Updates on the RINA light implementation

NEXTWORKS
ENGINEERING FORWARD

Vincenzo Maffione,
Marco Capitani (Presenter)
23/05/2018

Overview

- A Free and Open Source implementation of RINA for Linux
- Implementation split between user-space and kernel-space
  - Out-of-tree kernel modules loaded on the unmodified Linux kernel
- Keep it simple approach: codebase is clean and essential
  - ~ 37 Klocs
- Focus:
  - stability and performance: support deployments with hundreds to thousands of nodes
  - minimality: avoid over-engineering for ease of maintenance, use and improvement
- Main goal: be a baseline implementation for future RINA products
- Code and documentation available at https://github.com/rlite/rlite

Current main features

- Arbitrary composition and stacking of DIFs.
- Ability to run over legacy media like Ethernet, WiFi, TCP or UDP.
- Programmability of the data transfer constants
  - bit-width of addresses, sequence numbers and other protocol fields
- Programmability support (policies) for some layer management components:
  - Application Directory (DFT), Routing, Address Allocator and Flow allocator
- Support for flow control, retransmission control, congestion control, TTL and checksum
- Inspection tools to show current configuration and dynamic information
  - RIB contents, active flows and their statistics, locally registered applications, RMT statistics
- An implementation of the CDAP protocol, used by the IPCP management layer
- Support for integration tests and tests based on emulated networks
- A daemon to realize the RINA configuration specified by a given configuration file

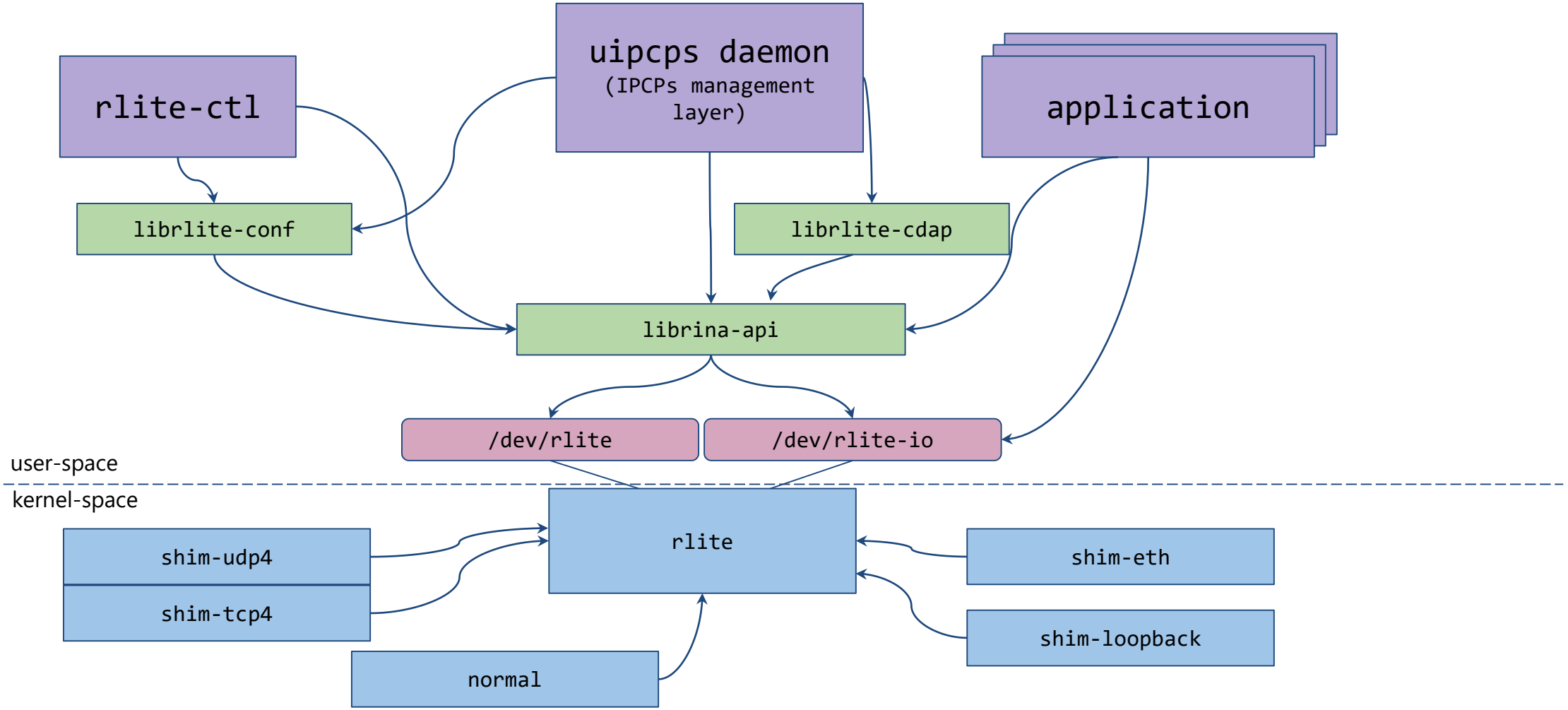
APIs for applications and administrators

- POSIX-like RINA API exposed to user applications (C/C++, Python)
  - Application registration and unregistration (`rina_register`, ...)
  - Flow allocation and deallocation, with QoS specification (`rina_flow_alloc`, `rina_flow_accept`, ...)
  - Data transfer (`write`, `read`, `writev`, `readv`)
  - Support for blocking and non-blocking operation (`select`, `poll`, `epoll`)
- Stack administration (`rlite-ctl`):
  - Creation, deletion and configuration of IPCPs
  - Registration and unregistrations between IPCPs
  - Enrollment and disconnection to/from DIFs
  - Configuration of policies and parameters
  - Performance monitoring (per-flow statistics, RMT statistics)
  - Inspection (IPCPs, RIB, policies)

Some performance and stability indicators

- Some raw throughput numbers with 1500 bytes PDUs (normal over shim-eth)
  - Up to 13 Gbps (1.1 Mpps) for a single flow without flow control and retransmission
  - About 11 Gbps (0.92 Mpps) for a single flow with flow control and retransmission
- Stability
  - Weeks long VM-based experiments with up 350 nodes, two levels of normal DIFs and 50 flows allocations per second
  - Done experiments with up to 10 levels of DIFs

Software architecture



Software architecture

- kernel-space
  - Supports control operations (IPCP lifecyle)
  - Implements datapath
  - Keeps state (except for RIB)
- user-space
  - Libraries to abstract interaction with kernel-space functionalities
  - A daemon to implement the layer management part of multiple IPCPs
    - RIB state is kept here
  - An *iproute2-like* command-line tool to administer the stack (rlite-ctl)
- kernel-space and user-space interact through character devices (i.e. through file descriptors)
  - `/dev/rlite` for control operations
  - `/dev/rlite-io` for data transfer and synchronization

Current policies

- **Addess allocator**, to assign an address to an IPCP joining a DIF
  - *static*: Manually assigned
  - *distributed*: Distributed algorithm run by current DIF members to allocate a new unused address
- **Application naming directory**
  - *fully-replicated*: A copy of the Directory Forwarding Table is stored on each node, and synchronized
  - *centralized-fault-tolerant*: DFT stored in a cluster of (usually 5) IPCPs in the DIF, which run a *consensus* algorithm (Raft) to achieve strong consistency and fault-tolerance
- **Routing**
  - *static*: Routes manually set through `rlite-ctl`
  - *link-state*: A simple Link State routing algorithm (full replication of the routing information across the IPCPs)
  - *link-state-lfa*: Link State routing enhanced with the Loop Free Alternate algorithm (fast re-route) to improve reliability in case of link/node failures

Current tunables

- About 20 parameters to configure IPCPs and policies, and enable/disable features
  - Keepalive, timeouts and auto-reconnect for management N-1-flows
  - Broadcast enrollment
    - Anycast enrollment against the DIF name (rather than a specific neighbor)
  - Use of reliable management N-1-flows
  - Use of reliable management N-flows where reliable N-1-flows are not available
    - In other words an IPCP can use its own IPC services
  - RIB synchronization and aging tunables
    - Synchronization interval, aging increment and interval, …
  - Default EFCP parameters for retransmission, flow control and queue sizing
  - Initial value for PDU TTL
  - PDU checksum to be used (no checksum or Internet checksum)

Other features

- The uipcps daemon builds a graph of local IPCPs and flows allocated among them
  - A node for each IPCP, an edge for each inter-IPCP flow
  - Graph used for automatic computation of:
    - per-IPCP Maximum SDU size (using the constraints provided by shim DIFs)
    - per-IPCP PCI header space to be reserved at kernel buffer allocation
  - Results pushed to the kernel for optimized operation → no PDU reallocations needed in the datapath
- Programmability of data transfer constants through *recompilation* of the kernel datapath
  - A different kernel module for each combination → no serialization run-time overhead and simpler code
  - No real memory concerns: only a few combinations (*flavours*) are expected on a real system
- Automatic handover over WiFi, based on signal strength
  - Cross-technology manual handover is possible, with no service disruption to applications

Some kernel-space internals

- Reference counters widely used to manage lifetime of objects (e.g. IPCPs, flows, registered applications, PDUs, etc.)
- *sk_buff-like* approach to avoid copies throughout the datapath (no PDU serialization)
- dynamic allocation of PDU buffers
- All PDU queues are limited in size to keep memory usage under control
- Deferred work (workqueues) used only when necessary, to keep latency low
- Vertical flow control, to prevent upper DIFs to overrun lower DIFs (e.g. network interfaces)
  - Synchronization mechanism in place to block upper IPCPs and wake them up when they can proceed

Next steps

- Generalize centralized-fault-tolerant functionalities
  - They would be used also by other components (Address Allocator, bandwidth allocator, ...).
- Add kernel support for Linux network namespaces to enable isolation between containers
  - Currently all the containers on the same machine share the same IPCPs and flows
  - It would be extremely useful to scale out single-machine integration tests to thousands of containers
- More policies
  - Distributed Flow Allocator with bandwidth allocation capabilities
  - Support for RMT scheduling (e.g. QTA mux)
  - Authentication policies for enrollment
  - Fully decentralized scalable Directory Forwarding Table (e.g. using DHT)
- SMP-scalable (kernel) datapath implementation
  - Minimize lock contention and cache coherency traffic between CPUs transmitting and receiving PDUs
- Unit tests for the EFCP state machine.