RINA: Recursive InterNetwork Architecture

Last advances from the PRISTINE project

Leonardo Bergesio <leonardo.bergesio@i2cat.net>
on behalf of
The PRISTINE consortium

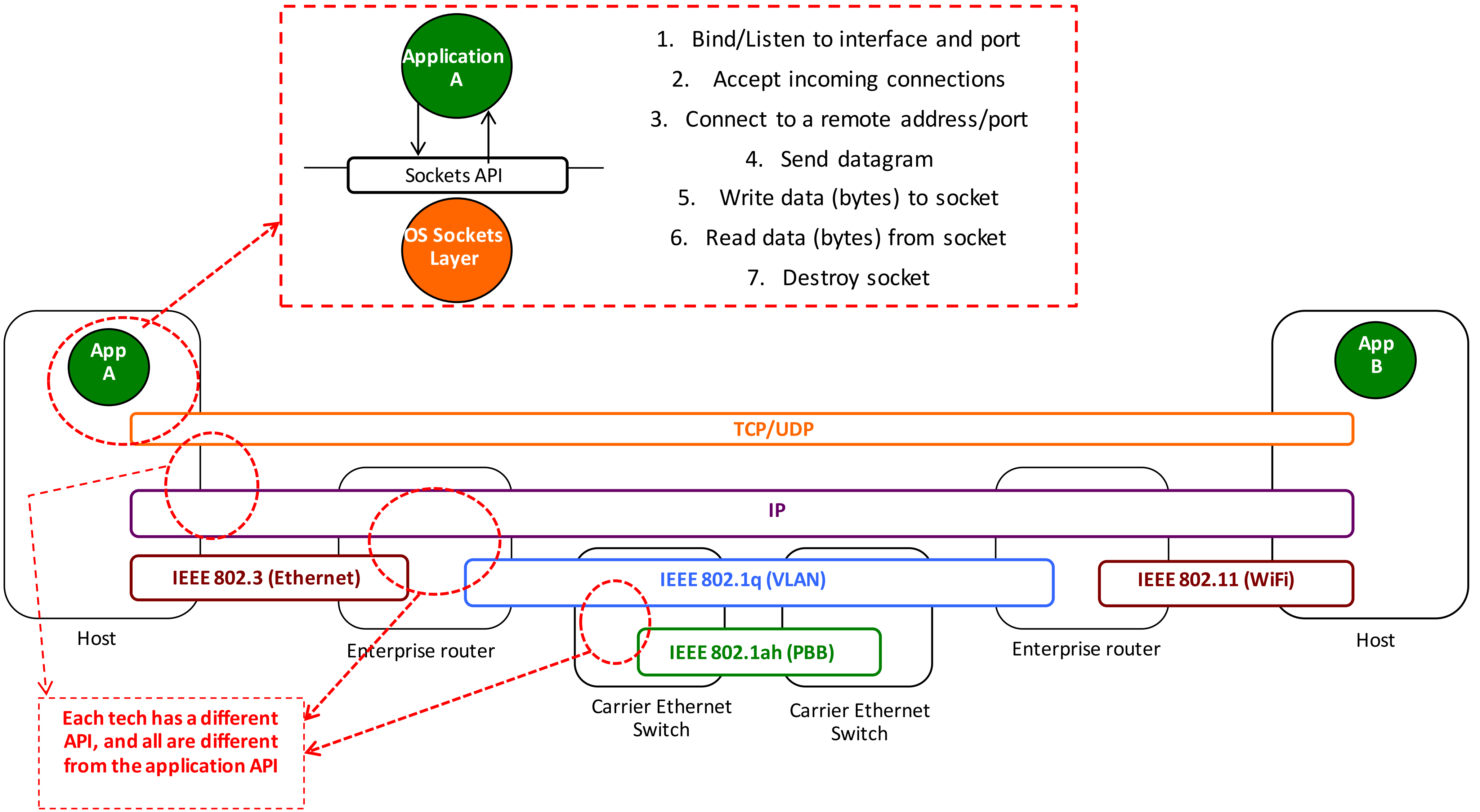
#ict-pristine

1 RINA INTRODUCTION

RINA higlights

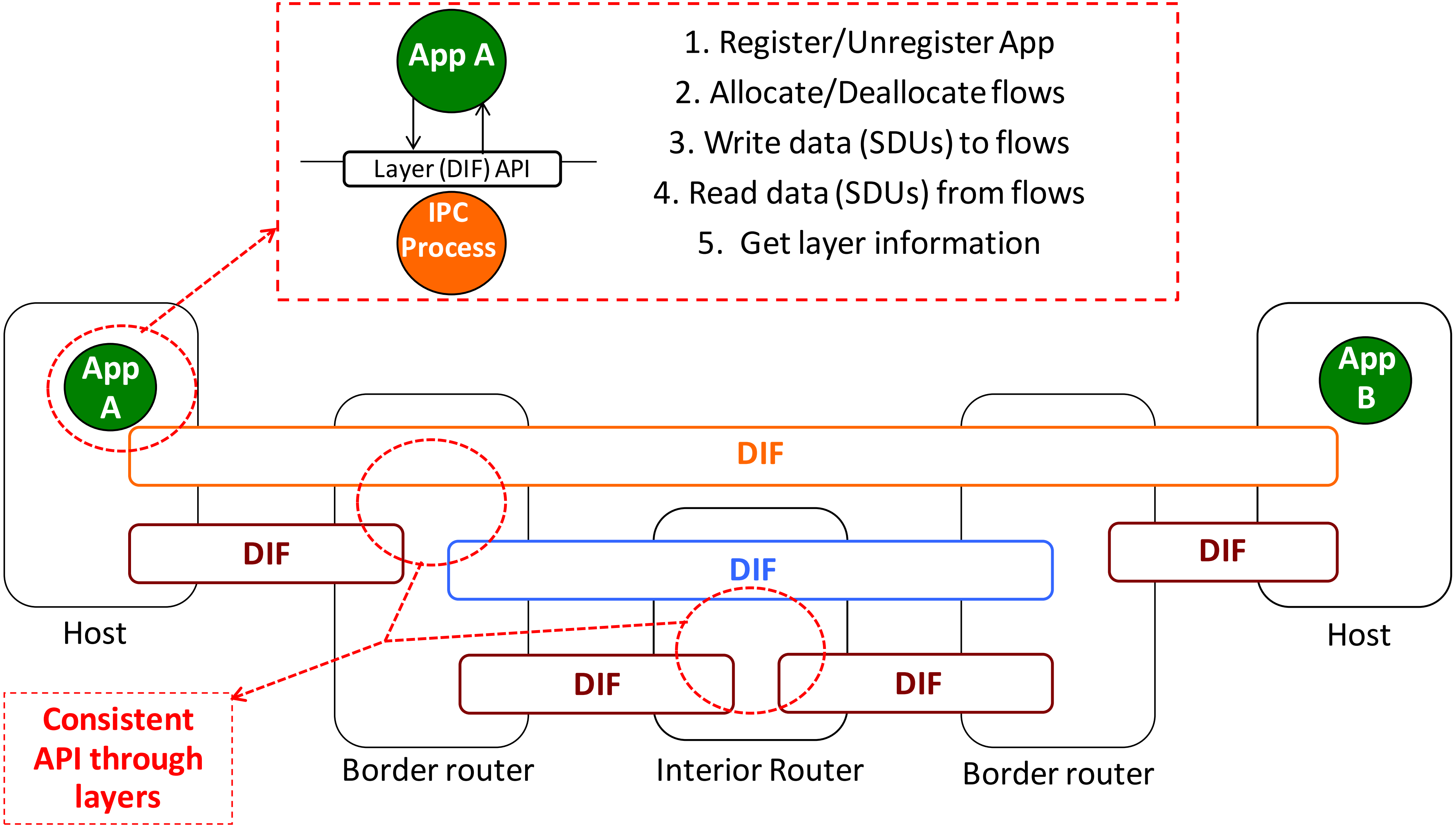
1. Network architecture resulting from a fundamental theory of computer networking
2. Networking is InterProcess Communication (IPC) and only IPC. Unifies networking and distributed computing: the network is a distributed application that provides IPC
3. There is a single type of layer with programmable functions, that repeats as many times as needed by the network designers
4. All layers provide the same service: a communication instance (flow) to two or more application instances, with certain characteristics (delay, loss, in-order-delivery, etc)
5. There are only 3 types of systems: hosts, interior and border routers. No middleboxes (firewalls, NATs, etc) are needed
6. Deploy it over, under and next to current networking technologies

From here …

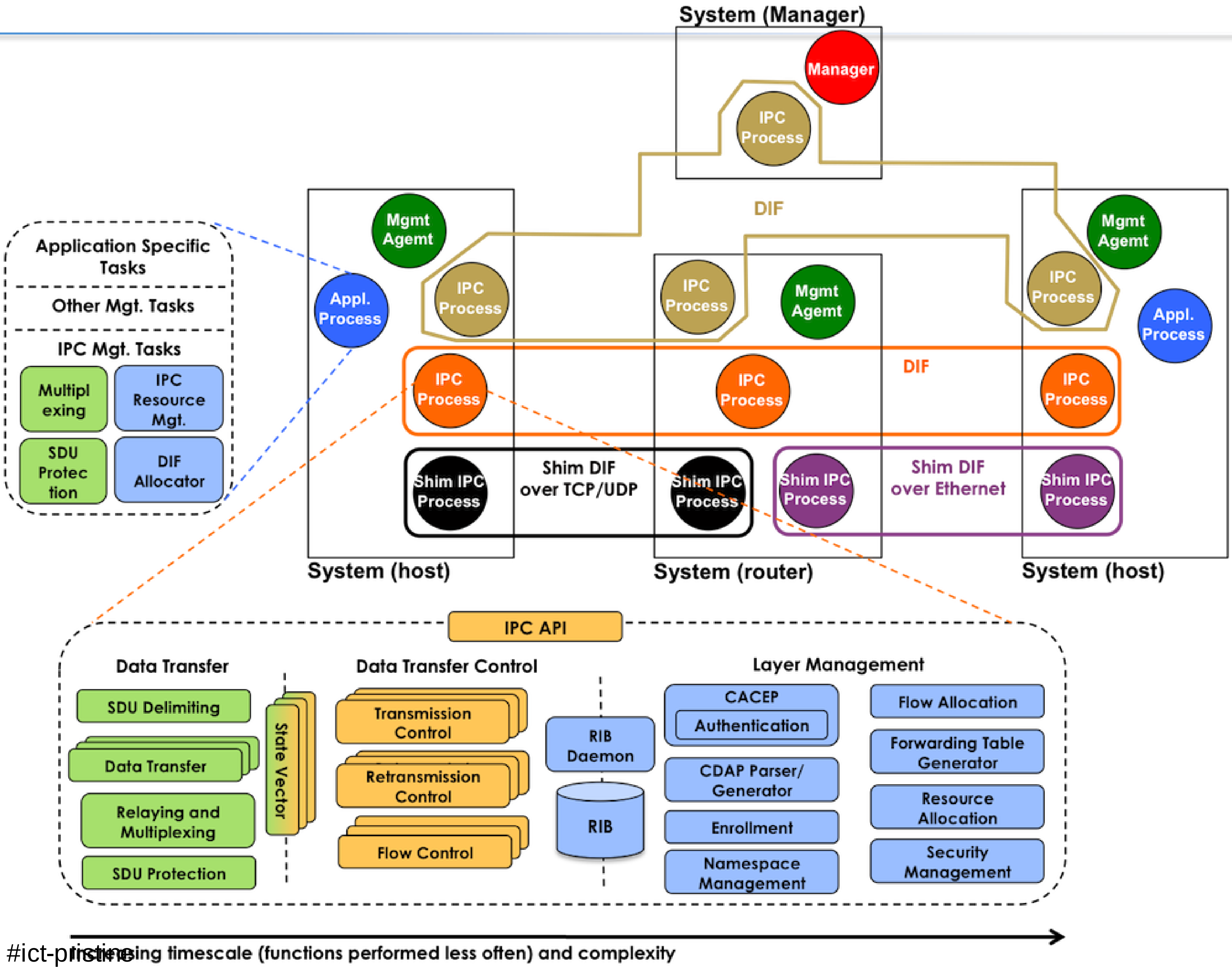


Application A
Sockets API
OS Sockets Layer
1. Bind/Listen to interface and port
2. Accept incoming connections
3. Connect to a remote address/port
4. Send datagram
5. Write data (bytes) to socket
6. Read data (bytes) from socket
7. Destroy socket
App A
App B
TCP/UDP
IP
IEEE 802.3 (Ethernet)
IEEE 802.1q (VLAN)
IEEE 802.11 (WiFi)
IEEE 802.1ah (PBB)
Host
Enterprise router
Carrier Ethernet Switch
Carrier Ethernet Switch
Enterprise router
Host
Each tech has a different API, and all are different from the application API

To here!


App A
Layer (DIF) API
IPC Process
1. Register/Unregister App
2. Allocate/Deallocate flows
3. Write data (SDUs) to flows
4. Read data (SDUs) from flows
5. Get layer information
App A
App B
DIF
DIF
DIF
DIF
DIF
DIF
Host
Host
Consistent API through layers
Border router
Interior Router
Border router

Internal layer organization



2

IRATI: OPEN SOURCE RINA IMPLEMENTATION

RINA implementation goals

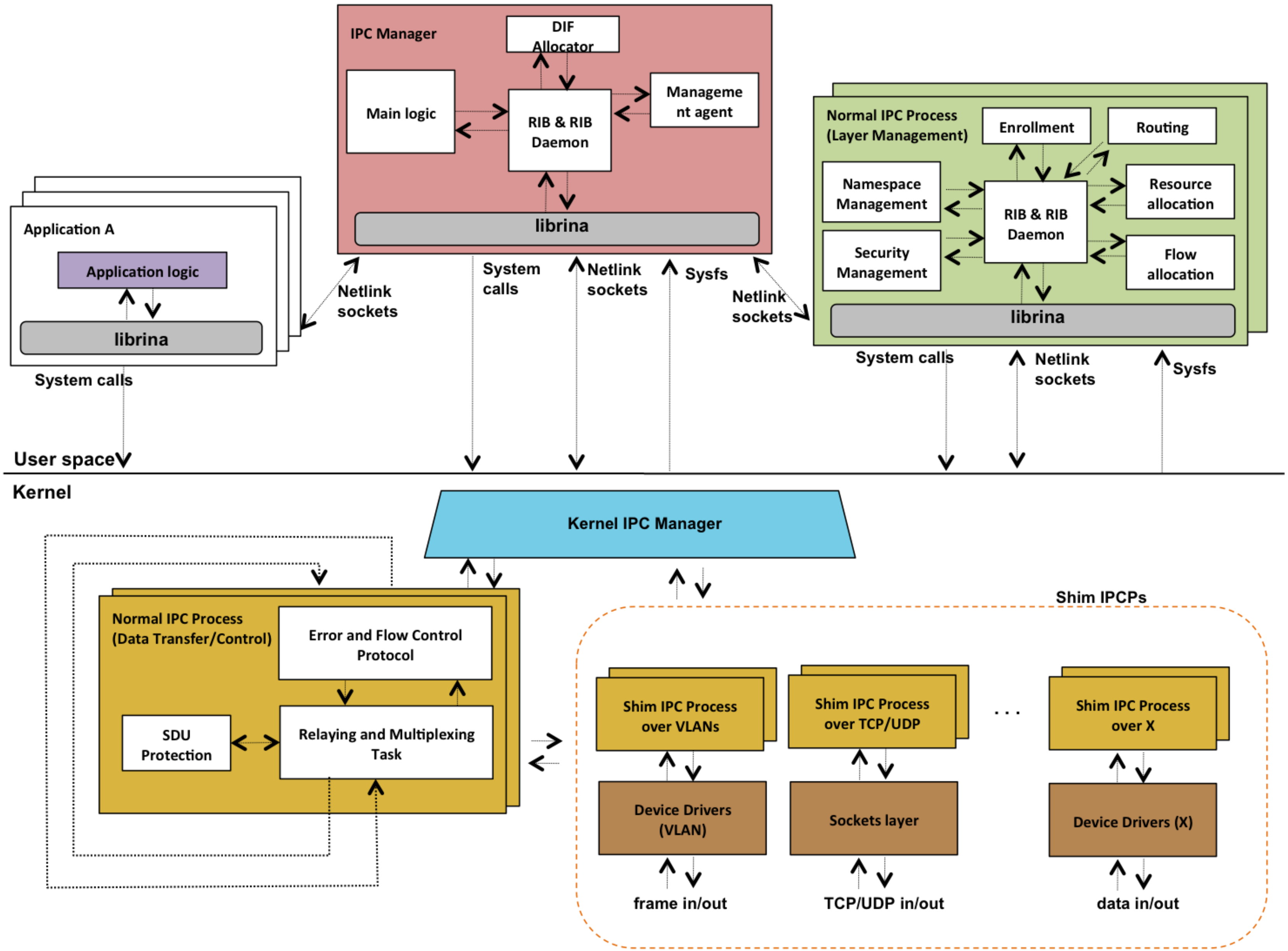
- Build a platform that **enables RINA experimentation** …
  1. Flexible, adaptable (host, interior router, border router)
  2. Modular design
  3. Programmable
  4. RINA over X (Ethernet, TCP, UDP, USB, shared memory, etc.)
  5. Support for native RINA applications

- … but can also be the **basis of RINA-based products**
  1. Tightly integrated with the Operating System
  2. Capable of being optimized for high performance
  3. Enables future hardware offload of some functions
  4. Capable of seamlessly supporting existing applications
  5. IP over RINA

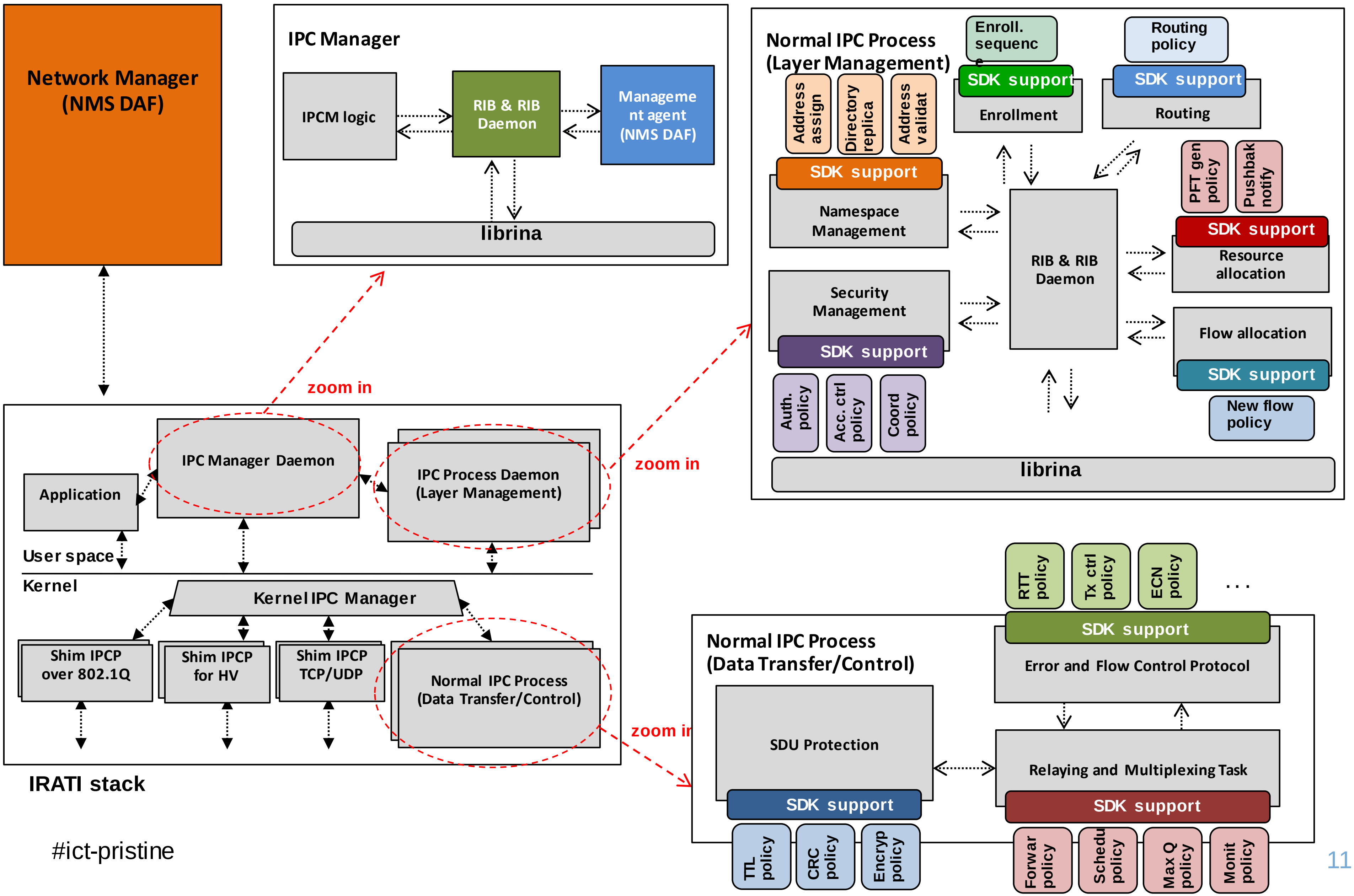
Some decisions and tradeoffs

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

High-level software arch.



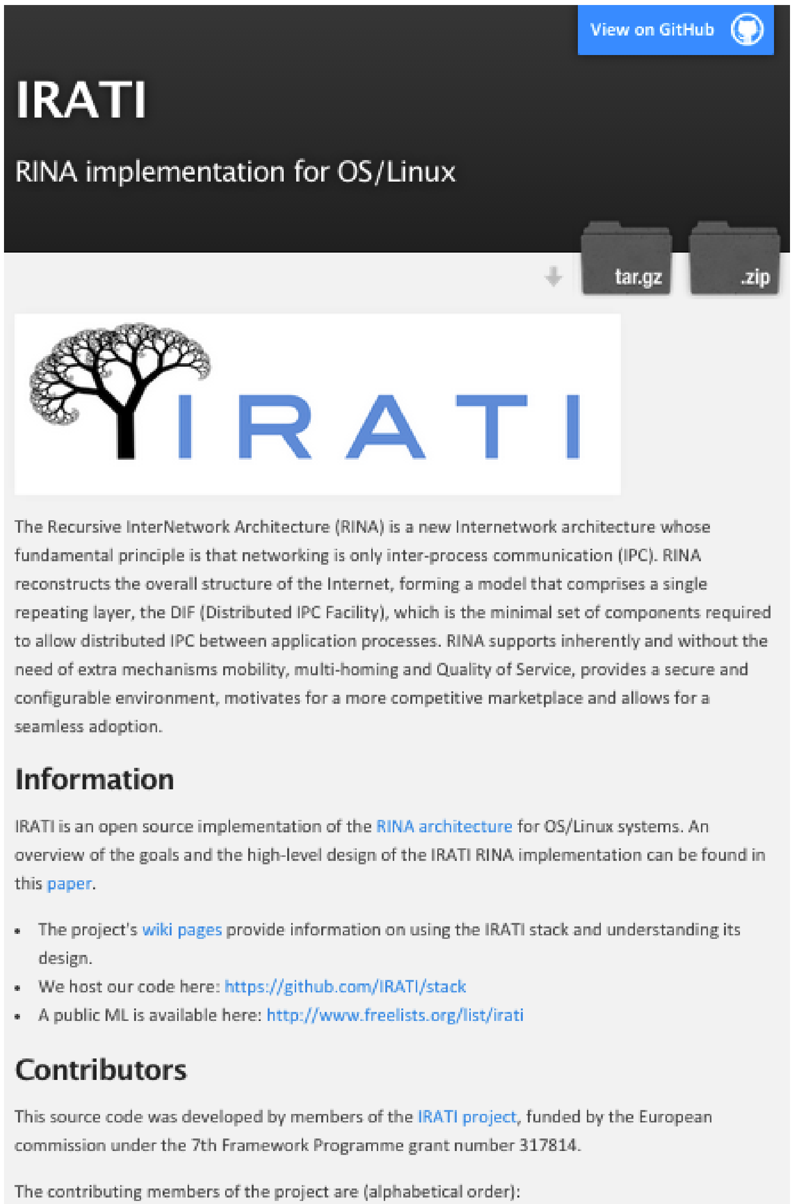
PRISTINE contributions: SDK, policies, NMS



Implementation status (I)
IPCP components

IPCP component	SDK	Available policies / comments
CACEP	Y	No authentication, password-based, cryptographic (RSA keys)
SDU Protection	N	On/off hardcoded default policies, no SDK support yet: CRC32 (Error Check), hopcount (TTL enforcement), AES encryption
CDAP	N	Google Protocol Buffers (GPB) encoding, no support for *filter* op
Enrollment	Y	Default enrollment policy based on enrollment spec
Flow Allocation	Y	Simple QoS-cube selection policy (just reliable or unreliable)
Namespace Mgr.	Y	Static addressing, fully replicated Directory Forwarding Table
Routing	Y	Link-state routing policy based on IS-IS
Res. Allocator	Y	PDU Fwding table generator policy with input from routing
EFCP	Y	Retx. Control policies, window-based flow control, ECN receiver
RMT	Y	Multiplexing: simple FIFO, cherish/urgency. Forwarding: longest match on dest. address, multi-path forwarding, LFA. ECN marking

Open source IRATI



View on GitHub

IRATI

RINA implementation for OS/Linux

tar.gz .zip

IRATI

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

Information

IRATI is an open source implementation of the RINA architecture for OS/Linux systems. An overview of the goals and the high-level design of the IRATI RINA implementation can be found in this paper.

- The project's wiki pages provide information on using the IRATI stack and understanding its design.
- We host our code here: https://github.com/IRATI/stack
- A public ML is available here: http://www.freelists.org/list/irati

Contributors

This source code was developed by members of the IRATI project, funded by the European commission under the 7th Framework Programme grant number 317814.

The contributing members of the project are (alphabetical order):

- **IRATI** github side
  - *http://irati.github.io/stack*
- Hosts code, docs, issues
  - Installation guide
  - Experimenters (tutorials)
  - Developers (software arch)
- Mailing list for users and developers
  - *irati@freelists.org*
- Procedures to contribute under discussion, doc ongoing

3

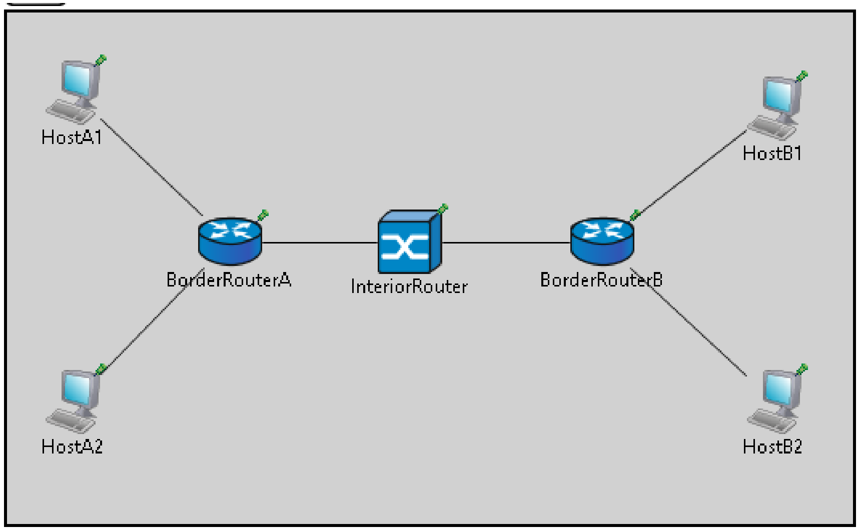
RINASIM: RINA SIMULATOR

RINASim

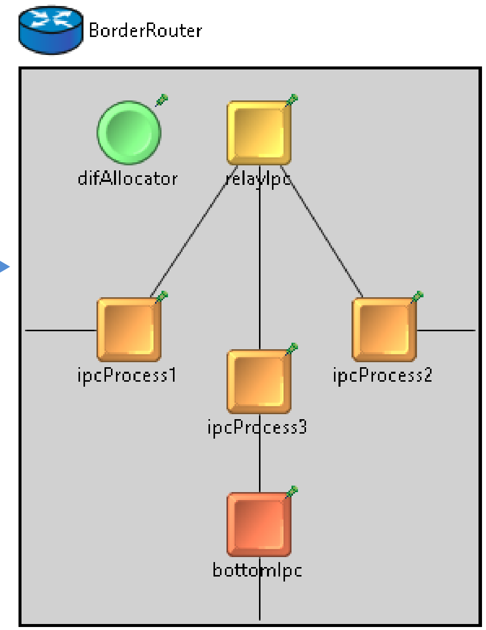
- RINASim is independent framework implemented for OMNeT++
- Source code is publicly available on github (https://github.com/kvetak/RINA)
  - Easy issue submitting
  - Fast integration of partners contribution
- Documentation is automatically generated using Doxygen
- Partners contribute via dedicated branch (fork/merge procedure)
- New release ~every month
  - Available also as an virtual-machine OVF appliance for VmWare or VirtualBox

RINA Simulator Model
Main building blocks

Main RINA Simulator model (systems and physical links)

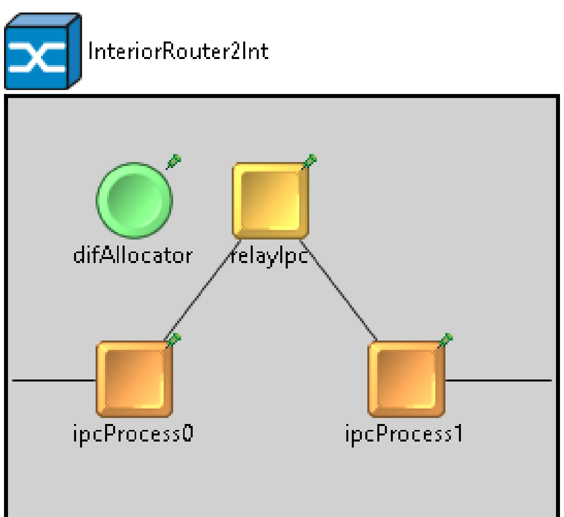


Border Router



- *1 N-level IPC Process*
- *X >= 2 N-1 level IPC Processes*
- *X >= 1 N-2 level IPC Processes*

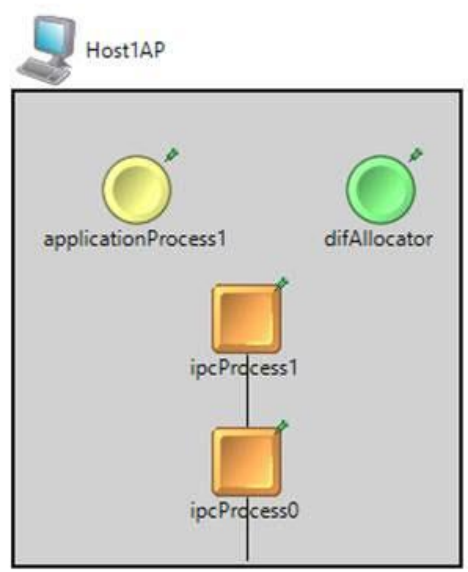
Interior Router



1 N-level IPC Process

X > 2 N-1 level IPC

Hosts

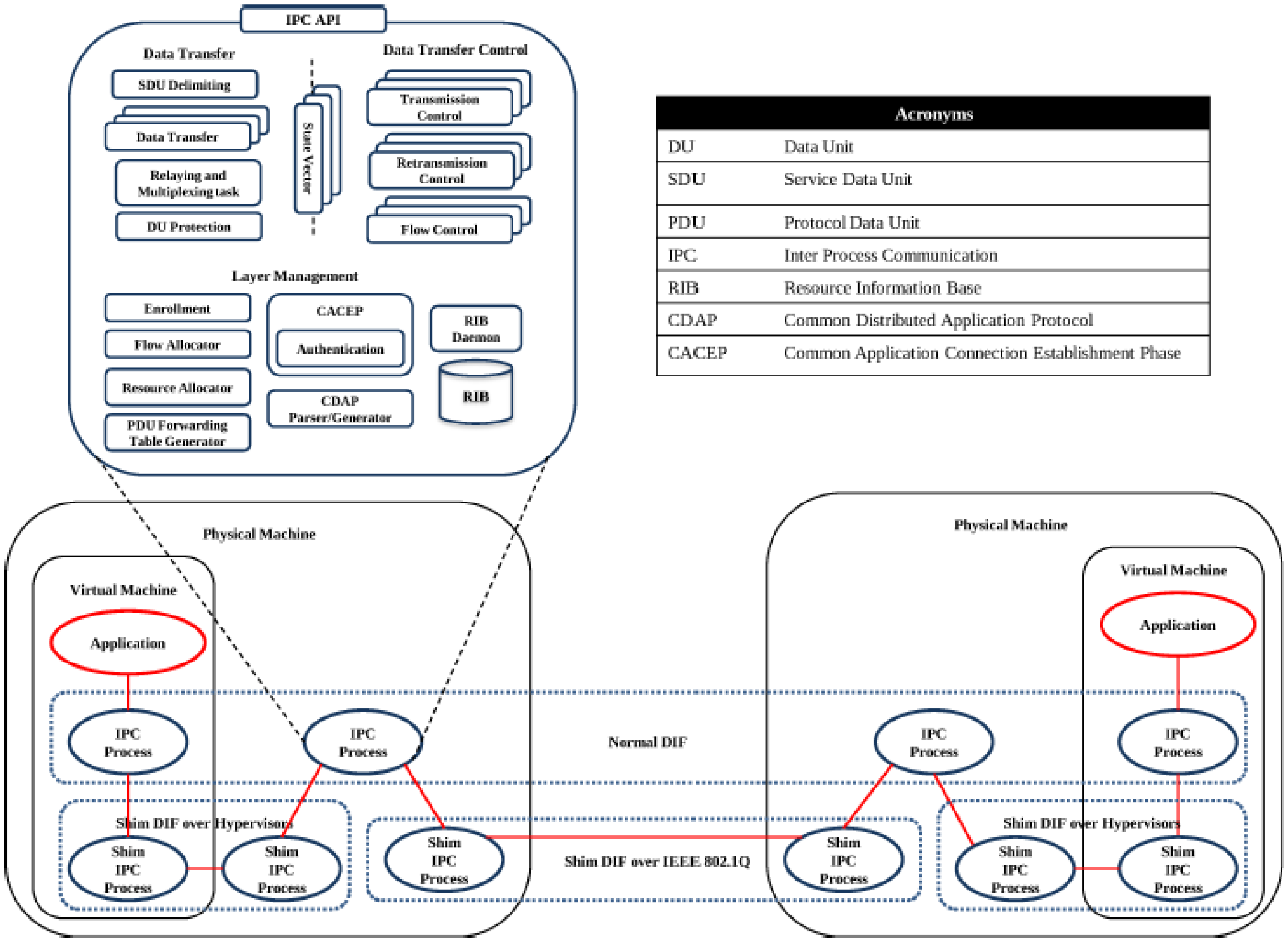


- *A flexible number of IPCPs, but not for relaying*
- *X >= 1 Application Processes*
- *1 DIF Allocator*

4

SOME ONGOING WORK AND RESULTS

Simplifying VM Networking with RINA

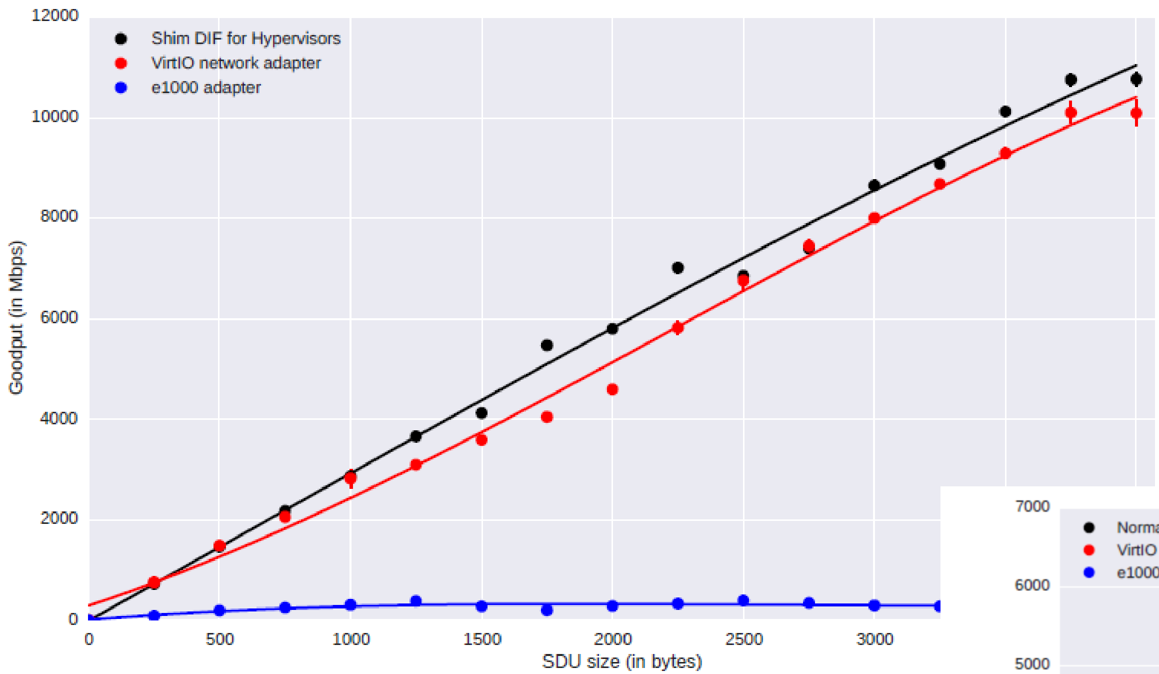


Acronyms	
DU	Data Unit
SDU	Service Data Unit
PDU	Protocol Data Unit
IPC	Inter Process Communication
RIB	Resource Information Base
CDAP	Common Distributed Application Protocol
CACEP	Common Application Connection Establishment Phase

- No need to implement complex and expensive NIC emulation.
- No need to generate and assign MAC addresses,
- No need to create and configure software L2 bridges to connect VMs and hypervisor physical NICs together.
- Users of the shim DIF are not restricted to the Ethernet MTU (1500/9000 bytes)
  - Commonly bypassed using the TCP Segmentation Offloading (TSO).
- No need to perform TCP/UDP checksumming since shared memory communication is protected from corruption
  - Checksumming is not actually performed by modern paravirtualized NICs (e.g. virtio-net, xen-netfront)

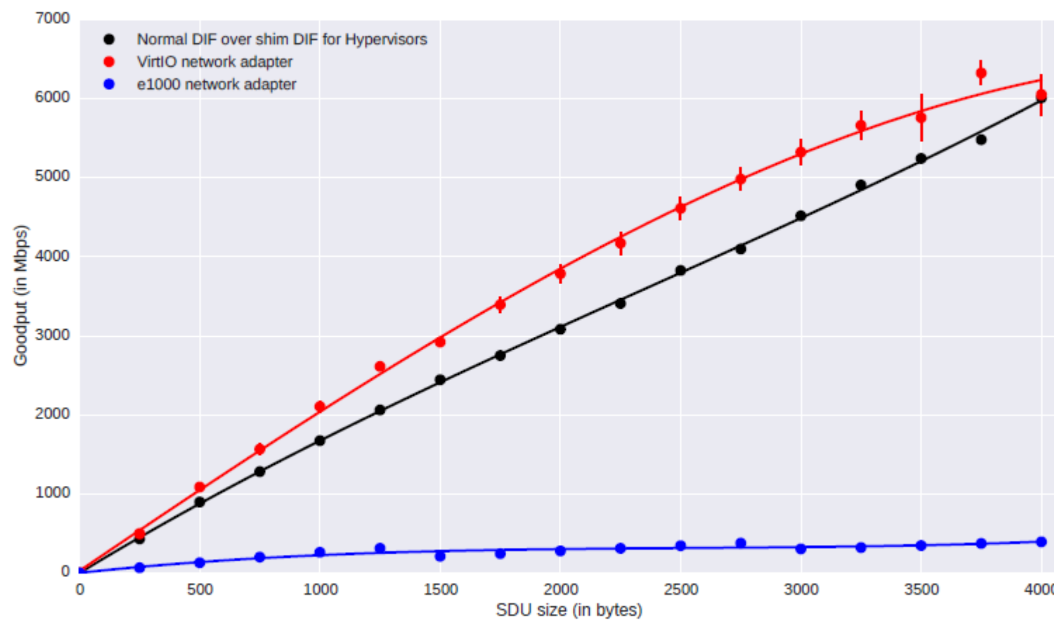
Experimental results (prototype)

Host to VM communication



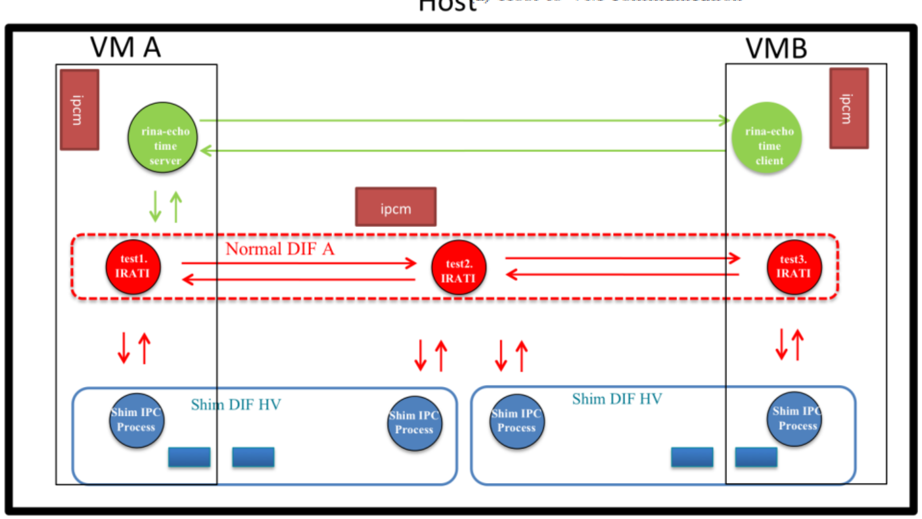
(a) Host to VM communication

VM to VM communication



(b) VM to VM communication

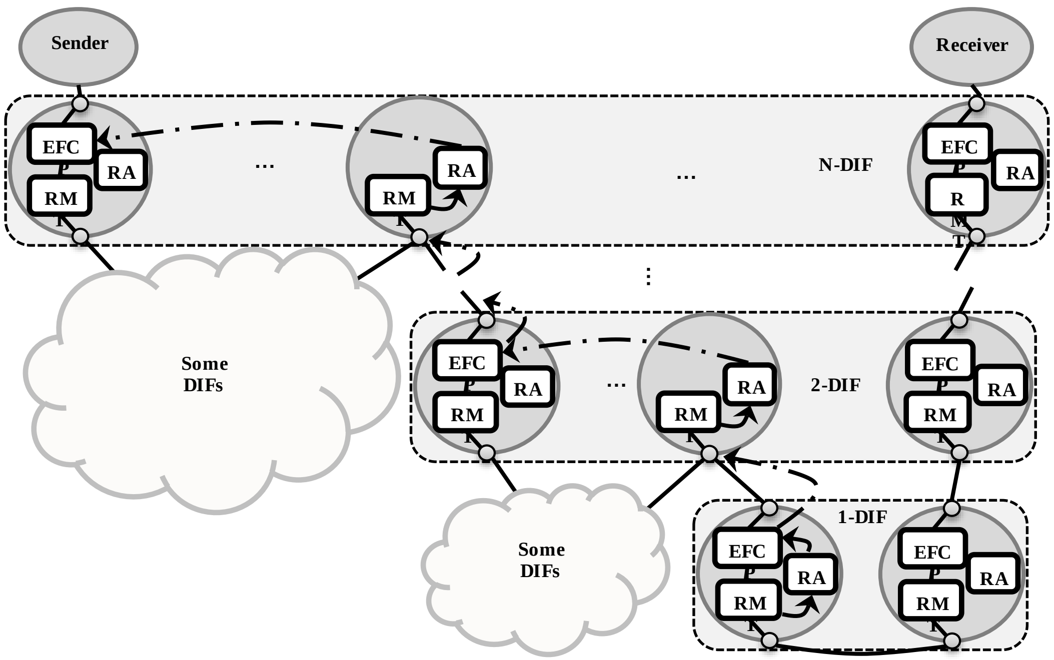
Fig. 5. Goodput of different network virtualization technologies



Experimental scenario

Congestion control (I)

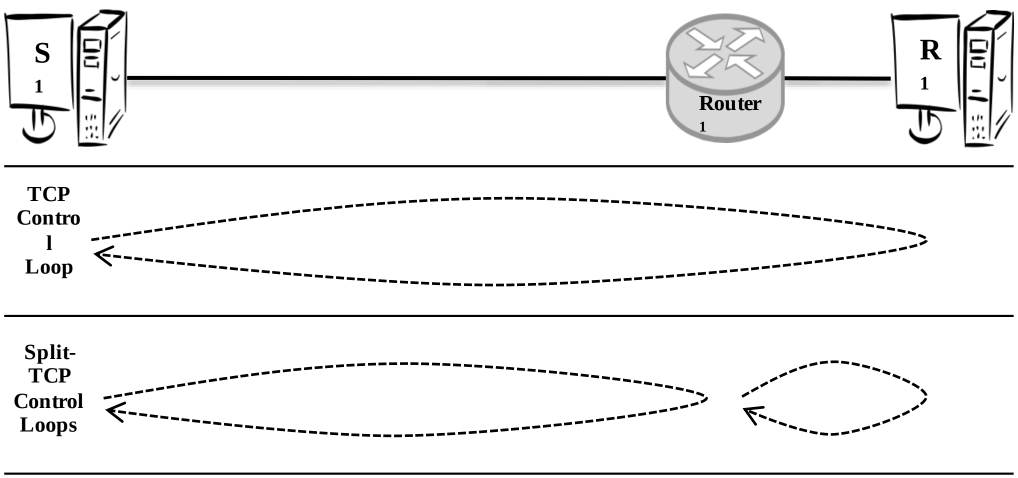
- In RINA CC is a generalization of how it is done in the Internet

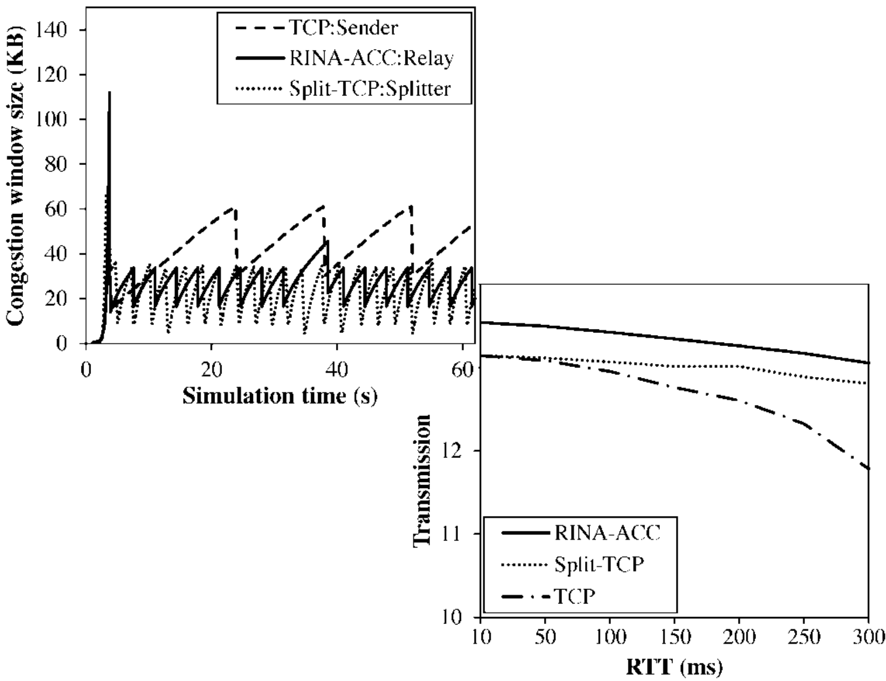


- Benefits:
  - "Naturally" gaining from properties of various previous improvements in the Internet, without inheriting their problems (PEPs), flow aggregation
  - Customization of CC policies to each layer needs (not one size fits all)
  - With explicit detection, congestion effects can be confined to a single layer (faster and more local response to congestion)

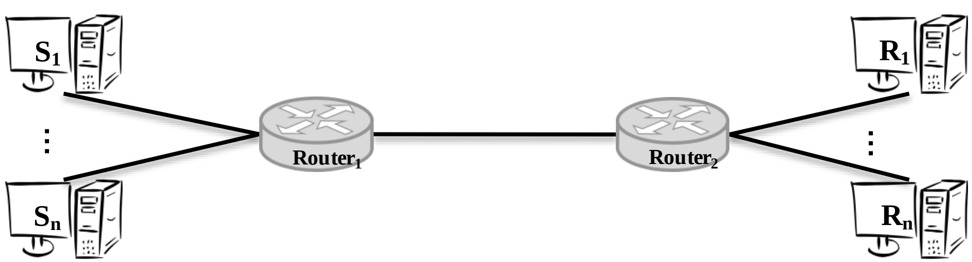
Congestion control (II): simulations

- Horizontal: consecutive DIFs



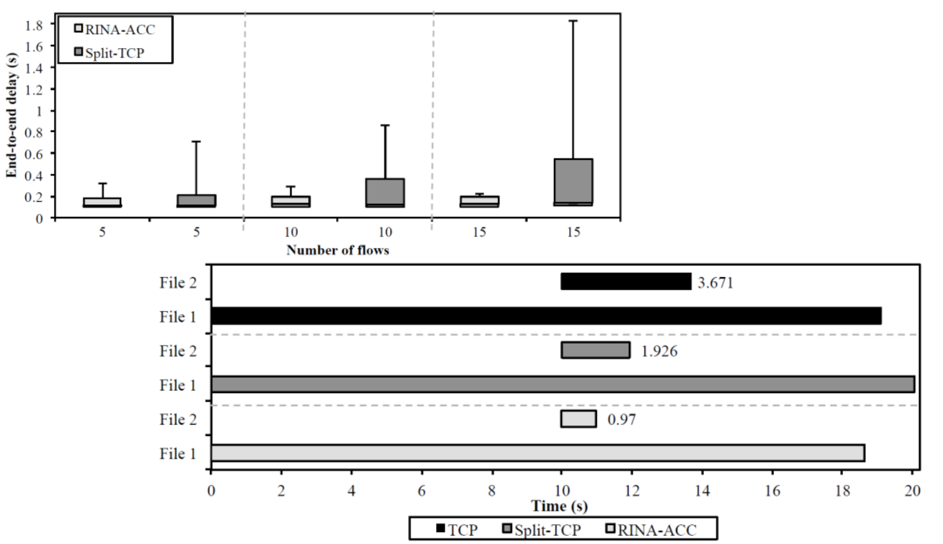


- Vertical: stacked DIFs



n = 5, 10, 15

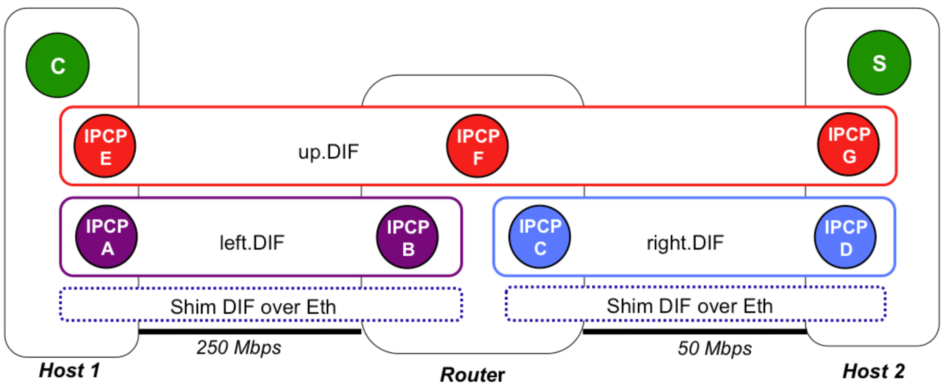
n = 1



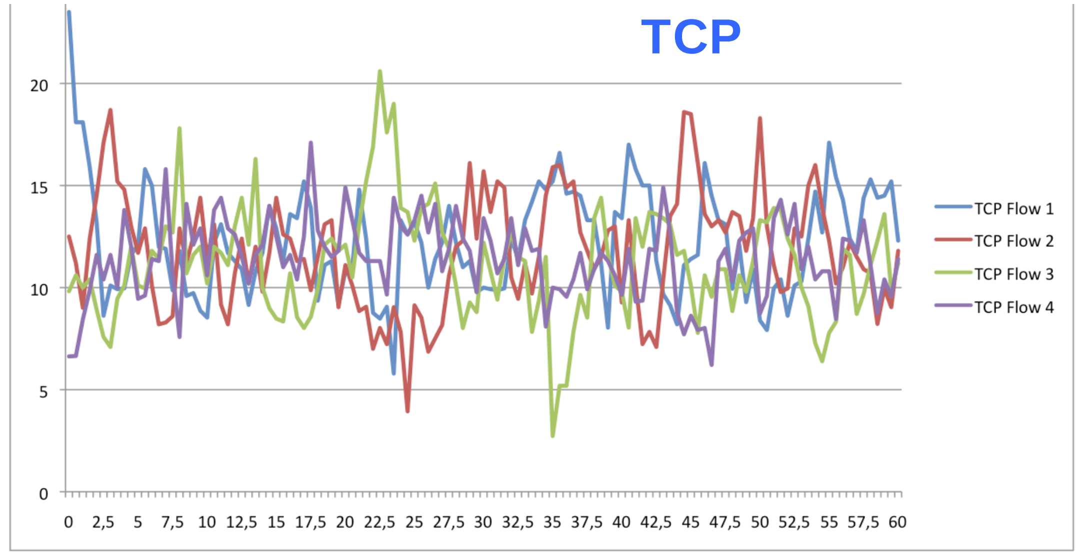
#ict-pristine

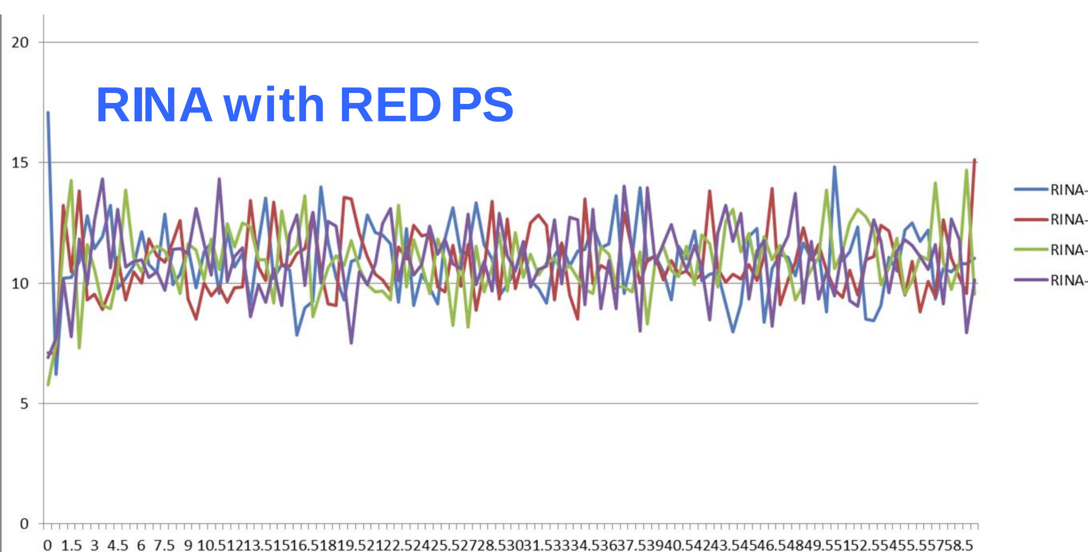
Congestion control (III): prototype

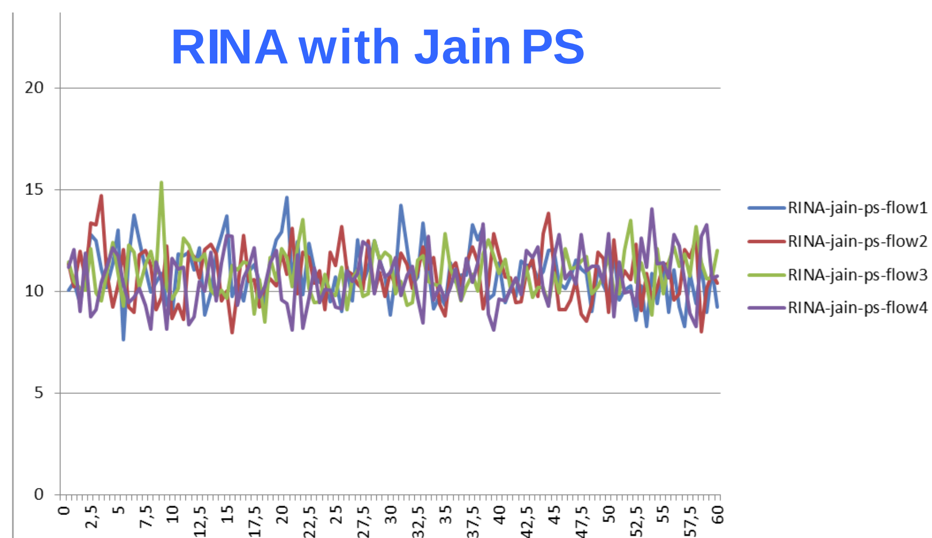
- TCP (iperf) vs. RINA with two different cc policy sets deployed in the red DIF. 4 flows, Throughput vs. Experiment time



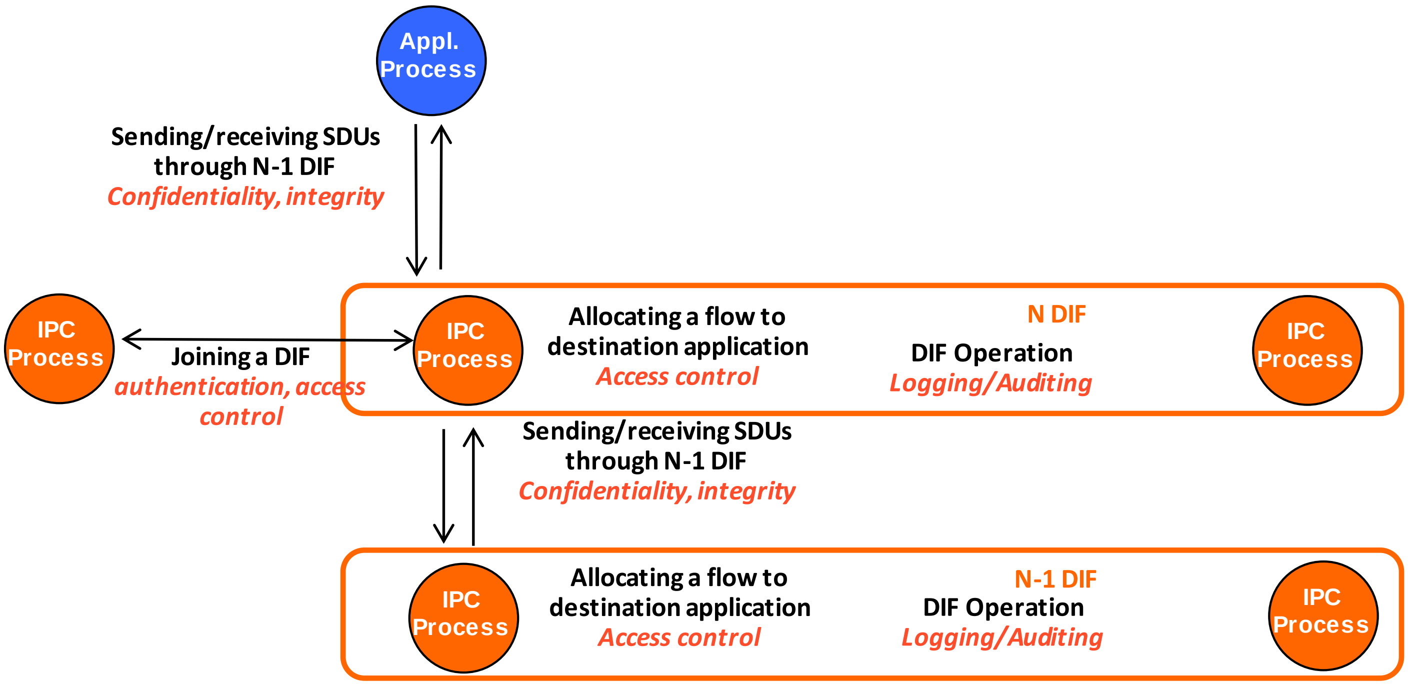
Experiment setup





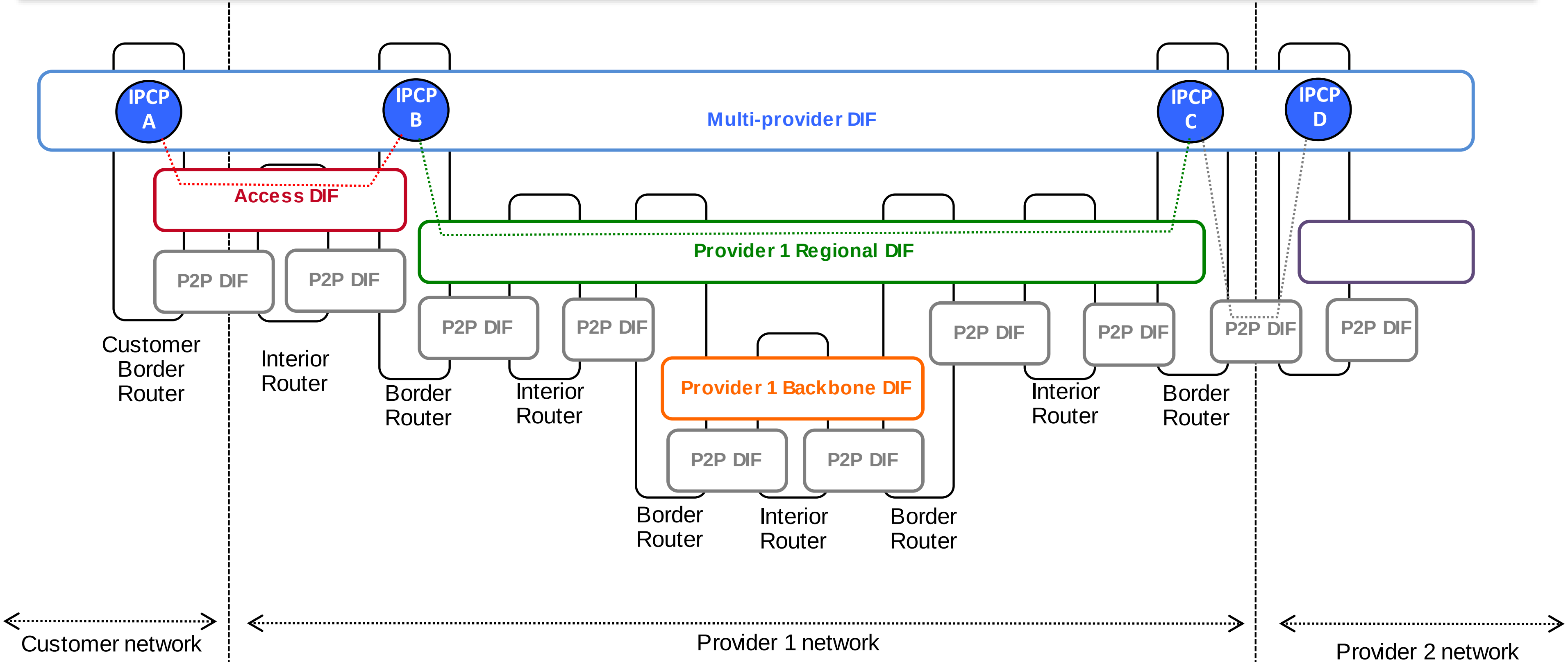


Security overview: placement of security functions in RINA



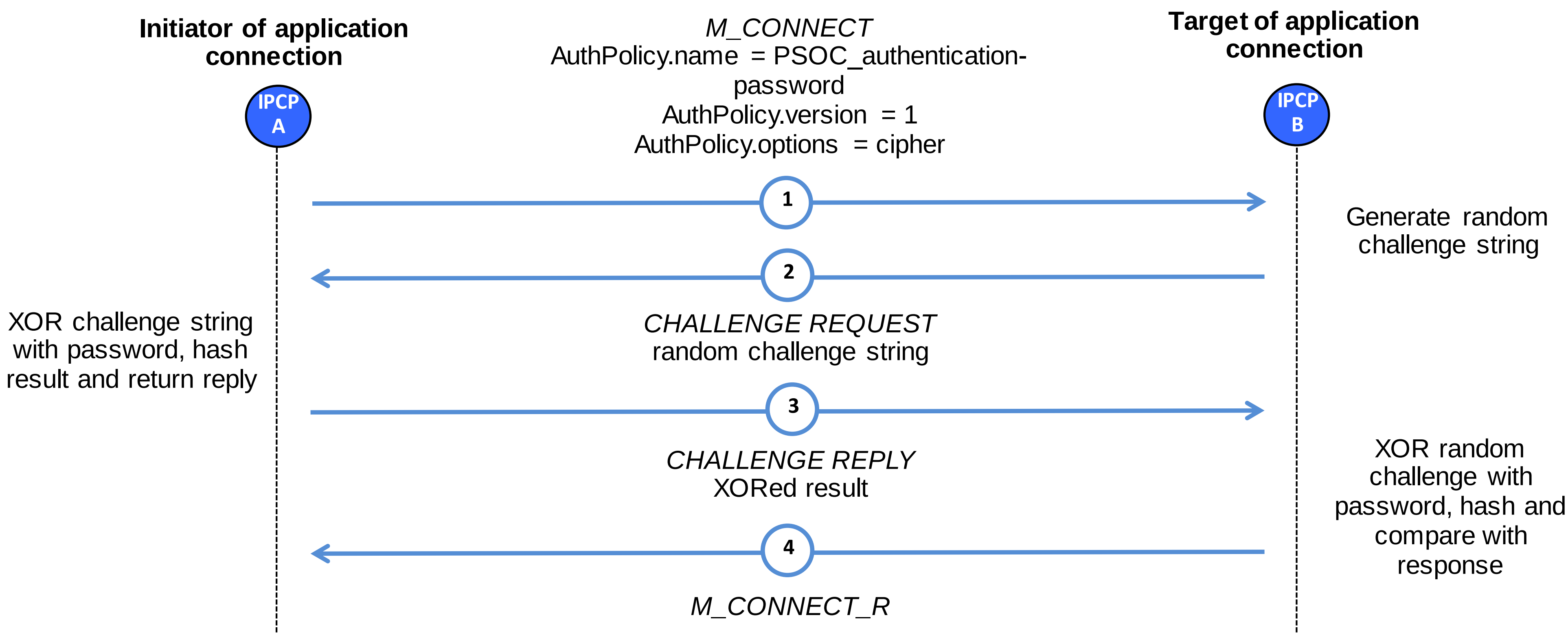
- IPC Process components involved in security
  - CACEP: Authentication policies
  - Security Coordination: Credential Management, Access Control Decisions (allow new IPC Processes in the DIF, accept flows to applications), Intrusion Detection/Prevention?, other?
  - SDU Protection: Confidentiality mechanisms (encryption)
  - RIB Daemon: Logging of operations in the DIF

Security: DIFs are securable containers

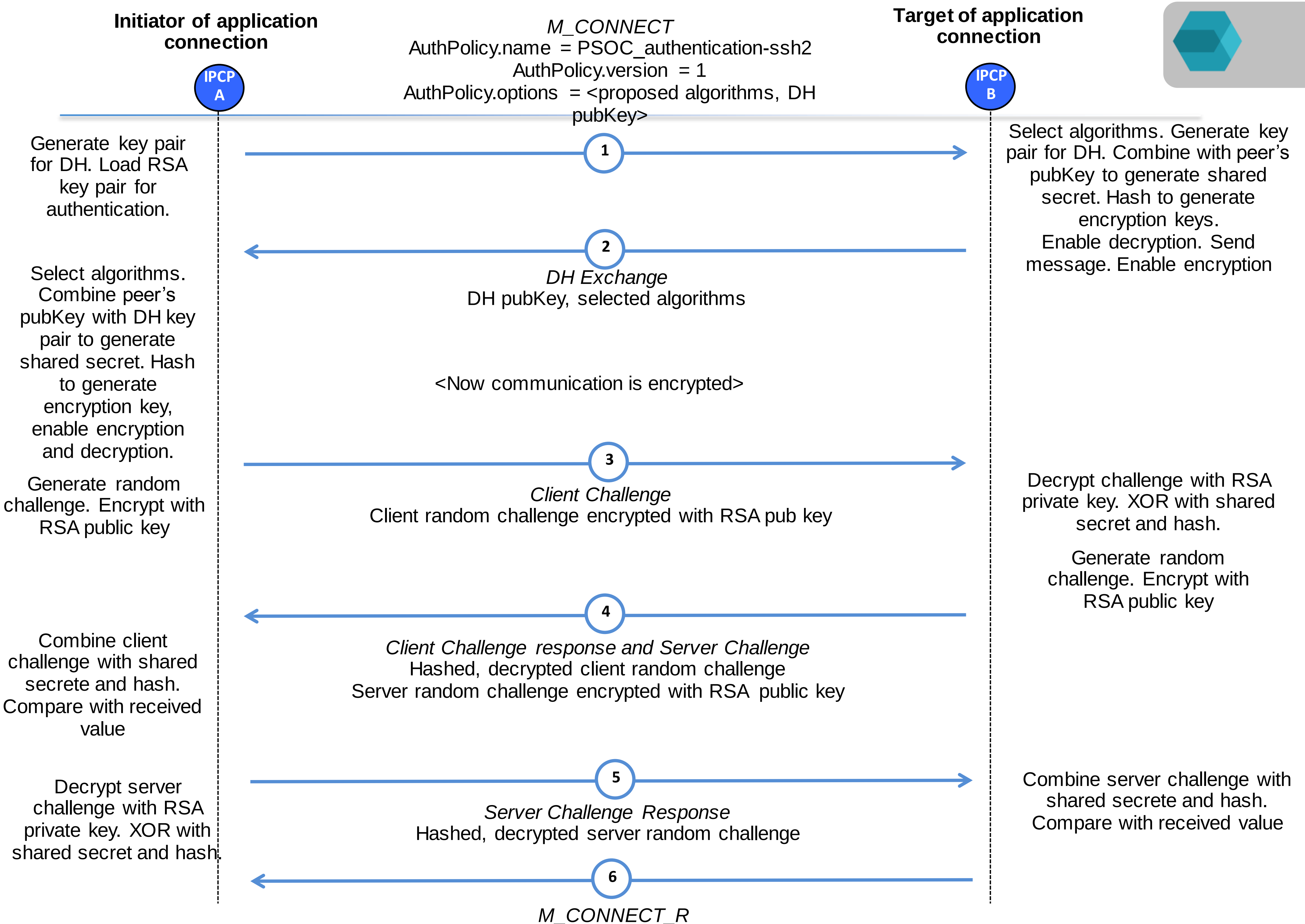


- Different security policies depending on who can join the DIF and trust on N-1 DIFs
- Recursion provides isolation: internal provider layers are not visible to other customer or provider networks (unless the provider's systems are physically compromised)

Example: AuthPassword policy
(implemented in prototype)



Example: AuthSSH2 policy (implemented in prototype)



5 FINAL REMARKS

Final remarks

1. Progress on improvement of core protocols (EFCP, CDAP). Started working on policy specifications for authentication, access control, SDU Protection, routing, congestion control and resource allocation.

2. Prototype (programmable via SDK) and Simulator maturing as they are used in experiments. Getting ready to become usable to newcomers (experiment tutorials under development).

3. Started quantifying RINA benefits on particular areas and specific scenarios/use cases (more to come during the following year). Current focus is congestion control, resource allocation, routing, security and network management.

4. Started working with SDOs to educate them on RINA and consider possible standardisation activities (ISO, ETSI). 5G and IoT are areas of potential interest.

<Thank you!>

Further information can be found here.

Twitter @ictpristine
www www.ict-pristine.eu