Prototipo de RINA sobre Ethernet

PFM Software Libre, Administración de Redes y Sistemas Operativos

Autor: Leonardo Bergesio Consultor: Miguel Martín Mateo Tutor Externo: Eduard Grasa

ÍNDICE

- Entorno del proyecto
- Breve introducción a RINA
- Objetivos
- Requerimientos
- Diseño
- Desarrollo
- Testing y validación
- Resultados y conclusiones

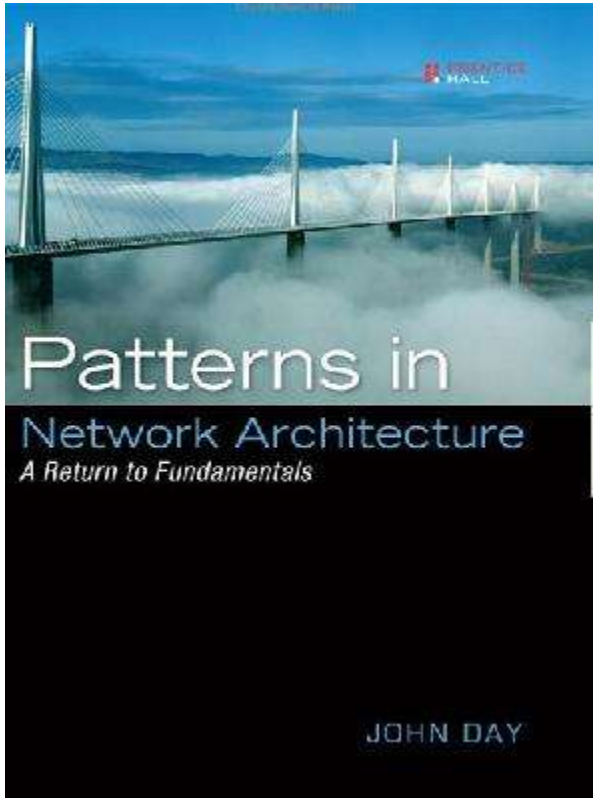
ENTORNO DEL PROYECTO

La Fundació i2CAT es un centro de investigación e innovación, que centra sus actividades en el desarrollo de la internet del futuro

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). El objetivo general de IRATI es conseguir una mayor compresión y exploración de RINA

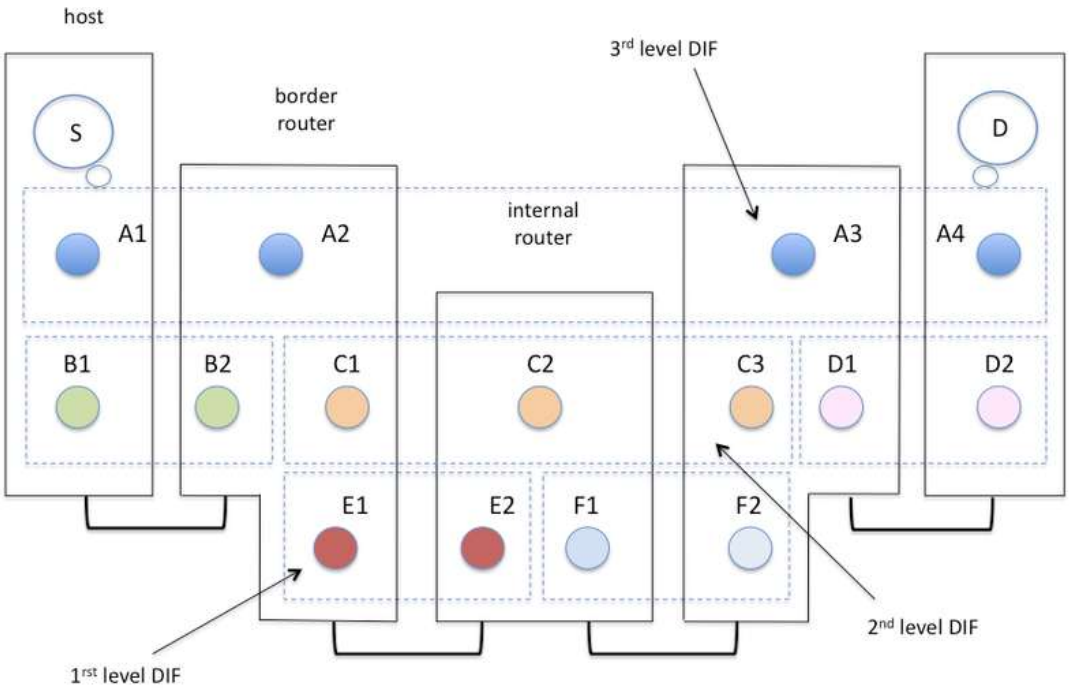
BREVE INTRODUCCIÓN A RINA

- La Recusive InterNetwork Architecture (RINA) es una arquitectura de red propuesta por John Day en su libro Patterns in Network Architecture: A Return to Fundamentals



- Presenta una arquitectura recursiva, diseñada desde cero, basada en una sólida teoría fundamental de networking y con un solo principio básico: networking es sólo IPC

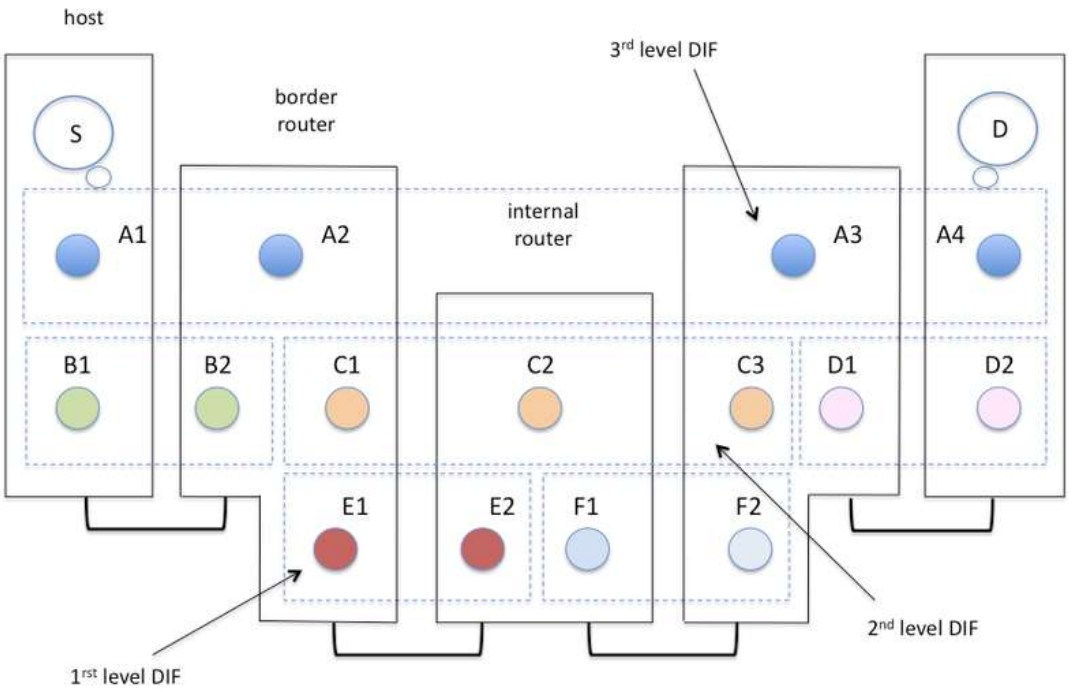
La arquitectura de RINA



- Plantea una estructura recursiva de capas que proveen servicios de IPC(**Inter Process Communication**) a las aplicaciones de la capa superior
- Sólo existe **un único tipo de capa** que se **repite** tantas veces como decida el diseñador de la red
- **Separación de mecanismos y políticas**

- **Todas las capas tienen las mismas funciones**, con distinto abasto y rango.
  - No todas las capas pueden necesitar todas las funciones, pero no requerirán más
- Una **capa** es una **Aplicación distribuida que realiza y gestiona IPC**.
  - Una Distributed IPC Facility (DIF)
- Esto conlleva a una teoría y una **arquitectura** que **escala indefinidamente**,
  - Cualquier límite impuesto no es una propiedad de la arquitectura en sí.

Nombres y direcciones en RINA

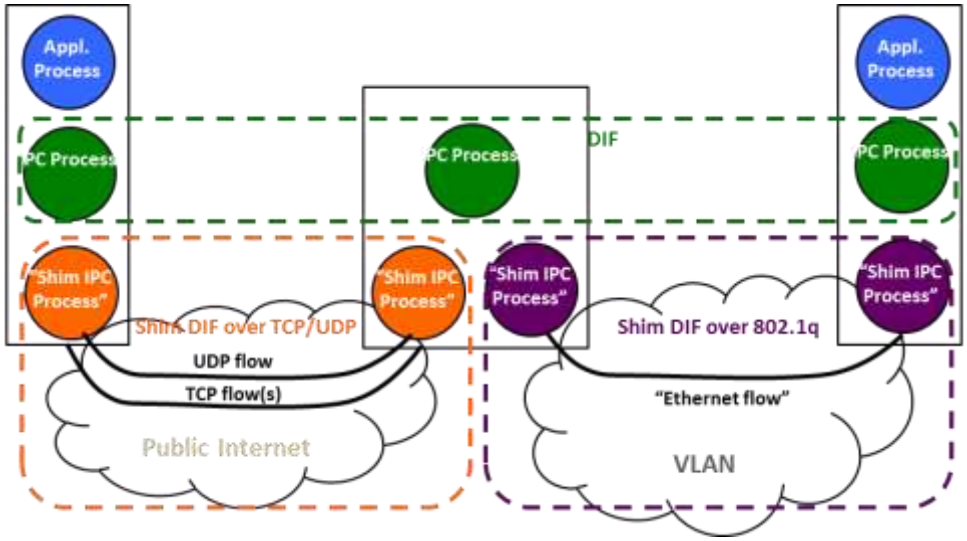


- Todos los procesos de aplicación (incluídos los IPC Processes) tienen un nombre único que los identifica en el espacio de nombres de aplicaciones..
- Con tal de facilitar su operación dentro de la DIF, cada IPC Process en una DIF obtiene un sinónimo que puede tener significado topológico dentro de la DIF: una dirección.

- El abasto de la dirección es la propia DIF, no son visibles fuera de la DIF.
- Cada DIF tiene un directorio que mapea el nombre de la aplicación de destino al nombre del IPC Process en una DIF mediante el cual se llega a la aplicación.
- Dado que la arquitectura es recursiva, las aplicaciones, nodos y Points of Attachment (PoAs) son relativos:
  - Para una DIF de nivel N, el proceso en la capa N+1 es una aplicación, y el proceso en la capa N-1 es un PoA.

La Shim DIF

- Es una capa que se posiciona sobre una capa no RINA (p.ej. Cable, Internet, Ethernet) y presenta una API RINA (quizá parcial) para que la aplicación encima pueda tratarla como una DIF normal.



- El IPC Process es una aplicación RINA poco demandante, sólo necesita flows no fiables a sus vecinos para funcionar.

- Las Shim DIFs actualmente siendo trabajadas son:
  - Shim DIF sobre TCP/UDP
  - Shim DIF sobre 802.1q (VLANs)
- Las principales responsabilidades de una Shim DIF son:
  - Mapear nombres de aplicaciones de la capa N+1a direcciones dentro de la DIF.
  - Crear y destruir flows dentro de la shim DIF
    - El flow es un recurso de comunicación bien definido en la DIF (p.ej: conexión TCP, UDP, conjunto de tramas Ethernet con las mismas MACs de origen/destino, etc)

OBJETIVOS

- Prototipo open source de RINA sobre Ethernet para un Operating System (OS) UNIX.
- Mejora del modelo de referencia de la arquitectura RINA y sus especificaciones, centrándose en la DIF sobre Ethernet.
- Validación experimental del prototipo y de RINA y comparación con TCP/IP.
- Interoperabilidad con el prototipo RINA sobre UDP/IP de la Pouzin Society (PSOC)

REQUERIMIENTOS

Los requerimientos provienen de 4 apartados:

- **Especificaciones y modelo de referencia**
- **Interoperabilidad**: con otros prototipos existentes.
- **Diseño, implementación y guías de referencia**: Reúne aquellos requerimientos impuestos por cuestiones de diseño, restricciones de la implementación, optimizaciones del funcionamiento del sistema, etc.
- **Casos de uso**: escenarios que marcan la manera en que debe funcionar el prototipo

La memoria contempla los 29 requerimientos generales identificados

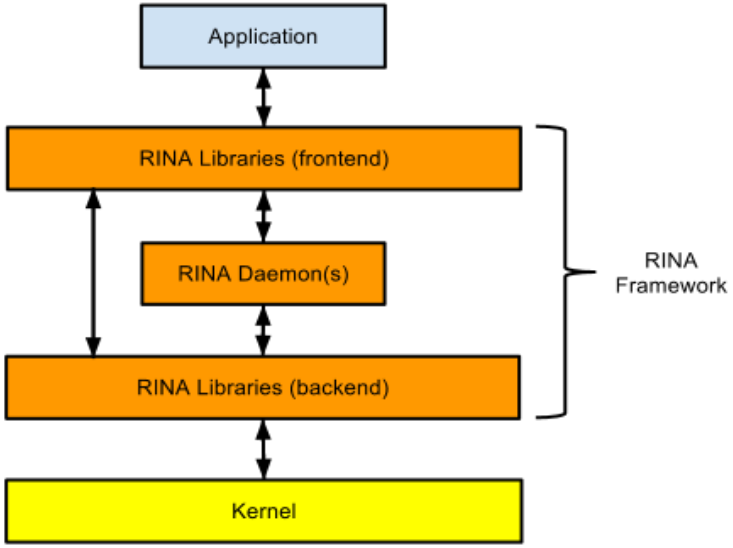
DISEÑO DE LA ARQUITECTURA DE SOFTWARE

- Descripción de los diferentes componentes y su relación en la implementación del prototipo de RINA sobre Ethernet para OS basados en UNIX.
- **Linux** es la plataforma escogida:
  - Es ámpliamente usado en distintos contextos
  - Es open source soportado por una gran comunidad y documentación

- La arquitectura (y por tanto la implementación) se extiende tanto al user-space como al kernel, dado que:
  - El fast-path: los problemas de rendimiento deben ser minimizados en aquellas tareas que se realizan muy frequentemente (leer/escribir datos) -> los componentes involucrados deben residir en el kernel
  - Se necesita acceder a los device drivers del hardware que se debe adaptar a RINA (ej: tarjetas Ethernet), lo que implica desarrollar en el kernel.
  - El slow-path: las tareas de administración y configuración no necesitan residir en el kernel.

El framework de software

- Los componentes se han organizado en 3 categorías.
  - Daemons (en user-space)
  - Librerias (en user-space)
  - Componentes del kernel



- Las librerías abstraen los detalles de la comunicación entre los componentes del user-space y entre el user-space y el kernel

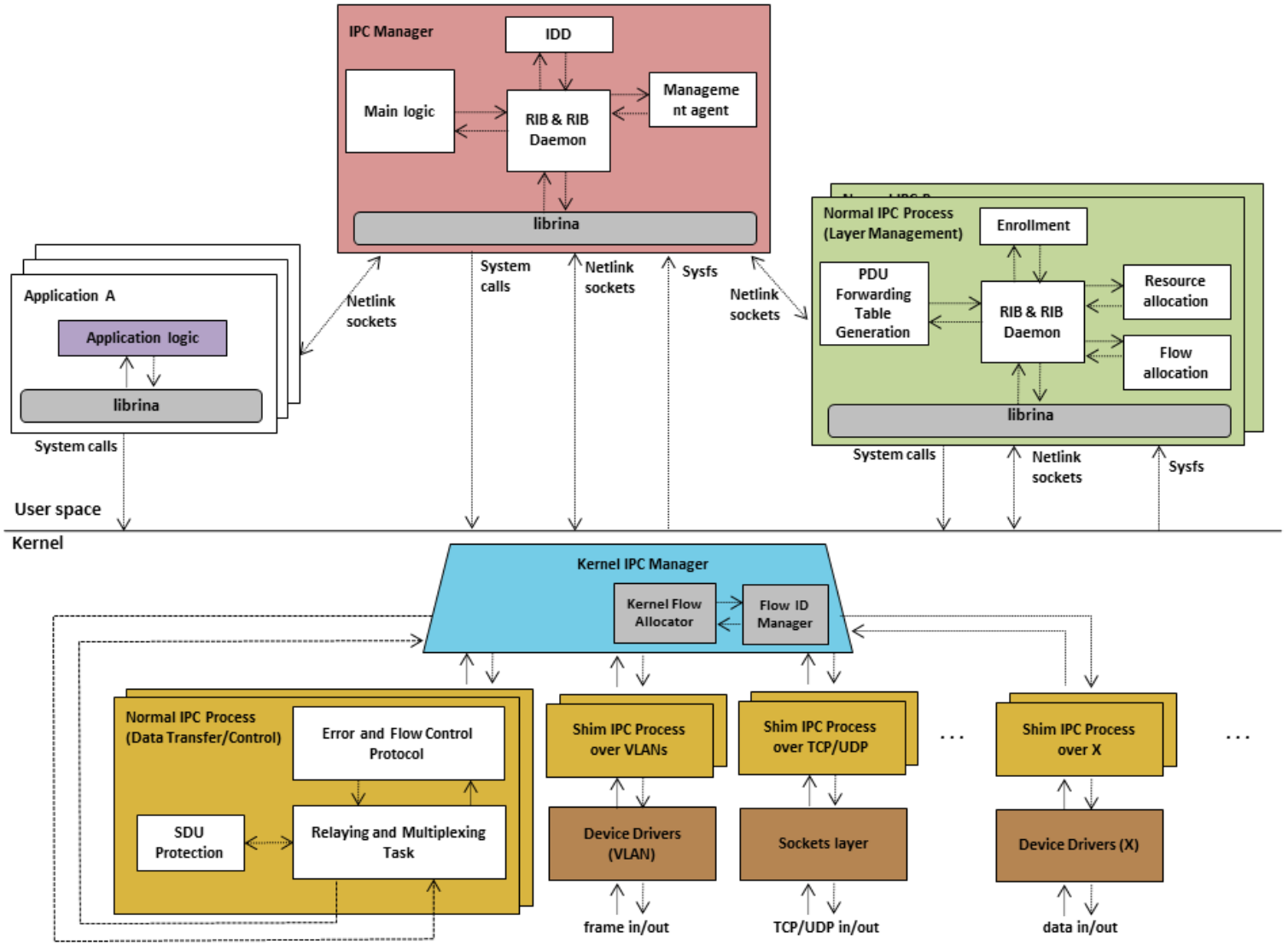
- Los componentes del kernel implementan las partes de "transmisión de datos" y "control de la transmisión de datos" de los IPC Processes normales y las shim DIFs ("fast path")

- Los dameons implementan la "capa de administración" de los IPC Processes y el IPC Manager.

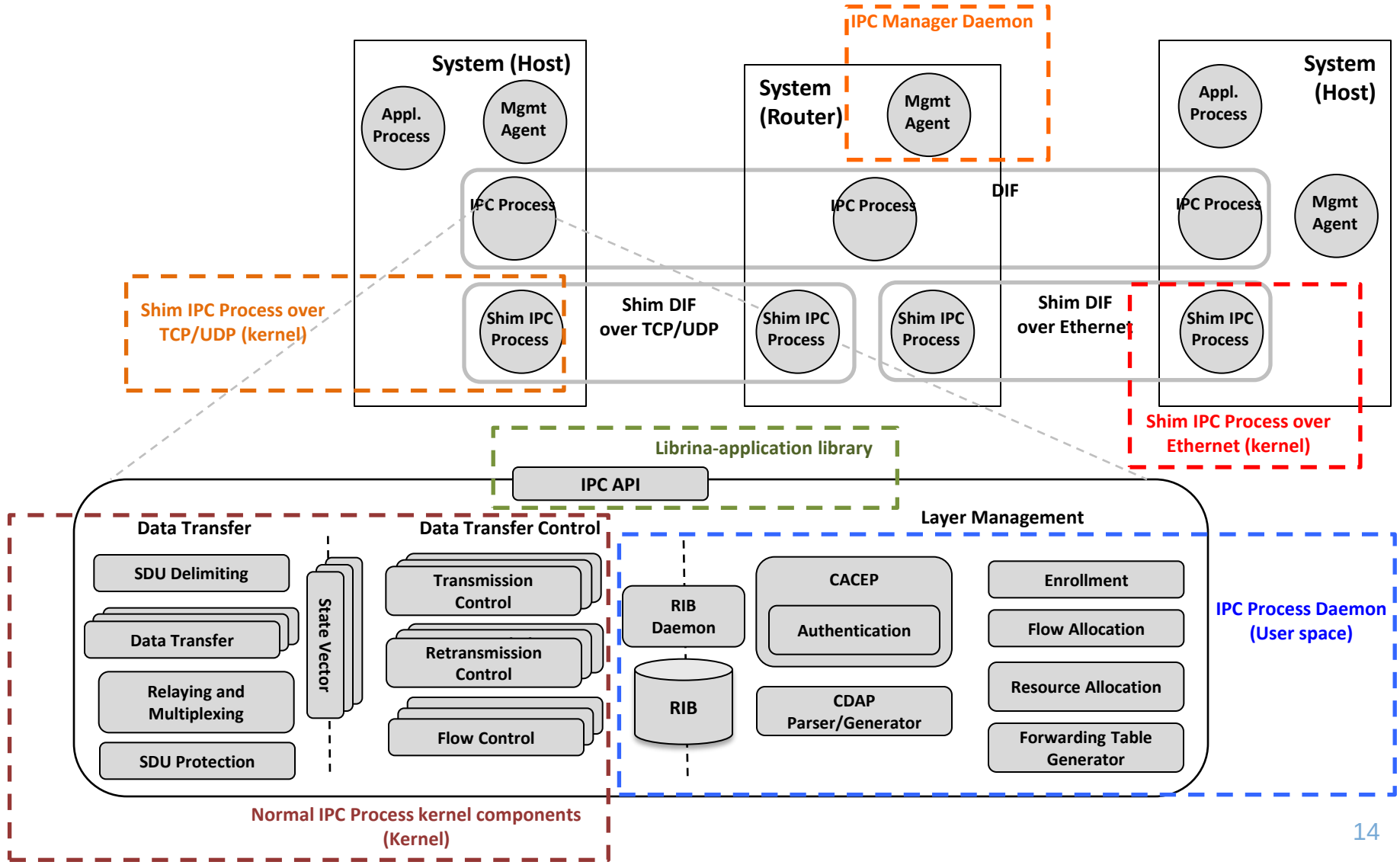
Comunicación entre componentes

- **Netlink sockets (user-space <-> user-space/ user-space -> kernel / kernel -> user space)**
  - Cumple con todos los requerimientos (comunicación 1:N, N:M, asíncrona, iniciada en cualquiera de los espacios). Netlink soporta comunicación desde el kernel a user-space gracias a sus canales full-duplex, lo que permite avisar a la aplicación sobre eventos ocurridos internamente en el kernel.
- **Llamadas a sistema (user-space -> kernel)**
  - Para acciones atómicas originadas por el usuario o pertenecientes al fast-path. Son más eficientes y consumen menos recursos. Síncronas.
- **Sysfs (configuración y monitorización del kernel)**
  - Permite configurar distintos componentes de RINA en el kernel y obtener estadísticas de su operación.

Arquitectura del prototipo

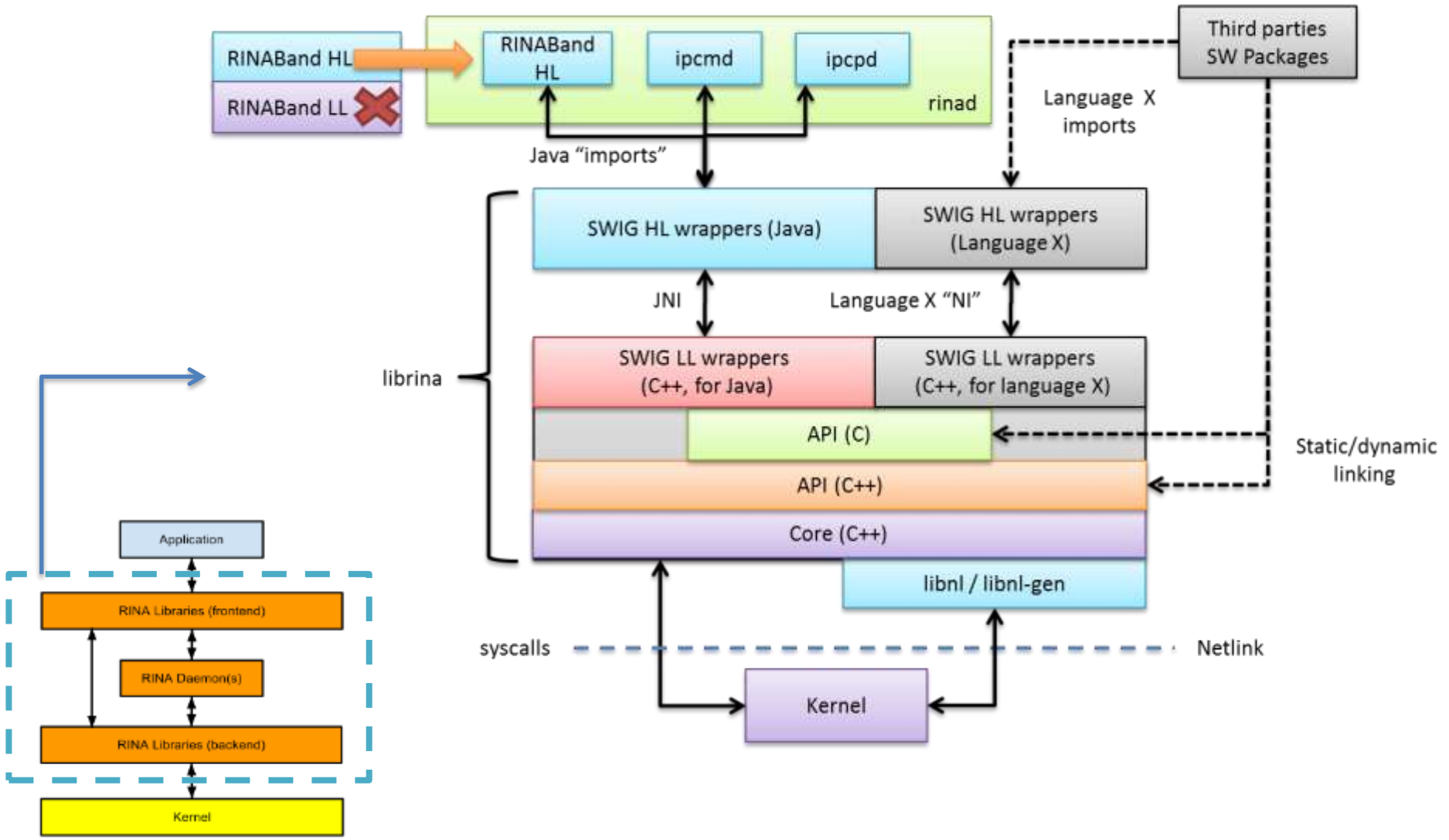


Mapping del modelo de RINA a la implementación del prototipo



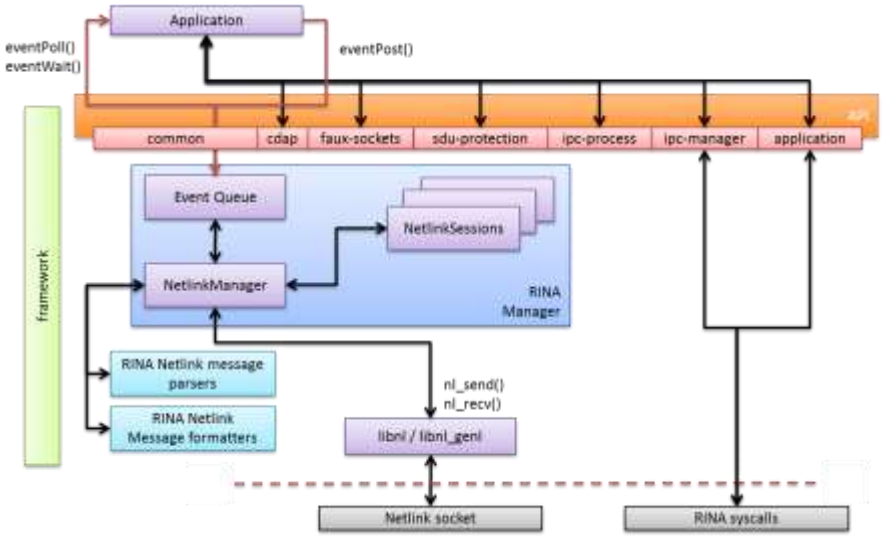
14

Componentes en user-space



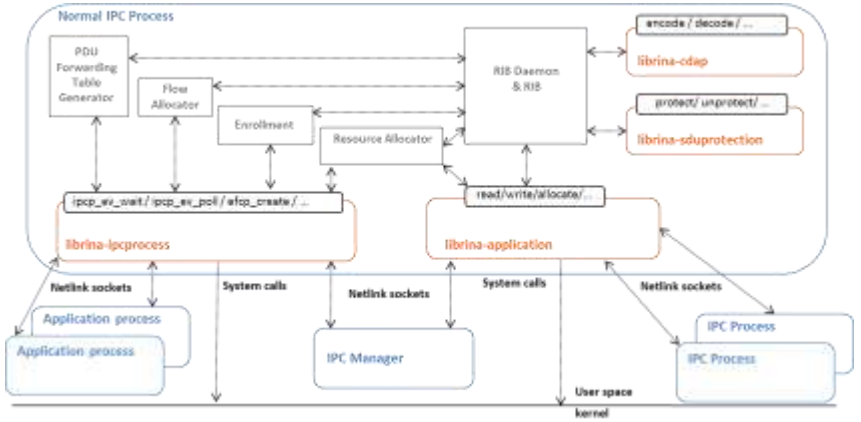
librina

- Abstrae las interacciones con el kernel (syscalls, Netlink)
- Provee la API RINA a las aplicaciones en user-space
- Se trata más de un framework/middleware que una simple librería (su propio modelo de datos y memoria)
- Basada en eventos
- Desarrollada en C++

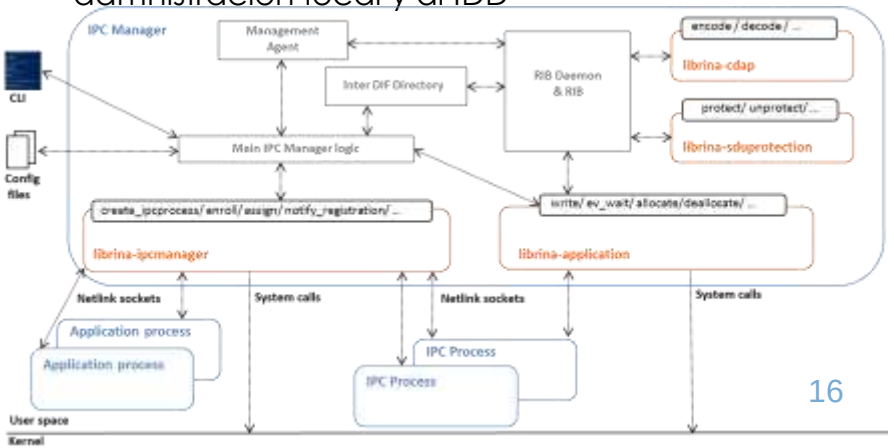


rinad

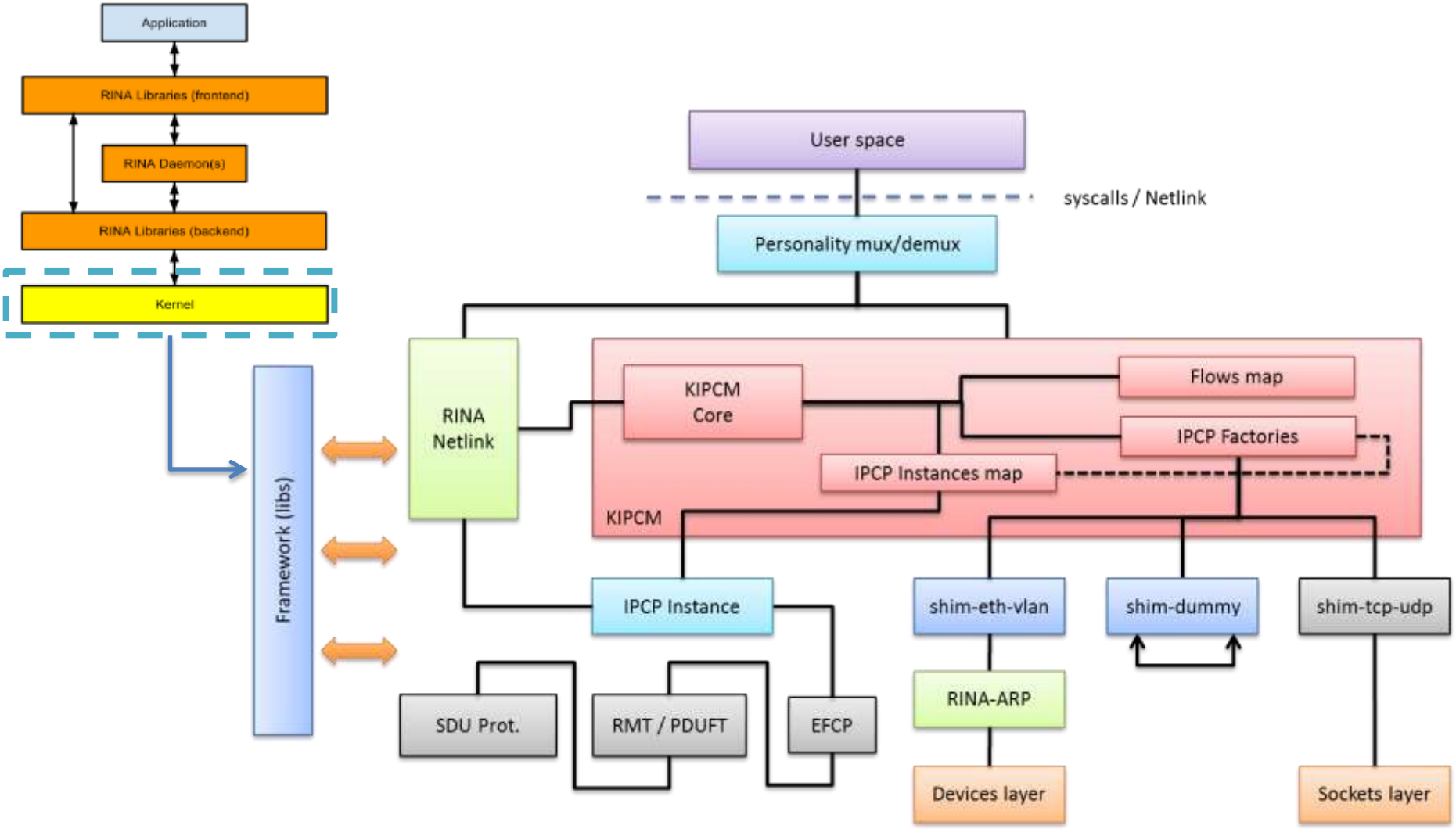
- **IPC process daemon (normal)**: Implementa la capa de administración del IPC Process "normal"(1 instancia por IPC Process normal)



- **IPC Manager daemon**: Administración y configuración de IPC Processes daemons y componentes del kernel; alberga al agente de admnistración local y al IDD



Componentes en el kernel



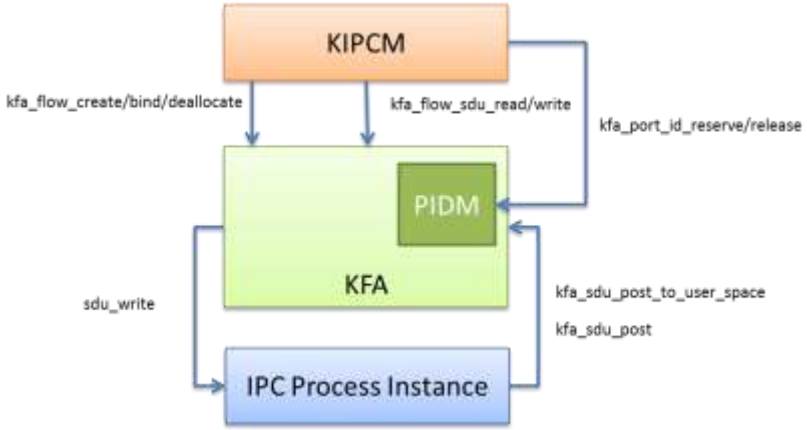
Personalities

KFA

- La Personality aspira a soportar distintas implementaciones del stack en el kernel de forma dinámica.
- Actúa como una capa de mux/demux entre las interfaces de user-space/kernel y la implementación concreta.
  - Tienen un id único
  - Demux user → kernel: las llamadas a las APIs se dirigen a la personality elegida (activa)
  - Mux kernel → user: Dado que los componentes tienen links a sus "padres" en una organizción jerárquica, la Personality es la raiz, y conecta a los componentes con la API entre espacios

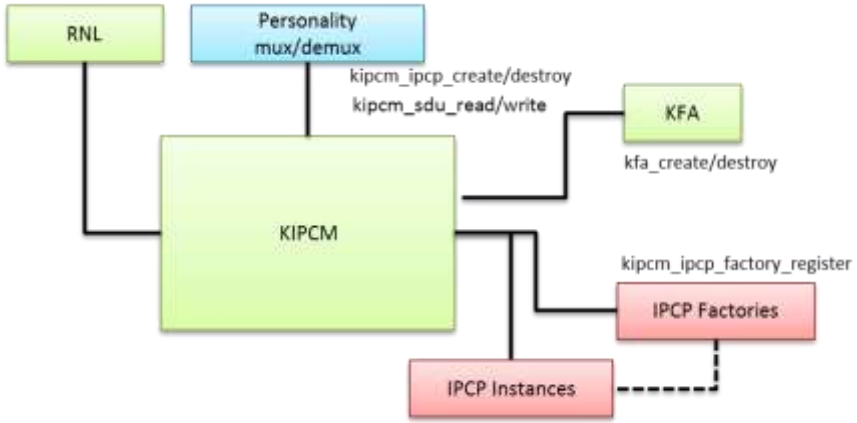
El Kernel Flow Allocator (KFA) se encarga de:

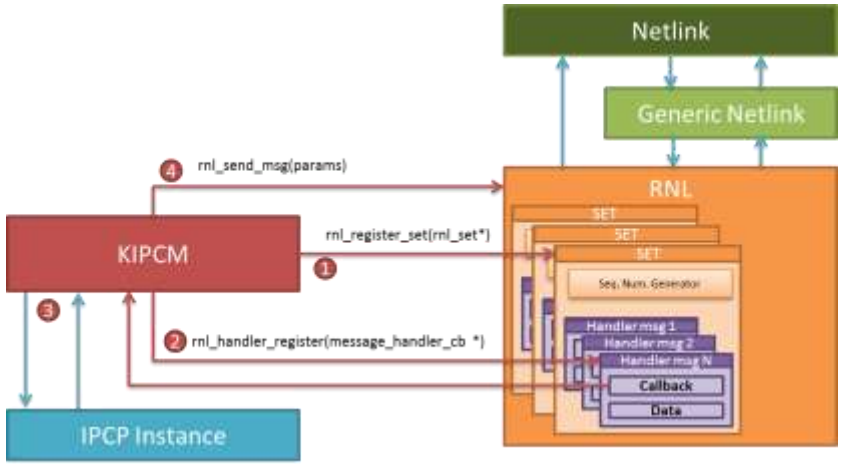
- Administracíon de flows
- Binding de IPC Processes y el KIPCM a través del flow
- Gestión de identificadores de puertos (portIDs)



La interfaz entre user-space y kernel: KIPCM + RNL

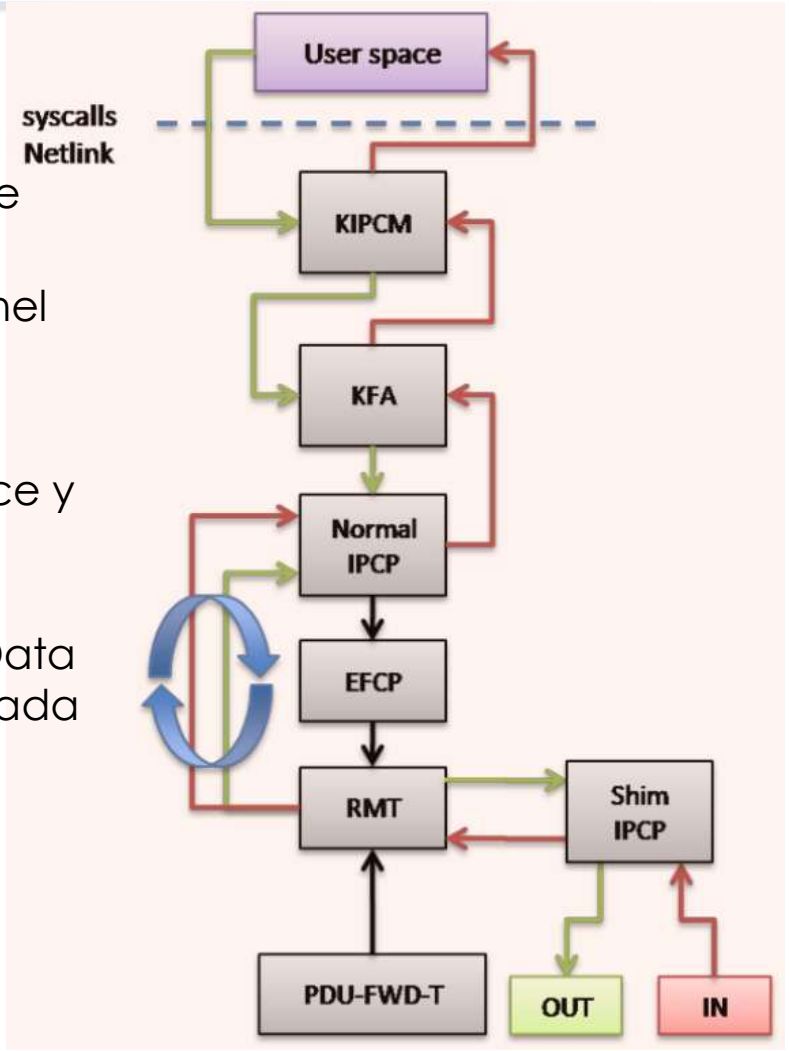
- Interfaz del Kernel = syscalls + Netlink
- El Kernel IPC Manager (KIPCM) gestiona las syscalls
- La RINA Netlink Layer (RNL) gestiona los mensajes de Netlink
- La distribución syscall/netlink se debe a:
  - Fast/slow paths
    - Netlink en el "slow-path" (principalmente configuración y adminstración)
    - Syscalls en el "fast-path" (read/write SDUs)
  - Bootstrapping requiere:
    - Syscalls para crear los componentes del kernel que luego utilizarán la capa de Netlink





El Core

- El KIPCM:
  1. Gestiona los IPC Processes
  2. Es el punto inicial donde "recursividad" se transforma en "iteración"
  3. Representante del IPC Manager en el kernel
- El KFA
  1. Gestiona flows
  2. Hace el binding entre IPC Process Instance y KIPCM
- Para llevar a cabo la recursividad, el KIPCM consulta al KFA quien inyecta/recibe Service Data Units (SDUs) de los elementos que iteran (por cada IPC Process):
  - OUT: IPCP → EFCP → RMT → PDU-FWD
  - IN: RMT → EFCP → IPCP

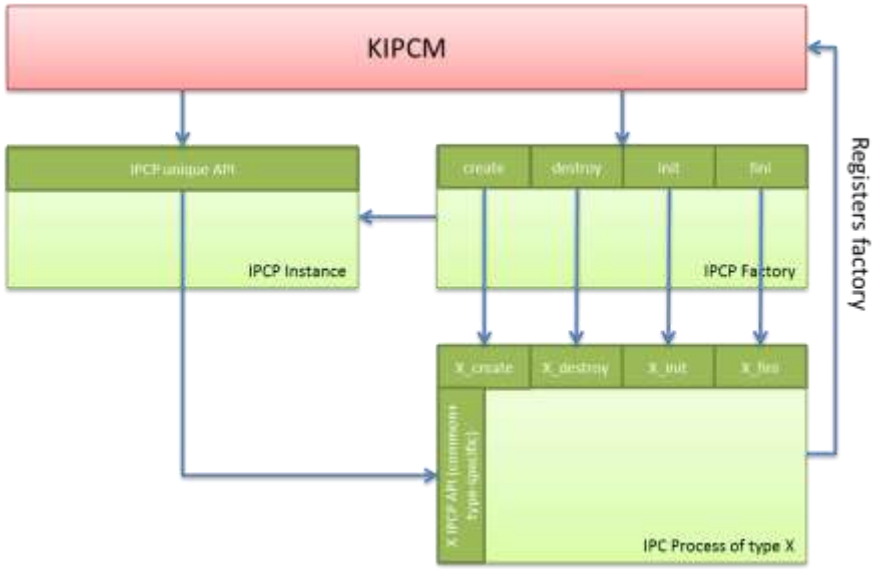


IPC Process Factories

Instances

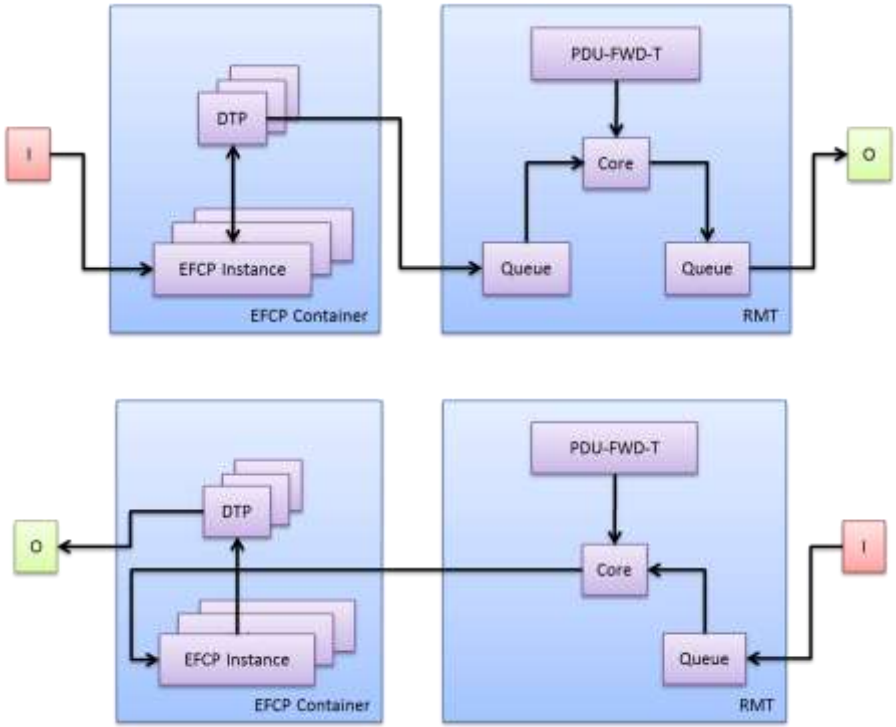
- Son utilizadas por los IPC Processes para publicar su disponibilidad.
- Se registran/desregistran ante el KIPCM aportando su nombre (tipo)
- Después del regustro, una factory publica la API de administración del IPC Process instance mediante la cual se pueden crear/destruir
- En la creación la Factory devuelve una instancia de IPC Process Instance que exporta una API común, pero esta ligada abstractamente a un IPC Process de un tipo concreto que tiene su API específica

- El método .create que proveen las Factories devuelven un objeto (instance) IPC Process
- Hay dos grandes tipos de IPC Process: "normal" y "shim"
- Independientemente del tipo, la instancia de IPC Process cumple:
  - La API es la misma
  - Cada IPC Process implementa su "core" (lógica interna)



IPC Processes:
El IPC Process normal Los shim IPC Process

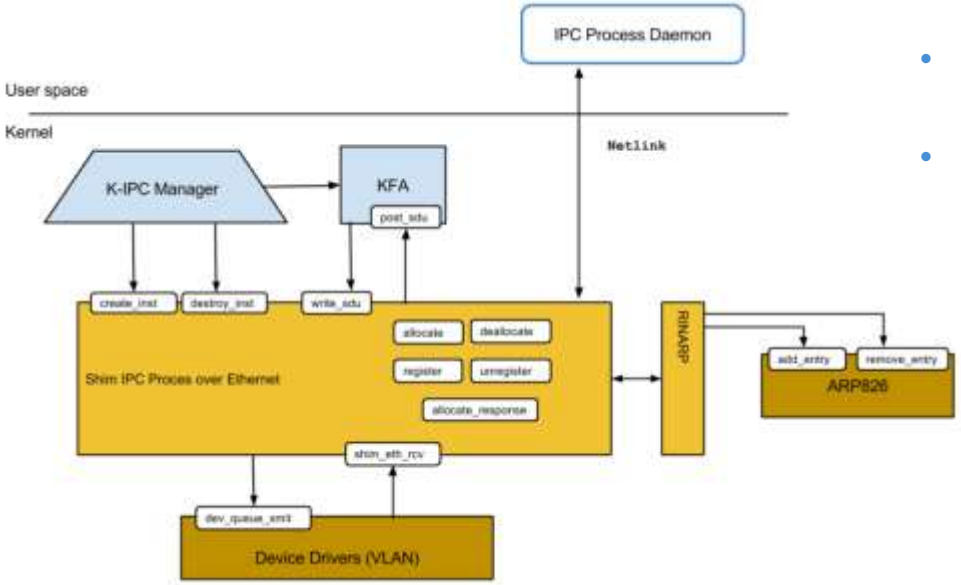
- El IPC Process normal en el kernel implementa aquellos módulos del fast-path que iteran durante la transferencia de SDUs/PDUs:
  - EFCP Container, EFCP (DTP+DTCP), RMT, PFT



- Se diseñaron 3 shims:
  - shim-dummy:
    - Es una shim para testing en un único host:
      - Emula una especie de interfaz "loopback" dentro de un único host
      - Sirve para testear y debuggear en un entorno de un único sistema.
  - shim-tcp-udp:
    - Representa el gateway RINA-TCP
    - Sólo diseño, no disponible
  - shim-eth-vlan.
    - Adapta Ethernet a RINA

La shim-eth-vlan

- Funcionalidad mínimas para adaptar Ethernet a RINA
- Una DIF es mapeada a una VLAN
- Utiliza ARP pero no la implementación de Linux
- Se comunica con los device drivers de las tarjetas
- Cumple las especificaciones desarrolladas durante el PFM



ARP826 y RINARP

- ARP de Linux está muy condicionado por IPv4:
  - Está limitado a la traducción de direcciones IP
  - Sólo se pueden asignar IPs (4bytes) a las interfaces
  - No hay manera de registrar handlers ante la recepción de requests/replies ARP (sólo el stack TCP/IP es avisado)
- Aunque se aceptaran estas limitaciones, IP y RINA no podrían convivir
- Se decide implementar un ARP que cumpla el estandar RFC-826
- RINARP es una capa de abstracción que permite:
  - El desarrollo en paralelo del shim IPC Process y de ARP826
  - Cambiar la implementación de ARP826 sin que el shim IPC Process se vea afectado
  - Ser utilizada por otros IPC Processes shim que utilizaran Ethernet (WIFI)

DESARROLLO

Entorno de desarrollo

- Una VM de VirtualBox con Debian 7 Wheezy y Linux v3.10.0
  - 1 CPU, 1024 MB RAM, 15GB disco duro
  - automake: Para generar makefiles
  - autoconf: Para generar los scripts de configuración
  - libtool: Para generar e instalar librerías.
  - pkg-config: Para la instalación de dependencias.
  - SWIG: 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 desarrollado en JAVA.
  - g++: Compilador de C++ para las librerías librina.
  - gcc: Compilador C.
  - gmake: Manipulador de dependencias.
  - git: Respositorio utilizado.
  - ncurses: Librerías para construir la herramienta Kconfig
  - Vim + Cvim: editor

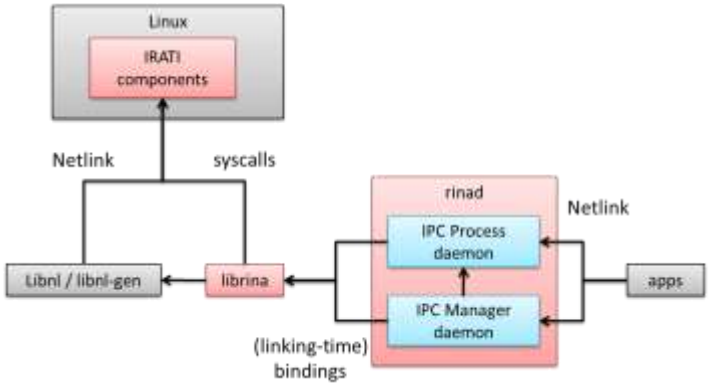
Repositorio y Github

- Ramas del repositorio central:
  - 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.
- Tags para las versiones: v<Mayor><Menor><Micro>
- Seguimientos de incidencias: Issue tracker de Github (gestión automática de SPRs)

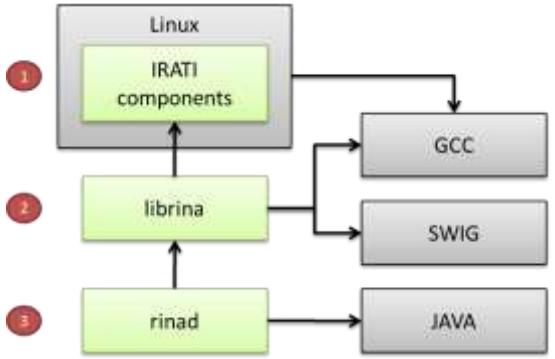
Paquetes, dependencias, compilación e instalación

- Paquetes:
  - Kernel
  - rinad
  - librina

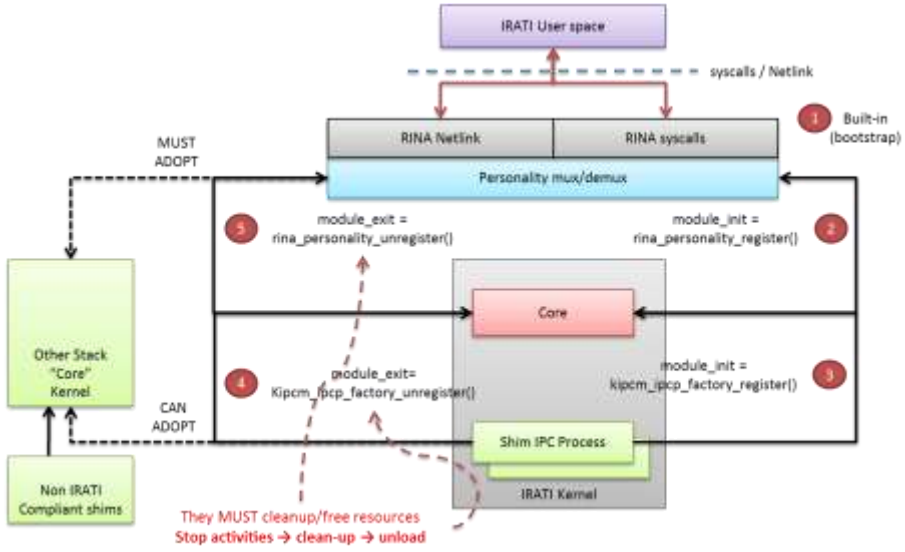
Dependencias de ejecución:



Dependencias de compilación:

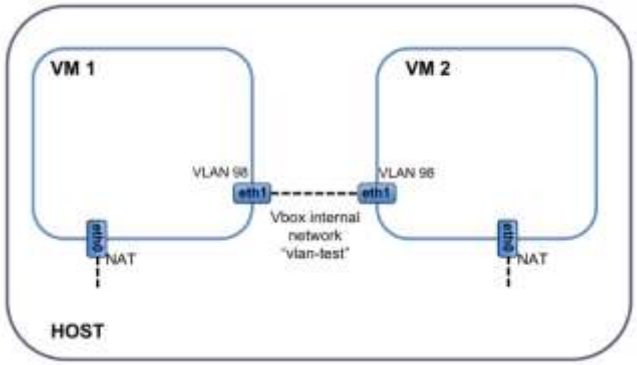


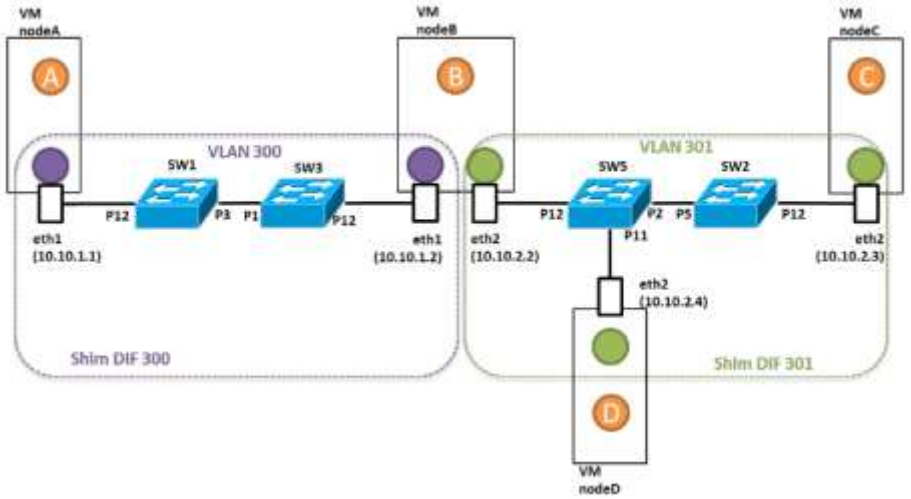
- Compilar e instalar user-space:
  - `./install-user-space-from-scratch <PREFIX>`
- Compilar e instalar el kernel:
  - `./install-kernel-from-scratch`
- Compilar e instalar todo :
  - `./install-from-scratch <PREFIX>`
- Desinstalar:
  - `./uninstall-and-clean`
- Cargar el sistema:
  - `modprobe rina-personality-default`
  - `modprobe shim-eth-vlan`
  - `modprobe shim-dummy`
  - `modprobe normal-ipcp`
  - `modprobe 8021q`



TESTING Y VALIDACIÓN

- Entornos de test





VirtualBox

- Local y autocontenido en un único PC.
- Brinda agilidad e independencia para el testing.
- Dos VMs con 2 tarjetas virtuales en cada una

OFELIA

- Testbed OpenFlow desplegado por el proyecto OFELIA FP7
- VMs y switches OpenFlow que permiten formar la topología deseada
- Configurable mediante el OFELIA Control Framework (OCF) via su interfaz web

- La aplicación echo
  - Programa desarrollado en JAVA para testear el ancho de banda en un flow RINA. El echo-client se conecta al echo-server, envía SDUs y el server las reenvía de vuelta

Tests unitarios, de regresión y funcionales

- Tests unitarios:
  - Aprovechando la estructura modular del código y las funciones __init y __exit de los módulos del kernel.
  - La función __init se utiliza para alimentar las funciones a testear y comprobar el resultado.
  - Un sistema de debugging mediante logs por niveles (DEBUG, WARNING, ERROR, etc) y dmesg.
- Test de regresión:
  - Técnica similar, seleccionables durante la compilación
- Tests funcionales en el entorno local:
  - Se siguió una estrategia de tarjetas para cada test funcional a realizar. En total se realizaton 19 tests pertenecientes a 4 categorías.
  - Tests básicos: comprueban que el stack del prototipo se puede cargar y descargar correctamente.
  - Tests del setup: comprueban que el stack puede crear y destruir distintas instancias de los componentes necesarios por el sistema.
  - Tests de transferencia de datos: comprueban que existe transferencia de datos entre distintos IPC Processes.
  - Tests de cleanup (limpieza): comprueban el comportamiento del sistema cuando debe cerrarse o una aplicación cae y como mantienen el estado.

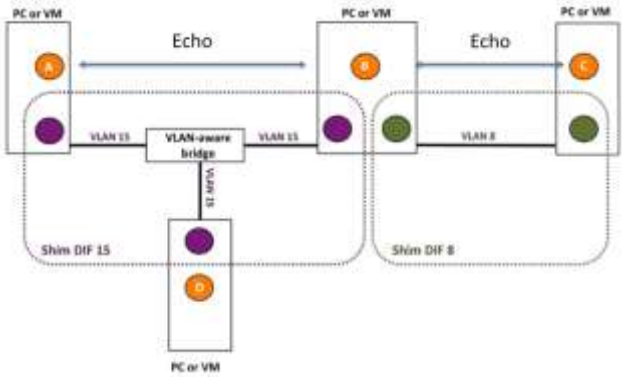
Etiqueta	IRATI-<NAME>-<NUMBER>
Nombre	<Nombre descriptivo de la tarjeta>
Objetivo	<Objetivos del test>
Tarjetas relacionadas	<Lista opcional de tarjetas relacionadas>
DUT	<Diagrama de la topología y detalles del entorno de test>

Paso	**Descripción**
1	
2	
...	
N	

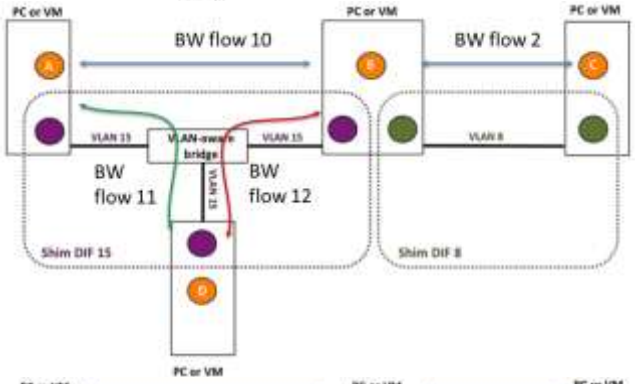
Comentarios adicionales
Notas, aclaraciones, etc...

Estado del test
PASADO/NO PASADO

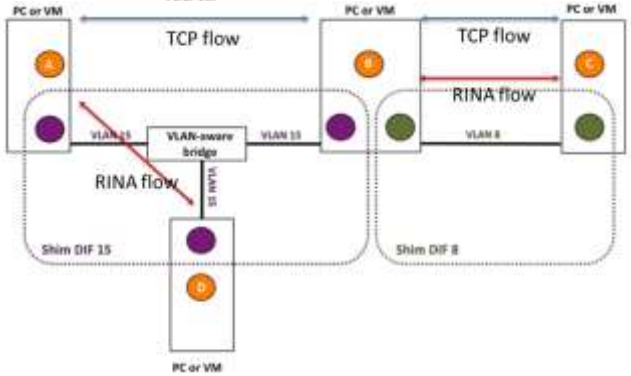
Tests funcionales en OFELIA (casos de uso)



- **Único flow por DIF**
  Ejecuta el echo-server/echo-client para testear la conexión básica con un único flow por DIF. Valida:
  - que las diferentes fases de la DIF se completan satisfactoriamente (enrollment, creación de flow, transferencia)
  - que la shim DIF sobre Ethernet funciona correctamente y no invalida paquetes sobre la VLAN



- **Múltiples flows por DIF**
  Ejecuta el echo-server/echo-client para testear la conexión básica con múltiples flows por DIF. Valida:
  - que las diferentes fases de reserva de flows y transferencia de datos se completan satisfactoriamente en condiciones altas de carga
  - que operación de transferencia de datos permanece estable durante el tiempo



- **Aplicaciones RINA e IP concurrentes**
  Ejecuta el 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 se utilizan los stacks UDP/IP y RINA a la vez. Valida:
  - la interoperabilidad de IP y RINA
  - que la operación de transferencia permanece estable durante el tiempo en ambos stacks
  - la interoperabilidad de los stacks sobre el mismo device driver de la tarjeta Ethernet

RESULTADOS Y CONCLUSIONES

- Los componentes software del prototipo RINA sobre Ethernet fueron diseñados, desarrollados, integrados y testeados.
- El problema con la implementación de Linux de ARP fue resuelto con RINARP.
- El prototipo actual implementa:
  - Los componentes básicos (core)
  - Un framework de utilidades basado en objetos del kernel
  - Distintos tipos de IPC Processes.
    - Normal (a falta de DTCP)
    - Shim dummy
    - Shim Ethernet
  - Claras interfaces entre componentes
  - Claras interfaces entre user-space y kernel
- Esqueleto fiable y funcional para continuar el desarrollo
- Todos los requisitos validados excepto el de interoperabilidad con otros prototipos por falta de tiempo y disponibilidad del otro prototipo

Gracias por su atención