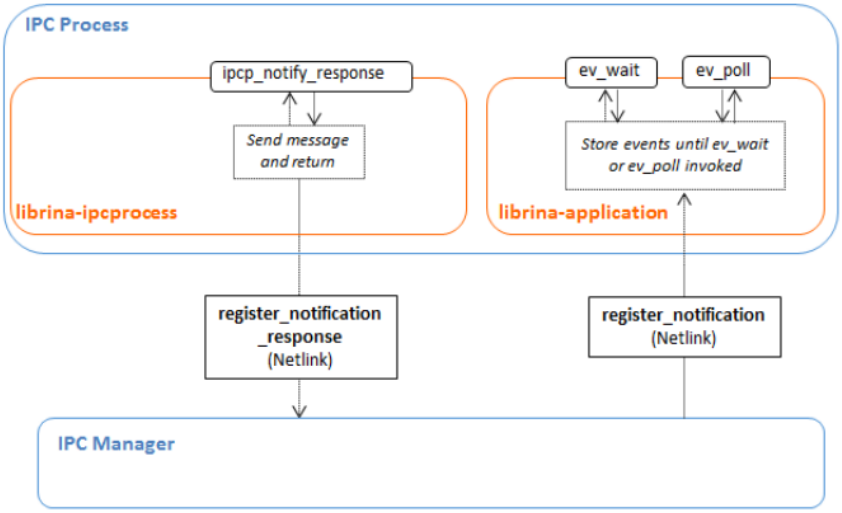


Figura 3.31: Comportamiento del IPC process daemon cuando es notificado del registro en una N-1 DIF

3.1.2.3.7.7 Enrollment

Cuando el IPC Manager quiere que un IPC process inicie el enrollment con otro IPC process le envia un mensaje `enroll_request`. Al recibir el evento, el IPC Process iniciará la petición de un N-1 flow de management al IPC process de destino. Finalmente enviará un mensaje de `enroll_response` al IPC Manager con el resultado.

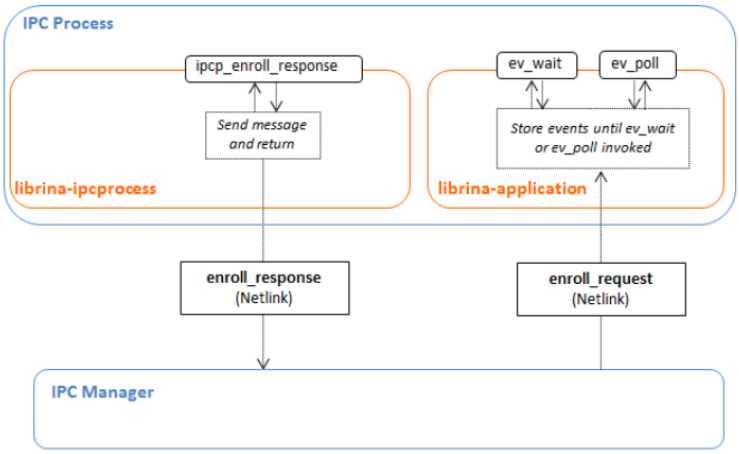


Figura 3.32: Comportamiento del IPC process daemon cuando se le ordena el enrollment con otro IPC process

3.2 Componentes de RINA en el kernel

Una introducción de los componentes de RINA de este prototipo ubicados en el kernel, y las razones de esta disposición, ya fue hecha en el Sección 3. La Fig. 3.33 muestra la disposición de los componentes en la arquitectura.

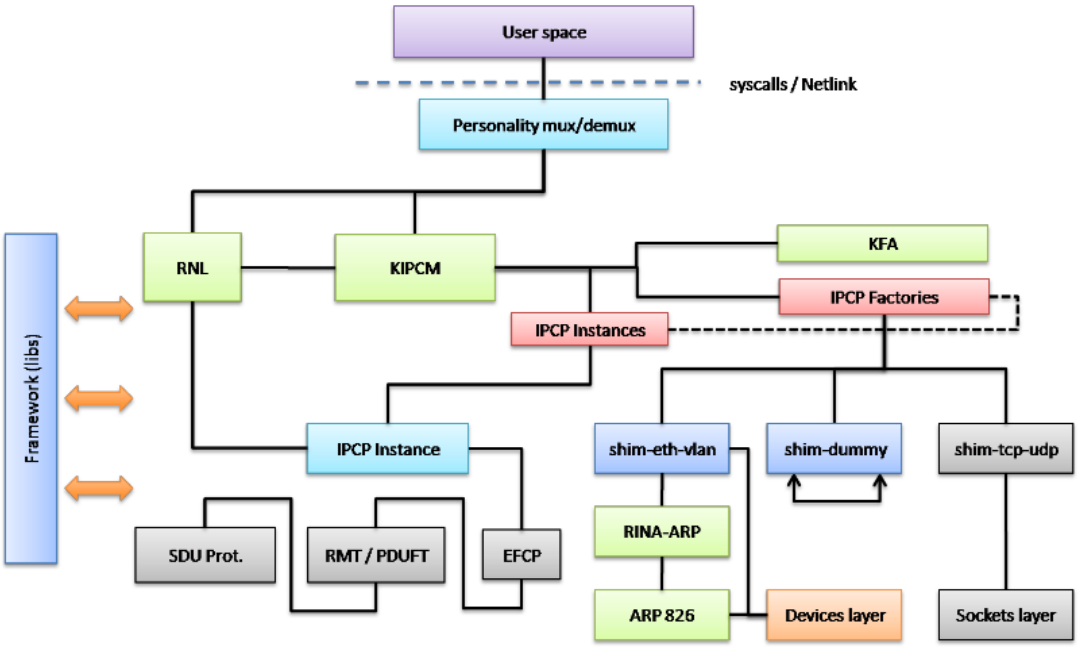


Figura 3.33: Arquitectura de los componentes del kernel

3.2.1 Interfaces de los componentes

Los componentes en el kernel tendrán "dos" interfaces north-bound en el sentido que algunas llamadas requerirán privilegios de root, mientras que otras no:

- La interfaz de aplicación: la misma que la interfaz south-bound de librina-ipc-process, estará disponible para las aplicaciones.
- La interfaz de administración y configuración: usada por el administrador con privilegios de root para configurar los componentes del kernel.

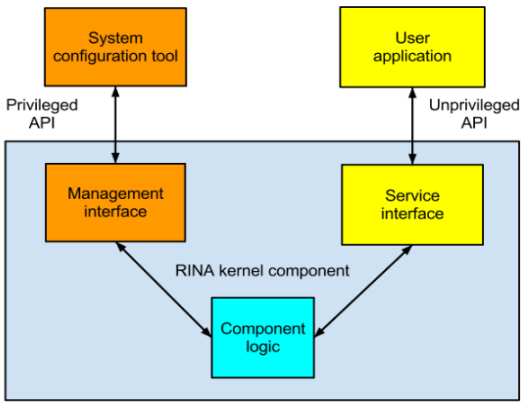


Figura 3.34: Interfaces de los componentes del kernel

3.2.2 La "Personality" y la interfaz de user-space/kernel

La capa de "Personlaity" aspira a soportar distintas implementaciones del stack del kernel de RINA coexistiendo en el mismo sistema. Para conseguirlo, una única interfaz entre user-space y el kernel debe ser establecida. Ésta está compuesta por llamadas a sistema y la capa RINA Netlink Layer (RNL).

La capa de Personality actúa como un multiplexor/demultiplexor que permite conectar dinámicamente diferentes implementaciones en el core del stack. Cada implementación se asocia con una instancia de Personality, la cual posee un identificador único. Desde user-space al kernel, la capa de Personality demultiplexa la interfaz hacia la Personality activa. En la dirección contraria, la capa de Personality multiplexa la instancia de la Personality hacia la entidad en user-space que pidió un servicio. Esto es posible dado que los componentes del kernel core tienen bindings a sus "padres" o "hermanos" en una arquitectura jerárquica donde la Personality es la raíz.

3.2.3 El Core del stack

Los componentes del kernel están divididos en dos categorías: el core y los IPC processes. El core es el conjunto de componentes que ofrecen el conjunto de funcionalidades fundamentales del sistema y los nexos que lo unen a las distintas instancias de IPC processes. Los componentes en el core orquestan las interacciones con el user-space, el ciclo de vida de los IPC processes y las operaciones de transmisión de datos. Está formado por el KFA, el KIPCM, la RNL y las IPC process Factories.

3.2.3.1 El Kernel Flow Allocator (KFA)

El KFA está a cargo de las operaciones de administración de flows como la creación, binding a los IPC processes, destrucción, etc. Estas funcionalidades se ofrecen al KIPCM por la interfaz superior, y a los IPC processes por la inferior como se ve en Fig. 3.35. Las dos funcionalidades principales del KFA son: i) la gestión de los identificadores de puertos (portIDs) mediante el Port ID Manager (PIDM) y ii) el binding del KIPCM y los IPC processes a través del flow que maneja.

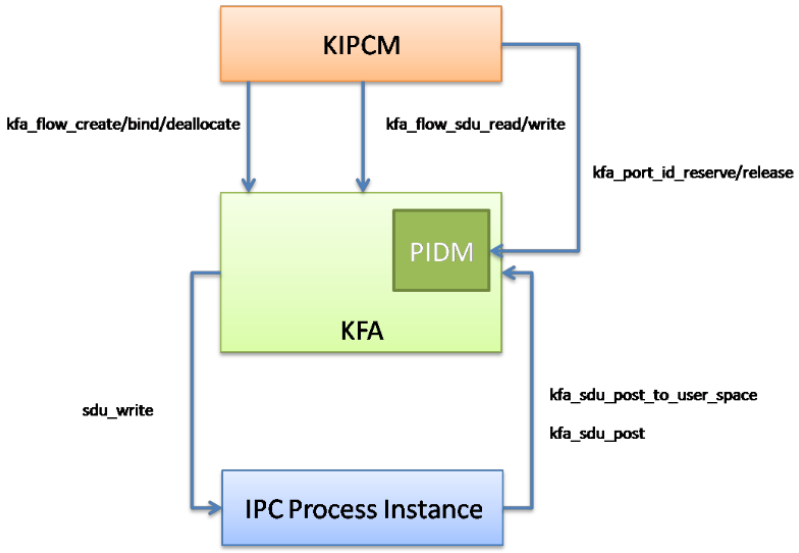


Figura 3.35: Detalle de las interacciones del KFA

3.2.3.2 El Kernel IPC Manager (KIPCM)

El KIPCM es el representante del IPC Manager en el kernel, con lo cual sus funciones son las mismas que el IPC Manager (Sección 3.1.2.2) pero en el contexto de los componentes del kernel. En concreto se encarga de:

- recuperar las instancias de los distintos componentes de un IPC process en particular, independientemente de la clase de proceso (normal, shim-dummy, shim Ethernet, shim TCP/UDP).
- Abstraer la naturaleza del IPC process respecto a la escritura y lectura de SDUs.
- Administración del ciclo de vida de los componentes del stack en el kernel (IPC processes, KFA, etc).
- Proveer las interfaces principales a user-space y actuar como hub de la RNL.

3.2.3.3 La RINA Netlink Layer (RNL)

En conjunto con un conjunto de llamadas a sistema y el procfs/sysfs, los sockets Netlink [15] son una de las tecnologías que conforman el framework de comunicación entre el user-space y el kernel en el stack de RINA. Además de Netlink, la extensión Generic Netlink, la cual trabaja como un multiplexor de protocolos, es utilizada para definir la familia de mensajes de RINA.

La RNL presenta una capa que abstrae y se encarga de todas las tareas relacionadas con la configuración, generación y destrucción de sockets y mensajes de Netlink escondiendo esta complejidad a los usuarios de la capa. Las funcionalidades concretas que ofrece son:

- Formatea y parsea los mensajes de Netlink de la familia de RINA.
- Permite a los componentes registrar manipuladores para los mensajes recibidos.
- Actúa como hub que multiplexa los mensajes recibidos desde user-space hacia los manipuladores registrados.

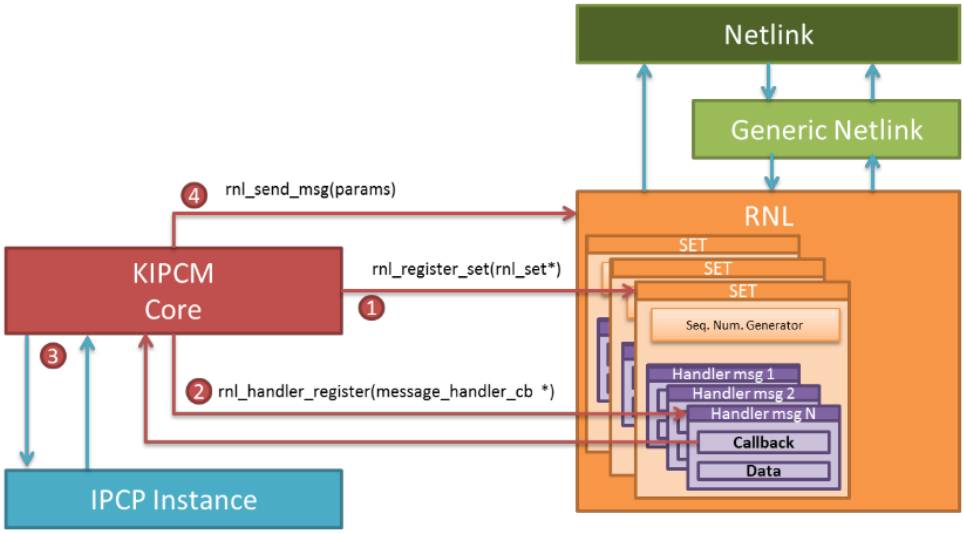


Figura 3.36: Interacciones de la RNL

3.2.3.3.1 Comportamiento de recepción

En la recepción de un nuevo mensaje desde user-space, la RNL busca entre los manipuladores registrados el adecuado al tipo de mensaje recibido. Entonces utiliza la función call-back contenida en el manipulador para pasar el mensaje al componente a cargo de atender dicho mensaje. El componente utilizará la RNL para parsear el mensaje y traducirlo a una estructura interna conocida y proseguir con su trabajo.

3.2.3.3.2 Comportamiento de envío

En la dirección contraria, cuando un componente quiere enviar un mensaje Netlink a user-space, llamará a RNL especificando el tipo de mensaje que quiere enviar y el valor de los parámetros que debe contener. RNL entonces comprobará que los parámetros sean correctos, generará un mensaje Netlink y un número de secuencia y lo enviará utilizando el identificador del socket de Netlink pasado.

3.2.3.4 IPCP Factories

Las factorías de IPC processes abstraen la naturaleza del IPC process (normal o shim) a sus usuarios (KIPCM, el user-space) mediante la producción de instancias de IPC processes abstractas. Cáda tipo de IPC process tiene su propia factoría, la cual registra en el KIPCM durante su inicialización (cuando el módulo es cargado en el kernel). El KICPM, en su lugar, utiliza la factoría cuando debe crear un IPC process.

La instancia de IPC process generada por la factoría ofrece una API común que abstrae la verdadera naturaleza del proceso pero que por debajo está conectada a la API específica del correspondiente tipo. La Fig. 3.37 muestra esta situación.

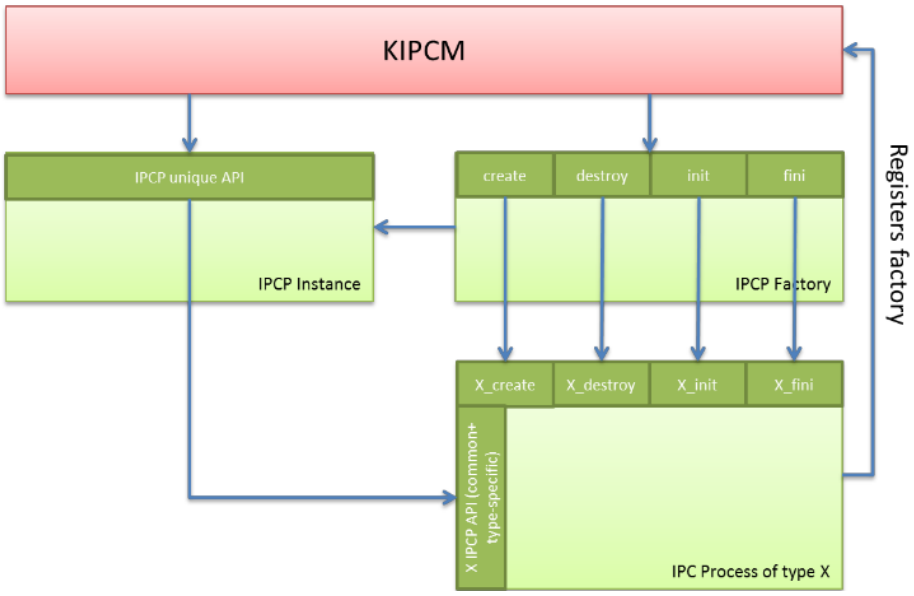


Figura 3.37: La factoría de IPC Proccess, la instancia IPCP abstracta y sus interacciones

3.2.4 Los IPC processes

Con el objetivo de minimizar el número de cambios de contexto durante las operaciones del fast-path, los diferentes componentes del IPC process se han distribuido entre el kernel y el user-space. Con esta división, el EFCP, RMT PFT y SDU protection forman parte de la

instancia de IPC process normal en el kernel. Sin embargo, para el caso de los IPC processes shim todos sus componentes se encuentran en el kernel ya que son específicos para la tecnología que adaptan (ej: Ethernet).

Aunque en el kernel puedan coexistir IPC processes de distinto tipo, las factorías descritas en Sección 3.2.3.4 proveen instancias abstractas que exportan una API común para ser usada por el resto de componentes del stack.

3.2.4.1 El IPC process normal

El IPC process normal es aquel IPC process, que a diferencia del shim IPC process, cumple con todos los requisitos de RINA y no adapta ninguna otra tecnología. El IPC process normal compone DIFs completamente compatibles con RINA y contiene todos los componentes del modelo de referencia que se explicarán a continuación.

3.2.4.1.1 Error and Flow Control Protocol (EFCP)

EFCP es el protocolo de transmisión para la comunicación entre dos procesos IPC. El protocolo asegura fiabilidad, orden y control de flow si es requerido por el QoS. EFCP está basado en Delta-T, diseñado por Richard Whatson [16].

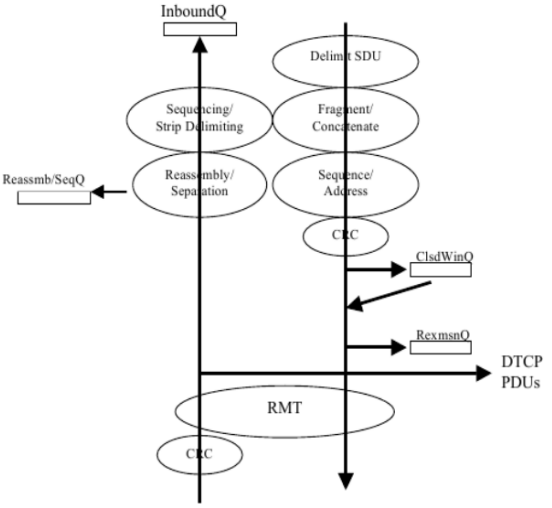
La cadena de octetos intercambiados entre dos máquinas de estado es una Protocol Data Unit (PDU), la cual está compuesta por una Protocol Protocol Control Information (PCI), entendida por la DIF; y la Service Data Unit (SDU), que contiene la información de usuario que no es entendida por la DIF y es pasada al usuario .

La máquina de estados de EFCP consiste en dos submáquinas con binding débil entre sí:

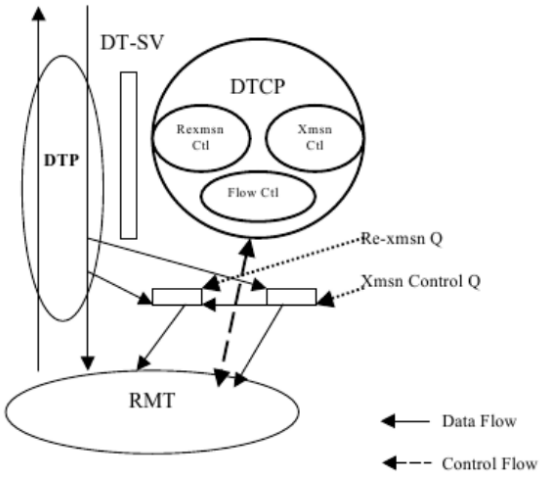
- La máquina de estados de Data Transfer Protocol (DTP): realiza la transmisión de PDUs, los mecanismos de binding fuerte como secuenciación, fragmentación/reensablado y concatenación/separación. Cada instancia de flow cuenta con una instacia de DTP.
- La máquina de estados de Data Transfer Control Protocol (DTCP): realiza los mecanismos de binding débil (feedback) como: retransmisión y control de flow. El estado de esta máquina se descarta después de largos periodos sin tráfico (2 Maximum Packet Lifetime (MPL)). Una instancia por flow es creada si éste requiere retransmisión o control de flow. DTCP utiliza los números de secuencia de las PDUs y tres timers para asegurar sincronismo con la instancia peer de DTCP: MPL, tiempo máximo que el receptor mantendrá una PDU antes de enviar un ack (A), y máximo tiempo que un emisor esperando un ack intentará la transmisión antes de desistir (R).

Las PDUs transmitidas por el EFCP pueden ir en dos direcciones:

- Las PDUs generadas por el EFCP a partir de SDUs de las aplicaciones son pasadas al RMT.
- Las PDUs recibidas por el RMT de un puerto N-1 son pasadas a la correspondiente instancia de EFCP para su posible reensamblado y entrega al correspondient proceso de aplicación.



(a) Esquema de DTP



(b) Esquema de DTCP

3.2.4.1.1.1 Interfaz de administración

La interfaz de administración del EFCP permite basicamente: i) recuperar las medidas tomadas por EFCP durante la ejecución; ii) permitir al administrador suspender o resumir el tráfico de una instancia de EFCP.

La interfaz esta basada en sysfs/procfs con la siguiente estructura:

- Directorio /proc/sys/rina/efcp/<process-id>/<cep-id>/statistics
- Un fichero por variable: pdus_received, bytes_received, average_pdu_lenght
- Un fichero por parámetro configurable en /proc/sys/rina/efcp/<process-id>/<cep-id>

3.2.4.1.2 Relating and Multiplexing Task (RMT) La RMT se sitúa lógicamente entre el EFCP y el módulo de SDU Protection. Tiene dos tareas principales:

- El rol de Relaying ("fiablidad") es para hacer el forwarding de PDUs que pasan por el IPC process al EFCP de destino comprobando la dirección de destino en la PCI de la PDU. La decisión de forwarding se basa en información de routing y el QoS acordado.
- El rol de la tarea de Multiplexing es precisamente multiplexar las PDUs entre las distintas instancias de EFCP y los distintos puntos de contacto (port-id) con los N-1 flows a las N-1 DIFs. El forwarding viene marcado por varias políticas que afectan al QoS entregado (adminstración de colas, scheduling, longitud de las colas, etc).

Además la RMT interactúa con la SDU Protection para proteger, si es requerido, las PDUs que se pasarán a la N-1 DIF.

3.2.4.1.3 El fast-path

Como ya se ha introducido, el fast-path corresponde con aquellos flujos de trabajo que se repiten frecuentemente y requieren ser rápidos y poco exigentes de recursos. En concreto lo forman las operaciones de read y write PDUs/SDUs. La Fig. 3.38 muestra las interacciones del EFCP y RMT con el resto de componentes durante las operaciones del fast-path. Las siguientes secciones se adentran en en los detalles de los procesos de lectura y escritura de Protocol o Service Data Units (DUs).

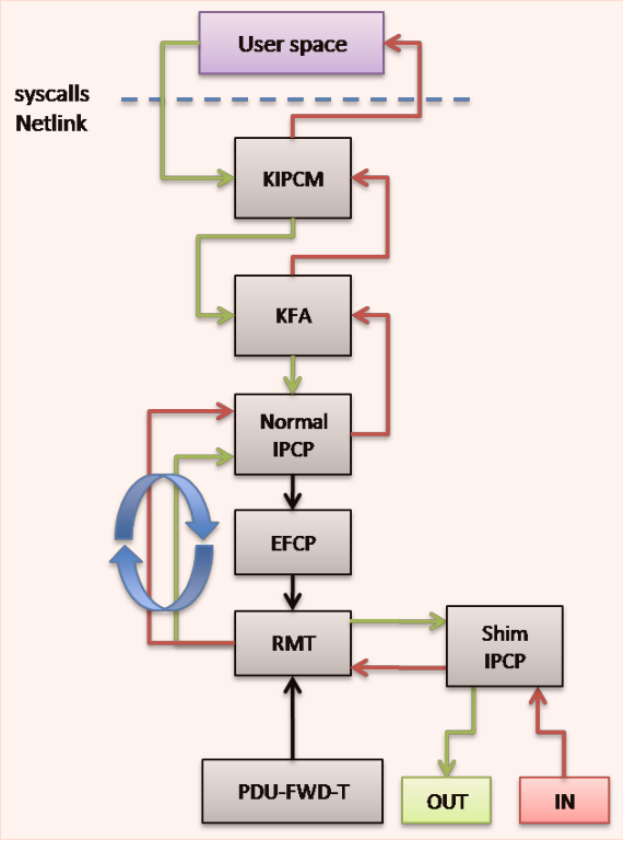


Figura 3.38: Interacción de los componentes durante el fasth-path

3.2.4.1.3.1 El flujo de salida de las DUs

El flujo de salida de DUs dentro del IPC process normal se puede resumir en los siguientes pasos:

1. Una SDU llega al EFCP Container via la la llamada `efcp_container_write`.
2. El EFCP Container obtiene la instancia correcta de EFCP y llama a `efcp_write`.
3. El EFCP pone la SDU en el DTP. Esto genera un nuevo ítem en la cola de trabajo del EFCP Container.
4. Cuando el ítem es ejecutado:
   (a) El DTP añade una PCI a la SDU produciendo una PDU.
   (b) La PDU es enviada al RMT mediante `rmt_send`, lo cual crea un ítem en la cola de trabajo del flujo de salida.
5. Cuando el ítem es ejecutado en el RMT:
   (a) Recupera el N-1 portID que debe ser utilizado para enviar la PDU a la dirección de destino con el QoS del flow.
   (b) Finalmente, regenera una SDU que es enviada. Esto es necesario por que para la N-1 DIF la DU recibida es considera una SDU.

Seguidamente se detalla el mismo proceso pero afectando a distintos IPC processes y por tanto haciendo énfasis en sus interacciones. En situaciones normales dentro de un sistema cuando existe un stack de IPC processes como el de la figura Fig. 3.40, el proceso descrito en la sección anterior e iniciado por una llamada al sistema de `sys_sdu_write`, se repite en los IPC processes hasta que llegan a un IPC process shim. Los siguientes pasos describen este flujo basándose en la Fig. 3.40 donde hay dos IPC processes normals y un shim Ethernet.

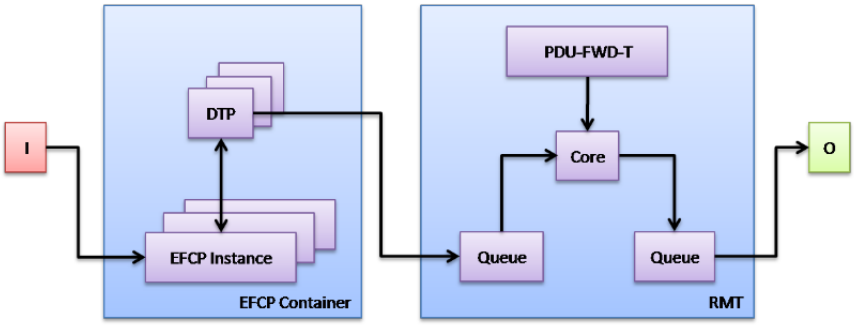


Figura 3.39: Flujo de trabajo de salida de DUs dentro del IPC process

1. La syscall toma como parámetros de entrada la SDU a ser enviada y el N-portID del flow. En el ejemplo, app2.
2. El KIPCM, vía `kipcm_flow_sdu_write`, recibe la SDU y la envía al KFA mediante `kfa_flow_sdu_write`.
3. El KFA recupera la instancia de IPC process desde la instancia de flow asociada al portID pasado y llama a `sdu_write` de la API del IPC process. Internamente, al ser un IPC process normal, IPC Process 2 utiliza `normal_sdu_write` que llama a `efcp_container_write`.
4. En este punto comienza el flujo interno descrito en la Fig. 3.39 anteriormente. La RMT llama otra vez a `kfa_flow_sdu_write` pero pasando un N-1 portID, en el ejemplo 21. Se debe aclarar que la sdu* de la figura hace referencia a que no es la misma sdu con la que se originó el proceso.
5. El mismo proceso es seguido ahora por el IPC process 1 que es también del tipo normal hasta que finalmente la sdu* es enviada al KFA al puerto 10.
6. El KFA recupera la estructura de flow, luego el IPC process asociado y llama a la función `sdu_write` sobre este último. En este caso, el IPC process es de tipo shim Ethernet, con lo cual internamente se ejecutará la función `shim_write`. El flujo interno de un IPC process shim Ethernet se distinto y se explicará en Sección 3.2.4.2, pero el resultado de éste es el envio de frames Ethernet hasta el sistema receptor.

3.2.4.1.3.2 El flujo de entrada de las DUs

El flujo de entrada de DUs dentro del IPC process normal se puede resumir en los siguientes pasos:

1. Una SDU llega a la RMT via la la llamada `rmt_receive` y es transformada a un ítem de trabajo en la cola de entrada.
2. Cuando el ítem es ejecutado, la SDU es transformada en una PDU y se comprueba la dirección de destino:
   (a) Si la dirección de destino no es la del IPC process la RMT utilizará la PFT para buscar un puerto N-1 para llegar a dicha dirección y se enviará. En caso de no encontrar un puerto para llegar a la dirección, la SDU se descarta.
   (b) Si la dirección de destino es la del IPC process, la RMT llama a `efcp_container_receive` pasándole la PDU al DTP y generando un nuevo ítem de trabajo en la cola de entrada del EFCP Container.
3. Cuando el ítem es ejecutado en el EFCP Container:

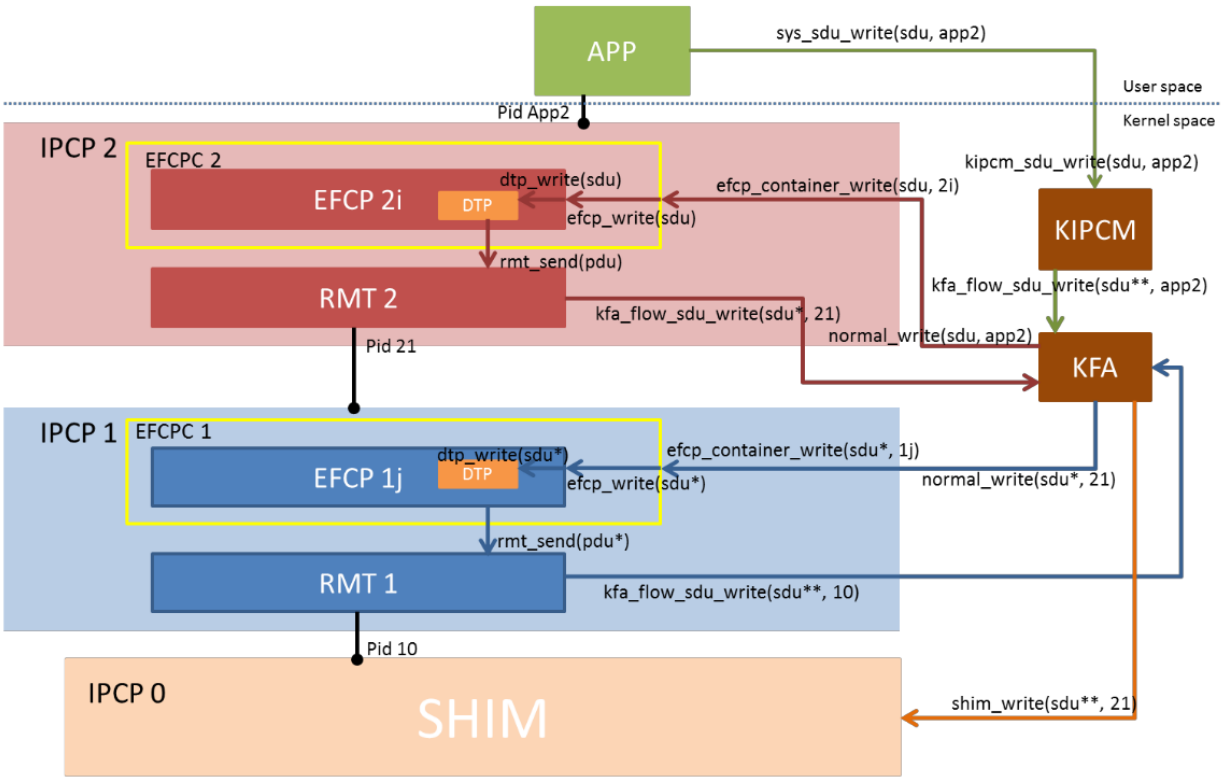


Figura 3.40: Flujo de trabajo de salida de DUs entre IPC processes

(a) El DTP recibe la información sobre la conexión mediante el PCI de la PDU, el cual es descartado.
(b) Finalmente, envía la SDU al KFA mediante `kfa_sdu_post` para que sea consumida.

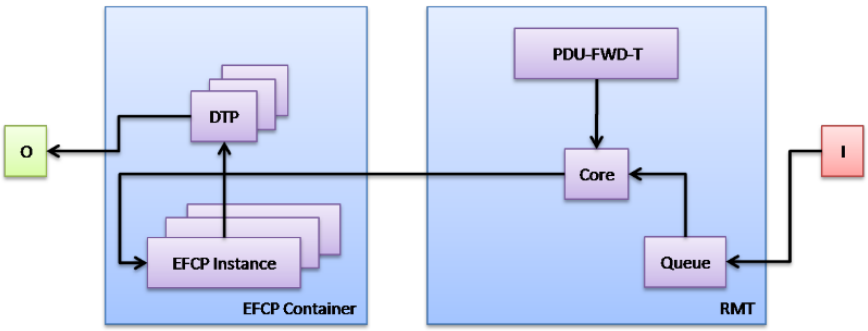


Figura 3.41: Flujo de trabajo de entrada de DUs dentro del IPC process

A continuación se describe el mismo proceso pero afectando al mismo stack de IPC processes de la figura Fig. 3.40 se la sección anterior. La Fig. 3.42 refleja las interacciones para el flujo de entrada.

1. Una SDU llega al sistema y después de ser procesada por el device driver de la tarjeta Ethernet es recibida por el IPC process 0, el cual realiza las tareas propias que se explicarán en Sección 3.2.4.2 resultando en una llamada a `kfa_sdu_post` al portID 10.

2. El KFA recupera la estructura de flow asociada al puerto y comprueba si dicho puerto está conectado a una instancia de RMT. En ese caso llama a `rmt_receive` para pasar la SDU al IPC process 1.
3. En este punto comienza el flujo interno descrito en la Fig. 3.41 anteriormente. El EFCP llama otra vez a `kfa_sdu_post` pasando el portID 21
4. Los pasos 2 y 3 son repetidos hasta que el KFA no encuentra una instancia de RMT conectada al puerto, considerando que la SDU es destinada a una aplicación en user-space.
5. El KFA entonces, coloca la SDU en cola del flow que contiene las SDUs listas para ser entregadas a la aplicación.
6. Cuando la aplicación invoca la llamada a sistema `sdu_read` recupera la SDUs que esperaba en la cola.

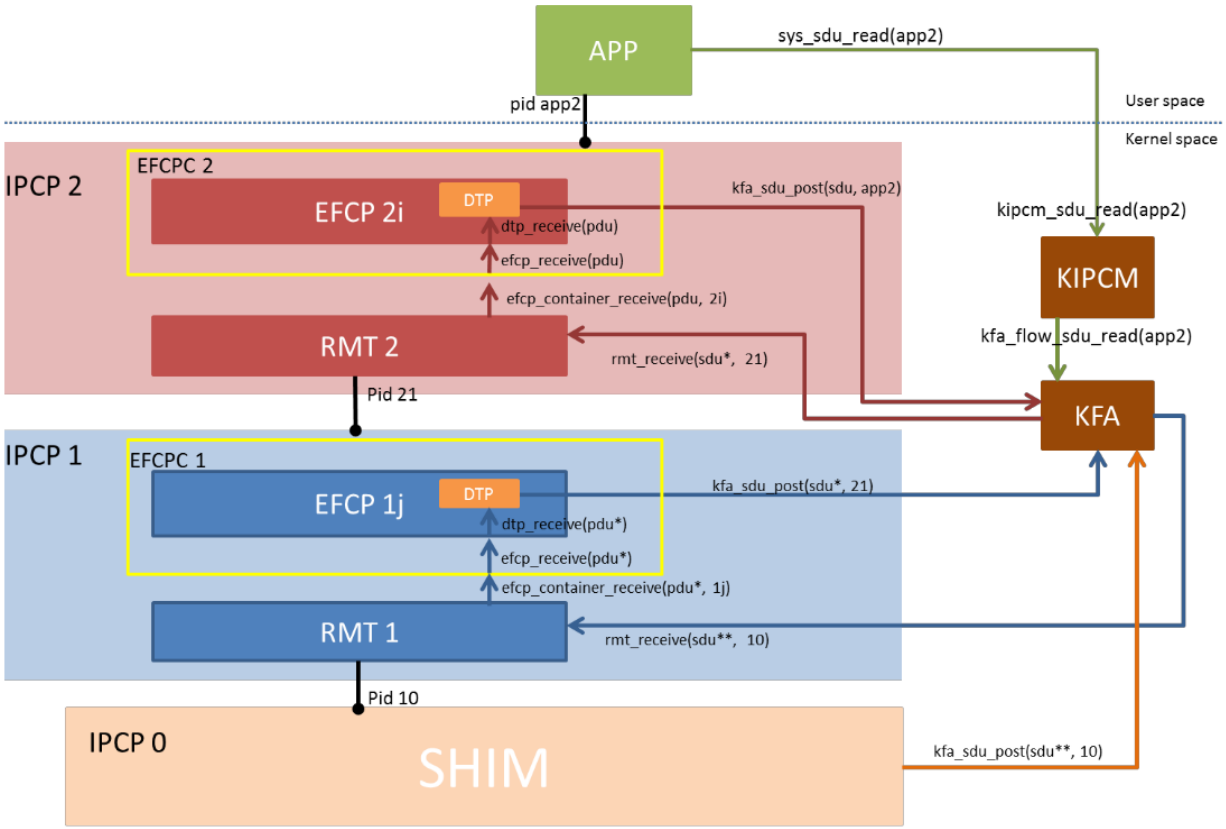


Figura 3.42: Flujo de trabajo de entrada de DUs entre IPC processes

3.2.4.1.4 SDU Protection

El módulo de SDU protection comprueba si la PDU del N-1 flow necesita más procesado para añadir o remover mecanismos de protección. Entre estos mecanismos se encuentra: CRCs, encriptación, TTL/Hop-count, etc, dependiendo de las característica del flow. El resultado de la protección puede suponer añadir campos a la PDU (PDU-CRC, etc).

En el primer prototipo, los workflows de la SDU vendrán determinados por la RMT de la sigueiente manera:

- La RMT entregará las PDUs listas para enviar al módulo quien se lo devolverá con la

correspondiente protección para ser colocado en la cola correspondiente al puerto del flow N-1.
- Cuando la RMT reciva una PDU del N-1 flow, la enviará al módulo de protección para aplicar el mecanismo inverso y recuperar la PDU desprotegida con tal de realizar las tareas necesarias.

3.2.4.2 Shim IPC process sobre Ethernet

La DIF shim Ethernet presenta una capa Ethernet como si fuera una DIF normal de RINA a las otras DIFs mediante una API RINA nativa. El objetivo de una shim DIF es implementar la funcionalidad mínima necesaria para poder usar un protocolo heredado dentro de RINA. Los procesos IPC shim Ethernet trabajan sobre Ethernet aumentados con el standard 802.1Q (VLAN).

Si bien el estándar del protocolo ARP no especifica que las direcciones a mapear con MACs deban ser IPs, la implementación de Linux se limita a este caso de uso. Esto impuso la necesidad de implementar una versión liviana de ARP que cumpla las especificaciones RFC-826[17] y sea adecuado para el shim IPC process sobre Ethernet. Con este fin de creó el módulo ARP826. Por último, como capa intermedia entre el shim IPC process y ARP826 existe la capa de RINARP.

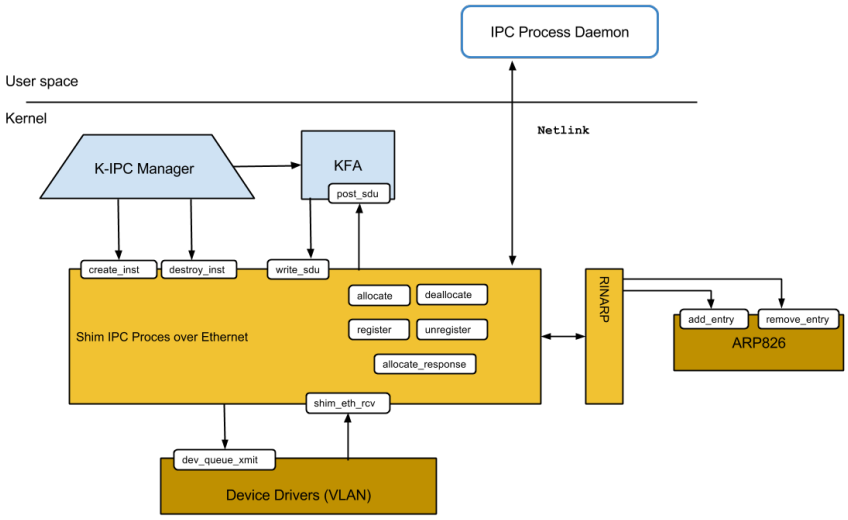


Figura 3.43: Interacción del shim IPC process sobre Ethernet con el resto de componentes

3.2.4.2.1 ARP826

El módulo ARP826 es una implementación liviana del estándar RFC-826[17]. Su desarrollo fue necesario dado que la implementación de Linux tiene los siguientes problemas:

- Linux ARP solo permite traducir direcciones IPv4.
- Cuando una nueva entrada está disponible en el cache ARP, sólo el stack de TCP/IP es notificado.
- Sólo permite tener una dirección de protocolo de red por interfaz. Linux ARP obtiene la dirección IP de la interfaz, con lo cual sería necesario que los nombres de aplicaciones fueran direcciones IP o el stack TCP/IP quedaría inutilizado al usar RINA.
- Las peticiones ARP de Linux sólo pueden ser enviadas a IPs de la misma subred, lo que agregaría muchas restricciones a los nombres de aplicación.

El diseño de ARP826 puede verse en Fig. 3.44. Sus componentes son: el core, mapas y tablas (cache), el Address Resolution Module (ARM) y la parte de RX/TX.

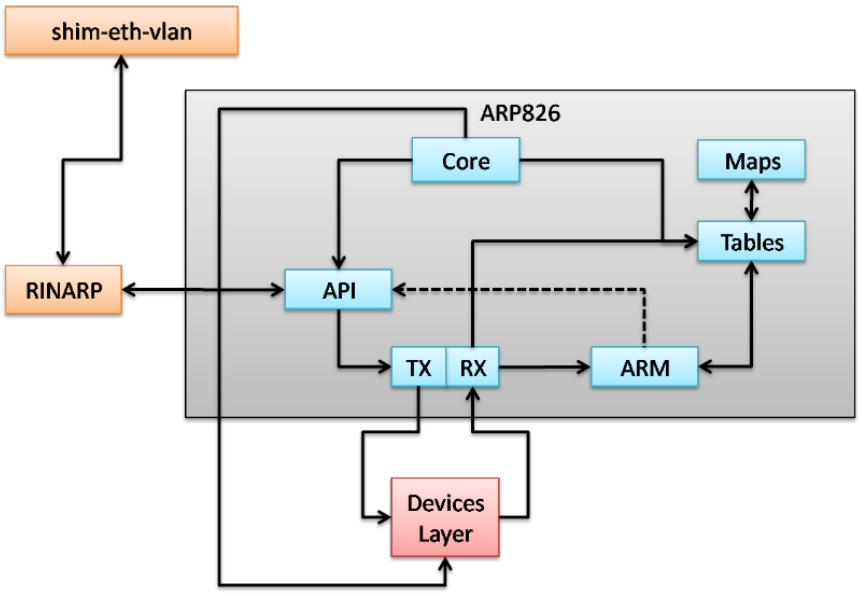


Figura 3.44: Interacción de los componentes de ARP826

3.2.4.2.1.1 Los Generic Protocol Addresss (GPAs) y Generic Hardware Addresss (GHAs)

El módulo ARP826 se puede resumir como un directorio que mapea direcciones de un protocolo de red con una dirección física. Las estructuras de datos que representan estas direcciones son las GPAs y GHAs respectivamente. Además, estas estructuras abstraen la naturaleza de las direcciones permitiendo aumentar o disminuir dinámicamente el tamaño de estas direcciones (requerimiento de algunos protocolos) y creando direcciones según la tecnología que haya debajo (Ethernet para el caso del stack presentado).

3.2.4.2.1.2 El core de ARP826

El core se encarga de la inicialización y configuración del módulo del kernel, además del registro de los protocolos de red en módulo. Durante la inicialización, la función `arp_receive` de la parte de RX/TX es registrada en la capa de device drivers del sistema. Esta función es llamada ante la llegada de una trama Ethernet con etherype ARP.

3.2.4.2.1.3 El Address Resolution Module (ARM)

El ARM se encarga de procesar las peticiones de traducción de direcciones. Este proceso de hace de manera asíncrona. Las peticiones son puestas en una cola de trabajo y atendidas oportunamente. Cuando se obtiene un resultado, el módulo utiliza la función de call-back que haya especificado el usuario que generó la petición, pasando el resultado como parámetro.

3.2.4.2.1.4 La parte de RX/TX

Este subcomponente se encuentra en la parte inferior del modelo. Es un componente pasivo que solo responde a eventos externos:

- Cuando se recibe una trama Ethernet, RX comprueba que pueda procesarla y que se trate de una petición o respuesta ARP. Si es una petición, la cache es consultada para

recuperar la dirección del protocolo de red. Si se obtiene, la respuesta es enviadaTX). Si la trama es una respuesta, ésta es redirigida al ARM.
- TX es ejecutado cuando un usuario utiliza su API. Simplemente crea una nueva petición ARP y la transmite por la interfaz especificada.

3.2.4.2.2 RINARP

RINARP es simplemente una capa de abstracción interpuesta entre el shim IPC process sobre Ethernet y ARP826. De esta manera es posible el desarrollo en paralelo de ambos componentes. Además, con este diseño se obtienen dos ventajas. Por un lado, es posible cambiar la implementación de ARP826 sin que los shim IPC processes se vean involucrados. Por el otro lado, la capa RINARP puede ser utilizada por otros procesos shim basados en Ethernet como podría ser un eventual shim IPC Process WIFI.

3.2.4.2.3 Comportamiento detallado del shim IPC process sobre Ethernet

3.2.4.2.3.1 Creación/destrucción de un shim IPC process

Durante la creación, un proceso shim Ethernet se inicializa con: i) el nombre del proceso de aplicación, ii) la interfaz Ethernet a la cual conectarse y iii) el nombre de la DIF shim, que es en realidad el ID de la VLAN a utilizar.

El último punto implica una precondición, ya que la interfaz virtual deberá haber sido creada antes de la creación del IPC process o ésta fallará. En caso contrario, el IPC process preguntará por la lista de drivers de las interfaces hasta encontrar el correspondiente a la interfaz VLAN de la shim DIF, al cual guardará como handler para tratar las PDUs que lleguen.

3.2.4.2.3.2 Registro de aplicación

Cuando una aplicación debe registrarse, el IPC process recibirá un mensaje `register_app` del IPC Manager. En lugar de implementar su propio mecanismo de directorio, se reutiliza el protocolo ARP (ARP826) en modo request/response. Para registrar una aplicación se agrega en la tabla ARP una entrada estática que mapea el nombre de la aplicación a la dirección MAC de la interfaz por donde llega el flow. Después el IPC process envía un mensaje `register_app_response` al IPC Manager. Sólo una aplicación a la vez puede registrarse en un IPC process shim Ethernet debido a la limitación que solo una entrada aplicación-MAC se puede poner en la tabla por cada MAC.

3.2.4.2.3.3 Desregistro de aplicación

Este mecanismo es lanzado por la recepción de un mensaje `unregister_app`, ante lo cual el IPC process removerá la entrada correspondiente del cache de la tabla ARP y enviará el correspondiente mensaje de respuesta `unregister_app_response` al IPC Manager.

3.2.4.2.3.4 Reserva de flow

Las peticiones de flows son comunicados al IPC process con un mensaje `allocate_request`. Si ya hay un flow establecido a la aplicación de destino (ya hay un portID en estado ALLOCATED), o en proceso (un portID en estado INITIATOR_ALLOCATE_PENDING), una respuesta negativa es devuelta al IPC Manager. Si ya existe una entrada en la cache ARP para la aplicación de destino, el IPC process creará un nuevo portID, responderá positivamente y pondrá el portID en estado ALLOCATED. Si no hay ninguna entrada en la cache

ARP, el IPC process creará un nuevo portID y enviará una request ARP. El portID cambia a estado INITIATOR_ALLOCATE_PENDING.

Si el portID se encuentra en estado INITIATOR_ALLOCATE_PENDING, y una petición ARP llega desde la aplicación que se está intentando alcanzar, el portID cambia a estado ALLOCATED y un mensaje `allocate_response` es enviado el IPC Manager.

Si el portID está en estado INITIATOR_ALLOCATE_PENDING, y una respuesta ARP llega, una respuesta positiva es generada y enviada junto con el portID al IPC Manager vía un mensaje `allocate_response`. Se añade una nueva entrada al directorio mapeando el nombre de la aplicación de destino con la dirección MAC. En este caso el portID pasa a estado ALLOCATED. Si el portID está en estado ALLOCATED y una respuesta ARP llega, el directorio se actualiza con el nuevo mapping de aplicación-MAC.

Cuando un IPC process recibe una trama Ethernet, comprueba si ya existe un flow para esta combinación de MACs origen/destino. Si este es el caso y el portID correspondiente se encuentra en estado ALLOCATED, la SDU es entregada al portID (será manipulada por el KFA). Si no es el caso, la trama es puesta en cola y un nuevo flow y portID son creados, inicializando este último en estado RECIPIENT_ALLOCATE_PENDING. Un mensaje `allocate_request_arrived` pidiendo la creación de un flow es enviado al IPC process daemon mediante el cual se llega a la aplicación de destino.

El IPC process responderá con un mensaje `allocate_response`. Si la respuesta es positiva, las frames que estaban en cola son entregadas a la aplicación de destino. El portID cambia a estado ALLOCATED. Si es negativa, la creación del flow falla. Entonces, todas las frames de la MAC de origen son descartadas hasta que la entrada ARP para dicha MAC de origen es removida del cache en el IPC process de destino. En ese caso el IPC process llama `shim_eth_deallocate_flow` y el portID pasa a estado NULL.

3.2.4.2.3.5 Destrucción de flow

Cuando un portID pasa a estado NULL (porque se recibió un mensaje `deallocate_request` o por decisión del IPC process) todas las estructuras de datos correspondientes al flow son eliminadas.

3.2.4.2.3.6 Lectura/escritura de SDUs

El KIPCM invocará la llamada `shim_eth_write_sdu` en el portID cuando tenga SDUs listas para ser escritas en el flow hacia el IPC process shim Ethernet. Si el port-id está en estado ALLOCATED, el IPC process llamará a `dev_queue_xmit` del device driver correspondiente. De la misma manera, cuando el device driver avise al proceso IPC que hay una trama Ethernet disponible, el proceso IPC recuperará la SDU y la enviará al correspondiente N+1 portID llamando al KIPCM.

3.2.4.3 Shim IPC process sobre TCP/UDP

La DIF shim sobre TCP/UDP presenta una capa TCP+IP como si fuera una DIF normal de RINA a a las otras DIFs mediante una API RINA nativa. El objectivo de una shim DIF es implementar la funcionalidad mínima necesaria para poder usar un protocolo heredado dentro de RINA. Los procesos IPC shim TCP/UDP trabajan directamente sobre la capa de sockets provista por el kernel.

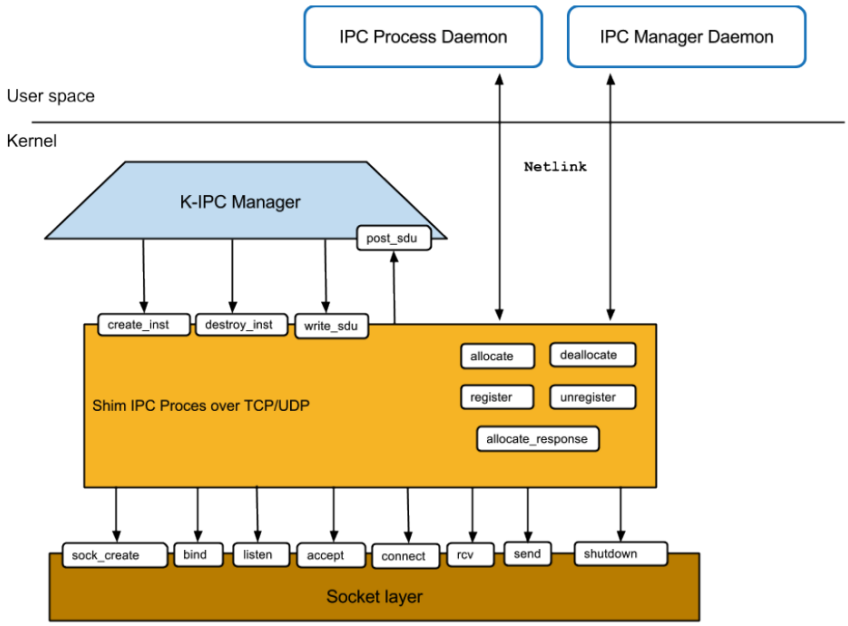


Figura 3.45: Interacción del shim IPC process sobre TCP/UDP con el resto de componentes

3.2.4.3.1 Comportamiento detallado

3.2.4.3.1.1 Creación/destrucción de IPC process

En tiempo de creación, el shim IPC process sobre TCP/UDP es inicializado con:

- el nombre del shim IPC process
- el nombre de la shim DIF
- la interfaz sobre la que el IPC process hará el binding
- las aplicaciones permitidas para registrarse en la DIF representadas con el puerto TCP/UDP que utilizan.
- Directorio con el mapping entre nombres de aplicaciones y direcciones IP y puerto.

Cuando el IPC Manager en user-space requiere la creación de un IPC process shim TCP/UDP se comunica con el KIPCM quien invoca `shim_tcp_udp_create` al shim IPC process TCP/UDP. Este componente comprueba que no haya otra instancia de IPC process en la interfaz requerida e inicializa las estructuras de datos necesarias.

La destrucción se realiza mediante la invocación de `shim_tcp_udp_destroy`.

3.2.4.3.1.2 Registro de aplicación

Cuando una aplicación debe ser registrada, el IPC process recibe un mensaje `register_app` del KIPCM (como proxy del IPC Manager). Entonces mira en la tabla de aplicaciones, si existe una entrada para la aplicación recupera el puerto y hace un binding a un socket en dicho puerto, donde comienza a escuchar. En ese momento devuelve la respuesta con un mensaje `register_app_response`.

3.2.4.3.1.3 Desregistro de aplicación

Operación activada por un mensaje `unregister_app`. El shim IPC process destruirá el socket creado y responderá con el mensaje de respuesta.

3.2.4.3.1.4 Reserva de flow

La operación se comunica con un mensaje `allocate_request`. El IPC process provee dos tipos de flows: i) confiables (TCP socket) y no confiable (UDP socket). A la recepción de la petición el IPC process comprueba el directorio en busca de la IP y puerto que deben ser utilizados para llegar a la aplicación de destino. Un socket /TCP o UDP) es abierto en dicha IP:puerto y las estructuras necesarias en el kernel son creadas. Finalmente se envía el mensaje de respuesta al IPC Manager.

El IPC process remoto extremo del flow creado tiene dos formas de enterarse de la petición de flow. Si es un flow confiable (TCP) tendrá que aceptar una conexión TCP entrante. En caso de un flow no confiable (UDP) no recibirá ninguna conexión entrante hasta que no llegue el primer paquete UDP. Por cada combinación de dirección IP de origen/destino y puerto UDP origen/destino, el IPC process comprueba que se trate de un flow ya reservado, y si no, crea las estructuras necesarias para servir al flow y avisar al IPC process destino del flow.

3.2.4.3.1.5 Destrucción de flow

Cuando el IPC Manager envía un mensaje `deallocate_request`, el IPC process comprueba si hay un flow activo en la dirección y puerto especificados por el mensaje. En caso afirmativo, cierra el socket y remueve las estructuras del flow antes de enviar el mensaje de resultado.

Si el flow es confiable, el IPC process sabrá que el flow fue destruido por un IPC process remoto ya que el socket TCP aparecerá como desconectado. En ese caso removerá las estructuras del kernel y enviará un mensaje `flow_deallocated` al IPC process que estaba utilizando dicho flow.

3.2.4.3.1.6 Registro de aplicación

Cuando una aplicación debe ser registrada, el IPC process recibe un mensaje `register_app` del KIPCM (como proxy del IPC Manager). Entonces mira en la tabla de aplicaciones, si existe una entrada para la aplicación recupera el puerto y hace un binding a un socket en dicho puerto, donde comienza a escuchar. En ese momento devuelve la respuesta con un mensaje `register_app_response`.

3.2.4.3.1.7 Lectura/Escritura de SDUs

El KIPCM invocará la llamada `shim_tcp_udp_write` en un puerto cuando tenga SDUs listas para ser enviadas al shim IPC process. Si el puerto ha sido reservado el IPC process llamará a la función `send` de la API de sockets. De la misma forma, el IPC process leerá datos del socket abierto y reensamblará las SDUs. Cuando tenga una SDU completa la enviará al KIPCM.

3.2.4.4 El Shim IPC process dummy

El shim IPC process dummy puede ser visto como una especie de IPC process de loopback. Su operación no atraviesa los límites del sistema con lo cual no puede comunicarse con procesos IPC en otros sistemas. Puede ser utilizado por IPC processes normales o directamente por aplicaciones en user-space, como es el caso de las desarrolladas para el testing. Como cualquier otro proceso IPC, el IPC process dummy registra sus propias factorías en el core (KIPCM) durante su inicialización.

4. Desarrollo

Esta sección se adentra en los detalles de la producción del código que implementa la arquitectura descrita en la sección anterior. El código del prototipo no se incluye en este reporte por la propia naturaleza del documento. En este momento, el repositorio del código es privado para el grupo de desarrolladores, sin embargo, es posible disponer de él para su valoración o estudio. Cuando el prototipo se encuentre en un estado más estable y completo se librerará con licencia GPL a la comunidad. La Fig. 4.1 muestra el sitio de Github con la pantalla principal del repositorio.

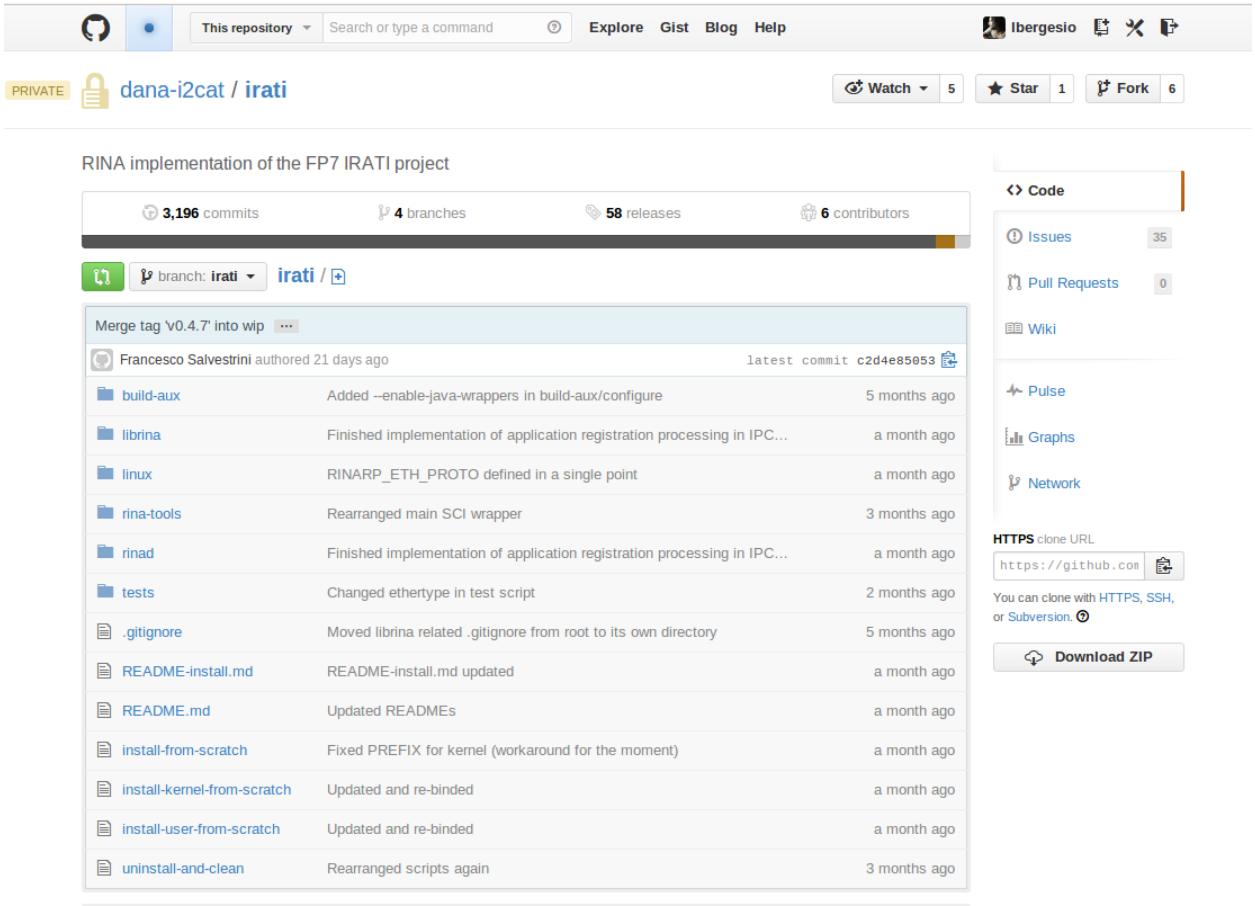


Figura 4.1: Repositorio del prototipo en IRATI

4.1 Entorno de desarrollo

El entorno de desarrollo y testing local se ha desarrollado utilizando la herramienta VirtualBox. Para esto se creó una máquina virtual (VM) con una instalación de Debian 7 Wheezy y el kernel v3.10.0. Las características más importantes de la máquina son:

- 1 CPU
- 1024 MB de RAM
- 15 GB de disco duro

Por otro lado, se procedió a la instalación de todas las herramientas necesarias para la construcción de los distintos componentes del prototipo. Estas son:

- automake [18]: Para generar makefiles
- autoconf [19]: Para generar los scripts de configuración
- libtool[20]: Para generar e instalar librerías.
- pkg-config [21]: Para la instalación de dependencias.
- SWIG [22]: Para la adaptación de las cabeceras y APIs entre los distintos lenguajes utilizados.
- JAVA: La VM y RE de JAVA dado que algunos de los componentes de user space son adoptados de un prototipo anterior desarrollado en JAVA.
- g++: Compilador de C++ para las librerías librina.
- gcc: Compilador C.
- gmake [23]: Manipulador de dependencias.
- git [24]: Respositorio utilizado.
- ncurses [25]: Librerías para construir la herramienta Kconfig.

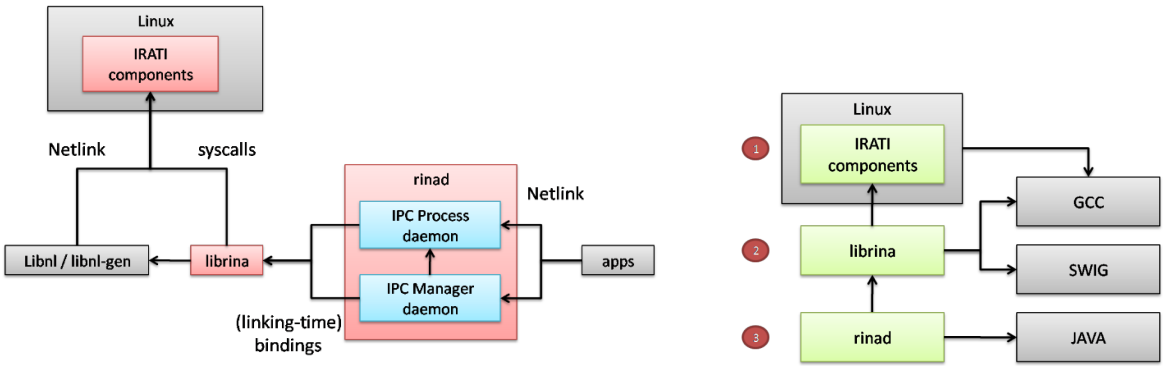
En cuanto a los entornos de desarrollo, se instalaron GNOME y el IDE Eclipse como complemento al editor habitual utilizado que fue VIM. VIM se ha configurado mediante la utilización de CVim [26] y un fichero `.vimrc` personaliado para la edición de código en C y utilizando las reglas de identación y formato de Linux. Otro factor de configuración del entorno es la personalización del fichero `.bash_rc` con una serie de comandos personalizados (alias) para la realización de las tareas más comunes (acceso SSH, compilación e instalación del kernel y user space, debugging de los logs, etc).

4.2 Dependencias de los paquetes software

El software del prototipo está compuesto finalmente por tres paquetes:

- El kernel que incluye los componentes del stack en el propio kernel.
- rinad que incluye los damons del stack corriendo en user space.
- librina como librería de referencia y framework para la interacción entre user-space y kernel.

La Fig. 4.2a muestra las dependencias entre los paquetes del stack, mientras que la Fig. 4.2b añade las dependencias externas para la compilación. Estas dependencias son comprobadas durante la compilación e instalación de los componentes mediante Autotools.



(a) Dependencias de ejecución (b) Dependencias de compilación

La compilación e instalación de estos paquetes se ha automatizado completamente mediante el uso de scripts que serán presentados en las siguientes secciones.

4.3 Entornos de compilación

4.3.1 Compilación del kernel

Para la compilación del kernel se utilizan los frameworks Kconfig[27] y Kbuild[28]. El código del stack presente en `linux/net/rina` contiene un fichero `Kconfig` y un fichero `Makefile` permitindo la carga dinámica de los módulos del stack al compilar el kernel. Estos módulos son:

Módulo	Descripción	Prerequisitos
rina-personality-default	La "personality" contiene todos los componentes básicos del stack permitiendo cargar distinas implementaciones del stack.	
normal-ipcp	IPC Process normal de RINA	rina-personality-default
shim-dummy	IPC Process shim Dummy	rina-personality-default
shim-eth-vlan	IPC Process shim Ethernet	rina-personality-default, rinarp
shim-tcp-udp	IPC Process shim TCP/UDP	rina-personality-default
rinarp	El módulo RINARP	arp286
arp826	El módulo ARP826	

Tabla 4.1: Dependencias entre módulos

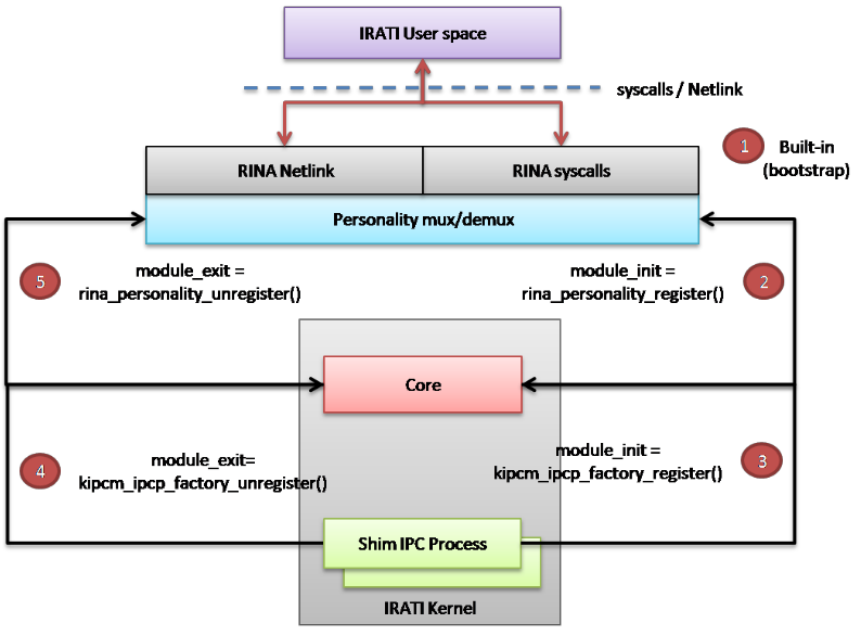


Figura 4.2: Carga/descarga de los módulos en el kernel

4.3.2 Compilación del user-space

Los componentes del prototipo en el espacio de usuario se compilan e instalan con el sistema de GNU autotools. De esta manera una simple llamada a `./configure && make && make install` compila e instala el software en el espacio de usuario. Autotools se compone por autoconf, automake, libtool y pkg-config.

4.4 Entornos de configuración e instalación

4.4.1 Instalación del kernel

La instalación del kernel se hace siguiendo el flujo típico de Linux. Desde el directorio `linux` del código fuente:

1. Copiar el fichero plantilla: `cp config-IRATI .config`.
2. Compilar kernel y módulos: `make bzImage modules`.
3. Instalar: `make modules_install install`.

4.4.2 Instalación en user-space

La instalación de rinad y librina se hace de la misma manera, donde `<PREFIX>` especifica el directorio de instalación:

1. Hacer el bootstrap del entorno: `./bootstrap`.
2. Configurar: `./configure -prefix=<PREFIX>`.
3. Compilar e instalar: `make && make install`.

4.4.3 Instaladores

El proceso de compilación e instalación está todavía mucho más simplificado mediante el uso de unos scripts utilitarios:

1. Compilar e instalar user-space: `./install-user-space-from-scratch <PREFIX>`.
2. Compilar e instalar el kernel: `./install-kernel-from-scratch`.
3. Compilar e instalar todo : `./install-from-scratch <PREFIX>`.
4. Desinstalar: `./uninstall-and-clean`

4.4.4 Carga del sistema

Una vez instalados todos los componentes el sistema debe ser reinciado para que la imagen de dicho kernel sea cargada. Una vez inicializado el sistema se deben cargar los módulos del stack y algunos utilitarios:

1. `modprobe rina-personality-default`.
2. `modprobe shim-eth-vlan`.
3. `modprobe shim-dummy`.
4. `modprobe normal-ipcp`.
5. `modprobe 8021q`.

Estos pasos pueden automatizarse modificando adecuadamente el dichero `etc/modules`. Si el kernel se compila en modo no modular (monolítico) entonces estos pasos no son necesarios ya que todos los módulos se compilan dentro de la imagen del kernel y se inicializan dinámicamente si es necesario.

4.5 Repositorio GIT

El repositorio central utilizado esta hospedado en Github (en estos momentos mediante un repositorio privado hasta que se alcance un grado de madurez suficiente como para poder abrirlo a la comunidad). El workflow diseñado para el trabajo sobre él se basa en las siguientes ramas:

master contiene el código que:

- Compila sin warnings al menos en la VM de desarrollo.
- Se ejecuta sin fallos mayores (kernel panics/segnmentation faults)

irati contiene el código que:

- Compila al menos en la VM de desarrollo.
- Se ejecuta con fallos menores (memory leaks, etc)

wip (work in progress) contiene el código que:

- Está en progreso y debe ser compartido por todos los desarrolladores.
- Puede no compilar ni ser ejecutable.
- Require testing y/o debugging.

Se uitilzó un sistema de tags para marcar las mayores releases en las ramas irati o master con el formato: v<Mayor><Menor><Micro>. Un tag que afecte al dígito de Mayor afectará a la rama master mientras que un tag que afecte al dígito Menor afecta a la rama irati. El issue tracker provisto por Github es utilizado dada su integración con el repositorio siendo posible abrir, comentar y cerrar issues directamente desde el commit de git.

4.5.1 Seguimiento de incidencias (Issues management)

Github además de repositorios GIT ofrece uns issue tracker sincronizado con el desarrollo del código[29] que permite gestionar los Software Problem Reports (SPR) como bugs, pedido de funcionalidades o definición de milestones. Además ofrece automatismo para hacer referencia a los SPR desde los propios commits del repositorio.

4.6 Lenguajes utilizados

Hay varias causas por las que se necesitó utilizar distintos lenguajes a la hora de programar el prototipo:

- El abasto del sistema va desde el user-space al kernel. Por este motivo los componentes del prototipo en el kernel obviamente debieron ser codificados en C.
- La existencia de un prototipo anterior menos avanzado realizado completamente en user-space en JAVA propició el desarrollo de los componentes del user-space (daemons) en JAVA con tal de reaprovechar la mayor cantidad de código posible, aunque hiciera falta cambiar y adaptar el código existente.
- Dado que las librerías librina debían, además de ofrecer la funcionalidad de RINA a user-space, abstraer la comunicación con el kernel, se eligió realizarlas en C++ para poder contar con la orientación a objetos y la fácil integración de C y C++. Por otro lado, con la herramienta SWIG es relativamente fácil hacer los bindings entre los componentes en JAVA y C++.

5. Testing

Esta sección presenta la metodología de los tests realizados durante el desarrollo del prototipo para su validación. Se han considerados tests unitarios, de regresión y funcionales. Como se verá en las siguientes secciones, los tests unitarios y de regresión no han seguido una metodología estricta sinó que se han aprovechado de la arquitectura modular del kernel. Para los tests funcionales, en cambio, sí que se ha seguido una metodología más clásica.

5.1 Entornos de test

Dada la naturaleza y los objetivos de los tests que se presentan en este documento, se diseñaron dos entornos de testing. Por un lado, uno local y autocontenido que se puede ejecutar en un único PC de forma de poder ganar agilidad en el proceso de testing. Este escenario puede verse en la Fig. 5.1, la cual describe gráficamente el entorno basado en dos VMs con las mismas características que la de desarrollo presentada en la Sección 4.1, pero con dos interfaces de red, una para proveer acceso a la VM y la otra para instalarle una interfaz virtual con VLAN para soportar la shim DIF sobre Ethernet que se testeará.

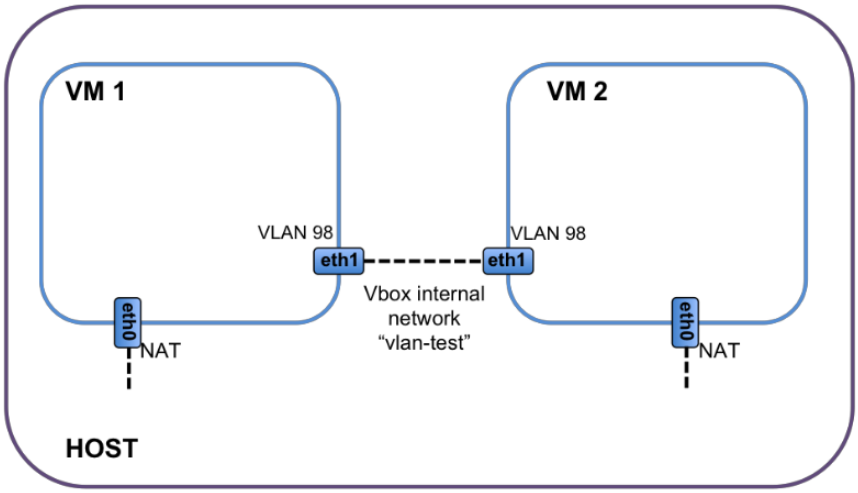


Figura 5.1: Escenario de test basado en VirtualBox

Por otro lado, el segundo escenario es el testbed OFELIA [30] que se utilizó únicamente para los tests funcionales. Dado que la puesta a punto de este entorno requiere alguna acciones de configuración y el uso de herramienta propias de OFELIA, la validación de este escenario se explicará en la Anexo B.

5.2 La aplicación echo

La aplicación echo ha sido diseñada para testar el stack RINA y especialmente la Shim-DIF. RINAband [31], es un programa JAVA para testear el bandwith en un flow RINA. El problema con esta aplicación es que espera que las DIFs soporten varios flows entre el mismo par de IPC Processes, lo que no es posible en la Shim-DIF por las limitaciones ya comentadas. Por este motivo, se desarrolló una versión más ligera de RINAband bautizada como "echo", cuyo comportamiento puede resumirse de la siguiente manera:

- La aplicación puede operar en modo servidor, modo en que soporta tests concurrentes, o modo cliente (echo-server, echo-client respectivamente).
- Cuando el cliente de la aplicación se ejecuta se registra en una DIF, reserva un flow al servidor y espera por el resultado. Si es satisfactorio, negocia los parámetros del test con el servidor (número y tamaño de SDUs a ser enviadas) y comienza la transferencia de datos. Cuando el servidor recibe una SDU, hace el echo de la misma SDU al cliente-
- El test se completa cuando todas las SDUs fueron enviadas y recibidas o cuando transcurre un cierto valor de tiempo sin recibir una SDU por parte de alguno de los extremos.
- Tanto el cliente como el servidor reportan estadísticas de la transferencia realizada como el número de SDUs recibidas y el tiempo abarcado.

  Si bien medir estos valores desde la perspectiva de la aplicación no es un dato del todo fiable para medir el rendimiento del stack, da una aproximación suficiente para este punto del desarrollo.

5.3 Tests unitarios y de regresión

Como se ha comentado en la introducción, no se ha seguido una metodología de tests unitarios realmente marcada en el sentido de que no existe una batería de tests independiente del código. La estrategia de debugging y testing difiere para el caso de los componentes en el kernel o en user-space. Dado que para el caso del kernel se ha utilizado una cierta práctica reglada, se ahondará en los detalles para este entorno, así como también para los elementos pertenecientes al framework de comunicación entre espacios.

El prototipo sigue una arquitectura modular, en coherencia con la arquitectura de Linux. Por esta razón, se ha utilizado la capacidad de los módulos para el desarrollo de las funcionalidades de RINA y de los tests unitarios y de regresión. La estructura básica del código de un nuevo módulo en el kernel es la siguiente:

Cuadro 5.1: Estructura básica del código de un módulo en el kernel

```
1  #define MODULE
2  #define LINUX
3  #define __KERNEL__
4
5  #include <linux/module.h> // Needed by all modules
6  #include <linux/kernel.h> // Needed for KERN_ALERT
7  #include <linux/init.h> // Needed for the macros
8
9
10 static int hello_2_init(void)
11 {
12    printk(KERN_ALERT "Hello, world 2\n");
13    return 0;
14 }
15
16
17 static void hello_2_exit(void)
18 {
19    printk(KERN_ALERT "Goodbye, world 2\n");
20 }
21
22
23 module_init(hello_2_init);
24 module_exit(hello_2_exit);
```

Los macros `module_init` y `module_exit` definen las funciones que el módulo ejecutará al cargarse y descargarse del sistema respectivamente. Con esto en mente, la función

`_init` se utilizó para poner código temporal que creará las estructuras necesarias para alimentar el resto de funciones del módulo y comprobar su salida. Básicamente la batería de tests unitarios atacando a las funciones del módulo. En el kernel las herramientas para debuggear no alcanzan mucho más que la utilización de la función `printk` (la variante en el kernel de `printf` y la posterior visualización de las trazas en el log del sistema. Por esto mismo, así fue la estrategia utilizada, con la salvedad de la creación de un conjunto de funciones (macros) utilitarias en el fichero `logs.h` donde se definen distintos niveles de mensajes envolviendo la llamada a `printk` como si de un loging de lenguajes de más alto nivel se tratara. El código puede verse en Cuadro 5.2, donde se definen `LOG_DBG`, `LOG_ERR`, etc.

Cuadro 5.2: Logging en el prototipo. logs.h

```
1  #ifndef RINA_LOGS_H
2  #define RINA_LOGS_H
3
4  #ifndef RINA_PREFIX
5  #error You must define RINA_PREFIX before including this file
6  #endif
7
8  #include <linux/kernel.h>
9
10 /* The global logs prefix */
11 #define __GPFX "rina-"
12
13 #ifdef CONFIG_RINA_UNFILTERED_LOGS
14 #define __LOG(PFX, LVL, FMT, ARGS...)  \
15         do { printk(KERN_NOTICE __GPFX PFX ": " FMT "\n", ##ARGS); } while (0)
16 #else
17 #define __LOG(PFX, LVL, FMT, ARGS...)  \
18         do { printk(LVL __GPFX PFX ": " FMT "\n", ##ARGS); } while (0)
19 #endif
20
21 /* Sorted by "urgency" (high to low) */
22 #define LOG_EMERG(FMT, ARGS...) __LOG(RINA_PREFIX, KERN_EMERG, FMT, ##ARGS)
23 #define LOG_ALERT(FMT, ARGS...) __LOG(RINA_PREFIX, KERN_ALERT, FMT, ##ARGS)
24 #define LOG_CRIT(FMT, ARGS...) __LOG(RINA_PREFIX, KERN_CRIT, FMT, ##ARGS)
25 #define LOG_ERR(FMT, ARGS...) __LOG(RINA_PREFIX, KERN_ERR, FMT, ##ARGS)
26 #define LOG_WARN(FMT, ARGS...) __LOG(RINA_PREFIX, KERN_WARNING, FMT, ##ARGS)
27 #define LOG_NOTE(FMT, ARGS...) __LOG(RINA_PREFIX, KERN_NOTICE, FMT, ##ARGS)
28 #define LOG_INFO(FMT, ARGS...) __LOG(RINA_PREFIX, KERN_INFO, FMT, ##ARGS)
29 #define LOG_DBG(FMT, ARGS...)  __LOG(RINA_PREFIX, KERN_DEBUG, FMT, ##ARGS)
30
31 #ifdef RINA_DEBUG_HEARTBEATS
32 #define LOG_HBEAT LOG_DBG("I'm in %s (%s:%d)", \
33                           __FUNCTION__, __FILE__, __LINE__)
34 #else
35 #define LOG_HBEAT
36 #endif
37
38 #define LOG_OBSOLETE_FUNC LOG_ERR("Function %s is obsolete and it will be " \
39                                   "removed soon, do not use", \
40                                   __FUNCTION__)
41 #define LOG_MISSING LOG_ERR("Missing code in %s:%d", __FILE__, __LINE__)
42 #define LOG_UNSUPPORTED LOG_WARN("Unsupported feature hit in %s:%d", \
43                                  __FILE__, __LINE__)
44
45 #endif
```

Complementariamente, la función `_exit` se utilizaba para destruir todas aquellas estructuras creadas para el testing.

La otra serie de tests unitarios realizados fue en la capa de Netlink y de llamadas a sistema, tanto del kernel como de user-space. Utilizando las herramientas de loging antes descritas, todos los parsers y crafters para cada uno de los mensajes se fue ejecutando siem-

pre que se añadía un tipo de mensaje nuevo.

Los tests de regresión siguieron todos la misma estrategia, pero a diferencia de los tests unitarios, los tests de regresión engloban un conjunto de tests unitarios con el fin de comprobar una funcionalidad mayor. Por otro lado, si bien los tests unitarios se ejecutan en el momento en que la funcionalidad específica es modificada, los tests de regresión se ejecutan siempre que alguna función importante del código es modificada. Por este motivo, los tests de regresión son activables desde el menú de configuración de compilación del kernel (Kconfig framework), siendo posible recompilar el kernel y ejecutar estos tests en el momento de cargar cada módulo y comprobar los resultados con el comando `dmesg`. La Fig. 5.2 muestra una de estas opciones.

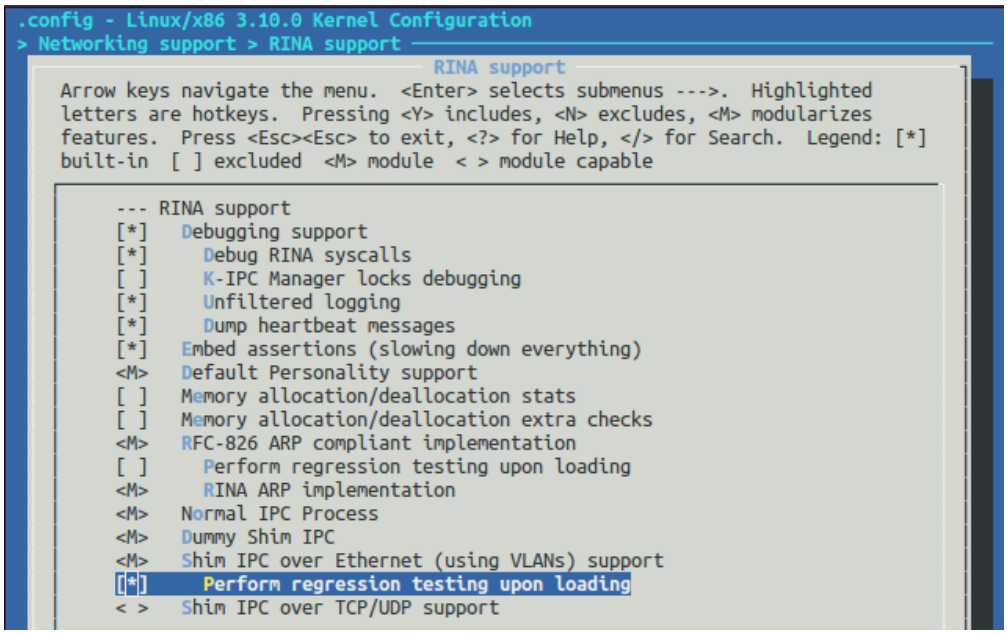


Figura 5.2: Tests de regresión configurables con Kconfig

5.4 Test funcionales en el entorno local

La Tabla 5.1 muestra el modelo utilizado para cada uno de los tests de la bateria. Cada una de las tarjetas contiene un conjunto de campos para identificar y describir los tests:

- **Etiqueta:** valor único que representa a la tarjeta.
- **Nombre:** Breve identificación del test.
- **Objetivos:** expone los resultados esperados en el test.
- **Tarjetas relacionadas:** lista de las tarjetas relacionadas en términos de prerequisitos o pasos comunes importados.
- **Topología y dispositivo bajo test (Device Under Test (DUT)):** describe la topología y la configuración utilizada para el test.
- **Descripción:** del test paso a paso.
- **Comentarios adicionales:** reporta las pre y postcondiciones del test no incluidas en la "Descripción".
- **Estado:** provee la confirmación del estado que identifica si el test ha sido pasado o no.

Etiqueta	IRATI-<NAME>-<NUMBER>
Nombre	<Nombre descriptivo de la tarjeta>
Objetivo	<Objetivos del test>
Tarjetas relacionadas	<Lista opcional de tarjetas relacionadas>
DUT	<Diagrama de la topología y detalles del entorno de test>

Paso	**Descripción**
1	
2	
...	
N	

Comentarios adicionales
Notas, aclaraciones, etc...

Estado del test
PASADO/NO PASADO

Tabla 5.1: Tarjeta de test

Esta metodología de tarjetas se utilizó para los primeros tests funcionales del stack a realizarse el el escenario de la Fig. 5.1. Se utilizaron 19 tarjetas divididas en 4 áreas principales. Cada tarjeta verifica un conjunto de funcionalidades expresadas como los objetivos a ser conseguidos por el test. El modelo elegido para las tarjetas puede verse en Tabla 5.2. El Anexo A aúna las 19 tarjetas completadas durante la realización de los distintos tests funcionales en el escenario de VirtualBox.

Tests básicos		
Nr.	**Etiqueta de la tarjeta**	**Nombre de la tarjeta**
1	IRATI-BASE-01	El stack se puede cargar
2	IRATI-BASE-02	El stack se puede descargar
3	IRATI-BASE-03	El stack se puede cargar y descargar múltiples veces
4	IRATI-BASE-04	El IPC Manager daemon se inicializa correctamente
Tests del setup		
Nr.	**Etiqueta de la tarjeta**	**Nombre de la tarjeta**
5	IRATI-CREATE-01	Creación de un IPC Process del tipo shim-eth-vlan usando un fichero de configuración
6	IRATI-CREATE-02	Creación de múltiples IPC Processes del tipo shim-eth-vlan usando un fichero de configuración
7	IRATI-CREATE-03	Creación de un IPC Process del tipo shim-eth-vlan usando comandos de la CLI
8	IRATI-DESTROY-01	Destrucción de un IPC Process
9	IRATI-DESTROY-02	Destrucción de múltiples IPC Processes

10	IRATI-ASSIGN-01	Asignar un IPC Process shim-eth-vlan a una DIF usando un fichero de configuración
11	IRATI-ASSIGN-02	Asignar un IPC Process shim-eth-vlan a una DIF usando la CLI
12	IRATI-DESTROY-03	Destruir un IPC process shim-eth-vlan asignado a una DIF
13	IRATI-REGISTER-01	Registrar la aplicación echo-server a una shim DIF
14	IRATI-REGISTER-02	Registrar dos aplicaciones a una IPC Process shim-eth-vlan
Tests de transferencia de datos		
Nr.	**Etiqueta de la tarjeta**	**Nombre de la tarjeta**
15	IRATI-CLIENT-01	Ejecutar la aplicación echo-client (una vez, 100 SDUs)
16	IRATI-CLIENT-02	Ejecutar la aplicación echo-client (dos veces, 100 SDUs)
17	IRATI-CLIENT-03	Ejecutar la aplicación echo-client (100 veces, 100 SDUs)
Test de cleanup		
Nr.	**Etiqueta de la tarjeta**	**Nombre de la tarjeta**
18	IRATI-KILL-01	Detener la aplicación echo-server
19	IRATI-KILL-02	Detener el IPC Manager daemon

Tabla 5.2: Conjunto de tests funcionales para el escenario de VirtualBox

5.5 Test funcional en OFELIA

5.5.1 Caso de uso

El caso de uso estudiado para realizar los tests funcionales sobre el stack producido, con especial atención sobre la Shim Ethernet, en el testbed de OFELIA puede ser observado en la Fig. 5.3.

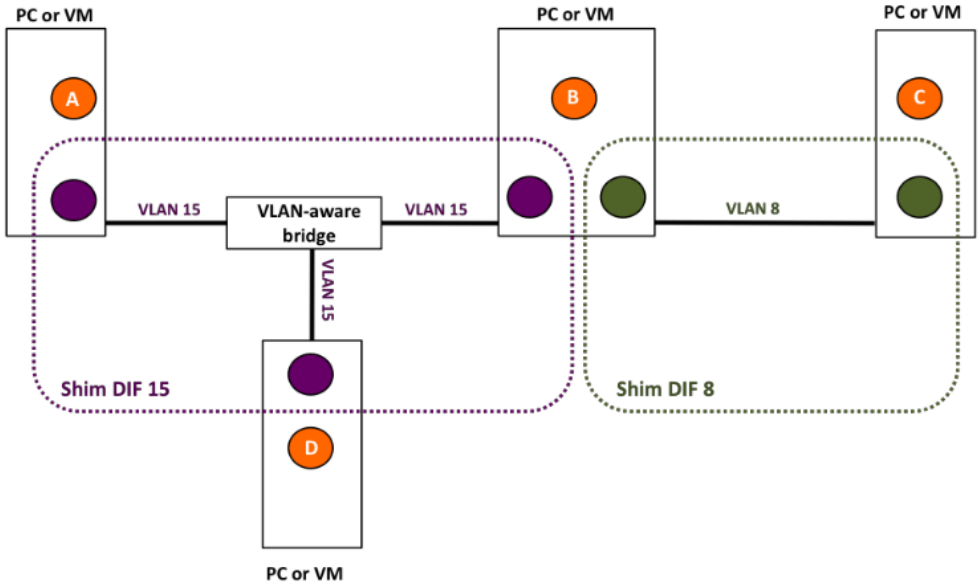


Figura 5.3: Caso de uso para la Shim Ethernet en OFELIA

Dentro de las diferentes instalaciones que ofrece el testbed de OFELIA se utilizó la isla de i2CAT. La abstracción de la Fig. 5.3 puede verse mapeado a los recursos físicos de la isla

en Fig. 5.4. Los cuatro nodos son VMs corriendo el kernel modificado con el stack producido. Los puentes con soporte para VLANs de la imagen corresponden con los switches OpenFlow [32] del testbed.

Los recursos OpenFlow del testbed se particionan mediante la segmentación del FlowSpace disponible. El FlowSpace es un espacio vectorial formado por los posibles valores de las 16 cabeceras que OpenFlow 1.0 utiliza para identificar el tráfico. Estos campos corresponde con parámetros que van desde la L2 a la L4.

OFELIA Control Framework (OCF) [33] es el programa de control del testbed que mediante una interfaz web permite la reserva de recursos a los experimentadores, lo que se llama "slice". Dentro de esta slice, el OCF reserva un subconjunto del FlowSpace sobre unos puertos y switches específicos y lo provee a un usuario. Este subset siempre contiene un tag VLAN mediante el cual se aísla el tráfico de un usuario o de otro. Este mecanismo es ideal para el use case que se presenta, ya que la Shim Ethernet utiliza VLANs como el nombre de la DIF, como se describe en las especificaciones [11].

En conclusión, después de identificar los recursos físicos necesarios para llevar a cabo el use case, se obtiene una relación directa entre las DIFs Shim Ethernet, las slices de OFELIA y los tags de VLANs. Las VLANs que fueron proporcionadas para soportar las dos Shim DIFs son la 300 y la 301. Además de las VMs que se pueden ver en Fig. 5.4, se creó una adicional para correr el controlador OpenFlow NOX [34] en modo MAC learning switch.

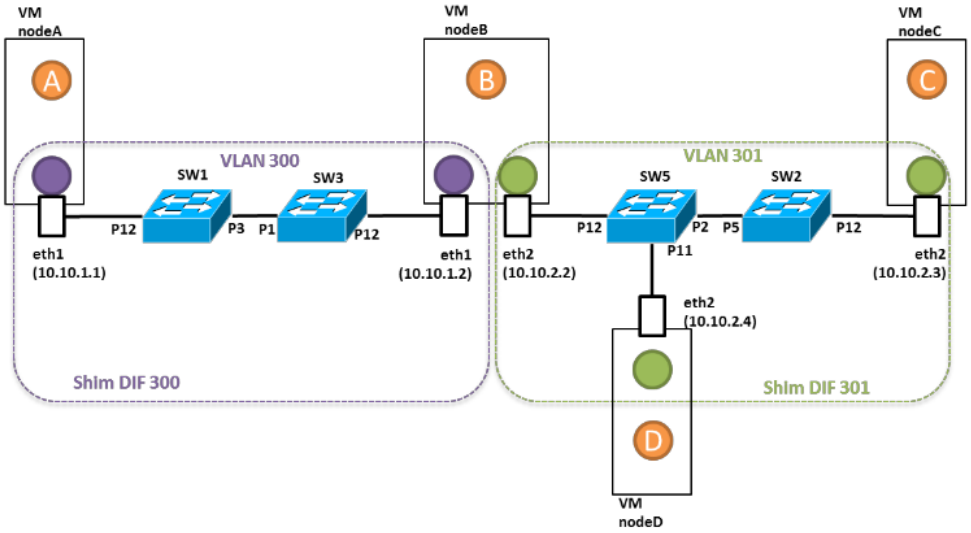


Figura 5.4: Mapeo del use case con los recursos físicos de OFELIA

5.5.2 Experimentos de validación

Dentro del caso de uso para OFELIA se han diseñado tres experimentos con especial foco en la shim DIF sobre Ethernet: i) un experimento para validar el setup un flow simple sobre una Shim-DIF; ii) un experimento que establezca varios flows concurrentes entre los mismos dos hosts (debido a las limitaciones de la Shim-DIF); y iii) un experimento corriendo UDP/IP concurrentemente con la implementación del stack para comprobar interferencias y la interoperabilidad al enviar y recibir tráfico RINA e IP sobre la misma VLAN.

5.5.2.1 Test con un único flow por DIF

Este test ejecuta las aplicaciones echo-server/echo-client para testear la conexión básica con un único flow por DIF. Sus objetivos son validar :

- Que las diferentes fases de la DIF (enrolment, reserva de flow, y transferencia de datos) se completan satisfactoriamente.
- Que la implementación de la Sim DIF sobre Ethernet (802.1Q) no invalida paquetes en referencia a los estándares de Ethernet y VLAN involuntariamente (comprobable si los paquetes atraviesan los switches).

La tarjeta del test puede verse en Anexo A.Tabla A.20.

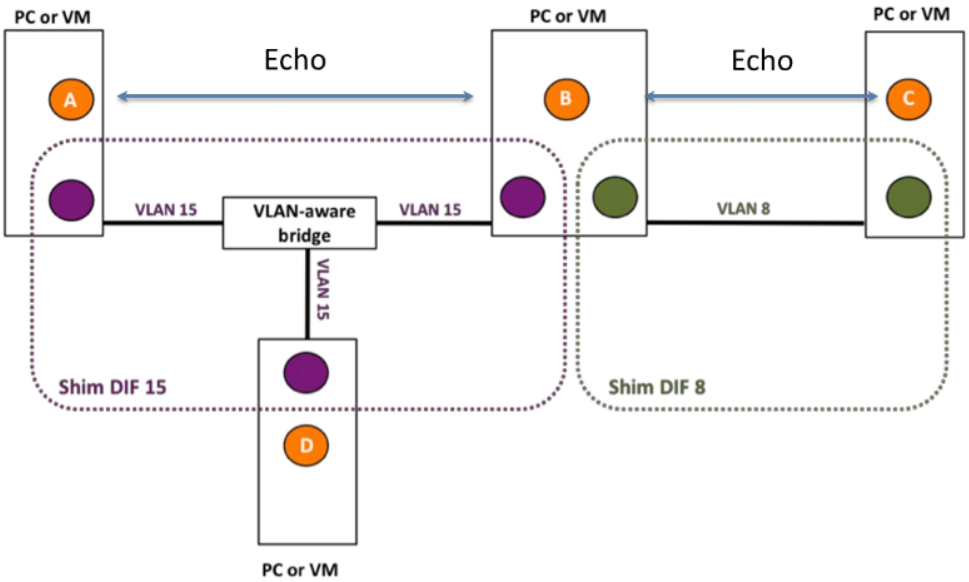


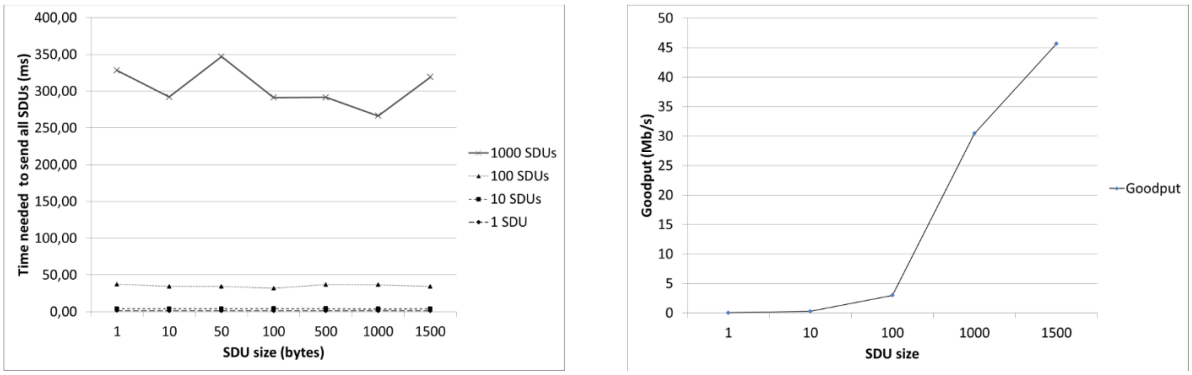
Figura 5.5: Test echo con un flow simple

5.5.2.1.1 Resultados

El test fue ejecutado considerando dos variables: número de SDUs ha ser enviadas (SDU_COUNT) y tamaño de las SDUs (SDU_SIZE). Un script Bash fue desarrollado con tal de automatizar las distintas ejecuciones modificando cada vez el valor de las variables. Cada combinación fue ejecutada tres veces y el mínimo valor del tiempo necesitado para enviar y recibir todas las SDUs fue considerado como el resultado. Fig. 5.6a muestra el tiempo necesito para enviar distintas cantidades de SDUs dependiendo de su tamaño, mientras que Fig. 5.6b replejan el throughput en Mbps obtenido al enviar 1000 SDUs de distintos tamaños.

5.5.2.1.2 Conclusiones

Como primera conclusión se pudo ver ver que el tiempo requerido para enviar una cierta cantidad de SDUs no depende del tamaño de éstas. Esto es debido a que el factor predominante no es la capacidad de los links sinó el tiempo necesario para que el stack pueda procesar una SDU completamente. Es meritorio aclarar que en este punto de la implementación no se lleva a cabo ninguna operación dependiente del tamaño de la SDU como cigrado,



(a) Tiempo de transferencia

(b) Throughput obtenido

Figura 5.6: Resultados de IRATI-SINGLEFLOW-01

reensabmblaje, etc. Cuando estas operaciones sean implementadas el tiempo necesario para procesar una SDU se incrementará pero la conclusión seguiŕá siendo válida ya que el tiempo de transmisión será aún menor en comparación.

Por otro lado, como se puede ver en Fig. 5.6b, el throughput de cada experimento obviamente se incrementa al aumentar el tamaño de la SDU. Dado que el tiempo de procesado no depende del tamaño, cuanto mayor sean las SDUs mayor será el throughput.

En pruebas locales se observaron mejores tiempos de transferencia que en OFELIA. Esto podría ser resultante de al menos dos motivos: i) el escenario de OFELIA es más complejo y los paquetes deben atravesar 2 switches y 8 interfaces entre físicas y virtuales para llegar de un host a otro; ii) por otro lado, un test con iperf previo mostró una capacidad de los links de aproximadamente 75 Mbps, quizá por problemas de congestionamiento o múltiples experimentos concurrentes corriendo en el testbed. Con lo cual, el resultado obtenido de más de 45Mbps supone alrededor de un 60 % del ancho de banda disponible.

Teniendo todo esto en cuenta, se observó que el stack fue capaz de completar todas las fases del ciclo de vida de un flow y transmitir SDUs sobre VLANs con un rendimiento razonable para una primera implementación.

5.5.2.2 Test con múltiples flows por DIF

Este test ejecuta las aplicaciones echo-server/echo-client para testear la conexión básica con múltiples flows por DIF. Sus objetivos son:

- Validar que las fases de reserva de flow y transferencia de datos se completan satisfactoriamente en un host ante condiciones altas de carga.
- Validar que la operación de transferencia de datos permanece estable durante el tiempo (5 minutos es el tiempo del test de resistencia).

La tarjeta del test puede verse en Anexo A.Tabla A.21.

5.5.2.2.1 Resultados

El experimento se ejecutó enviando siempre 1000 SDUs desde cada cliente pero variando el tamaño de las SDUs. Se utilizó el mismo script Bash que en el caso anterior debidamente

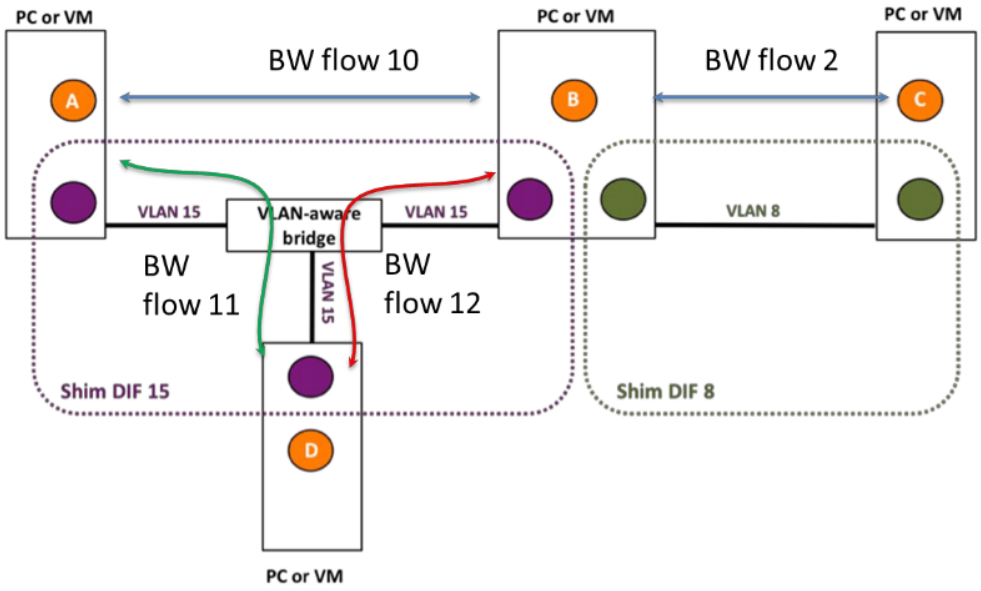


Figura 5.7: Test echo con múltiples flows

configurado. Cada experimento fue ejecutado tres veces para cada tamaño de SDU. En este caso, el valor medio de tiempo obtenido fue considerado como resultado dado que se observó que cuando dos clientes se conectaban al mismo echo-server, el mínimo throughput de uno estaba correlado con el máximo del otro debido a que el echo-server en ese momento servía más a un cliente que al otro. Fig. 5.8a, Fig. 5.8b y Fig. 5.8c muestran el tiempo necesario para enviar las 1000 SDUs dependiendo del tamaño en los nodos A, B y C respectivamente. Por otro lado, Fig. 5.8d, Fig. 5.8e y Fig. 5.8f muestran el throughput en Mbps al enviar 1000 SDUs de distintos tamaños en los nodos A, B y C respectivamente.

5.5.2.2.2 Conclusiones

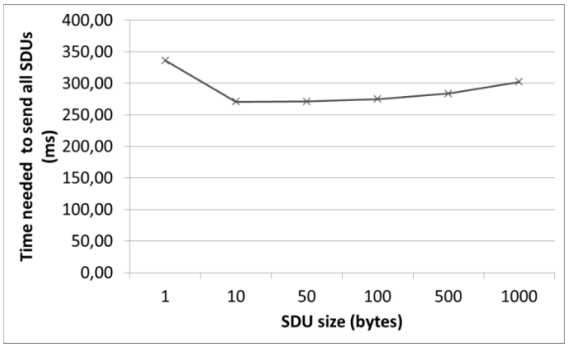
Los resultados muestran que los nodos con un echo-client compartiendo un echo-server (clientes en B y C y server en D) obtienen aproximadamente el doble de tiempo y mitad del troughput que el nodo que no comparte al servidor (cliente en A y server en B). Esto es debido a que la capacidad de los links seguramente no fue suficiente para soportar el doble de tráfico, además de la propia limitación del servidor que debe atender dos clientes. A pesar de este hecho, el stack se comportó correctamente ante la presencia de múltiples flows.

5.5.2.3 Test con aplicaciones RINA e IP concurrentes

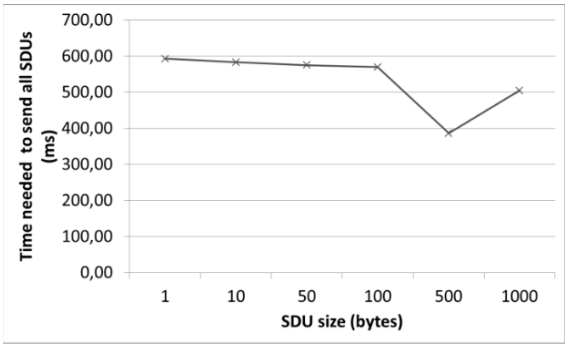
Este test ejecuta las aplicaciones echo-server/echo-client e iperf a la vez entre los mismos hosts y sobre la misma VLAN para testear la estabilidad del kernel cuando utiliza los stacks UDP/IP y RINA y, a la vez, comprobar como interactúan. Sus objetivos son:

- Validar la interoperabilidad de IP y RINA.
- Validar que la operación de transferencia de datos permanece estable durante el tiempo (más de 1 minuto) cuando múltiples flows corren simultáneamente en ambos stacks.
- Investigar como ambos stacks interactúan con el mismo device driver de la tarjeta Ethernet.

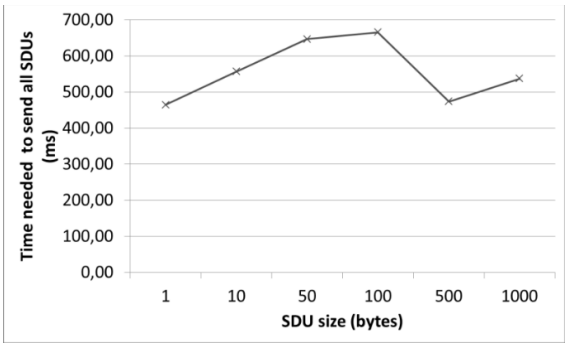
La tarjeta del test puede verse en Anexo A.Tabla A.22.



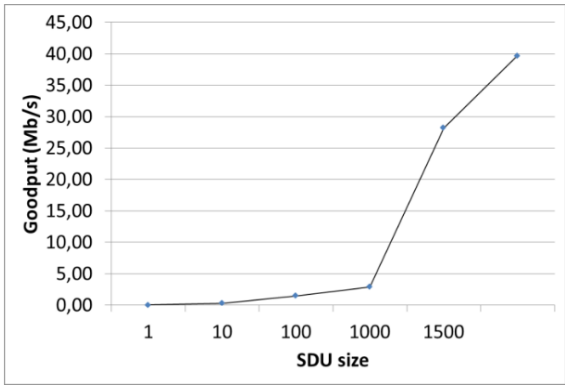
(a) Tiempo de transferencia en A



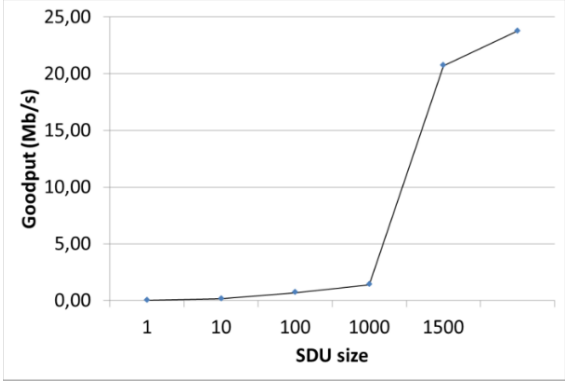
(b) Tiempo de transferencia en B



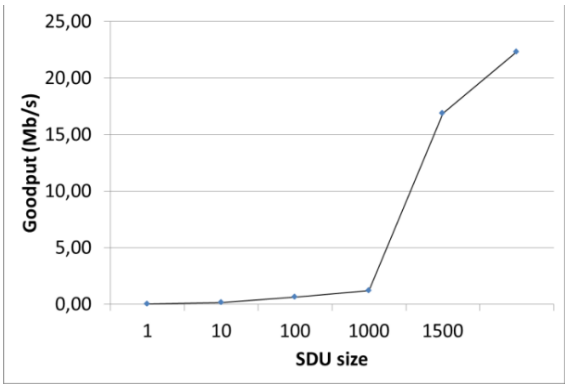
(c) Tiempo de transferencia en C



(d) Throughput de A



(e) Throughput de B



(f) Throughput de C

Figura 5.8: Resultados de IRATI-MULTIFLOW-01

5.5.2.3.1 Resultados

El despliegue de este experimento fue el mismo que en Sección 5.5.2.1 pero ejecutando iperf sobre la misma interfaz virtual que soporta a la shim DIF. Al mismo tiempo que se producía el intercambio de SDUs entre el echo-client y el echo-server, iperf enviaba tráfico UDP entre los mismos hosts y las mismas interfaces. Fig. 5.10 muestra el throughput obtenido al enviar 1000 SDUs de distintos tamaños mientras la Tabla 5.3 muestra los resultados obtenidos por iperf.

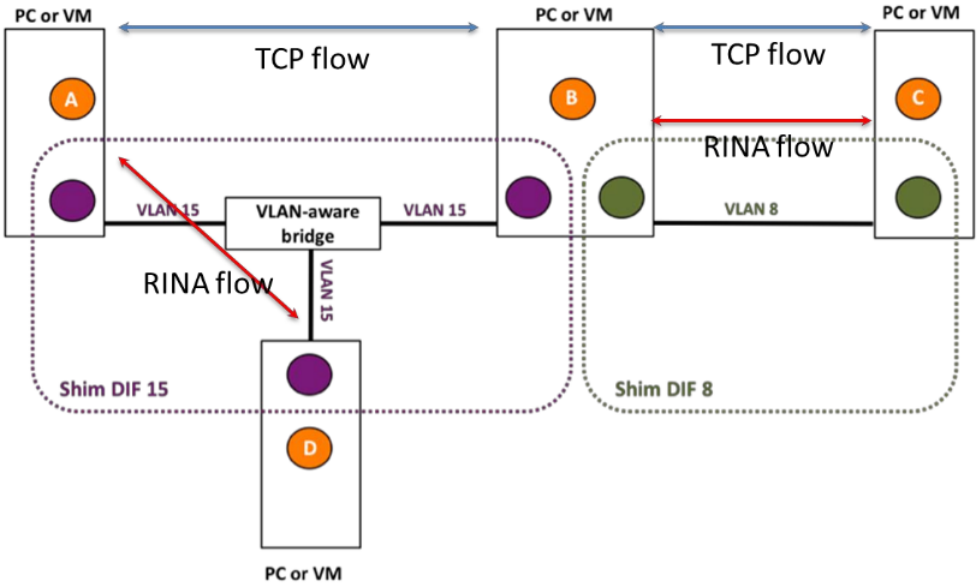


Figura 5.9: Test con IP y RINA corriendo simultáneamente

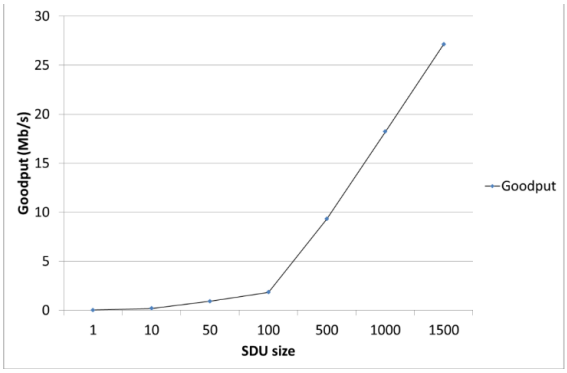


Figura 5.10: Throughput para distintos tamaños de SDU en el nodo B ejecutando iperf

Intervalo de tiempo	Nº datagramas	Datos enviados	Ancho de banda
90s	554915	778 MB	75,5 Mbps

Tabla 5.3: Resultados de iperf con UDP en nodo B

5.5.2.3.2 Conclusiones

Los tests mostraron que los sistemas implicados pudieron soportar UDP y RINA a la vez, siendo posible la coexistencia de ambos stacks en la misma VLAN. Sin embargo, en un escenario más cercano a la realidad, sería necesario asegurar unos mecanismos de priorización y/o aislamiento para asegurar la compartición de los recursos. Un ejemplo de este escenario podría ser el la separación de tráfico IP y RINA mediante VLANs.

6. Resultados, conclusiones y trabajo futuro

Los componentes software necesarios para un primer prototipo de RINA en Linux han sido diseñados, desarrollados, integrados y han sido sometidos a una batería de tests funcionales.

Las actividades de diseño y desarrollo prograsaron en su mayor medida como había sido planificado [35] (aunque nuevos requerimientos afectaron el normal desempeño). Principalmente, se hace referencia a la implementación de ARP presente en Linux. Su difícil integración con la shim Ethernet, uno de los objetivos principales del prototipo, propició la necesidad de desarrollar una versión del protocolo ARP que cumpla con el estándar RFC-826 [17]. Este hecho junto con el ambicioso plan de tareas provocó la imposibilidad de realizar un testing profundo de las funcionalidades del IPC process normal. Éstas se han diseñado y desarrollado pero no testeado de forma integral, motivo por el cual no se presentan tests de esta capa. Aún así, los requerimientos fueron cumplidos y los cambios en la arquitectura durante el proceso de diseño y desarrollo dieron respuesta a los problemas que los motivaron. El prototipo actual implementa los siguientes componentes para un sistema basado en Linux:

- Los componentens básicos (el core)
- Un framework de utilidades basado en objetos del kernel
- Distintos tipos de procesos IPC:
  - normal (a falta de DTCP)
  - dummy
  - shim Ethernet
- Claras interfaces entre componentes.
- Claras interfaces entre user-space y kernel implementadas con procfs/sysfs, llamadas a sistema y sockets Netlink.

El software del prototipo RINA sobre Ethernet actual, supone un esqueleto fiable para continuar su desarrollo, ya que cuenta con un framework de desarrollo, compilación e instalación robusto y que utiliza herramientas extendidas y conocidas. Este factor es muy importante de cara a promover el desarrollo de RINA.

A nivel funcional, el stack cumple el objetivo principal de soportar la creación de DIFs sobre Ethernet con niveles de servicio similares a UDP. En este aspecto también se cumple el segundo objetivo de mejora del modelo de referencia de la arquitectura y de sus especificaciones, añadiendo las de la shim Ethernet.

En cuanto a la validación y el tercero de los objetivos, se han cumplido los requisitos de funcionalidad y estabilidad para hacer posible la experimentación. En este aspecto, una serie de tests se han diseñado y ejecutado satisfactoriamente en dos ambientes de experimentación. Por un lado, utilizando herramientas de virtualización locales como VirtualBox, y por otro lado, en una plataforma con hardware real como el testbed OFELIA, que otorga al stack producido mayor confiabilidad y robustez.

El único requisito que no ha sido completamente testeado es el de interoperabilidad con el otro prototipo existente realizado en JAVA sobre TCP/IP y UDP/IP. Aún así dado que

parte del código de user-space ha sido importado de este prototipo, se espera una compatibilidad conseguible sin mucho coste. Aún así, este punto pasa a ser parte del trabajo a futuro que se describe a continuación.

El trabajo a futuro está claramente marcado por los objetivos del proyecto IRATI dentro del cual se enmarca este PFM. Dentro del subconjunto relacionado con el desarrollo se identificaron claros puntos de mejora o de necesidad de nuevas funcionalidades, además de las ya planificadas. Éstas son:

- Rediseño incremental de los flujos de entrada y salida de DUs con tal de obtener una arquitectura distribuida que no ocasione cuellos de botella causados por la actual centralización en el KIPCM y KFA.
- Reducir el mapa de memoria necesario en el kernel mediante técnicas como optimización de las estructuras, optimizaciones del uso de memoria con Copy-On-Write (COW), caching, etc.
- Mejorar la integración de sysfs con tal de exponer los objetos del kernel adecuadamente.
- Mejorar el rendimiento de los componentes de user-space traduciéndolos de Java a C++.
- Mejorar la compatibilidad del stack con POSIX de cara a ganar portabilidad hacia otros sistemas POSIX como FreeBSD.
- Comprobar la interoperabilidad con otros prototipos.

Como conclusión final, se ha demostrado la validez de la arquitectura con una primera implementación que si bien no ofrece la totalidad de las funcionalidades teóricas, hace una primera incursión en ellas dejando señales positivas de las posibilidades que ofrece RINA y de la posibilidad real de implementarla. En este camino, después del trabajo realizado se tienen dos soportes básicos para el trabajo futuro: un sustrato desde donde comenzar y unas directrices claras de cómo continuar, objetivos del PFM.

Del lado de la experiencia personal, el PFM ha sido la culminación de los estudios del máster. La temática elegida ha incorporado muchos de los temas estudiados, entre los cuales se pueden citar administración de sistemas, desarrollo comunitario y el entorno del sofware libre. Por estas razones creo que la elección fue acertada y desde ya la experiencia y formación obtenida han mejorado mi capacitación y conocimientos.

Apéndice A. Tarjetas de test

A.1 IRATI-BASE-01

Etiqueta	IRATI-BASE-01
Nombre	El stack se puede cargar
Objetivo	Comprobar la correcta inicialización de los componentes del stack en el kernel.
Tarjetas relacionadas	-
DUT	Una única VM como especifica la Sección 4.1 y el módulo 8021q cargado.

Paso	**Descripción**
1	Encender la VM y logearse como `root`

2	Cargar los módulos bajo test con los siguientes comandos:
	`modprobe rina-personality-default`
	`modprobe 8021q`
	`modprobe arp826`
	`modprobe rinarp`
	`modprobe normal-ipcp`
	`modprobe shim-dummy`
	`modprobe shim-eth-vlan`
	Cada comando debe completarse satisfactoriamente, sin reportar errores tanto en la CLI como en los logs del kernel

Comentarios adicionales

Estado del test
PASADO

Tabla A.1: Tarjeta de test IRATI-BASE-01

A.2 IRATI-BASE-02

Etiqueta	IRATI-BASE-02
Nombre	El stack se puede descargar
Objetivo	Comprobar la correcta descarga de los componentes del stack en el kernel.
Tarjetas relacionadas	IRATI-BASE-01
DUT	Una única VM como especifica la Sección 4.1 y el módulo 8021q cargado.

Paso	**Descripción**

1	Encender la VM y logearse como `root`

2	Descargar los módulos bajo test con los siguientes comandos:
	
	`rmprobe shim-eth-vlan`
	`rmdprobe rinarp`
	`rmdprobe arp826`
	`rmprobe 8021q`
	`rmprobe normal-ipcp`
	`rmdprobe shim-dummy`
	`rmprobe rina-personality-default`
	
	Cada comando debe completarse satisfactoriamente, sin reportar errores tanto en la CLI como en los logs del kernel

Comentarios adicionales

Estado del test
PASADO

Tabla A.2: Tarjeta de test IRATI-BASE-02

A.3 IRATI-BASE-03

Etiqueta	IRATI-BASE-03
Nombre	El stack se puede cargar y descargar múltiples veces
Objetivo	Comprobar la correcta descarga de los componentes del stack en el kernel.
Tarjetas relacionadas	IRATI-BASE-01, IRATI-BASE-02
DUT	Una única VM como especifica la Sección 4.1 y el módulo 8021q cargado.

Paso	**Descripción**
1	Encender la VM y logearse como `root`
2	Realizar los pasos descritos en IRATI-BASE-01 (debe PASAR)
3	Realizar los pasos descritos en IRATI-BASE-02 (debe PASAR)
4	Realizar los pasos descritos en IRATI-BASE-01 (debe PASAR)

Comentarios adicionales

Estado del test
PASADO

Tabla A.3: Tarjeta de test IRATI-BASE-03

A.4 IRATI-BASE-04

Etiqueta	IRATI-BASE-04
Nombre	El IPC Manager dameon se inicializa
Objetivo	Comprobar la correcta inicialización del IPC Manager.
Tarjetas relacionadas	IRATI-BASE-01
DUT	Una única VM como especifica la Sección 4.1 y el módulo 8021q cargado.

Paso	**Descripción**
1	Encender la VM y logearse como `root`
2	Realizar los pasos descritos en IRATI-BASE-01 (debe PASAR)

3	Editar el fichero `$RINAD/ipcmanager.conf` con:
	`{`
	`  ''localConfiguration'' : { <keep the default value here> }`
	`}`

4	Ejecutar el IPC Manager con el comando `$RINAD/bin/ipcmanager`. El IPC Manager debe arrancar, incializarse sin producir errores en su log (con un tiempo de 5 minutos sin errores se considera el test PASADO).

Comentarios adicionales

Para conseguir mayor evidencia del test pasado correctamente es posible inspeccionar los logs del kernel mediante:
`sudo dmesg > temp.txt`
`vim temp.txt`
En caso satisfactorio debería verse:
`rina-rnl: Multiplexing message type 29`
`rina-rnl: Fetching handler callback and data`
`rina-rnl: Gonna call f821dc60(f44fcff0, f4507980, f17bdc94)`
`rina-kipcm: IPC Manager started. It is listening at NL port-id 1`
`rina-rnl: Message 29 handled successfully`

Estado del test
PASADO

Tabla A.4: Tarjeta de test IRATI-BASE-04

A.5 IRATI-CREATE-01

Etiqueta	IRATI-CREATE-01

Nombre	Creación de un IPC Process shim-eth-vlan usando un fichero de configuración
Objetivo	Crear un IPC Process shim-eth-vlan después de la correcta inicialización del IPC Manager.
Tarjetas relacionadas	IRATI-BASE-01
DUT	Una única VM como especifica la Sección 4.1 y el módulo 8021q cargado.

Paso	Descripción
1	Encender la VM y logearse como `root`
2	Realizar los pasos descritos en IRATI-BASE-01 (debe PASAR)

3	Editar el fichero `$RINAD/ipcmanager.conf` con:
	`{`
	`"localConfiguration" : { <keep the default values here> },`
	`"ipcProcessesToCreate" : [ {`
	`"type" : "shim-eth-vlan",`
	`"applicationProcessName" : "test",`
	`"applicationProcessInstance" : "1"`
	`} ]`
	`}`

4	Ejecutar el IPC Manager con el comando `$RINAD/bin/ipcmanager`. El IPC Manager debe arrancar, incializarse y pedir al kernel la creación de un IPC Process shim-eth-vlan sin producir errores en los logs.

Comentarios adicionales

Para conseguir mayor evidencia del test pasado correctamente es posible inspeccionar los logs del kernel mediante:

```
sudo dmesg > temp.txt
vim temp.txt
```

En caso satisfactorio debería verse:

```
rina-syscalls: Parms:
rina-syscalls:   Process name     = test
rina-syscalls:   Process instance = 1
rina-syscalls:   Entity name      =
rina-syscalls:   Entity instance  =
rina-syscalls:   Type             = shim-eth-vlan
rina-syscalls: Handling personality hook ipc_create
rina-syscalls: Calling personality hook ipc_create
rina-personality-default: Calling wrapped function
rina-kipcm: Creating IPC process:
rina-kipcm:   name:      test/1//
rina-kipcm:   id:        1
rina-kipcm:   factory:   shim-eth-vlan
rina-ipcp-factories: Checking name
rina-ipcp-factories: Name 'shim-eth-vlan' is ok
rina-ipcp-utils: Copying name f08b53c0 into f08b5c00
rina-ipcp-utils: Name at f08b5c00 finalized successfully
rina-ipcp-utils: Name f08b53c0 copied successfully into f08b5c00
rina-shim-eth-vlan: KFA instance f0883840 bound to the shim eth
rina-ipcp-utils: Name at f08b53c0 finalized successfully
rina-ipcp-utils: Name at f08b53c0 destroyed successfully
```

Estado del test
PASADO

Tabla A.5: Tarjeta de test IRATI-CREATE-01

A.6 IRATI-CREATE-02

Etiqueta	IRATI-CREATE-02
Nombre	Creación de múltiples IPC Processes shim-eth-vlan usando un fichero de configuración
Objetivo	Crear dos IPC Processes shim-eth-vlan después de la correcta inicialización del IPC Manager mostrando que el sistema suporta la presencia de múltiples IPC Processes del mismo tipo.
Tarjetas relacionadas	IRATI-BASE-01
DUT	Una única VM como especifica la Sección 4.1 y el módulo 8021q cargado.

Paso	**Descripción**
1	Encender la VM y logearse como `root`
2	Realizar los pasos descritos en IRATI-BASE-01 (debe PASAR)

3	Editar el fichero `$RINAD/ipcmanager.conf` con:
	
	`{`
	`  "localConfiguration" : { <keep the default values here> },`
	`"ipcProcessesToCreate" : [ {`
	`  "type" : "shim-eth-vlan",`
	`  "applicationProcessName" : "test",`
	`  "applicationProcessInstance" : "1"`
	`  }, {`
	`  "type" : "shim-eth-vlan",`
	`  "applicationProcessName" : "second",`
	`  "applicationProcessInstance" : "1"`
	`  } ],`
	
	`}`

4	Ejecutar el IPC Manager con el comando `$RINAD/bin/ipcmanager`. El IPC Manager debe arrancar, incializarse y pedir al kernel la creación de los IPC Processes shim-eth-vlan sin producir errores en los logs.

Comentarios adicionales
Para conseguir mayor evidencia del test pasado correctamente es posible inspeccionar los logs del kernel de la misma manera que la tarjeta anterior pero con las líneas resultantes duplicadas (una vez por cada IPC Process)

Estado del test
PASADO

Tabla A.6: Tarjeta de test IRATI-CREATE-02

A.7 IRATI-CREATE-03

Etiqueta	IRATI-CREATE-03
Nombre	Creación de múltiples IPC Processes shim-eth-vlan usando la CLI
Objetivo	Comprobar que la CLI del IPC Manager es capaz de crear IPC Processes.
Tarjetas relacionadas	IRATI-BASE-04
DUT	Una única VM como especifica la Sección 4.1 y el módulo 8021q cargado.

Paso	**Descripción**
1	Encender la VM y logearse como `root`
2	Realizar los pasos descritos en IRATI-BASE-04 (debe PASAR)
3	Logearse en la misma VM en otra sesión de la consola.

4	Acceder a la consola mediante `telnet localhost 32766`. Lo siguiente deberá aparecer:
	
	`{`
	`root@irati-vm:~# telnet localhost 32766`
	`Trying ::1...`
	`Connected to localhost.`
	`Escape character is '^]'.`
	`Welcome to the IPC Manager console. Type one or more commands.`
	`ipcmanager>`

5	Crear el primer IPC Process con `createipcp test 1 shim-eth-vlan`. En la consola deberá aparecer:
	
	`ipcmanager> createipcp test 1 shim-eth-vlan`
	`IPC Process 1 created successfully.`

6	Repetir el paso 5 para el segundo IPC Process `createipcp second 1 shim-eth-vlan`

7	Comprobar que los procesos están el sistema con `listipcp`:
	
	`ipcmanager> listipcp`
	`Id: 1`
	`  Type: shim-eth-vlan`
	`  Name: Process name: test; Process instance: 1; Entity name: ; Entity instance:`
	
	`Id: 2`
	`  Type: shim-eth-vlan`
	`  Name: Process name: second; Process instance: 1; Entity name: ; Entity instance:`

Comentarios adicionales
Para conseguir mayor evidencia del test pasado correctamente es posible inspeccionar los logs del kernel.

Estado del test
PASADO

Tabla A.7: Tarjeta de test IRATI-CREATE-03

A.8 IRATI-DESTROY-01

Etiqueta	IRATI-DESTROY-01
Nombre	Destrucción de un IPC Process
Objetivo	Destrucción de un IPC Process usando la CLI
Tarjetas relacionadas	IRATI-CREATE-01
DUT	Una única VM como especifica la Sección 4.1 y el módulo 8021q cargado.

Paso	Descripción
1	Encender la VM y logearse como `root`
2	Realizar los pasos descritos en IRATI-CREATE-01 (debe PASAR)
3	Logearse en la misma VM en otra sesión de la consola.

4	Acceder a la consola mediante `telnet localhost 32766`. Lo siguiente deberá aparecer:
	`{`
	`root@irati-vm:~# telnet localhost 32766`
	`Trying ::1...`
	`Connected to localhost.`
	`Escape character is '^]'.`
	`Welcome to the IPC Manager console. Type one or more commands.`
	`ipcmanager>`

5	Listar los procesos en el sistema con `listipcp`:
	`ipcmanager> listipcp`
	`Id: 1`
	`  Type: shim-eth-vlan`
	`  Name: Process name: test; Process instance: 1; Entity name: ; Entity instance:`

6	Destruir el IPC Process con `destroyipcp 1`:
	`ipcmanager> destroyipcp 1`
	`IPC Process 1 destroyed successfully.`

Comentarios adicionales

Los logs del kernel deberán contener las siguientes líneas:
`rina-syscalls: Handling personality hook ipc_destroy`
`rina-syscalls: Calling personality hook ipc_destroy`
`rina-personality-default: Calling wrapped function`
`rina-shim-eth-vlan: Looking for the eth-vlan-instance to destroy`
`rina-ipcp-utils: Name at f08b5b40 finalized successfully`
`rina-ipcp-utils: Name at f08b5b40 destroyed successfully`
`rina-shim-eth-vlan: Eth-vlan instance destroyed, returning`
`rina-syscalls: Handling personality hook ipc_destroy`
`rina-syscalls: Calling personality hook ipc_destroy`
`rina-personality-default: Calling wrapped function`
`rina-shim-eth-vlan: Looking for the eth-vlan-instance to destroy`
`rina-ipcp-utils: Name at f08b5240 finalized successfully`
`rina-ipcp-utils: Name at f08b5240 destroyed successfully`
`rina-shim-eth-vlan: Eth-vlan instance destroyed, returning`

Estado del test
PASADO

Tabla A.8: Tarjeta de test IRATI-DESTROY-01

A.9 IRATI-DESTROY-02

Etiqueta	IRATI-DESTROY-02
Nombre	Destrucción de múltiples IPC Processes
Objetivo	Destrucción de múltiples IPC Processes usando la CLI del IPC Manager
Tarjetas relacionadas	IRATI-CREATE-02, IRATI-DESTROY-01
DUT	Una única VM como especifica la Sección 4.1 y el módulo 8021q cargado.

Paso	**Descripción**
1	Encender la VM y logearse como `root`
2	Realizar los pasos descritos en IRATI-CREATE-01 (debe PASAR)
3	Realizar los pasos descritos en IRATI-DESTROY-01 (debe PASAR)

4	Destruir el segundo IPC Process con `destroyipcp 2`:
	`ipcmanager> destroyipcp 2`
	`IPC Process 2 destroyed successfully.`

Comentarios adicionales
Los logs del kernel deberán contener las mismas líneas que en el test anterior pero duplicadas.

Estado del test
PASADO

Tabla A.9: Tarjeta de test IRATI-DESTROY-02

A.10 IRATI-ASSIGN-01

Etiqueta	IRATI-ASSIGN-01
Nombre	Asignar un IPC Process shim-eth-vlan a una DIF usando un fichero de configuración
Objetivo	Mostrar la correcta asignación de un IPC Process shim-eth-vlan a una DIF de modo que el IPC process pueda empezar a operar satisfactoriamente.
Tarjetas relacionadas	IRATI-BASE-01
DUT	Una única VM como especifica la Sección 4.1 y el módulo 8021q cargado.

Paso	**Descripción**
1	Encender la VM y logearse como `root`
2	Realizar los pasos descritos en IRATI-BASE-01 (debe PASAR)

3	Editar el fichero `$RINAD/ipcmanager.conf` con:
	
	``
	{
	  "localConfiguration" : { <keep the default values here> },
	  "ipcProcessesToCreate" : [{
	  "type" : "shim-eth-vlan",
	  "applicationProcessName" : "test ",
	  "applicationProcessInstance" : "1",
	  "difName" : "98"
	  }],
	  "difConfigurations" : [{
	  "difName" : "98",
	  "configParameters" : [{
	  "name" : "interface-name",
	  "value" : "eth1"
	  }]
	  }]
	}
	``

4	Ejecutar el IPC Manager con el comando `$RINAD/bin/ipcmanager`. El IPC Manager debe arrancar, incializarse y pedir al kernel la creación de un IPC Process shim-eth-vlan y su asignación a la DIF 98. Las siguientes líneas deberán verse en el log del IPC Manager: ``` DEBUG rina.ipcmanager.impl.helpers.IPCProcessManager - Requested the assignment of IPC Process 1 to DIF 98 INFO rina.ipcmanager.impl.helpers.IPCProcessManager - Successfully assigned IPC ```

Comentarios adicionales

Los logs del kernel deberán contener líneas similares a las siguientes:

```
rina-rnl: Multiplexing message type 1
rina-rnl: Fetching handler callback and data
rina-rnl: Gonna call f821e3a0(f44fcff0, f130fb00, f05e5c94)
rina-rnl-utils: RINA Netlink parser started ...
rina-rnl-utils: msg at f0e745a0 / msg->attrs at f44a1508
rina-rnl-utils:   src-ipc-id: 0
rina-rnl-utils:   dst-ipc-id: 1
rina-ipcp-utils: Name at f08b5b40 finalized successfully
rina-kipcm: Found IPC Process with id 1
rina-ipcp-utils: Copying name f08b5b40 into f13683c0
rina-ipcp-utils: Name at f13683c0 finalized successfully
rina-ipcp-utils: Name f08b5b40 copied successfully into f13683c0
rina-shim-eth-vlan: Building complete VLAN interface name ('eth1', 98)
rina-shim-eth-vlan: VLAN id is '98'
rina-shim-eth-vlan: Complete vlan-interface-name length will be 8
rina-shim-eth-vlan: Complete vlan-interface-name = 'eth1.98'
rina-shim-eth-vlan: Got device 'eth1.98', trying to register handler
rina-shim-eth-vlan: Configured shim eth vlan IPC Process
rina-ipcp-utils: Name at f08b5b40 finalized successfully
Nov 28 13:47:16 irati-vm kernel: [ 9897.501239] rina-ipcp-utils: Name at f08b5b40 destroyed successfully
rina-rnl-utils: Result of genlmesg_end: 32
rnl-utils: Going to send NL unicast message (type = 2, seq-num 1385642837, port = 1)
rina-rnl-utils: Unicast NL message sent (type = 2, seq-num 1385642837, port = 1)
rina-rnl: Message 1 handled successfully
```

Estado del test

PASADO

Tabla A.10: Tarjeta de test IRATI-ASSIGN-01

A.11 IRATI-ASSIGN-02

Etiqueta	IRATI-ASSIGN-02
Nombre	Asignar un IPC Process shim-eth-vlan a una DIF usando la CLI
Objetivo	Mostrar que la CLI del IPC Manager permite la correcta asignación de un IPC Process shim-eth-vlan a una DIF.
Tarjetas relacionadas	IRATI-CREATE-01
DUT	Una única VM como especifica la Sección 4.1 y el módulo 8021q cargado.

Paso	**Descripción**
1	Encender la VM y logearse como `root`
2	Realizar los pasos descritos en IRATI-CREATE-01 (debe PASAR)editando el fichero `$RINAD/ipcmanager.conf` con: `{ "localConfiguration" : { <keep the default values here> }, "ipcProcessesToCreate" : [{ "type" : "shim-eth-vlan", "applicationProcessName" : "test ", "applicationProcessInstance" : "1" }], "difConfigurations" : [{ "difName" : "98", "configParameters" : [{ "name" : "interface-name", "value" : "eth1" }] }] }`
3	Logearse en la misma VM en otra sesión de la consola.

4	Acceder a la consola mediante `telnet localhost 32766`. Lo siguiente deberá aparecer:
	`root@irati-vm:~# telnet localhost 32766`
	`Trying ::1...`
	`Connected to localhost.`
	`Escape character is '^]'.`
	`Welcome to the IPC Manager console. Type one or more commands.`
	`ipcmanager>`

5	Listar los procesos en el sistema con `listipcp`:
	`ipcmanager> listipcp`
	`Id: 1`
	`Type: shim-eth-vlan`
	`Name: Process name: test; Process instance: 1; Entity name: ; Entity instance:`

6	Pedir la asignación del IPC Process a la DIF 98 con `assigndif 1 98`:
	
	`ipcmanager> assigndif 1 98`
	`Successfully assigned IPC Process 1 to DIF 98`

7	Listar los procesos en el sistema de nuevo con `listipcp`:
	
	`Id: 1`
	`  Type: shim-eth-vlan`
	`  Name: Process name: test; Process instance: 1; Entity name: ; Entity instance:`
	`  Member of DIF: 98`

Comentarios adicionales
Los logs del kernel deberán contener las mismas líneas que el test anterior.

Estado del test
PASADO

Tabla A.11: Tarjeta de test IRATI-ASSIGN-01

A.12 IRATI-DESTROY-03

Etiqueta	IRATI-DESTROY-03
Nombre	Destruir un IPC process shim-eth-vlan asignado a una DIF
Objetivo	Mostrar que cuando el IPC Process shim-eth-vlan es destruido estando asignado a una DIF, correctamente se desregistra del device driver de la VLAN de la DIF y elimina todas sus estructuras.
Tarjetas relacionadas	IRATI-ASSGIN-02
DUT	Una única VM como especifica la Sección 4.1 y el módulo 8021q cargado.

Paso	Descripción
1	Encender la VM y logearse como `root`
2	Realizar los pasos descritos en IRATI-ASSIGN-01 (debe PASAR)

3	Ejecutar el siguiente comando: `destroyipcp 1` obteniendo:
	
	`ipcmanager> destroyipcp 1`
	`IPC Process 1 destroyed successfully`

Comentarios adicionales
Los logs del kernel deberán contener las mismas líneas que el test IRATI-DESTROY-01.

Estado del test
PASADO

Tabla A.12: Tarjeta de test IRATI-DESTROY-03

A.13 IRATI-REGISTER-01

Etiqueta	IRATI-REGISTER-01
Nombre	Registrar la aplicación echo-server a una shim DIF
Objetivo	Comprobar que una aplicación puede registrarse en una DIF shim-eth-vlan.
Tarjetas relacionadas	IRATI-ASSGIN-01
DUT	Una única VM como especifica la Sección 4.1 y el módulo 8021q cargado.

Paso	Descripción
1	Encender la VM y logearse como `root`
2	Realizar los pasos descritos en IRATI-ASSIGN-01 (debe PASAR)
3	Ejecutar el echo-server en una nueva terminal con el comando `$RINAD/bin/echo-server`. El echo server deberá arrancar, inicializarse y pedir al IPC Manager el registro a una DIF. Las siguientes líneas deben verse en el log del echo-server:

```
INFO rina.utils.apps.echo.server.EchoServer - Requested registration of control AE:
      Process name: rina.utils.apps.echo.server; Process instance: 1; Entity name: ;
      Entity instance:
INFO IPC Event has happened: REGISTER_APPLICATION_RESPONSE_EVENT
INFO rina.utils.apps.echo.server.EchoServer - Succesfully registered AE Process name:
       rina.utils.apps.echo.server; Process instance: 1; Entity name: ; Entity instance
      :  to DIF 98
```

En el log del IPC Manager deberá verse:

```
INFO rina.ipcmanager.impl.IPCManager - Got event of type:
      APPLICATION_REGISTRATION_REQUEST_EVENT and sequence number: 1385644263
DEBUG rina.ipcmanager.impl.helpers.ApplicationRegistrationManager - Requested
      registration of application Process name: rina.utils.apps.echo.server; Process
      instance: 1; Entity name: ; Entity instance:  to DIF 100. Got handle 1385644257
INFO rina.ipcmanager.impl.IPCManager - Got event of type:
      IPCM_REGISTER_APP_RESPONSE_EVENT and sequence number: 1385644257
INFO rina.ipcmanager.impl.helpers.ApplicationRegistrationManager - Successfully
      registered application Process name: rina.utils.apps.echo.server; Process
      instance: 1; Entity name: ; Entity instance:  to DIF Process name: 100; Process
      instance: ; Entity name: ; Entity instance:
```

Comentarios adicionales

Los logs del kernel deberán contener las siguientes líneas o similares:

```
rina-rnl: Multiplexing message type 18
rina-rnl: Fetching handler callback and data
rina-rnl: Gonna call f821eb60(f44fcff0, f1518000, f0711c94)
rina-rnl-utils: RINA Netlink parser started ...
rina-rnl-utils: msg at f0e742d0 / msg->attrs at f45b42e0
rina-rnl-utils:   src-ipc-id: 0
rina-rnl-utils:   dst-ipc-id: 1
rina-ipcp-utils: Name at f05220c0 finalized successfully
rina-ipcp-utils: Name at f0522b40 finalized successfully
rina-ipcp-utils: Copying name f05220c0 into f0522480
rina-ipcp-utils: Name at f0522480 finalized successfully
rina-ipcp-utils: Name f05220c0 copied successfully into f0522480
rina-arp826-tables: Adding GPA/GHA couple to the 0xd1f0 ptype table
rina-arp826-tables: Creating a new table entry for this ARP-add request
rina-arp826-tables: Creating new table entry
rina-arp826-tables: Table entry f0522540 created successfully
rina-arp826-tables: Adding the GPA/GHA entry to the 0xd1f004 table
rina-arp826-tables: Adding entry f0522540 to the table
rina-arp826-tables: Entry f0522540 added successfully to the table
rina-arp826-tables: GPA/GHA couple for ptype 0xd1f0 added successfully
rina-ipcp-utils: Name at f05220c0 finalized successfully
rina-ipcp-utils: Name at f05220c0 destroyed successfully
rina-ipcp-utils: Name at f0522b40 finalized successfully
rina-ipcp-utils: Name at f0522b40 destroyed successfully
rina-rnl-utils: Result of genlmesg_end: 32
rina-rnl-utils: Going to send NL unicast message (type = 19, seq-num 1385644257, port = 1)
rina-rnl-utils: Unicast NL message sent (type = 19, seq-num 1385644257, port = 1)
rina-rnl: Message 18 handled successfully
```

Estado del test
PASADO

Tabla A.13: Tarjeta de test IRATI-REGISTER-01

A.14 IRATI-REGISTER-02

Etiqueta	IRATI-REGISTER-02
Nombre	Registrar dos aplicaciones a una IPC Process shim-eth-vlan
Objetivo	Comprobar que sólo una aplicación registrada a la vez es soportada por el IPC Process shim-eth-vlan.
Tarjetas relacionadas	IRATI-REGISTER-01
DUT	Una única VM como especifica la Sección 4.1 y el módulo 8021q cargado.

Paso	**Descripción**
1	Encender la VM y logearse como `root`
2	Realizar los pasos descritos en IRATI-REGISTER-01 (debe PASAR)

3	Ejecutar el echo-client en una nueva terminal con el comando `$RINAD/bin/echo-client`. El echoc-client deberá arrancar y pedir al IPC Manager el registro a una DIF. Dado que es la única DIF en el sistema, el IPC Manager pedirá el registro de la aplicación al IPC Process shim-eth-vlan, lo cual deberá fallar. El echo-client producirá las siguientes líneas:
	
	`INFO rina.utils.apps.echo.client.EchoClient - Requested registration of AE: Process name: rina.utils.apps.echo.client; Process instance: 1; Entity name: ; Entity instance:`
	`}IPC Event has happened: REGISTER_APPLICATION_RESPONSE_EVENT`
	`ERROR rina.utils.apps.echo.client.EchoClient - Problems registering AE Process name: rina.utils.apps.echo.client; Process instance: 1; Entity name: ; Entity instance: . Error code: -1`
	
	En el log del IPC Manager deberá verse:
	
	`INFO rina.ipcmanager.impl.IPCManager - Got event of type: APPLICATION_REGISTRATION_REQUEST_EVENT and sequence number: 1385645059`
	`DEBUG rina.ipcmanager.impl.helpers.ApplicationRegistrationManager - Requested registration of application Process name: rina.utils.apps.echo.client; Process instance: 1; Entity name: ; Entity instance:  to DIF 100. Got handle 1385644258`
	`INFO rina.ipcmanager.impl.IPCManager - Got event of type: IPCM_REGISTER_APP_RESPONSE_EVENT and sequence number: 1385644258`
	`INFO rina.ipcmanager.impl.helpers.ApplicationRegistrationManager - Could not register application Process name: rina.utils.apps.echo.client; Process instance: 1; Entity name: ; Entity instance:  to DIF Process name: 100; Process instance: ; Entity name: ; Entity instance:`

Comentarios adicionales
Los logs del kernel deberán contener las siguientes líneas o similares: `rina-shim-eth-vlan: Application rina.utils.apps.echo.server/1// is already registered`

Estado del test
PASADO

Tabla A.14: Tarjeta de test IRATI-REGISTER-02

A.15 IRATI-CLIENT-01

Etiqueta	IRATI-CLIENT-01
Nombre	Ejecutar la aplicación echo-client (una vez, 100 SDUs)
Objetivo	Asegurar que 100 SDUs son intercambiadas entre dos DUT.
Tarjetas relacionadas	IRATI-BASE-04
DUT	Dos VMs como especifica la Fig. 5.1.

Paso	**Descripción**
1	Realizar los pasos descritos en IRATI-BASE-04 en las dos VMs (debe PASAR)

2	Abrir una consola en VM1 y logearse como `root`
3	Ejecutar el echo-server con el comando `$RINAD/bin/echo-server`. El echo-server deberá manterse corriendo sin mostrar errores en ningún momento.
4	Abrir una consola en VM2 y logearse como `root`
3	Ejecutar el echo-server con el comando `$RINAD/bin/echo-server`. El echo-server deberá manterse corriendo sin mostrar errores en ningún momento.
5	Ejecutar el echo-client con el comando `$RINAD/bin/echo-client -c 100`. El echo-client deberá ejecutarse sin errores.

Comentarios adicionales

Estado del test
PASADO

Tabla A.15: Tarjeta de test IRATI-CLIENT-01

A.16 IRATI-CLIENT-02

Etiqueta	IRATI-CLIENT-02
Nombre	Ejecutar la aplicación echo-client (dos veces, 100 SDUs)
Objetivo	Asegurar que la ejecución previa no compromete otras ejecuciones del mismo test.
Tarjetas relacionadas	IRATI-CLIENT-01
DUT	Dos VMs como especifica la Fig. 5.1.

Paso	**Descripción**
1	Realizar los pasos descritos en IRATI-BASE-01 en las dos VMs (debe PASAR)
2	Cerrar el echo-client en la VM2, el cual debe salir sin errores
3	Cerrar el echo-server en la VM1, el cual debe salir sin errores
4	Ejecutar los pasos de IRATI-CLIENT-01 de nuevo (debe PASAR).

Comentarios adicionales

Estado del test
PASADO

Tabla A.16: Tarjeta de test IRATI-CLIENT-02

A.17 IRATI-CLIENT-03

Etiqueta	IRATI-CLIENT-03

Nombre	Ejecutar la aplicación echo-client (100 veces, 100 SDUs)
Objetivo	Asegurar que múltiples ejecuciones del test IRATI-CLIENT-01 no compromete los componentes del kernel y que múltiples tests se pueden ejecutar sin comprometer las DUTs.
Tarjetas relacionadas	IRATI-BASE-04
DUT	Dos VMs como especifica la Fig. 5.1.

Paso	Descripción
1	Realizar los pasos descritos en IRATI-BASE-04 en las dos VMs (debe PASAR)
2	Abrir una consola en VM1 y logearse como `root`
3	Ejecutar el echo-server con el comando `$RINAD/bin/echo-server`. El echo-server deberá manterse corriendo sin mostrar errores en ningún momento.
4	Abrir una consola en VM2 y logearse como `root`
5	Ejecutar el echo-client con el comando: `for ((i=0 ; i<100; i=i+1)); do echo $i && $RINAD_PREFIX/bin/echo-client c 100 || break ; done && echo PASS` El comando no deberá presentar ningún error y la palabra PASS deberá aparecer en los logs.

Comentarios adicionales

Estado del test
PASADO

Tabla A.17: Tarjeta de test IRATI-CLIENT-03

A.18 IRATI-KILL-01

Etiqueta	IRATI-KILL-01
Nombre	Detener la aplicación echo-server
Objetivo	Comprobar que el IPC Manager detecta que un proceso del OS fue destruido y consecuentemente le pide eliminar todos los recursos (flows, registros9 que tenía dicho proceso.
Tarjetas relacionadas	IRATI-REGISTER-01
DUT	Una única VM como especifica la Sección 4.1 y el módulo 8021q cargado.

Paso	Descripción
1	Encender la VM y logearse como `root`
2	Realizar los pasos descritos en IRATI-REGISTER-01 (debe PASAR)

2	Destruir el programa echo-server mediante el comando `CTRL+C` o meduante `kill`. El IPC Manager debe recibir un evento indicando que el proceso de OS fue eliminado y pedir al kernel que desregistre la aplicación de la DIF. Si la operación se ejecuta correctamente, el log del IPC Manager debería verse tal que: `INFO rina.ipcmanager.impl.IPCManager - Got event of type: OS_PROCESS_FINALIZED and sequence number: 0` `INFO rina.ipcmanager.impl.IPCManager - OS process finalized. Naming info: Process name: rina.utils.apps.echo.server; Process instance: 1; Entity name: ; Entity instance:` `INFO rina.ipcmanager.impl.helpers.FlowManager - 0 flows are going to be deallocated` `DEBUG rina.ipcmanager.impl.helpers.ApplicationRegistrationManager - Requested unregitration of application Process name: rina.utils.apps.echo.server; Process instance: 1; Entity name: ; Entity instance: from DIF Process name: 100; Process instance: ; Entity name: ; Entity instance: . Got handle 1385644259` `INFO rina.ipcmanager.impl.IPCManager - Got event of type: IPCM_UNREGISTER_APP_RESPONSE_EVENT and sequence number: 1385644259` `INFO rina.ipcmanager.impl.helpers.ApplicationRegistrationManager - Successfully unregistered application Process name: rina.utils.apps.echo.server; Process instance: 1; Entity name: ; Entity instance: from DIF Process name: 100; Process instance: ; Entity name: ; Entity instance:`

Comentarios adicionales

Los logs del kernel deberían contener las siguientes líneas:
 	
`rina-rnl-utils: Going to send NL unicast message (type = 28, seq-num 0, port = 1)`
`rina-rnl-utils: Unicast NL message sent (type = 28, seq-num 0, port = 1)`
`rina-rnl: Dispatching message (skb-in=f4573500, info=f0711c94)`
`rina-rnl: Multiplexing message type 20`
`rina-rnl: Fetching handler callback and data`
`rina-rnl: Gonna call f821e100(f44fcff0, f4573500, f0711c94)`
`rina-rnl-utils: RINA Netlink parser started ...`
`rina-rnl-utils: msg at f06cc870 / msg->attrs at f16098a0`
`rina-rnl-utils:  src-ipc-id: 0`
`rina-rnl-utils:  dst-ipc-id: 1`
`rina-ipcp-utils: Name at f0522600 finalized successfully`
`rina-ipcp-utils: Name at f0522a80 finalized successfully`
`rina-ipcp-utils: Name at f0522480 finalized successfully`
`rina-ipcp-utils: Name at f0522480 destroyed successfully`
`rina-ipcp-utils: Name at f0522600 finalized successfully`
`rina-ipcp-utils: Name at f0522600 destroyed successfully`
`rina-ipcp-utils: Name at f0522a80 finalized successfully`
`rina-ipcp-utils: Name at f0522a80 destroyed successfully`
`rina-rnl-utils: Result of genlmesg_end: 32`
`rina-rnl-utils: Going to send NL unicast message (type = 21, seq-num 1385644259, port = 1)`
`rina-rnl-utils: Unicast NL message sent (type = 21, seq-num 1385644259, port = 1)`
`rina-rnl: Message 20 handled successfully`

Estado del test
PASADO

Tabla A.18: Tarjeta de test IRATI-KILL-01

A.19 IRATI-KILL-02

Etiqueta	IRATI-KILL-02

Nombre	Detener el IPC Manager daemon
Objetivo	Comprobar que el kernel detecta que el IPC Manager daemon fue destruido y actualiza su estado correctamente.
Tarjetas relacionadas	IRATI-BASE-04
DUT	Una única VM como especifica la Sección 4.1 y el módulo 8021q cargado.

Paso	Descripción
1	Encender la VM y logearse como `root`
2	Realizar los pasos descritos en IRATI-BASE-04 (debe PASAR)
3	Destruir el programa IPC Manager daemon mediante el comando `CTRL+C` o mediante `kill`. El kernel deberá darse cuenta de esto y mostrar la siguiente línea en el log: `rina-rnl: IPC Manager process has been destroyed`

Comentarios adicionales
Tras la detección de que el IPC Manager fue destruido, los componentes del kernel deberán liberar todos los recursos que estaban siendo utilizados (reigstros, flows, IPC Processes). Este funcionalidad no está todavía implementada, con lo cual si se para al IPC Manager habiendo IPC Processes en el sistema, éste debera ser reiniciado para corregir su estado.

Estado del test
PASADO

Tabla A.19: Tarjeta de test IRATI-KILL-02

A.20 IRATI-SINGLEFLOW-01

Etiqueta	IRATI-SINGLEFLOW-01
Nombre	Test con un único flow por DIF
Objetivo	Comprobar la conexión básica con un único flow por DIF.
Tarjetas relacionadas	IRATI-CLIENT-01, IRATI-CLIENT-02
DUT	Como en Fig. 5.4.

Paso	Descripción
1	Establecer las VLANs entre los hosts
2	Configurar los IPC Managers para que asignen las DIFs a las VLANs correctamente.
3	Registrar todos los echo-servers y echo-clients en cada hosts en la DIF correcta.
4	Ejecutar los echo-servers y echo-clients.
5	Registrar el tiempo para enviar las N SUDs de B bytes.

Comentarios adicionales
Durante la experimentación en OFELIA se encontraron algunos problemas con el campo ethertype de la trama Ethernet. Para explicarlo es preciso considerar los siguientes puntos: • Las direcciones del protocolo de red sobre las requests/replies de ARP826 pueden ser de cualquier longitud. • El campo Protocol Type de las requests/replies oara la shim DIF sobre Ethernet es 0xD1F0 como se especifica en [11], el cual identifica al tráfico RINA. • El campo ethertype de la trama Ethernet para el tráfico ARP826 producido por la shim DIF es 0x0806 (ARP) ya que ARP826 cumple con el estándar. El problema identificado fue que los switches hardware descartaban las requests/replies producidas por ARP826 porque al inspeccionar el Protocol Type y encontrar 0x0806 validaban el paquete adjunto como si fuera IP y busccaban direcciones de red de 4 bytes, con lo cual el paquete al no cumplir con estos criterios era descartado. La solución fue cambiar este campo por otro valor correspondiente a un protocolo conocido pero con el cual los switches no aplicaran validaciones. La solución definitiva será decidir un valor libre y asignarlo a ARP826 en las especificaciones de la shim DIF, ya que, a falta de mayores pruebas, se espera que la implementación de ARP soporte únicamente IP como ya pasa en Linux.

Estado del test
PASADO. Los paquetes siguen la cadena OpenFlow normal.

Tabla A.20: Tarjeta de test IRATI-SINGLEFLOW-01

A.21 IRATI-MULTIFLOW-01

Etiqueta	IRATI-MULTIFLOW-01
Nombre	Test con múltiples flows por DIF
Objetivo	Comprobar la conexión básica con múltiples flows por DIF.
Tarjetas relacionadas	IRATI-SINGLEFLOW-01
DUT	Como en Fig. 5.4.

Paso	Descripción
1	Establecer las VLANs entre los hosts
2	Configurar los IPC Managers para que asignen las DIFs a las VLANs correctamente.
3	Registrar todos los echo-servers y echo-clients en cada hosts en la DIF correcta.
4	Ejecutar los echo-servers y echo-clients.
5	Registrar el tiempo para enviar las N SUDs de B bytes.

Comentarios adicionales

Estado del test

PASADO

Tabla A.21: Tarjeta de test IRATI-MULTIFLOW-01

A.22 IRATI-CONCURRENT-01

Etiqueta	IRATI-CONCURRENT-01
Nombre	Test con aplicaciones RINA e IP concurrentes
Objetivo	Comprobar la estabilidad del kernel cuando utiliza los stacks UDP/IP y RINA y comprobar como interactúan
Tarjetas relacionadas	IRATI-SINGLEFLOW-01
DUT	Como en Fig. 5.4.

Paso	**Descripción**
1	Establecer las VLANs entre los hosts
2	Configurar los IPC Managers para que asignen las DIFs a las VLANs correctamente.
3	Registrar todos los echo-servers y echo-clients en cada hosts en la DIF correcta.
4	Ejecutar iperf usando UDP durante 5 minutos.
5	Ejecutar los echo-servers y echo-clients.
6	Registrar el ancho de banda obtenido por iperf.
7	Registrar el tiempo para enviar las N SUDs de B bytes.

Comentarios adicionales

Estado del test
PASADO

Tabla A.22: Tarjeta de test IRATI-CONCURRENT-01

Apéndice B. Validación del setup en el testbed de OFELIA

Algunos detalles del funcionamiento del testbed de OFELIA se han introducido en la Sección 5.5.1. Para profundizar en este aspecto y en la tecnología que lo soporta, OpenFlow, se puede acceder a las referencias [36] y [37].

El primer paso para acceder a OFELIA es obtener una cuenta de usuario. Esto puede conseguirse de forma gratuita desde el portal de OFELIA[38]. Seguidamente es necesario acceder a la red de control del testbed mediante la conexión a una Virtual Private Network (VPN) suministrada durante el proceso de registro. El setup de la conexión puede verse en Fig. B.1.

```
root@leo-Latitude-E5520:/etc/openvpn# openvpn ofelia.ovpn
Fri Jul 26 09:19:47 2013 OpenVPN 2.2.1 i686-linux-gnu [SSL] [LZO2] [EPOLL] [PKCS11] [eurephia] [M
H] [PF_INET6] [IPv6 payload 20110424-2 (2.2RC2)] built on Feb 27 2013
Enter Auth Username:leonardobergesio
Enter Auth Password:
Fri Jul 26 09:19:56 2013 WARNING: No server certificate verification method has been enabled.  Se
e http://openvpn.net/howto.html#mitm for more info.
Fri Jul 26 09:19:56 2013 NOTE: the current --script-security setting may allow this configuration
 to call user-defined scripts
Fri Jul 26 09:19:56 2013 UDPv4 link local (bound): [undef]
Fri Jul 26 09:19:56 2013 UDPv4 link remote: [AF_INET]157.193.215.150:1194
Fri Jul 26 09:19:56 2013 WARNING: this configuration may cache passwords in memory -- use the aut
h-nocache option to prevent this
Fri Jul 26 09:19:57 2013 [VPNSERVER_ROUTED] Peer Connection Initiated with [AF_INET]157.193.215.1
50:1194
Fri Jul 26 09:19:59 2013 TUN/TAP device tun0 opened
Fri Jul 26 09:19:59 2013 do_ifconfig, tt->ipv6=0, tt->did_ifconfig_ipv6_setup=0
Fri Jul 26 09:19:59 2013 /sbin/ifconfig tun0 10.216.126.4 netmask 255.255.255.0 mtu 1500 broadcas
t 10.216.126.255
Fri Jul 26 09:19:59 2013 /etc/openvpn/update-resolv-conf tun0 1500 1541 10.216.126.4 255.255.255.
0 init
dhcp-option DNS 10.216.4.3
Fri Jul 26 09:20:02 2013 Initialization Sequence Completed
```

Figura B.1: establecimiento de la conexión con la VPN de OFELIA

La página principal del OCF [33] cuando se accede como usuario puede verse en Fig. B.2. Esta página lista los proyectos del usuario, y en concreto se puede ver el proyecto creado para IRATI y este PFM.

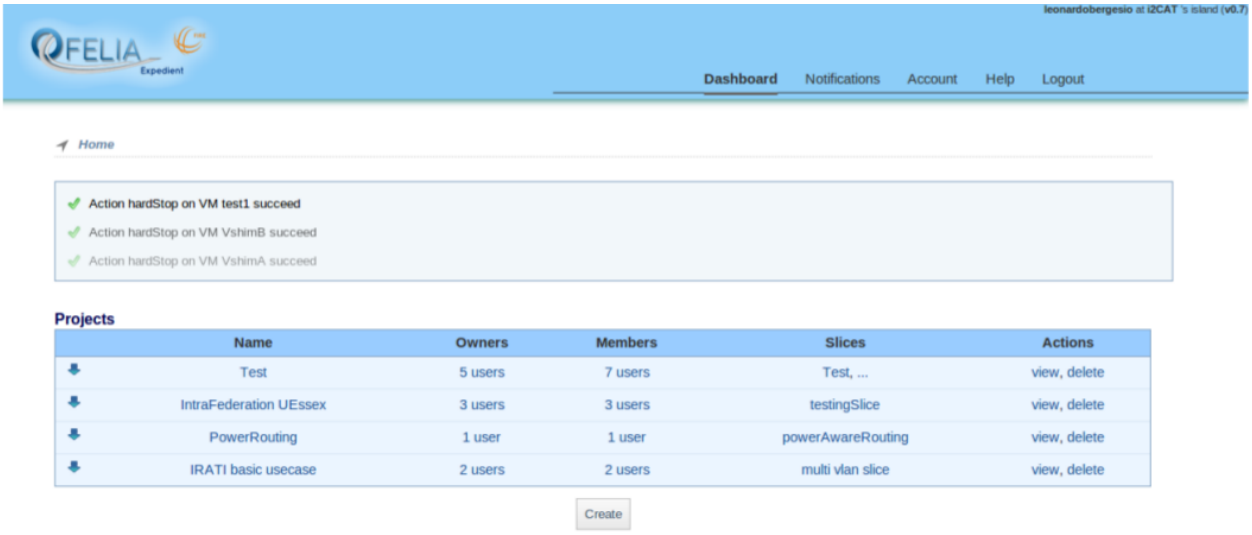


Figura B.2: Dashboard del OCF y el listado de proyectos

El setup del experimento se realiza en la página de la slice dentro del OCF. Los siguientes pasos deben ser seguidos:

1. Pedir el FlowSpace para el experimento. La Fig. B.3 muestra la topología física de la isla de i2CAT. Además, los switches y los puertos seleccionados aparecen resaltados.

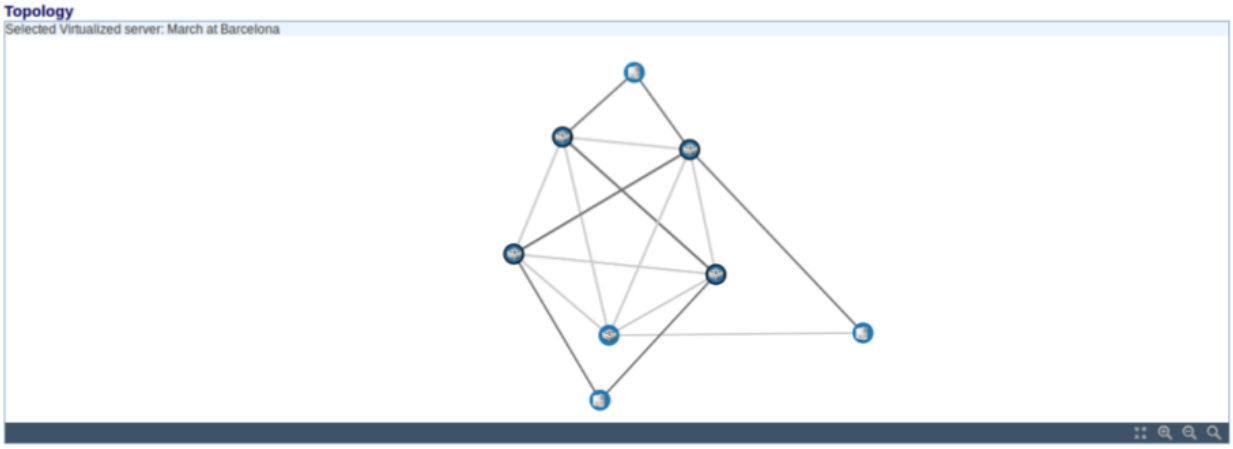


Figura B.3: Topología física de la slice para el experimento en la isla de i2CAT

2. Crear las VMs necesarias en los servidores adecuados como se ve en la Fig. B.4:

   - nodeA corresponde con el sistema A, hospedado en el servidor Verdaguer
   - nodeB corresponde con el sistema B, hospedado en el servidor Rodoreda
   - nodeC corresponde con el sistema C, hospedado en el servidor Verdaguer
   - nodeD corresponde con el sistema D, hospedado en el servidor March

Computational resources

• Virt. Aggregate: i2CAT servers

Name: i2CAT servers
Status:
Physical location: Barcelona
Resources:

Server Name	Virt. Tech.	Operating System	CPU	Memory	Disc
March	XEN	GNU/Linux Debian (6.0)	None	None	None

VM Name	State	Operating System	Memory	Mgmt IP	Actions
nodeD	running	GNU/Linux Debian (7.0)	1024	10.216.12.34	Stop Reboot

SSH access: ~# ssh root@10.216.12.34 (password: your user password)

SSH common details: to access as root just type su inside (password: openflow)

Server Name	Virt. Tech.	Operating System	CPU	Memory	Disc
Rodoreda	XEN	GNU/Linux Debian (6.0)	None	None	None

VM Name	State	Operating System	Memory	Mgmt IP	Actions
nodeB	running	GNU/Linux Debian (7.0)	1024	10.216.12.31	Stop Reboot

SSH access: ~# ssh root@10.216.12.31 (password: your user password)

SSH common details: to access as root just type su inside (password: openflow)

Server Name	Virt. Tech.	Operating System	CPU	Memory	Disc
Verdaguer	XEN	GNU/Linux Debian (6.0)	None	None	None

VM Name	State	Operating System	Memory	Mgmt IP	Actions
switch-controller	running	GNU/Linux Debian (6.0)	512	10.216.12.27	Stop Reboot
				SSH access: ~# ssh root@10.216.12.27 (password: your user password)	
nodeA	stopped	GNU/Linux Debian (7.0)	2048	10.216.12.29	Start Delete
				SSH access: ~# ssh root@10.216.12.29 (password: your user password)	
nodeC	stopped	GNU/Linux Debian (7.0)	1024	10.216.12.33	Start Delete
				SSH access: ~# ssh root@10.216.12.33 (password: your user password)	

SSH common details: to access as root just type su inside (password: openflow)

Figura B.4: VMs creadas para el experimento

3. Especificar la IP del controlador OpenFlow. En la Fig. B.5 puede verse esto en conjunto con el FlowSpace que ha sido otorgado a la slice. El controlador OpenFlow corresponde a una instancia de NOX corriendo como MAC learning switch en la VM switch_controller en el servidor Verdaguer. Su ejecución puede verse en Fig. B.6

Network resources

• OpenFlow Aggregate: i2CAT openflow infrastructure

Name: i2CAT openflow infrastructure
Status:
Physical location: Barcelona
Resources:

Requested Flowspace (1) More information
Granted Flowspace (1) More information

Flowspace	Associated OpenFlow Interfaces
VLAN ID: 300 - 301	OpenFlow Switch: 00:10:00:00:00:00:00:05 - Port 2 OpenFlow Switch: 00:10:00:00:00:00:00:05 - Port 12 OpenFlow Switch: 00:10:00:00:00:00:00:03 - Port 1 OpenFlow Switch: 00:10:00:00:00:00:00:03 - Port 12 OpenFlow Switch: 00:10:00:00:00:00:00:02 - Port 12 OpenFlow Switch: 00:10:00:00:00:00:00:02 - Port 5 OpenFlow Switch: 00:10:00:00:00:00:00:05 - Port 11 OpenFlow Switch: 00:10:00:00:00:00:00:01 - Port 12 OpenFlow Switch: 00:10:00:00:00:00:00:01 - Port 3

Openflow controller: tcp:10.216.12.27:6633 Update controller

Actions: Book Openflow resources

Remove from slice: Remove AM

Figura B.5: FlowSpace otorgado

```
Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Wed Nov 27 09:05:29 2013 from 10.216.126.3
$ su
Password:
root@switch-controller:/ofelia/users/leonardobergesio# cd /opt/ofelia/software/nox/build/src/
root@switch-controller:/opt/ofelia/software/nox/build/src# ./nox_core -i ptcp:6633 -v switch
00001|nox|INFO:Starting nox_core (/opt/ofelia/software/nox/build/src/.libs/lt-nox_core)
00002|pyrt|DBG:Loading a component description file 'nox/coreapps/simple_c_py_app/meta.json'.
00003|pyrt|DBG:Loading a component description file 'nox/coreapps/simple_c_app/meta.json'.
00004|pyrt|DBG:Loading a component description file 'nox/coreapps/coretests/meta.json'.
00005|pyrt|DBG:Loading a component description file 'nox/coreapps/examples/meta.json'.
00006|pyrt|DBG:Loading a component description file 'nox/coreapps/examples/t/meta.json'.
00007|pyrt|DBG:Loading a component description file 'nox/coreapps/messenger/meta.json'.
00008|pyrt|DBG:Loading a component description file 'nox/coreapps/switch/meta.json'.
00009|pyrt|DBG:Loading a component description file 'nox/coreapps/pyrt/meta.json'.
00010|pyrt|DBG:Loading a component description file 'nox/coreapps/snmp/meta.json'.
00011|pyrt|DBG:Loading a component description file 'nox/coreapps/testharness/meta.json'.
00012|pyrt|DBG:Loading a component description file 'nox/coreapps/hub/meta.json'.
00013|pyrt|DBG:Loading a component description file 'nox/netapps/networkstate/meta.json'.
00014|pyrt|DBG:Loading a component description file 'nox/netapps/switchstats/meta.json'.
00015|pyrt|DBG:Loading a component description file 'nox/netapps/switch_management/meta.json'.
00016|pyrt|DBG:Loading a component description file 'nox/netapps/tablog/meta.json'.
00017|pyrt|DBG:Loading a component description file 'nox/netapps/discovery/meta.json'.
00018|pyrt|DBG:Loading a component description file 'nox/netapps/tests/meta.json'.
00019|pyrt|DBG:Loading a component description file 'nox/netapps/lavi/meta.json'.
00020|pyrt|DBG:Loading a component description file 'nox/netapps/storage/meta.json'.
00021|pyrt|DBG:Loading a component description file 'nox/netapps/storage/t/meta.json'.
00022|pyrt|DBG:Loading a component description file 'nox/netapps/hoststate/meta.json'.
00023|pyrt|DBG:Loading a component description file 'nox/netapps/data/meta.json'.
00024|pyrt|DBG:Loading a component description file 'nox/netapps/bindings_storage/meta.json'.
00025|pyrt|DBG:Loading a component description file 'nox/netapps/bindings_storage/t/meta.json'.
00026|pyrt|DBG:Loading a component description file 'nox/netapps/routing/meta.json'.
00027|pyrt|DBG:Loading a component description file 'nox/netapps/authenticator/meta.json'.
00028|pyrt|DBG:Loading a component description file 'nox/netapps/topology/meta.json'.
00029|pyrt|DBG:Loading a component description file 'nox/netapps/flow_fetcher/meta.json'.
00030|pyrt|DBG:Loading a component description file 'nox/netapps/user_event_log/meta.json'.
00031|pyrt|DBG:Loading a component description file 'nox/netapps/route/meta.json'.
00032|pyrt|DBG:Loading a component description file 'nox/webapps/miscws/meta.json'.
00033|pyrt|DBG:Loading a component description file 'nox/webapps/webservice/meta.json'.
00034|pyrt|DBG:Loading a component description file 'nox/webapps/webserver/meta.json'.
00035|nox|DBG:Application installation report:
00036|nox|DBG:switch:
        Current state: INSTALLED
        Required state: INSTALLED
        Dependencies:

00037|nox|DBG:python:
        Current state: INSTALLED
        Required state: INSTALLED
        Dependencies:

00038|nox|DBG:built-in DSO deployer:
        Current state: INSTALLED
        Required state: INSTALLED
        Dependencies:

00039|nox|DBG:built-in event dispatcher:
        Current state: INSTALLED
        Required state: INSTALLED
        Dependencies:

00040|openflow|DBG:Passive tcp interface bound to port 6633
00041|nox|INFO:nox bootstrap complete
```

Figura B.6: Ejecución del controlador OF

4. Finalmente, es necesario configurar las interfaces de las VMs conectadas a la red de experimentación para darles una IP en redes distintas y configurar las VLANs. Fig. B.7 muestra la configuración del nodeA.

```
root@nodeA:/ofelia/users/leonardobergesio# ifconfig eth1 up
root@nodeA:/ofelia/users/leonardobergesio# vconfig add eth1 300
Added VLAN with VID == 300 to IF -:eth1:-
root@nodeA:/ofelia/users/leonardobergesio# ifconfig eth1.300 10.10.1.1/24 up
root@nodeA:/ofelia/users/leonardobergesio# ifconfig
eth0      Link encap:Ethernet  HWaddr 02:03:00:00:00:0c
          inet addr:10.216.12.29  Bcast:10.216.15.255  Mask:255.255.252.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:5048 errors:0 dropped:0 overruns:0 frame:0
          TX packets:269 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:247453 (241.6 KiB)  TX bytes:31225 (30.4 KiB)

eth1      Link encap:Ethernet  HWaddr 02:03:00:00:00:0d
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:8 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:1866 (1.8 KiB)

eth1.300  Link encap:Ethernet  HWaddr 02:03:00:00:00:0d
          inet addr:10.10.1.1  Bcast:10.10.1.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:8 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:1866 (1.8 KiB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
```

Figura B.7: Configuración de las interfaces del nodeA

En este punto el entorno para la experimentación es completamente funcional. Fig. B.8 y Fig. B.9 se observan las pruebas de conectividad entre el nodeA y el nodeB, y el nodeB y el nodeC. Esto es posible gracias a la conectividad L2 existente y requerida por la Shim DIF sobre Ethernet.

```
root@nodeA:/ofelia/users/leonardobergesio# ping 10.10.1.2
PING 10.10.1.2 (10.10.1.2) 56(84) bytes of data.
64 bytes from 10.10.1.2: icmp_req=1 ttl=64 time=51.0 ms
64 bytes from 10.10.1.2: icmp_req=2 ttl=64 time=21.0 ms
64 bytes from 10.10.1.2: icmp_req=3 ttl=64 time=21.0 ms
64 bytes from 10.10.1.2: icmp_req=4 ttl=64 time=21.1 ms
64 bytes from 10.10.1.2: icmp_req=5 ttl=64 time=21.2 ms
64 bytes from 10.10.1.2: icmp_req=6 ttl=64 time=25.7 ms
64 bytes from 10.10.1.2: icmp_req=7 ttl=64 time=21.2 ms
64 bytes from 10.10.1.2: icmp_req=8 ttl=64 time=21.3 ms
64 bytes from 10.10.1.2: icmp_req=9 ttl=64 time=21.3 ms
^C
--- 10.10.1.2 ping statistics ---
9 packets transmitted, 9 received, 0% packet loss, time 8009ms
rtt min/avg/max/mdev = 21.028/25.027/51.075/9.320 ms
root@nodeA:/ofelia/users/leonardobergesio#
```

Figura B.8: Test de conectividad entre el nodeA y el nodeB

```
root@nodeC:/ofelia/users/leonardobergesio# ping 10.10.2.2
PING 10.10.2.2 (10.10.2.2) 56(84) bytes of data.
64 bytes from 10.10.2.2: icmp_req=7 ttl=64 time=41.5 ms
64 bytes from 10.10.2.2: icmp_req=8 ttl=64 time=18.5 ms
64 bytes from 10.10.2.2: icmp_req=9 ttl=64 time=47.9 ms
64 bytes from 10.10.2.2: icmp_req=10 ttl=64 time=43.6 ms
64 bytes from 10.10.2.2: icmp_req=11 ttl=64 time=32.6 ms
64 bytes from 10.10.2.2: icmp_req=12 ttl=64 time=37.4 ms
64 bytes from 10.10.2.2: icmp_req=13 ttl=64 time=33.1 ms
64 bytes from 10.10.2.2: icmp_req=14 ttl=64 time=31.5 ms
64 bytes from 10.10.2.2: icmp_req=15 ttl=64 time=30.7 ms
64 bytes from 10.10.2.2: icmp_req=16 ttl=64 time=25.1 ms
^C
--- 10.10.2.2 ping statistics ---
16 packets transmitted, 10 received, 37% packet loss, time 15077ms
rtt min/avg/max/mdev = 18.533/34.224/47.956/8.326 ms
root@nodeC:/ofelia/users/leonardobergesio#
```

Figura B.9: Test de conectividad entre el nodeB y el nodeC

Apéndice C. Valoración económica

Como ya se explicó anteriormente este PFM tiene como objetivo el diseño y desarrollo de un prototipo de la arquitectura RINA presentada por John Day en el kernel de Linux sobre Ethernet. El trabajo se enmarca dentro del proyecto IRATI que es llevado a cabo por un consorcio europeo financiado por el Séptimo Programa Marco (FP7) de la Unión Europea. Dadas las características del proyecto real, la valoración económica presentada en este reporte se corresponde con una simulación donde el proyecto valorado corresponde a cuatro meses de trabajo realizado por una compañía española residente en Barcelona con el objetivo de un primer prototipo que presente unas funcionalidades básicas (primera fase de IRATI aproximadamente).

C.1 Recursos humanos

El equipo utilizado para llevar a cabo el proyecto en esta simulación para el PFM se corresponde en gran medida con el equipo real, aunque considerando a todos ellos residentes en Barcelona. Los roles integrantes son:

- 1 Gestor del proyecto: Se encarga de la gestión del proyecto, recursos humanos y materiales y de todo lo relacionado al reporte del trabajo hecho ante responsables superiores. Eventualmente realiza labores técnicas debido a su formación profesional.
- 1 Arquitecto de software: Se encarga de gestionar el equipo de desarrolladores a nivel de tareas y lidera el diseño del prototipo. También realiza tareas de codificación.
- 2 Desarrolladores a tiempo completo: Principalmente desarrollo de código, testing y participación en el diseño.
- 1 Desarrollador a tiempo parcial: Ídem que los anteriores pero a tiempo parcial.

La Tabla C.1 describe los gastos supuestos por los recursos humanos necesarios para el proyecto. Así mismo, también describe los costes indirectos que suponen gastos de alquiler de oficinas, limpieza, servicios, infraestructura, etc. Se calculan como el 30 % del coste del salario de cada trabajador. La Fig. C.1 muestra la proporción entre los salarios de los trabajadores (sin considerar costes indirectos) sobre el total.

	Sueldo anual bruto	Coste empresa (38.4 %)	Total anual	Total duración proyecto (4 meses)	Costes Indirectos (30 %)
Gestor	32000€	11136€	43136€	14.378,67€	4.313,60€
Arquitecto	30000€	10440€	40440€	13.480,00€	4.044,00€
Desarrollador TC 1	26000€	9048€	35048€	11.682,67€	3.504,80 €
Desarrollador TC 2	26000€	9048€	35048€	11.682,67€	3.504,80 €
Desarrollador TP 1	14000€	4872€	18872€	6.290,67€	1.887,20 €
TOTAL				57.514,67 €	17.254,40 €

Tabla C.1: Costes de personal para la realización del proyecto

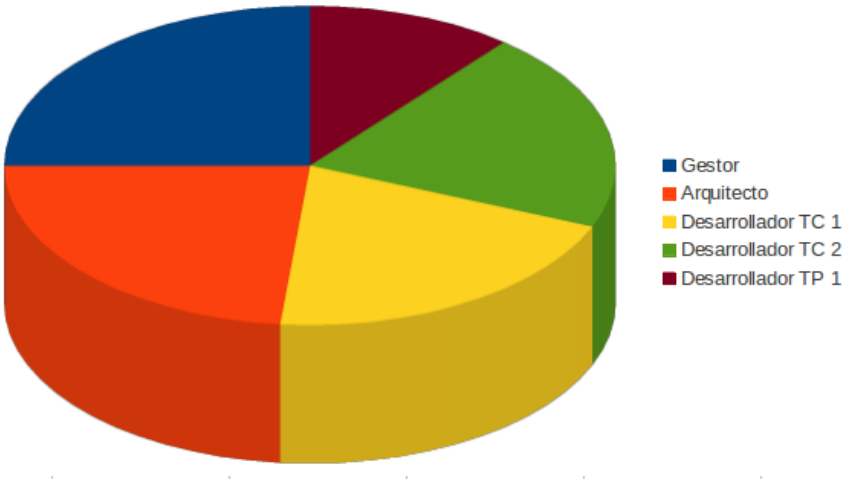


Figura C.1: Gráfico de costes de personal

C.2 Recursos materiales

En esta sección se realiza un análisis de los equipos necesarios para llevar a cabo el proyecto. Una vez identificados, se listan los modelos específicos obtenidos del análisis de las opciones disponibles.

A continuación se listan los equipos necesarios así como la justificación de su uso:

1 Servidor de compilación y testing : Será necesario un servidor capaz de albergar 3 máquinas virtuales por desarrollador, sin contar el gestor, éstas serían 12 máquinas virtuales. El numero viene determinado por el use case a testear (más sencillo que el presentado en OFELIA) que requiere 3 VMs. Por otro lado, el servidor se justifica también para aprovechar la potencia de procesado y memoria RAM dado que el proceso de compilación del kernel es costoso y conlleva tiempo. La compilación repetida en un ordenador portátil se haría más tediosa y duradera aunque será imposible evitarla.

Ordenadores portátiles: Uno por trabajador como herramienta principal de trabajo. Dentro de las posibilidades, una capacidad de procesado elevada es requerida para las tareas de compilación.

Pantallas, teclado, ratón y juego de auriculares y micrófono: Necesario para la disposición del escritorio de trabajo y la posibilidad de tener varias ventanas de código abiertas simultáneamente. Los auriculares y micrófonos serán necesarios para la comunicación del equipo mediante Skype o similares.

Testbed OFELIA: Como ya se comentado anteriormente el testbed de OFELIA también se utilizará como entorno de pruebas del código producido y su uso actualmente es gratuito.

Sofware: El software necesario será el reportado en anteriores PACs para poder llevar a cabo las tareas, en conjunto con aplicaciones típicas de ofimática, comunicación, etc. Todo el software será open source o en su defecto gratuito.

Los modelos específicos seleccionados son:

Equipo	Características
Servidor SUPERSERVER SYS-5017R-MTF SUPERMICRO	Intel Sandy Bridge 4C E5-2609 2.4G 10M 6.4 1066MHZ 4 x 8GB DDR3-1600 2Rx4 ECC REG 2 x SATAIII 1TB ST1000NM0011 SEAGATE 7200 64MB Garantía 3 años in situ
Mac Book Pro 13	Core i5 de Intel de doble núcleo a 2,5 GHz Turbo Boost de hasta 3,1 GHz 4 GB de memoria a 1.600 MHz Disco duro de 500 GB a 5.400 rpm1 HD Graphics 4000 de Intel Batería integrada (7 horas)
Dell XPS 13	Procesador Intel® Core™ i7-3687U 8 GB2 Un solo canal SDRAM DDR3 a 1600 MHz Sistema operativo Ubuntu Edition version 12.04 Pantalla WLED 1080p 13,3" con tecnología TrueLife Disco duro SSD de 512 GB Tarjeta gráfica Intel® HD 4000 1 año de servicio ProSupport y servicio in situ Peso de 1.36 kg
Pantalla Dell UltraSharp U2412M	24"(60,96 cm)(16:10) Panel IPS, antirreflejo con capa resistente 3H Resolución 1920 x 1200 a 60 Hz Relación de contraste: 1000:1 (típica) 2.000.000:1(dinámico) Brillo 300 cd/m2 Tiempo de respuesta 8 ms Ángulo máx. de visión (vertical/horizontal) 178°/178° 16,7 millones de colores Espectro de colores 82 % (CIE 1976) 1 conector DVI-D con HDCP 1 DisplayPort (DP) 1 VGA 1 puerto USB 2.0 de entrada 4 puertos USB 2.0 de salida (hub) Conector de alimentación de CC para la barra de sonido Dell
Auriculares con micrófono Turtle Beach Ear Force ZLA	Altavoces de 40 mm de diámetro con imanes de neodimio Respuesta de frecuencia 20Hz - 20kHz Longitud del cable 6 pies Audio / Mic Conexión 3.5mm, 4 polos Cable adaptador PC 3.5mm doble

Tabla C.2: Especificaciones de los equipos elegidos

Finalmente, el análisis de costes materiales se pueden ver en Tabla C.3.

Equipo	Cantidad	Precio Unitario	IVA Unitario	Total sin IVA
Servidor SUPERSERVER SYS-5017R-MTF SUPERMICRO	1	1.770,08€	371,72€	1.770,08€
MacBook Pro 13	1	970,91€	258,09€	970,91€
Dell XPS 13	4	1.119,00€	234,99€	4.476,00€
Pantalla Dell UltraSharp U2412M	5	208,26€	43,73€	1.041,30€
Teclado y ratón Logitech Desktop MK120	5	13,18€	2,77€	65,90€
Auriculares con micrófono Turtle Beach Ear Force ZLA	5	14,01€	2,94€	70,05€
Testbed OFELIA	-	-	-	-
Software	-	-	-	-
TOTAL				8394,24€

Tabla C.3: Costes de equipos para la realización del proyecto

C.3 Diseminación

Los costes de diseminación corresponden a aquellos gastos destinados a cubrir toda acción referente a la comunicación de los resultados o el trabajo del proyecto con tal de aumentar su visibilidad, y con esto asegurar su continuidad y adopción. Entre estos gastos se consideran asistencias a congresos, eventos, etc. esto supone gastos de viaje, hospedaje, entradas, etc. De manera poco estricta se considera 1 viaje para dos personas cada mes en asistencias a eventos más la organización de un workshop sobre RINA. En total 6000€.

C.4 Resumen

La Tabla C.4 resume los costes totales identificados en la valoración económica y el gráfico de Fig. C.2 muestra la relación de cada uno respecto al total de los 89.163,31€calculados.

Costes personal	57.514,67€
Costes indirectos	17.254,40€
Costes equipos	8.394,24€
Costes diseminación	6.000,00€
TOTAL	89.163,31€

Tabla C.4: Tabla resumen de costes de la valoración económica

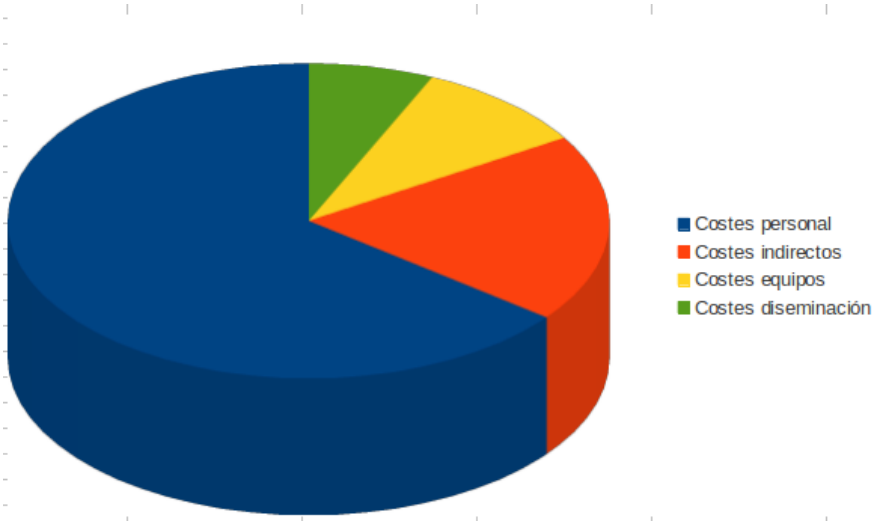


Figura C.2: Gráfico de resumen de costes

GNU Free Documentation License

Preamble

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "**Document**", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "**you**". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "**Modified Version**" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "**Secondary Section**" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "**Invariant Sections**" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not

allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "**Cover Texts**" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "**Transparent**" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "**Opaque**".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "**Title Page**" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

The "**publisher**" means any person or entity that distributes copies of the Document to the public.

A section "**Entitled XYZ**" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "**Acknowledgements**", "**Dedications**", "**Endorsements**", or "**History**".) To "**Preserve the Title**" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute.

However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer

review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements".

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and will automatically terminate your rights under this License.

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See `http://www.gnu.org/copyleft/`.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software

Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation. If the Document specifies that a proxy can decide which future versions of this License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Document.

11. RELICENSING

"Massive Multiauthor Collaboration Site" (or "MMC Site") means any World Wide Web server that publishes copyrightable works and also provides prominent facilities for anybody to edit those works. A public wiki that anybody can edit is an example of such a server. A "Massive Multiauthor Collaboration" (or "MMC") contained in the site means any set of copyrightable works thus published on the MMC site.

"CC-BY-SA" means the Creative Commons Attribution-Share Alike 3.0 license published by Creative Commons Corporation, a not-for-profit corporation with a principal place of business in San Francisco, California, as well as future copyleft versions of that license published by that same organization.

"Incorporate" means to publish or republish a Document, in whole or in part, as part of another Document.

An MMC is "eligible for relicensing" if it is licensed under this License, and if all works that were first published under this License somewhere other than this MMC, and subsequently incorporated in whole or in part into the MMC, (1) had no cover texts or invariant sections, and (2) were thus incorporated prior to November 1, 2008.

The operator of an MMC Site may republish an MMC contained in the site under CC-BY-SA on the same site at any time before August 1, 2009, provided the MMC is eligible for relicensing.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

> Copyright © YEAR YOUR NAME. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with ... Texts." line with this:

> with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

Bibliografía

[1] IRATI
urlhttp://irati.eu/

[2] Seventh Framework Programme for Research and Technological Development
http://cordis.europa.eu/fp7

[3] Fundació i2CAT - http://www.i2cat.net

[4] Pouzin Society
http://www.pouzinsociety.org/

[5] John Day
http://en.wikipedia.org/wiki/John_Day_(computer_scientist)

[6] John Day. Patterns in Network Architecture: A Return to Fundamentals. Prentice Hall, 2008

[7] L. Pouzin. Presentation and major design aspects of the CYCLADES computer network. Proceedings of the NATO Advanced Study Institute on Computer Communication Networks, páginas 415–434, 1973.

[8] Distributed Data Processing Group Digital Equipment Corporation. Decnet technical summary, 1980.

[9] Xerox Systems. Internet transport protocols, xsis 028112, 1981.

[10] The RINA Specification Handbook, Pouzin Society. Enero 2013

[11] IRATI D2.1 - First phase use cases updated RINA specifications and high-level software architecture - http://irati.eu/wp-content/uploads/2012/07/IRATI-D2.1.pdf

[12] Leonardo Bergesio, TFM PAC 2 Prototipo de RINA sobre Ethernet, Análisis de requerimientos y viabilidad. Q2 2012-2013

[13] CDAP - Common Distributed Application Protocol Reference (available on demand) - PNA Technical Draft D-Base-2010-xxy, draft 0.7.2, December 2010

[14] Google Protocol Buffers Developers Guide - https://developers.google.com/protocol-buffers/

[15] RFC 3549 Linux Netlink as an IP Services Protocol - http://www.ietf.org/rfc/rfc3549.txt

[16] R. Watson. Timer-based mechanism in reliable transport protocol connection management. ComputerNet works, 5:47–56, 1981

[17] RFC-826, An Ethernet Address Resolution Protocol – http://tools.ietf.org/search/rfc826

[18] GNU automake – http://www.gnu.org/software/automake

[19] GNU autoconf – http://www.gnu.org/software/autoconf

[20] GNU libtool – http://www.gnu.org/software/libtool

[21] pkg-config – http://www.freedesktop.org/wiki/Software/pkg-config

[22] SWIG: The Software Wrapper and Interface Generator – http://swig.org

[23] GNU Make – http://www.gnu.org/software/make

[24] git – http://git-scm.com/

[25] ncurses - http://www.gnu.org/software/ncurses/

[26] CVim - http://www.vim.org/scripts/script.php?script_id=213

[27] The Linux Kconfig configuration framework - https://www.kernel.org/doc/Documentation/kbuild/kconfig.txt

[28] The Linux Kbuild building framework - https://www.kernel.org/doc/Documentation/kbuild/makefiles.txt

[29] Issues tracker de IRATI (disponible bajo pedido) - https://github.com/dana-i2cat/irati/issues

[30] OFELIA FP7 Project
http://www.fp7-ofelia.eu/

[31] E. Grasa, E. Trouva, S. Bunch, M. Ponce de Leon, J. Day, P. deWolf; "eveloping a RINA prototype over UDP/IP using the TINOS framework", CFI 2012 , South Korea, September 2012.

[32] Open Network Foundation
https://www.opennetworking.org/

[33] OFELIA Control Framework - http://fp7-ofelia.github.io/ocf/

[34] NOX OpenFlow Controller - http://www.noxrepo.org/

[35] Leonardo Bergesio, Pr. Externas PAC 1 Plan de Trabajo. Q2 2012-2013

[36] Manual de Usuario de OFELIA - https://alpha.fp7-ofelia.eu/doc/index.php/Main_Page

[37] What is OpenFlow? - http://archive.openflow.org/wp/learnmore/

[38] - Portal web de OFELIA - https://register.fp7-ofelia.eu/login/login

[39] Software Defined Networking
http://en.wikipedia.org/wiki/Software-defined_networking

- FIN DEL DOCUMENTO -