ETSI GR NGP 009 V1.1.1 (2019-02)

Next Generation Protocols (NGP); An example of a non-IP network protocol architecture based on RINA design principles

Disclaimer

The present document has been produced and approved by the Next Generation Protocols (NGP) ETSI Industry Specification Group (ISG) and represents the views of those members who participated in this ISG.
It does not necessarily represent the views of the entire ETSI membership.

Reference
DGR/NGP-009

Keywords
API, architecture, internet, meta-protocol,
network, next generation protocol, protocol

ETSI

650 Route des Lucioles
F-06921 Sophia Antipolis Cedex - FRANCE

Tel.: +33 4 92 94 42 00 Fax: +33 4 93 65 47 16

Siret N° 348 623 562 00017 - NAF 742 C
Association à but non lucratif enregistrée à la
Sous-Préfecture de Grasse (06) N° 7803/88

Contents

Intellectual Property Rights

Essential patents

IPRs essential or potentially essential to normative deliverables may have been declared to ETSI. The information pertaining to these essential IPRs, if any, is publicly available for **ETSI members and non-members**, and can be found in ETSI SR 000 314: *"Intellectual Property Rights (IPRs); Essential, or potentially Essential, IPRs notified to ETSI in respect of ETSI standards"*, which is available from the ETSI Secretariat. Latest updates are available on the ETSI Web server (https://ipr.etsi.org/).

Pursuant to the ETSI IPR Policy, no investigation, including IPR searches, has been carried out by ETSI. No guarantee can be given as to the existence of other IPRs not referenced in ETSI SR 000 314 (or the updates on the ETSI Web server) which are, or may be, or may become, essential to the present document.

Trademarks

The present document may include trademarks and/or tradenames which are asserted and/or registered by their owners. ETSI claims no ownership of these except for any which are indicated as being the property of ETSI, and conveys no right to use or reproduce any trademark and/or tradename. Mention of those trademarks in the present document does not constitute an endorsement by ETSI of products, services or organizations associated with those trademarks.

Foreword

This Group Report (GR) has been produced by ETSI Industry Specification Group (ISG) Next Generation Protocols (NGP).

Modal verbs terminology

In the present document "**should**", "**should not**", "**may**", "**need not**", "**will**", "**will not**", "**can**" and "**cannot**" are to be interpreted as described in clause 3.2 of the ETSI Drafting Rules (Verbal forms for the expression of provisions).

"**must**" and "**must not**" are **NOT** allowed in ETSI deliverables except when used in direct citation.

Introduction

Network protocol architecture provides a set of patterns and methodology that guides network (protocol) designers in carrying out their task. It captures the rules and patterns that are invariant with respect to the specific requirements of each individual network (cellular, datacentre, sensor, access, core, enterprise, LAN, etc.). Today the prevalent network protocol architecture (usually referred to as "Internet protocol suite"), loosely based on OSI provides too little patterns and commonality and has fundamental design flaws in its structure, naming and addressing, service API and security. These issues contribute to an explosion in the number of the network protocols required, both to cover requirements of multiple use cases and to work around the fundamental design flaws.

The Recursive InterNetwork Architecture (RINA) is a "back to basics" approach learning from the experience with TCP/IP and other technologies in the past. Research results to date have found that many long-standing network problems can inherently be solved by the structure resulting from the theory of networking. Hence, additional mechanisms are not required.

RINA provides the adequate tools to solve the problems of the Internet architecture (complexity, scalability, security, mobility, quality of service or management to name a few). RINA is based on a single type of layer, which is repeated as many times as required by the network designer. The layer is called a Distributed IPC Facility (DIF), which is a distributed application that provides Inter Process Communication (IPC) services over a given scope to the distributed applications above (which can be other DIFs or regular applications). These IPC services are defined by the DIF API, which allows instances of applications -including other DIFs- to request IPC flows with certain characteristics (such as loss, delay, in-order delivery) to other application instances. Hence a layer can is a resource allocator that provides and manages the IPC service oven a given scope (link, network, internetwork, VPN, etc.). It allocates resources (memory in buffers, bandwidth, scheduling capacity) to competing flows.

All DIFs offer the same services through their API and have the same components and structure. Each layer features two sets of protocol frameworks: one for data transfer (called EFCP, Error and Flow Control Protocol), and one for layer management (CDAP, the Common Distributed Application Protocol). However, not all the DIFs operate over the same scope and environment nor do they have to provide the same level of service. Hence, invariant parts (mechanisms) and variant parts (policies) are separated in different components of the data transfer and layer management protocol frameworks. This makes it possible to customize the behaviour of a DIF to optimally operate in a certain environment with a set of policies for that environment instead of the traditional "one size fits all" approach or having to re-implement mechanisms in independent protocols over and over again.

Last but not least, RINA can be deployed incrementally where it has the right incentives, and interoperate with current technologies such as IP, Ethernet, MPLS, WiFi, Cellular or others.

1 Scope

Today most network protocols loosely follow the layering structure of the OSI network architecture. Protocols are organized in a static number of layers, in which each layer provides a different function to the layer above. The limitations of such structure have led to an explosion in the number of protocols at each layer with little or no commonality, layer violations and the need for ad-hoc extensions in the number of layers where the architecture could not model real-world networks with enough fidelity (e.g. layers 2,5 or 3,5, virtual networks, etc.). SDOs independently develop protocols for different layers of the protocol architecture, many times replicating each other's work and leading to inefficiencies at the system level. This results in:

a) networks that are highly complex to operate and troubleshoot;

b) specification and implementation of new protocols which add little value to the existing base; and

c) an overall networked system that is far from an optimal integration level from a systems design perspective.

The present document discusses the properties of a non-IP network architecture based on RINA design principles. Network architecture captures all the rules and patterns that are independent of the requirements addressed by individual network protocols. It solves the problems that are generic to any network (e.g. structure, naming and addressing, security models or QoS) at the architecture level, avoiding the need for individual protocols to solve these problems by themselves. RINA has been designed to capture the invariants of all forms of networking, providing SDOs and network designers with a common framework and methodology to design and build protocols for any type of network. Thus a network protocol architecture like RINA encourages networks with fewer protocols and more commonality, more cooperation between SDOs and simpler and more predictable networks.

2 References

2.1 Normative references

Normative references are not applicable in the present document.

2.2 Informative references

References are either specific (identified by date of publication and/or edition number or version number) or non-specific. For specific references, only the cited version applies. For non-specific references, the latest version of the referenced document (including any amendments) applies.

NOTE: While any hyperlinks included in this clause were valid at the time of publication, ETSI cannot guarantee their long term validity.

The following referenced documents are not necessary for the application of the present document but they assist the user with regard to a particular subject area.

[i.1] IETF RFC 62: "A System for Inter-Process Communication in a Resource Sharing Network", August 1970, D.C. Walden.

NOTE: Available at https://tools.ietf.org/rfc/rfc62.txt.

[i.2] INWG-96 (1975): "Proposal for an International End To End Protocol", V. Cerf, A. McKenzie, R. Scantleburie, H. Zimmerman.

NOTE: Available at http://dotat.at/tmp/INWG-96.pdf.

[i.3] IETF RFC 793: "Transmission Control Protocol", September 1981, University of Southern California.

NOTE: Available at https://tools.ietf.org/html/rfc793.

[i.4] ETSI GS NGP 007: "Next Generation Protocols (NGP); NGP Reference Model".

NOTE: Available at https://www.etsi.org/deliver/etsi_gs/NGP/001_099/007/01.01.01_60/gs_NGP007v010101p.pdf.

[i.5] ISO/IEC 7498-1:1994: "Information technology -- Open Systems Interconnection -- Basic Reference Model: The Basic Model".

[i.6] IETF draft-ietf-taps-arch-02: "An Architecture for Transport Services", January 2018, T. Pauly, B. Trammell, A. Brunstrom, G. Fairhurst, C. Perkins, P. Tiesel, C. Wood.

NOTE: Available at https://datatracker.ietf.org/doc/draft-ietf-taps-arch/?include_text=1.

[i.7] P. B. Hansen: "The Nucleus of a Multi Programming System", Communications of the ACM 13 (4): 238-241. April 1970.

NOTE: Available at https://dl.acm.org/citation.cfm?id=362278&dl=ACM&coll=GUIDE.

[i.8] J. Day: "Patterns in Network Architecture: A return to Fundamentals". Prentica Hall, 2008.

[i.9] R. Watson: "Timer-based mechanism in reliable transport protocol connection management", Computer Networks, 5:47-56, 1981.

[i.10] G. Gursun, I. Matta and K. Mattar: "On the Performance and Robustness of Managing Reliable Transport Connections", 8th International Workshop on PFLDNeT, November 2010.

[i.11] G. Boddapati, J. Day, I. Matta, L. Chitkushev: "Assessing the security of a clean-slate Internet architecture", 20th IEEE conference on Network Protocols, 2012.

[i.12] E. Grasa, O. Rysavy, O. Lichtner, H. Asgari, J. Day, L. Chitkushev: "From protecting protocols to protecting layers: designing, implementing and experimenting with security policies in RINA", IEEE ICC 2016.

[i.13] IETF RFC 4291: "IPv6 addressing architecture", February 2006, R. Hinden, S. Deering.

NOTE: Available at https://tools.ietf.org/html/rfc4291.

[i.14] IETF RFC 4192: "Procedures for renumbering an IPv6 network without a flag day", September 2005, F. Baker, E. Lear, and R. Droms.

NOTE: Available at https://tools.ietf.org/html/rfc4192.

[i.15] IETF RFC 5887: "Renumbering still needs work", May 2010, B. Carpenter, R. Atkinson, and H. Flinck.

NOTE: Available at https://tools.ietf.org/html/rfc5887.

[i.16] D. Leroy and O. Bonaventure: "Preparing network configurations for IPv6 renumbering," International Journal of Network Management, vol. 19, no. 5, pp. 415-426, September/October 2009.

[i.17] J. Small: "Threat analysis of recursive internetwork architecture distributed IPC facilities", BU Technical report, 2011.

NOTE: Available at http://pouzinsociety.org/research/publications.

[i.18] Eduard Grasa, Leonardo Bergesio, Miquel Tarzan, Diego Lopez, John Day and Lou Chitkushev: "Seamless Network Renumbering in RINA: Automate Address Changes Without Breaking Flows!", EUCNC 2017.

NOTE: Available at https://zenodo.org/record/1013204#.WeTDrROCxTY.

[i.19] IETF RFC 5944: "IP mobility support for IPv4, revised", November 2010, C. Perkins.

NOTE: Available at https://tools.ietf.org/html/rfc5944.

[i.20] IETF RFC 6275: "Mobility support in IPv6", July 2011, J. A. C. Perkins, D. Johnson.

NOTE: Available at https://tools.ietf.org/html/rfc6275.

[i.21] IETF RFC 5213: "Proxy mobile IPv6", August 2008, S. Gundavelli, K. Leung, V. Devarapalli, K. Chowdhury and B. Patil.

NOTE: Available at https://tools.ietf.org/html/rfc5213.

[i.22] IETF RFC 6830: "The Locator/ID Separation Protocol (LISP)", January 2013, D. Meyer, D. Lewis, D. Farinacci, V. Fuller.

[i.23] J. Day and E. Grasa: "Mobility Made Simple". PSOC Tutorial paper, May 2016.

NOTE: Available at http://psoc.i2cat.net/sites/default/files/PSOC-tutorial-Mobility-made-simple.pdf?_ga=2.45065702.356832367.1537337692-718608749.1512423093.

[i.24] V. Ishakian, J. Akinwumi, F. Esposito and I. Matta: "On supporting mobility and multihoming in recursive internet architectures", Comput. Commun., vol. 35, no. 13, pp. 1561-1573, July 2012.

[i.25] ETSI GS NGP 001 (V1.3.1): "Next Generation Protocols (NGP); Scenario Definitions".

NOTE: Available at https://www.etsi.org/deliver/etsi_gs/NGP/001_099/001/01.03.01_60/gs_NGP001v010301p.pdf.

[i.26] IETF RFC 1287: "Towards the Future Internet Architecture", December1991, D. Clark, L. Chapin, V. Cerf, R. Braden, R. Hobby.

NOTE: Available at https://tools.ietf.org/html/rfc1287.

[i.27] J. Day: "How in the heck do you lose a layer!?", International Conference on the Network of the Future, 2011.

[i.28] Atlantic Council, Frederik S. Pardee Center for International Futures, Zurich: "Risk Nexus. Overcome by Cyber Risks? Economic benefits and costs of alternate cyber futures", 2016.

NOTE: Available at http://publications.atlanticcouncil.org/cyberrisks//.

[i.29] IETF RFC 2535: "Domain name system security extensions", March 1999, D. Eastlake.

NOTE: Available at https://tools.ietf.org/html/rfc2535.

[i.30] IETF RFC 2401: "Security architecture for the IP Protocol", December 2005, S. Kent, K. Seo.

NOTE: Available at https://tools.ietf.org/html/rfc2401.

[i.31] IETF RFC 6863: "Analysis of OSPF security according to the keying and authentication for routing protocols (karp) design guidelines", March 2013, S. Hartman, D. Zhang.

NOTE: Available at https://tools.ietf.org/html/rfc6863.

[i.32] IETF RFC 8205: "BGPsec protocol specification", September 2017, M. Lepinski, K. Sriram.

NOTE: Available at https://tools.ietf.org/html/rfc8205.

[i.33] IETF RFC 5246: "The Transport Layer Security Protocol, version 1.2", August 2008, T. Diersk, R. Escola.

NOTE: Available at https://tools.ietf.org/html/rfc5246.

[i.34] J. Small: "Patterns in network security: An analysis of architectural complexity in securing RINA networks", Boston University Computer Science Department, Master thesis, 2012.

NOTE: Available at https://open.bu.edu/handle/2144/17155.

[i.35] IETF draft-ietf-tsvwg-natsupp: "Stream Control Transmission Protocol network address translation", July 2017, R. Stewart, M. Tuexen, I. Rengeler.

[i.36] R. Lychev, S. Goldberg and M. Schapira: "BGP security in partial deployment: Is the juice worth the squeeze?", ACM SIGCOMM 2013.

[i.37] M. Bari, R. Boutaba, R. Esteves, L. Granville, M. Podlesny, M. Rabbani, Q. Zhang and M. Zhani: "Data center network virtualization: A survey", IEEE Communications Surveys and Tutorials, vol. 15, no. 2, 2013.

[i.38] B. Schneier: "A plea for simplicity: You can't secure what you don't understand", Information Security, 1999.

NOTE: Available at https://www.schneier.com/essays/archives/1999/11/a_plea_for_simplicit.html.

[i.39] J. Mirkovic and P. Reiher: "A taxonomy of DDoS attacks and DDoS defense mechanisms", ACM SIGCOMM Computer Communications Review, vol. 34, no. 2, pages 39-53, 2004.

[i.40] IEEE 802.1aq™: "Standard for Local and Metropolitan Area Networks: Virtual Bridged Local Area Networks - Amendment 8: Shortest Path Bridging, 802.1aq", April 2012.

NOTE: Available at https://standards.ieee.org/standard/802_1aq-2012.html.

[i.41] E. Grasa, B. Gastón, S. van der Meer, M. Crotty and M. A. Puente: "Simplifying multi-layer Network Management with RINA", Proceedings of the TNC conference, 2016.

NOTE: Available at https://tnc16.geant.org/core/presentation/667.

[i.42] J. Day: "How Distributed is Distributed Management? Or can too many cooks spoil the broth?", Pouzin Society blog post.

NOTE: Available at http://pouzinsociety.org/node/55.

[i.43] PRISTINE Consortium: "D5.4 consolidated dif management system", PRISTINE deliverable D5.4, July 2016.

NOTE: Available at http://ict-pristine.eu/wp-content/uploads/2018/05/pristine-d54-consolidated-network-management-system_v1_0.pdf.

[i.44] ARCFIRE consortium, deliverable D3.1: "Integrated software ready for experiments: RINA stack, Management System and measurement framework", H2020 ARCFIRE, December 2016.

NOTE: Available at http://ict-arcfire.eu/wp-content/uploads/2017/10/arcfire_d31-final.pdf.

[i.45] V. Maffione: "Port of the dropbear ssh client/server for rina", December 2016.

NOTE: Available at https://github.com/vmaffione/rina-dropbear.

[i.46] M. Williams: "Prototype sockets emulator for rina", September 2017.

NOTE: Available at https://github.com/rlite/rina-dropbear.

[i.47] S. Vrijders, E. Trouva, J. Day, E. Grasa, D. Staessens, D. Colle, M. Pickavet and L. Chitkushev: "Unreliable inter process communication in Ethernet: migrating to RINA with the shim DIF," in 5th International Workshop on Reliable Networks Design and Modeling (RNDM-2013), 2013, pp. 97-102.

[i.48] IRATI Consortium: "IRATI Deliverable D2.4, third phase use cases, updated RINA specification and high-level software architecture", IRATI website, December 2014.

NOTE: Available at http://irati.eu/wp-content/uploads/2012/07/IRATI-D2.4-bundle.zip.

[i.49] IRATI Consortium: "Irati Deliverable D3.3, second phase integrated rina prototype for hypervisors for aunix-like OS", June 2014.

NOTE: Available at http://irati.eu/wp-content/uploads/2012/07/IRATI-D3.3-bundle.zip.

[i.50] V. Maffione: "Rina-tcp gateway implementation", September 2016.

NOTE: Available at https://github.com/rlite/rlite#71-rina-gw.

[i.51] S. Vrijders, V. Maffione, D. Staessens, F. Salvestrini, M. Biancani, E. Grasa, D. Colle, M. Pickavet, J. Barron, J. Day and L. Chitkushev: "Reducing the complexity of virtual machine networking", IEEE Communications Magazine, vol. 54, no. 4, pp. 152-158, April 2016.

[i.52] P. Teymoori, M. Welzl, S. Gjessing, E. Grasa, R. Riggio, K. Rausch and D. Siracussa: "Congestion control in the recursive internetwork architecture (RINA)", IEEE ICC 2016, Next Generation Networking and Internet Symposium, 2016.

[i.53] S. León, J. Perelló, D. Careglio, E. Grasa, D. Lopez and P. A. Aranda: "Benefits of programmable topological routing policies in rina-enabled large-scale datacentres", IEEE Globecom 2016, Next Generation Networks Symposium, December 2016.

[i.54] V. Maffione: "Prototype rina-enabled vrf implementation", May 2018.

NOTE: Available at https://github.com/rlite/rlite#72-iporinad.

[i.55] ARCFIRE consortium, deliverable D4.3: "Design of experimental scenarios; selection of metrics and kpis", H2020 ARCFIRE, January 2017.

NOTE: Available at http://ict-arcfire.eu.

[i.56] IEEE 802.11™: "IEEE Standard for Information technology--Telecommunications and information exchange between systems Local and metropolitan area networks--Specific requirements - Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications".

3 Definition of terms, symbols and abbreviations

3.1 Terms

For the purposes of the present document, the following terms apply:

Application Process (AP): instantiation of a program executing in a processing system intended to accomplish some purpose

NOTE: The Application Process definition aims to be quite abstract and applicable to a broad range of hardware and software (CPUs, FPGAs, ASICs and other platforms). See the definition of "Processing System" below.

Common Distributed Application Protocol (CDAP): application protocol component of a Distributed Application Facility (DAF) that can be used to construct arbitrary distributed applications, of which the DIF is an example

NOTE: CDAP enables distributed applications to exchange and operate structured data objects, rather than forcing applications to explicitly deal with serialization and input/output operations.

Data Transfer Control Protocol (DTCP): optional part of EFCP that provides the loosely-bound mechanisms

NOTE: Each DTCP instance is paired with a DTP instance to control the flow, based on its policies and the contents of the shared state vector.

Data Transfer Protocol (DTP): required data transfer part of EFCP consisting of tightly bound mechanisms found in all DIFs, roughly equivalent to IP and UDP

NOTE: When necessary DTP coordinates through a state vector with an instance of the Data Transfer Control Protocol. There is an instance of DTP for each flow.

Distributed Application Facility (DAF): collection of two or more cooperating Application Processes in one or more processing systems, which exchange information using the IPC services provided by a DIF and maintain shared state

NOTE: In some Distributed Applications, all members will be the same, i.e. a homogeneous DAF, or may be different, a heterogeneous DAF.

Distributed IPC Facility (DIF) layer: collection of two or more Application Processes cooperating to provide Interprocess Communication (IPC)

NOTE: A DIF is a DAF that does IPC. The DIF provides IPC services to Applications via a set of API primitives that are used to exchange information with the Application's peer.

Error and Flow Control Protocol (EFCP): data transfer protocol required to maintain an instance of a communication service within a DIF

NOTE: The functions of this protocol ensure reliability, order, and flow control as required. It consists of separate instances of DTP and optionally DTCP, which coordinate through a state vector.

flow: service provided by an EFCP-instance to an application process

NOTE: The binding between an EFCP-instance and the application process using it is called a port.

Flow Allocator (FA): layer management component of the IPC Process that responds to Allocation Requests from Application Processes

NOTE: A Flow Allocator Instance (FAI) is created for each Allocate Request. The FAI is responsible for:

1) finding the address of the IPC-Process with access to the requested destination-application;
2) determining whether the requesting Application Process has access to the requested Application Process;
3) selecting the policies to be used on the flow;
4) monitoring the flow; and
5) managing the flow for its duration.

Inter Process Communication (IPC): service provided by a DIF to two or more instances of Application Processes, allowing them to exchange information

IPC Process (IPCP): Application Process, which is a member of a DIF and implements locally the functionality to support and manage IPC using multiple sub-tasks

layer: set of protocol machines sharing state under a certain scope

NOTE: In the context of RINA, a layer is a Distributed IPC Facility.

(N)-DIF: nomenclature to indicate the rank of a DIF, as a basis to describe its relationship with DIFs in the ranks above ((N+1)-DIF)) and below ((N-1)-DIF)

peer IPCP: IPCP in the same DIF that is one hop away, without requiring another IPCP to act as a relay

NOTE: In general, peer IPCPs should have an N-1 DIF in common.

processing system: hardware and software capable of executing programs instantiated as Application Processes that can coordinate with the equivalent of a "test and set" instruction, i.e. the tasks can all atomically reference the same memory

Protocol-Data-Unit (PDU): string of octets exchanged among the Protocol Machines (PM)

NOTE: PDUs contain two parts. The PCI (Protocol Control Information), which is understood and interpreted by the DIF, and User-Data, that is incomprehensible to this PM and is passed to its user.

Protocol Machine (PM): implementation of the protocol logic that exchange state information with a peer PM by inserting protocol control information into a PDU on one side (sender), and striping it in the other side (receiver)

Relaying/Multiplexing-Task (RMT): RMT performs the real time scheduling of sending PDUs on the appropriate (N-1)-ports of the (N-1)-DIFs available to the RMT

NOTE: This task is an element of the data transfer function of a DIF. Logically, it sits between the EFCP and SDU Protection.

Resource Allocator (RA): component of the DIF that manages resource allocation and monitors the resources in the DIF by sharing information with other DIF IPC Processes and the performance of supporting DIFs

Resource Information Base (RIB): logical representation of the local repository of the objects exposing the externally visible state of an Application Process

NOTE: Each member of the DAF maintains a RIB. A Distributed Application may define a RIB to be its local representation of its view of the distributed application.

RIB Daemon: layer management component of the IPC Process that optimizes the requests for information from the other layer management tasks of the IPCP

NOTE: Each local Application Process participating in a Distributed Application may have several sub-tasks or threads. Each of these may have requirements for information from other participants in the distributed application on a periodic or event driven basis.

Service-Data-Unit (SDU): amount of data passed across the (N)-DIF interface to be transferred to the destination application process

NOTE: The integrity of an SDU is maintained by the (N)-DIF. An SDU may be fragmented or combined with other SDUs for sending as one or more PDUs.

3.2 Symbols

Void.

3.3 Abbreviations

For the purposes of the present document, the following abbreviations apply:

ACL	Access Control List
ADDR	Address
AP	Application Process
API	Application Programming Interface
AS	Autonomous System
ASIC	Application-Specific Integrated Circuit
ASN	Abstract Syntax Notation
BGP	Border Gateway Protocol
BS	Base Station
BSS	Basic Service Set
CACEP	Common Application Connection Establishment Phase
CDAP	Common Distributed Application Protocol
CEPID	Connection EndPoint IDentifier
CMIP	Common Management Information Protocol
CPU	Central Processing Unit
CRC	Cyclic Redundancy Check
CSP	Communication Service Provider
DAF	Distributed Application Facility
DC	Data Centre
DCCP	Datagram Congestion Control Protocol
DDoS	Distributed Denial of Service
DIF	Distributed IPC Facility
DMM	Distributed Mobility Management
DNS	Domain Name System
DNSSEC	DNS Security
DSCP	Differential Services Code Point
DST	Destination
DTCP	Data Transfer Control Protocol
DTP	Data Transfer Protocol
ECN	Explicit Congestion Notification
EFCP	Error and Flow Control Protocol

EPS	Evolved Packet System
FA	Flow Allocator
FAI	Flow Allocator Instance
FPGA	Field Programmable Gate Array
GRE	Generic Routing Encapsulation
GTP	GPRS Tunnelling Protocol
IP	Internet Protocol
IPC	Inter Process Communication
IPCP	IPC Process
IPsec	Internet Protocol Security
IRATI	Investigating RINA as an Alternative to TCP/IP
IS-IS	Intermediate System to Intermediate System
LAN	Local Area Network
LISP	Locator Identifier Separation Protocol
LTE	Long Term Evolution
MAC	Medium Access Control
MH	Mobile Host
MIPv4	Mobile IPv4
MIPv6	Mobile IPv6
MPLS	Multi-Protocol Label Switching
MP-TCP	Multi-Path Transmission Control Protocol
NAT	Network Address Translation
NMS	Network Management System
NSM	NameSpace Manager
NVGRE	Network Virtualization using Generic Routing Encapsulation
OS	Operating System
OSI	Open Systems Interconnection
OSPF	Open Shortest Path First
PBB	Provider Backbone Bridging
PCI	Protocol Control Information
PDCP	Packet Data Convergence Protocol
PDU	Protocol Data Unit
PE	Provider Edge
PEP	Performance Enhancing Proxy
PMIPv6	Proxy Mobile IPv6
POSIX	Portable Operating System Interface
QoS	Quality of Service
RA	Resource Allocator
RIB	Resource Information Base
RINA	Recursive InterNetwork Architecture
RIP	Routing Information Protocol
RLC	Radio Link Control
RMT	Relaying and Multiplexing Task
SCTP	Stream Control Transmission Protocol
SDN	Software Defined Networking
SDO	Standards Developing Organization
SDU	Service Data Unit
SD-WAN	Software Defined Wide Area Network
SID	Shared Information and Data model
SLA	Service Level Agreement
SMI	Structure of Management Information
SNMP	Simple Network Management Protocol
SRC	Source
SSH	Secure Shell
TAPS	Transport Services
TCP	Transmission Control Protocol
TLS	Transport Layer Security
UDP	User Datagram Protocol
UE	User Equipment
URL	Universal Resource Locator
VLAN	Virtual Local Area Network
VM	Virtual Machine

VPLS	Virtual Private LAN Service
VPN	Virtual Private Network
VRF	Virtual Routing and Forwarding
VXLAN	Virtual eXtensible Local Area Network
WAN	Wireless Area Network

4 Overview and motivation

4.1 What does the present document mean by "network protocol architecture"?

Architecture is the style of design and method of construction of buildings and physical structures. Architecture provides a set of patterns and methodology that guides building designers in carrying out their task. The same architecture is used to design many different buildings with different requirements: architecture captures the rules and patterns that are invariant with respect to the specific requirements of each individual building.

To illustrate the explanation of the paragraph above with an example, let us consider the Gothic architecture style. It features a set of common structural elements that are common across multiple buildings with different requirements. Amongst such common elements can be found:

a) grand, tall designs which swept upwards with height and grace;

b) flying buttresses;

c) pointed arches;

d) vaulted ceilings;

e) light, airy interiors; or

f) the emphasis upon the decorative style and the ornate.

With these elements architects designed a myriad of buildings like cathedrals, city halls, palaces, fish markets or city gates. Architects adapted the elements of the architecture to the program required by the buildings they were designing.

However, the tasks network protocol designers are concerned with are not related to building physical structures, but to design protocols that support networks of computing devices. The term 'computing' here is used in the broad sense of the word: devices that produce some kind of computation or equivalent. Hence this category includes sensors, smartphones, tablets, laptops, personal computers, high-end servers, routers, switches, etc.

Computer networking is an enabler for distributed computing. It provides communication services to instances of distributed application processes (again, distributed application processes used in a loose way, may be OS application processes, or sensors, actuators, etc.). The service that computer networking provides to applications is that of imperfect remote data replication services. Networks can be characterized as distributed data copying machines of finite capacity, that introduce delay and loss while carrying out their task. In other words, computer networking can be seen as distributed IPC (Inter Process Communication), a view that has been taken by the pioneers in the field [i.1], [i.2], [i.3]; and is also consistent with the generic protocol model described in ETSI GS NGP 007 [i.4].

Hence, network protocol architecture provides a set of patterns and methodology that guides network (protocol) designers in carrying out their task. It captures the rules and patterns that are invariant with respect to the specific requirements of each individual network (cellular, datacentre, sensor, access, core, enterprise, LAN, etc.). Therefore, for the purposes of the present document, network protocol architecture is defined as the *general rules and patterns to provide distributed IPC services to any type of application over any type of physical media*.

4.2 What is the current network protocol architecture?

Part of the problems in computer networking today are due to the lack of precision and consistency when defining the elements of computer networks and their interaction. It is hard to reason logically about loosely defined concepts, since it leaves too much room for ambiguity and misinterpretations. Network protocol architecture is a good example of this issue, since there is no formal specification of what is the network protocol architecture in use today (or even if there is just only one or multiple).

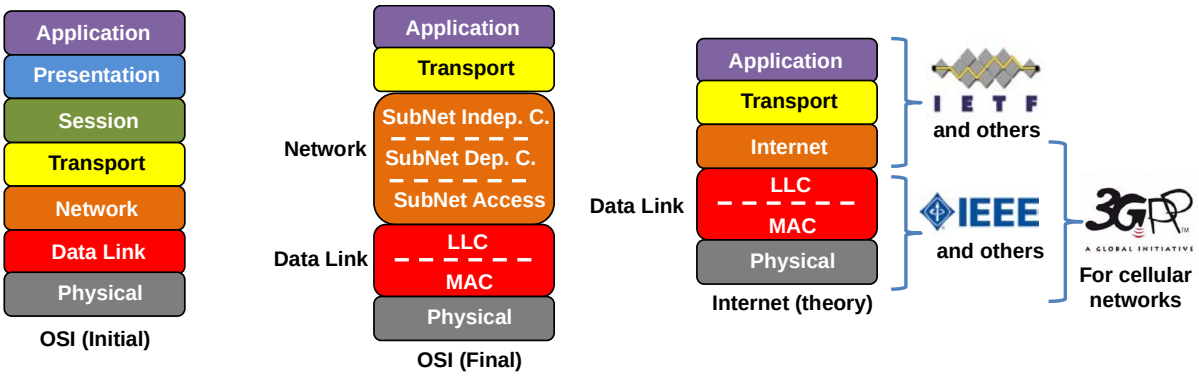


Figure 1: Initial and final OSI architecture models (left, centre) and Internet protocol suite model (right)

There is a rough consensus that the network protocol architecture is loosely based on OSI [i.5], the Open Systems Interconnection effort. The left part of Figure 1 shows the seven layers of the most widely known version of the OSI architecture. Although popular, this is not the final version of the OSI architecture, which is shown in the central part of Figure 1. It was realized that the three upper layers were in reality a single one (the application layer), and the network layer was divided into three independent sublayers:

a) subnet access layer, with network-dependent addresses, which performs relaying and multiplexing functions within a network;

b) subnet dependent convergence layer, which provides error and flow control over a single network; and

c) subnet independent convergence layer, with network-independent addresses, which performs relaying and multiplexing functions across networks.

Finally, the right part of Figure 1 illustrates the layers of what could be described as the current network /Internet architecture. The lower layers (physical and data link), are the same as in the OSI architecture. On top of the data link layer there is the "network" or "Internet" layer (depending on the sources it is called one way or the other) with addresses, and on top of it an end-to-end transport layer providing end to end error and flow control services to the application layer (featuring protocols that perform application-specific tasks). Most protocols in the data link and physical layers are standardized by the IEEE (notably Ethernet with all its variants), while protocols in the Internet, Transport and Application layers are standardized by the IETF. Other SDOs are focused on specific types of networks, such as the 3GPP for cellular networks and either use the protocols from the IEEE and/or IETF or define their own.

4.3 Summary of current issues at the network protocol architecture level

4.3.1 Structure

Layers as units of modularity. Network architectures in use today are mostly based on the functional layering paradigm, in which different layers perform different functions. The typical theoretical model describes 4 layers (physical, data link, network and transport) with application protocols on top. Layers are seen as a unit of modularity. The functions of each layer are performed by different protocols, which are independently designed from each other. This design pattern causes a lot of overhead and repetition (layer 2 protocols have many functions in common with layer 4 protocols, for instance).

Functions in different layers are not independent. The current partitioning of functions in different layers has proven to be problematic: IP fragmentation does not work because in order to operate correctly in needs information that TCP has, but IP has to operate independently from it (since it is in another layer).

A fixed number of layers limits the number of scopes. Layers enable the isolation of state of different scopes: this is their fundamental property. The theoretical architecture just has room for two scopes: data link and network; i.e. the whole Internet. In real internetworks this is clearly not sufficient - network providers want to isolate their internal resource allocation, routing and forwarding policies from the rest of the Internet, for example. This has caused the proliferation of new protocols which belong to layers 2,5 - such as MPLS -, virtual layers above transport - such as VXLAN - tunnels - such as GRE - or recursive encapsulations - such as MAC-in-MAC. These protocols have evolved with no organization to solve individual use-cases as they appeared, invalidating the original architectural model and increasing the complexity of networks, as seen in Figure 2.

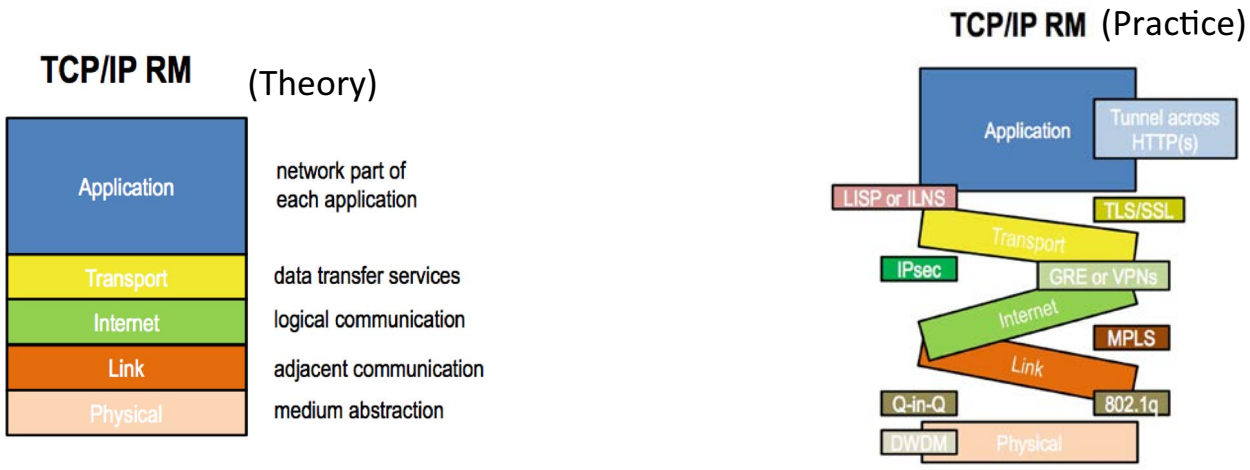


Figure 2: Internet architecture model is constantly extended to support new use cases

Incomplete and/or missing layer APIs. Most layers do not have a standard API that abstracts the service that the layer provides, therefore all users of a layer need to know the details about the specific protocols implementing the layer functionality. The only layer that has a (de-facto) standard API is the transport layer with the Sockets API. However Sockets do not hide the transport layer implementation details and expose individual transport protocols to applications. Therefore applications have to be aware of the choice of available transport protocols, and make a savvy decision based on its requirements. This fact not only makes networked applications more complex, but also makes the deployment of new transport protocols very hard [i.6].

4.3.2 Protocol design

Multiple protocols per layer. Even with the functional layering approach - in which each layer performs a different function - today there are multiple protocols that can be used in each layer. This is due to the fact that current protocols are hand-crafted to solve a particular use-case or to cover a small range of requirements. When these requirements cannot be met a new protocol is designed from scratch. The transport layer is a good example of this approach, in which different protocols like TCP, UDP, SCTP or DCCP mostly differ in the way they do flow and retransmission control. A protocol design strategy that identified major commonalities across protocols and tried to make a generic design adaptable to different environments would considerably reduce the number of protocols required in a network.

Protocols designed independently from each other. Closely related to the last bullet point, most protocols today are designed in isolation from each other, which maximizes specification and implementation work due to repeated functions that are re-invented by each protocol. For example all routing protocols in use today (OSPF, IS-IS, BGP, RIP, etc.) are about maintaining a partially distributed database across network nodes, and then applying different algorithms locally to compute forwarding tables. What changes from protocol to protocol is the strategy used for replication, the information exchanged and the local algorithms; but the mechanisms used to encode the information - abstract and concrete syntaxes - and to communicate updates can be common to all of them. Redesigning groups of protocols with common goals in order to maximize commonality between them would simplify designing, deploying, operating and managing network protocols.

4.3.3 Naming, addressing and routing

Lack of application names. A well-formed network architecture needs application names that are location-independent, so that applications attached to the network can maintain their identity regardless of where they are. In the Internet URLs may appear to play this role, but a close inspection reveals that this is not right. The URL is a concatenation of a domain name, a transport layer port and a string that is meaningful to the application. Domain names are just synonyms to IP addresses (they are resolved to IP addresses by DNS, which is a directory service external to the network). When an application requests a transport service to the sockets API it passes the address where the destination application is attached: the network does not know about names. If the application changes its point of attachment to the network, its IP address changes and all the transport connections to that application are broken. Also, access control rules (e.g. firewall rules) are written in terms of IP addresses and transport ports; which means that they have to be updated every time an application changes its attachment to the network.

Naming the interface instead of the node. Within a layer a node is the protocol machine that strips off the header of that protocol in that layer. Therefore protocol PDUs within a layer are addressed to nodes. However in the TCP/IP protocol suite addresses are assigned to interfaces (Points of attachment to the layer below), not nodes. This fact makes real multi-homing impossible to support (because the network does not know that two interface addresses reach the same node), makes router forwarding tables 3/4 times larger and greatly complicates mobility.

No names for layers, no layer directory. Although some layers do have names (BSS-id in the case of WLANs or VLAN-ids in the case of VLANs), the upper layers close to the applications do not have them. This fact, together with the lack of application names, cause applications to be available through a single layer at a time (usually the topmost one), or to multiple layers if the application is attached to interfaces of multiple top layers (e.g. an interface belonging to the public Internet, another interface belonging to a private IP network). The lack of layer directories makes discovering what is the best layer to reach an application impossible: this information has to be statically configured into client applications.

4.3.4 Mobility and multi-homing

Need for specialized protocols to support multi-homing. A number of approaches has been proposed to achieve multi-homing in current networks. Classical (or true) multi-homing requires provider-independent addresses - which cannot be aggregated in the global Internet routing table - a dynamic routing protocol (typically BGP) and at least two links: each one to a different provider. When one of the links to a provider fails BGP will recompute the AS path so that the other provider is selected. However this only allows multi-homing at the AS-level. Host multi-homing has partial support through protocols that exploit the use of multiple IP interfaces (such as SCTP, MP-TCP or SHIM6). However in this approach packets that are already en route towards an IP address belonging to an interface that is down will be lost (hence, only partial multi-homing support is achieved). The whole approach makes multi-homing complex and only partially supported.

Need for specialized protocols and artefacts to achieve mobility. That support for mobility - which can be just seen as dynamic multi-homing - is cumbersome should come to no surprise. The lack of application names and node addresses require the use of special protocols and artefacts to support some degree of host mobility. Most current mobility solutions require setting up tunnels through specialized protocols. Managing these tunnels results in significant overhead, since they have to be updated every time the mobile host changes its point of attachment to the network. Different types of mobile networks with varying support for host mobility require different protocols (e.g. WiFi vs. cellular). Tunnel-less mobility solutions based on true multi-homing such as the one achieved via BGP have been investigated but proven too slow, since BGP was never designed to cope with the fast-changing nature of mobile access networks.

4.3.5 Quality of Service, resource allocation, congestion control

Lack of a consistent QoS model across layers. The lack of a clear, consistent model to communicate performance requirements across layers make it hard to manage the performance of a multi-layer network so that it delivers consistent service experiences to its customers. Since there are no abstract mechanisms for the communication of performance requirements between layers, this communication has to be designed on a use-case per use-case basis, usually involving the Network Management System. The layer also needs to identify the flows belonging to different QoS groups, which also depends on the technology being used: DSCP code marking can be used in the case of IP, MPLS traffic engineering based on the values of MPLS labels, VLAN tags for Ethernet or I-SID (service identifiers) in the case of PBB.

Only end-to-end (too long) congestion control loops. Dealing with congestion at the transport layer only maximizes the length of the congestion control loop, which also maximizes the time to react to congestion and its variance (oscillations). As a consequence, TCP senders may be reacting to congestion that is no longer there; or by the time the TCP sender reacts the network is already in a very congested state. This is a particular problem on cellular networks, where the highly-variable signal strength experienced by a mobile host can cause the network's allocation of radio resources for that host to vary by three orders of magnitude in a time far less than the end-to-end round trip. If congestion was managed at the level of each individual network, shorter and more effective control loops would be possible.

Predatory (implicit) congestion control. The use of lost or duplicate packets as a signal of congestion causes TCP to interpret packet loss due to link failures as congestion (which is usual in TCP-over-wireless scenarios). As a consequence TCP overreacts to inexistent congestion, degrading the throughput and delay experienced by applications. Explicit Congestion Notifications (ECN) signalling by routers in the path traversed by the PDUs belonging to the TCP connection provides a better feedback signal to react to congestion, but this still requires an end-to-end round trip to trigger a reaction, rather than being dealt with at the point of congestion.

Homogeneous congestion control policies for heterogeneous networks. Transport protocol connections typically go through a number of different networks and/or network segments with different characteristics (wired/wireless access, aggregation, core, interconnect, datacentre, etc). Trying to find a single congestion controller that can behave optimally for the composed end to end loop control loop is just impossible: what is optimal for a certain segment is not efficient for another one. Again, if congestion is managed at the network or network segment level - through multiple, shorter control loops - each control loop can be optimized for the network segment it is controlling.

4.3.6 Security

Most protocols have their own security model. The TCP/IP protocol suite security model is usually based on building security functions for each protocol. For example, DNSSEC provides data integrity and authentication to security-aware resolvers. IPsec is a general framework for secure IP communications, supporting confidentiality, integrity, authentication or protection against replay attacks. However since IPsec works end-to-end within a layer, it either only protects the IP payload (transport mode) or makes IP connection-oriented (tunnel mode), encapsulating a protected IP packet into an unprotected IP packet. This makes IPsec a partial solution, not addressing the requirements of IP control plane protocols, which need to define their own security functions, such as OSPF or BGP. TLS, the Transport Layer Security Protocol, specifies a set of related security functions to enable secure communications over the transport layer. All in all, this approach results in a high overhead and unbounded complexity, since new protocols will in many cases require its companion security protocols.

Coupled port allocation and synchronization functions; use of well-known ports. Transport layer protocols overloads the port-id to be both a local handle (socket) and the connection-endpoint-id (CEPID). Furthermore the lack of application names overloads port-ids with application semantics: application endpoints are identified by a combination of IP address and a well-known port-id that is assigned when the application binds to an IP layer. Static destination port-id values have to be known by the source application when requesting a transport connection. Therefore an attacker wanting to intercept a particular transport connection only needs to guess/spoof the source port-id.

No application names: network addresses are exposed to applications. Since the TCP/IP protocol suite does not have application names (DNS is an external directory), IP layers expose addresses to applications. This disclosure of information facilitates spoofing of IP addresses, and in combination with the use of common monitoring tools such as traceroute or ping allows attackers on end hosts to learn about the addresses of potential targets in a layer as well as the network connectivity graph. Attackers can use this information to setup DDoS attacks by automating the discovery and infection of vulnerable machines, or to attack the network infrastructure by gaining control over routers.

4.3.7 Network Management

Too little commonality in network protocols. Different layers perform different functions, which require different protocols. Even within the same layer multiple protocols are required to adapt the layer to different operational requirements. Each protocol comes with its own protocol machine definition, its own configuration and its own state model. Managing networks designed this way is hard and cumbersome, since the NMS needs to understand the state, operation and configuration models of all the different protocols in the network, as well as their interactions.

No well-defined model for the interaction between layers. To manage a multi-layer network, not only needs the NMS to understand individual protocols, but also the interaction amongst the multiple protocol layers. Since there is no well-defined, consistent layer API, each layer interaction has to be dealt with on a case-by-case basis, complicating network configuration, performance and security management.

Different protocols and object models for network management. Although NETCONF and YANG are gaining traction as a protocol and object model for network configuration management, they are still far from covering all the network segments and technologies. CMIP and X.700, SNMP and SMI, and SID (the Shared Information and Data Model) are still prevalent in different networking areas. As a consequence converged operator networks today are likely to require multiple protocols and object models for management, complicating this task even more.

No bounded set of APIs for network programmability. SDN wants to enhance the programmability of network devices through the separation of the data forwarding and control planes. Current SDN approaches focus on defining APIs and protocols for data forwarding via centralized controllers. However, since SDN does not change the organization of network protocols and layers, and the number of network protocols is unbounded, the set of APIs required to support network programmability is also unbounded. Therefore programmability is achieved at the cost of increased complexity, making it difficult to agree on a set of standard APIs to facilitate programmability of network functions across a wide set of device manufacturers.

4.4 Goals for a generic network protocol architecture

A better network architecture than the one in place today needs to provide as much invariants as possible, so that the number of protocols required to cover all networking use cases can be minimized and their commonality maximized.

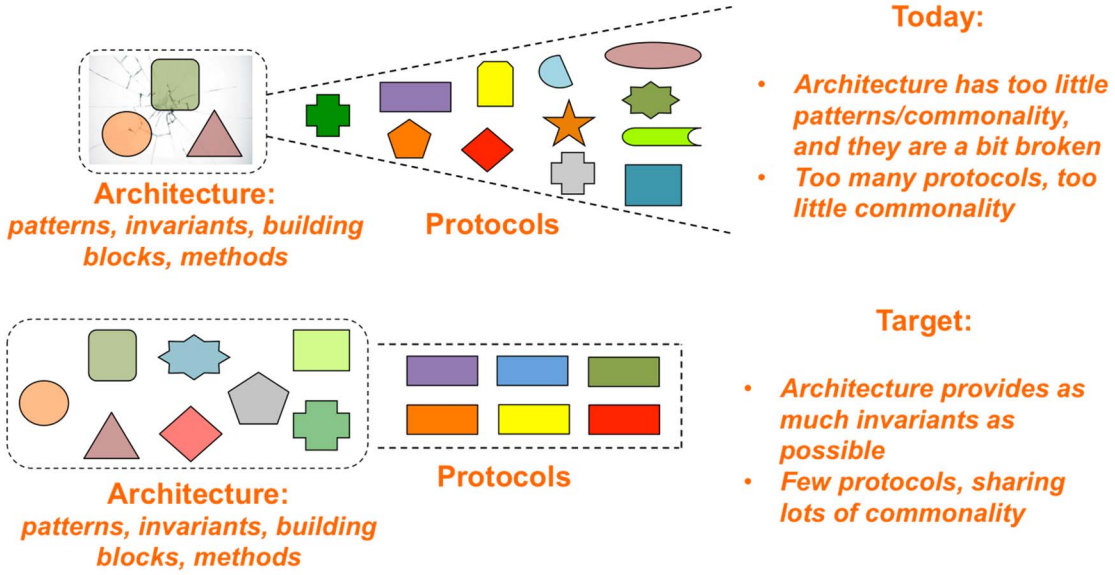


Figure 3: Abstract representation of current network architecture and protocols vs. the desired goal

Specifically, an improved network protocol architecture compared to the one prevalent today should:

- Provide a framework to facilitate re-use of mechanisms across layers, in order to be able to repeat functions without having to re-specify them.
- Avoid layer violations: layers should be seen as black boxes with a well-defined API.
- Allow for a variable number of scopes, which can be decided in real time by network designers. Avoid the need of introducing new protocols when new scopes are needed.
- Maximize the commonality across layers, in order to simplify its structure as much as possible.
- Provide a clear structure and division of functions within each layer, minimizing the number of protocols required to operate it.
- Provide a complete naming and addressing architecture, with support for application names.
- Support mobility and multi-homing due to the properties of the network structure, without requiring specialized protocols.
- Provide a consistent QoS model across layers, which can be applied from the application to the physical wire.

- Provide adequate support for congestion control, including explicit congestion detection, minimizing the length of control loops and allowing for their customization.
- Provide security and privacy by design, rather than as extension(s).
- Provide a consistent security model within and across layers, which is independent of individual protocols.
- Minimize the amount of internal state that needs to be exposed outside of a layer (such as addresses).
- Minimize the number of protocols required for network management (ideally one).
- Support a bounded and well-defined set of programmability points.

5 Structure

5.1 A network service definition

As introduced in clause 4.1, the role of computer networks is to support distributed computing; that is: to enable distributed instances of applications to communicate with each other. Hence the network allocates resources to support instances of communications between remote applications. The service provided by the network can be characterized as distributed Inter Process Communication or distributed IPC.

According to the definitions in ETSI GS NGP 007 [i.4], the term "*flow*" is used to describe an individual instance of a communication service. Hence applications request flows to the network in order to exchange information with other applications. When applications request a communication service (flow), they provide information regarding what other application(s) they would like to communicate with, and what (if any) should be the characteristics of the service.

Therefore, there is a need for names that identify instances of applications (a specific instance, any instance, a group of instances, etc.), and the need for a mechanism that enables the application to express constraints on the characteristics of the flow. Such constraints can be related to the performance of the service (goodput, latency), its reliability (packet loss rate, ordered delivery of data, etc.), security or maybe others. However, such constraints are always related to the service definition, they should never try to dictate the protocols, mechanisms or policies that the network internally has to use in order to fulfil the flow request: applications ask *what service characteristics* they want, the network decides *how to* provide the service.

Once the resources of the flow are allocated, applications can start sending and receiving data to its peers through the flow. When the application is done with the communication, it can inform the network that the resources tied to the flow can be released.

Hence, a generic network service according to the RINA mode can be defined as follows, including a very high-level abstract API definition.

Networks provide distributed Inter Process Communication (IPC) services to instances of distributed applications. Networks enable multiple instances of distributed applications to communicate with certain characteristics. Each instance of a communication service is called "flow". Applications can interact with the network via the following primitives:

- **flow_handle allocate_flow**(*source_app_name, destination app_name*, list of *service_characteristics*)
- **write_data_to_flow**(*flow_handle, data*)
- **read_data_from_flow**(*flow_handle, data*)
- **deallocate_flow**(*flow_handle*)

5.2 Networks and distributed computing

Before trying to identify generic patterns in the structure of networks, it is important to get a better understanding of their nature. Clause 5.1 argues that networks provide services to distributed applications, in order to facilitate distributed computing. But what are networks internally? Is the nature of networks very different from the one of the distributed applications they support?

Traditionally networks have been seen as something separate and special by those researching, designing and building them (except at the beginnings of network research, when the vision was that networking was all about distributed computing). But this is the point of view of the observer (humans looking at networks), not the one from the organism (networks). Taking a closer look into what networks do:

- They provide distributed IPC services to other distributed applications.
- Network functions execute in some form of Compute Entity (generic CPUs, FPGAs, ASICs, etc.) in a variety of platforms (PCs, laptops, routers, sensors, smartphones, tablets, switches, etc.).
- Instances of network functions can be implemented in software, hardware or a combination.
- Network functions/protocol machines have instances distributed through many systems, sharing state and exchanging information to carry out a collaborative task.

Just like the web, peer-to-peer distribution systems, email, etc. Hence networks are just a particular class of distributed applications, which are specialized to perform their task: providing IPC services to instances of other distributed applications (as other distributed applications are specialized to carry out their own tasks). This observation may not be ground-breaking, but it has two consequences that are important to identify invariances, common patterns and simplify the ways in which networks are modelled, designed and built:

- If networks are distributed applications, there may be common structural patterns shared by all types of distributed applications (which are also common to networks).
- If networks are distributed applications, by definition "networks" can also be the users of distributed IPC services (which hints to the recursive structure presented in clause 5.3).

5.3 A repeating structural pattern: recursive Distributed IPC Facilities (DIFs)

The network is a distributed application that provides IPC services to instances of other distributed applications, called Distributed IPC Facility (DIF). But a single instance of this DIF running in all the machines in the world and supporting all the applications in the world is not very scalable. There is a need to isolate scopes (link, network, Internet, VPN, etc.) both for security, performance and scalability reasons.

The solution is to have multiple instances of the DIF, dealing with different scopes and providing services to each other (after all, a DIF is just a distributed application). The RINA network protocol architecture features a single type of layer (the DIF) that repeats as many times as need by the network designer. A layer is a resource allocator that provides and manages the IPC service over a certain scope (link, network, Internet, VPN, etc.). A layer allocates resources (memory in buffers, scheduling capacity) to competing flows, which may have different quality requirements in terms of allowable data loss rate, latency, jitter or throughput.

An example of such a repeating structure is provided in Figure 4. The Figure depicts five systems: two hosts, two border routers and an interior router, interconnected by four point-to-point physical links. IPC over each physical link is managed by an instance of a DIF. Another DIF (in blue) provides IPC services between applications in the two border routers, hiding the interior router from the hosts. Finally, an "end-to-end" DIF (in orange) provides IPC services between applications running in hosts.

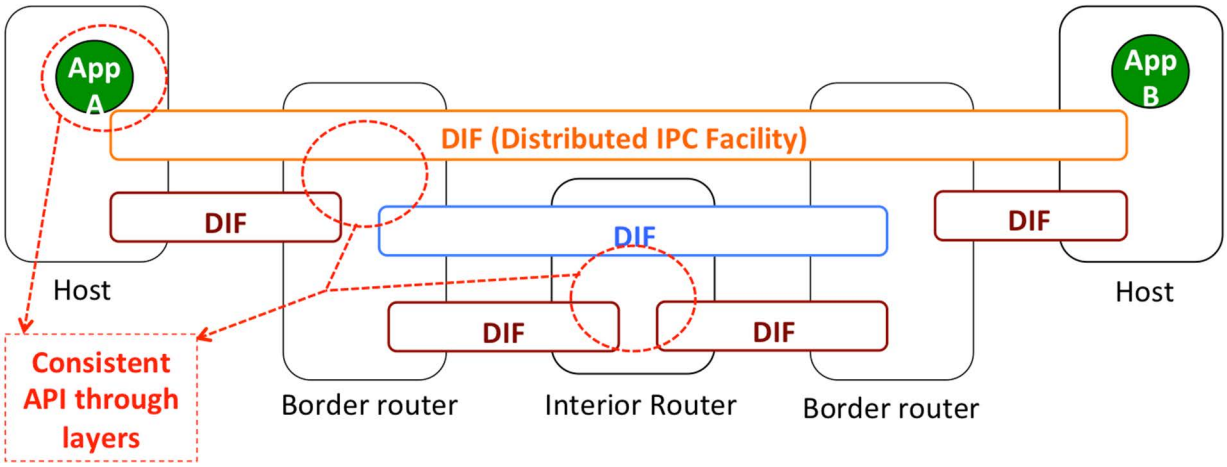


Figure 4: A repeating layer (the DIF) for different scopes is the basic structural pattern of RINA

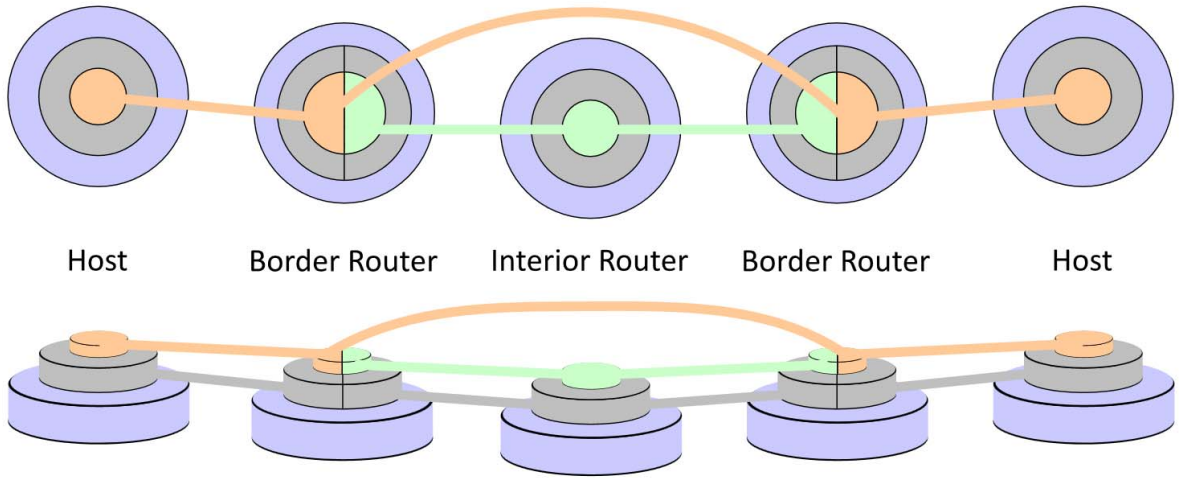


Figure 5: Illustrating DIFs to show different scopes

The scope of each layer (DIF) is configured to handle a given range of bandwidth, QoS and scale; thus dividing the problem of global network resource allocation into tractable parts. DIFs manage resources over a given range, therefore the policies of each layer will be selected to optimize that range. The number of DIFs required in any network or internetwork depends on the range of bandwidth, QoS and scale. Moreover, this number can change dynamically and be different in different parts of a network (access, aggregation, core, etc.). *This is a network design question, not a network protocol architecture question*.

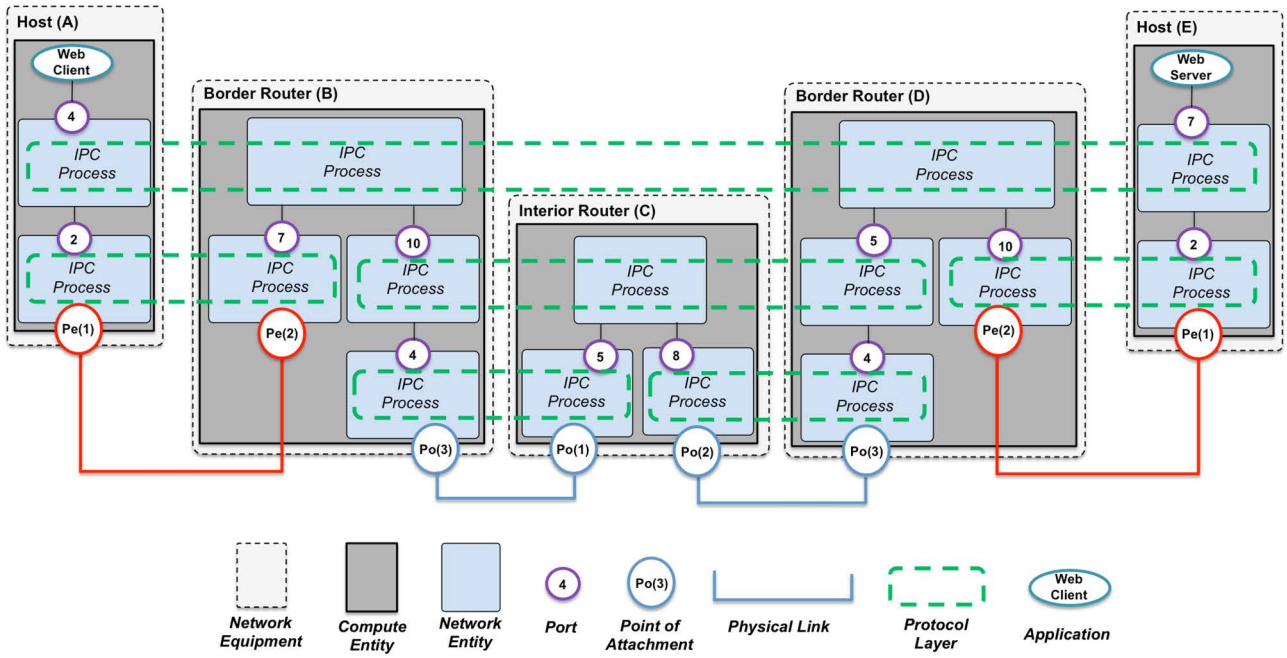


Figure 6: DIFs described using the protocol model in ETSI GS NGP 007 [i.4]

Figure 6 illustrates the example scenario featured in Figure 4, but describes it using the NGP generic protocol model introduced in ETSI GS NGP 007 [i.4]. Each DIF is composed by a number of "IPC Processes" (IPCPs), which collaborate together to provide and manage an IPC service over a certain scope. Each IPCP can be thought of as an instance of a distributed application, at the intersection of a protocol layer and a Network Equipment.

The IPCP provides multiple flows to applications on top of it, each flow being an instantiation of an IPC service identified by a port (purple circles in Figure 6). Applications using these flows may also be other higher-layer IPCPs, thus forming a recursive structure. IPCPs consume flows provided by other IPCPs below them or points of attachment to physical links of different technologies (electrical, optical, radio, etc.).

The job of an IPCP is to:

a) multiplex data from "customer" flows (the applications served by the IPCP) into the flows provided by lower DIFs; and

b) relay data from different lower DIF ports.

IPCPs use a common data transfer protocol to perform this task, which will be introduced in clause 6. This common data transfer protocol can be optimized for the requirements of each DIF via policies, maximizing the commonality across DIFs but allowing flexibility where it really makes a difference.

In order to carry out its data transfer tasks, the IPCPs requires a number of layer management functions to manage allocation of flows, resources, discovery of applications, authentication, access control, peer IPCP discovery, etc. Since IPCPs are the same for each DIF, a common layer management framework that can be adapted to the operational environment of each DIF is required. Such a framework is introduced in clause 6.

5.4 Examples of DIF configurations

5.4.0 Introduction

This clause introduces a few examples to facilitate understanding how DIFs may be used in real world networks.

5.4.1 Virtual Private LAN Service (VPLS)

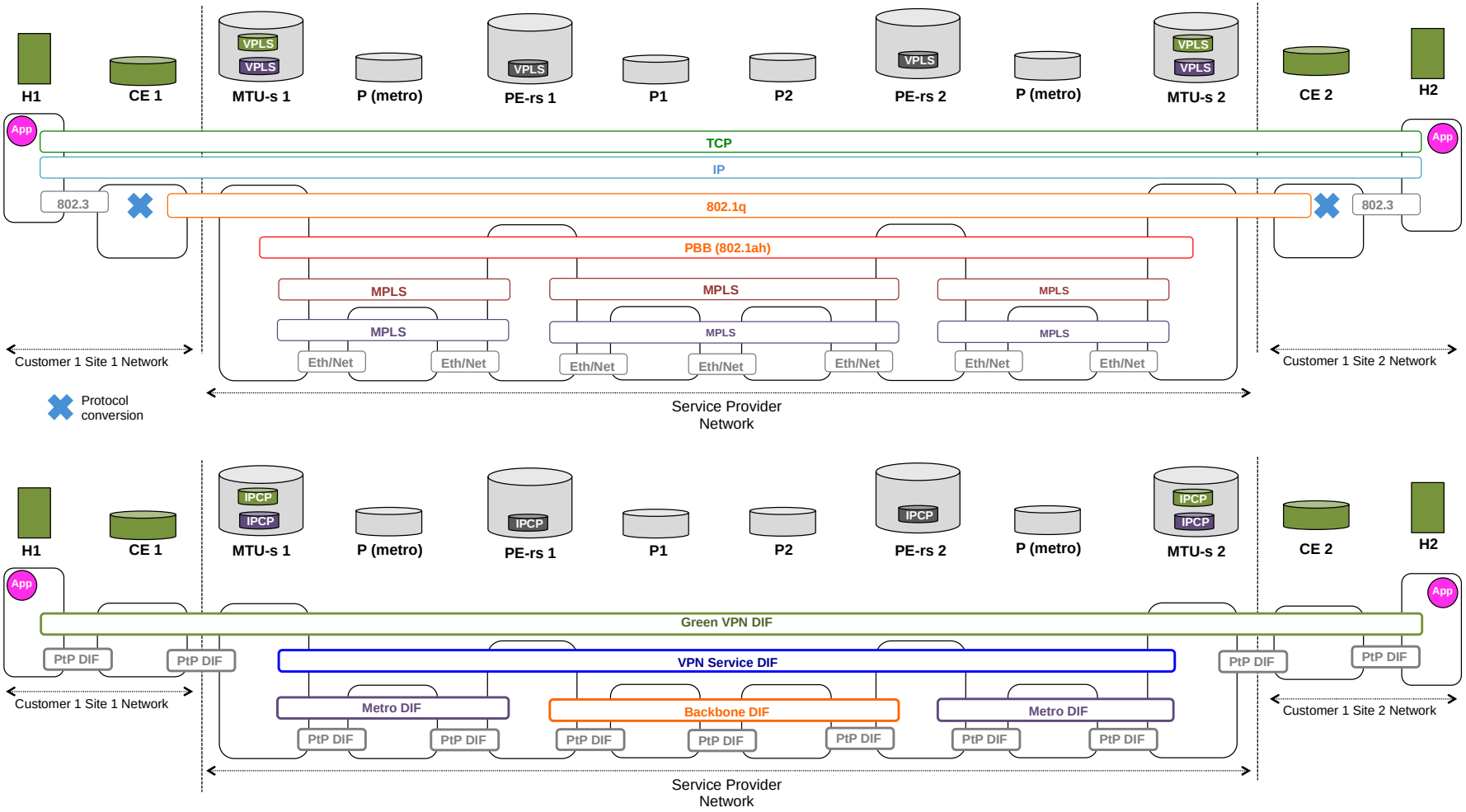


Figure 7: Example of provider network offering enterprise VPNs: current protocol architecture (top) and RINA (bottom)

The upper part of Figure 7 depicts an operator providing a Virtual Private LAN Service (VPLS) to interconnect two customer sites at layer 2. The service provider runs the layer 2 VPN service over two MPLS-based metropolitan area networks and an MPLS-based core. Provider Backbone Bridging (PBB) is used to isolate the routing of the VPN service over the provider network from the customer's address space. An equivalent RINA configuration is shown in the bottom part of Figure 7. Point-to-Point (PtP) DIFs provide IPC services over different types of point to point links. Metro DIFs provide IPC over metro aggregation networks, multiplexing the traffic of the different types of services the operator provides over the metropolitan segment. A backbone DIF provides IPC over the core segment of the network, interconnecting PoPs in different cities. A VPN service DIF is floating on top of the Metro and Core DIFs, allocating resources to different VPN DIFs over the entire scope of the provider's network.

5.4.2 LTE Evolved Packet System (EPS) User Plane

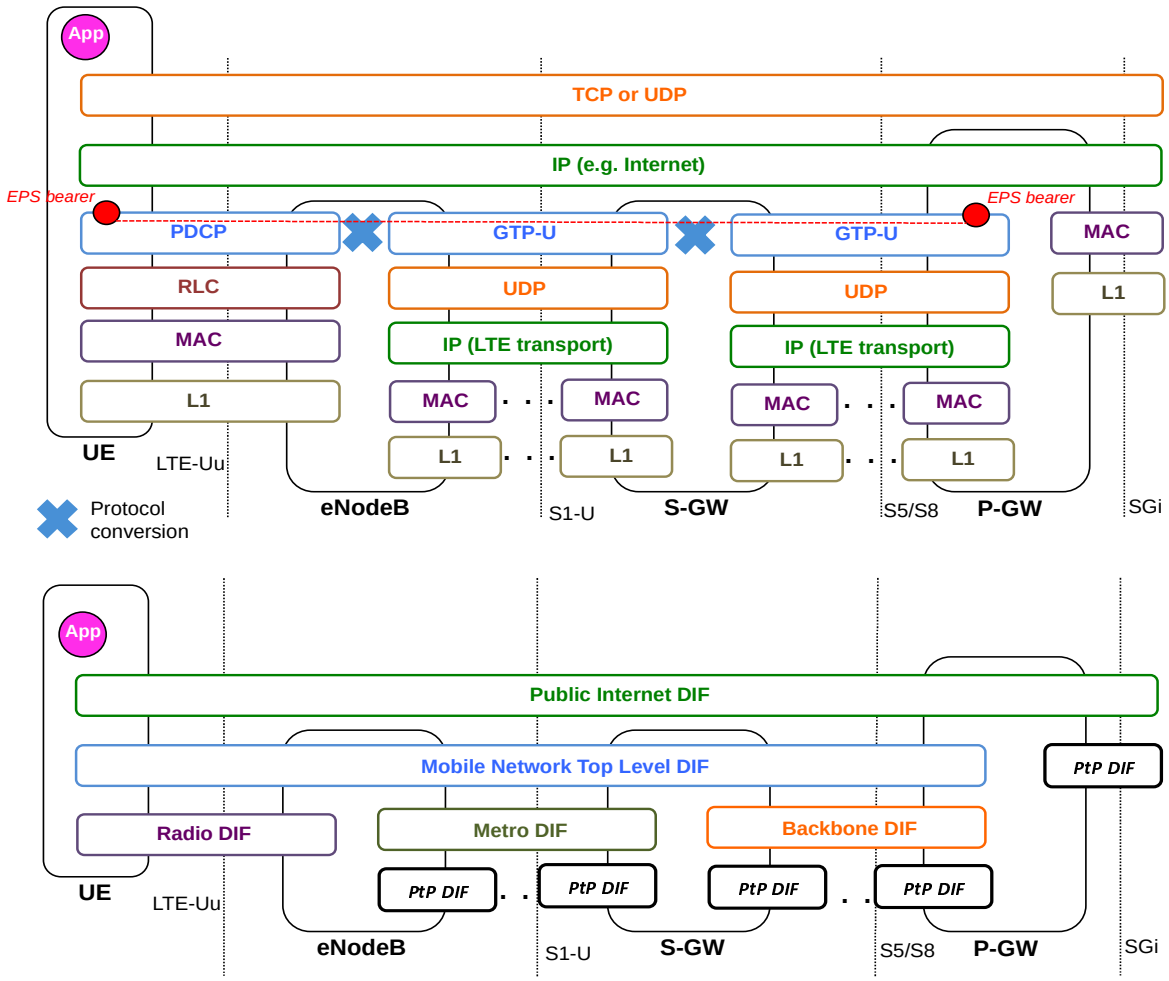


Figure 8: Cellular access network offering Internet services: LTE protocol stack (up) and RINA (down)

Figure 8 shows the user-plane protocol stack of an LTE network, with the i) radio protocols (RLC, MAC, L1), the tunnelling protocols to create bearers (flows) over the mobile network (PDCP and GTP) and the overlay TCP, UDP and IP layers of the public Internet. GTP tunnels are aggregated over the provider's IP-based aggregation and backbone layers, which multiplex the LTE bearers with other types of traffic belonging to other services offered by the operator. An equivalent structure can be re-created using RINA's generic layering, as shown in the bottom part of Figure 7. A radio multi-access DIF manages the radio resource allocation and IPC over the wireless media. A Mobile network top-level DIF provides flows over the scope of the mobile network - the policies of this DIF are optimized to deal with different rates of User Equipment (UE) mobility. Metro and Backbone DIFs multiplex and transport the traffic of the Mobile Network top-level DIF and other DIFs providing other service over the metro and core network segments. Finally, the public Internet DIF allows applications in the UE to connect to other applications available on the public Internet (other DIFs may also apply at this to level, for example a private Enterprise DIF for authorized users of an external Enterprise network).

Note that the LTE architecture is logically comparable to the 5G architecture for the sake of the present document. For a summary of differences between LTE EPS and 5G SBA (Service Based Architecture), please see ETSI GS NGP 001 (V1.3.1) [i.25] Annex B (5G Mobile network model).

5.4.3 Multi-tenancy Data Centre

Finally, Figure 9 shows an example of a Data Centre network supporting multi-tenancy. In this case the VXLAN protocol is used to create several layer 2 virtual networks over a shared IPv6- based Data Centre Fabric. The equivalent RINA configuration is depicted at the bottom of the same Figure, featuring two types of DIFs: a Data Centre Fabric DIF that allocates the resources in the leaf-spine fabric to flows used by competing tenant DIFs, which provide isolated and customized networking domains to different tenants.

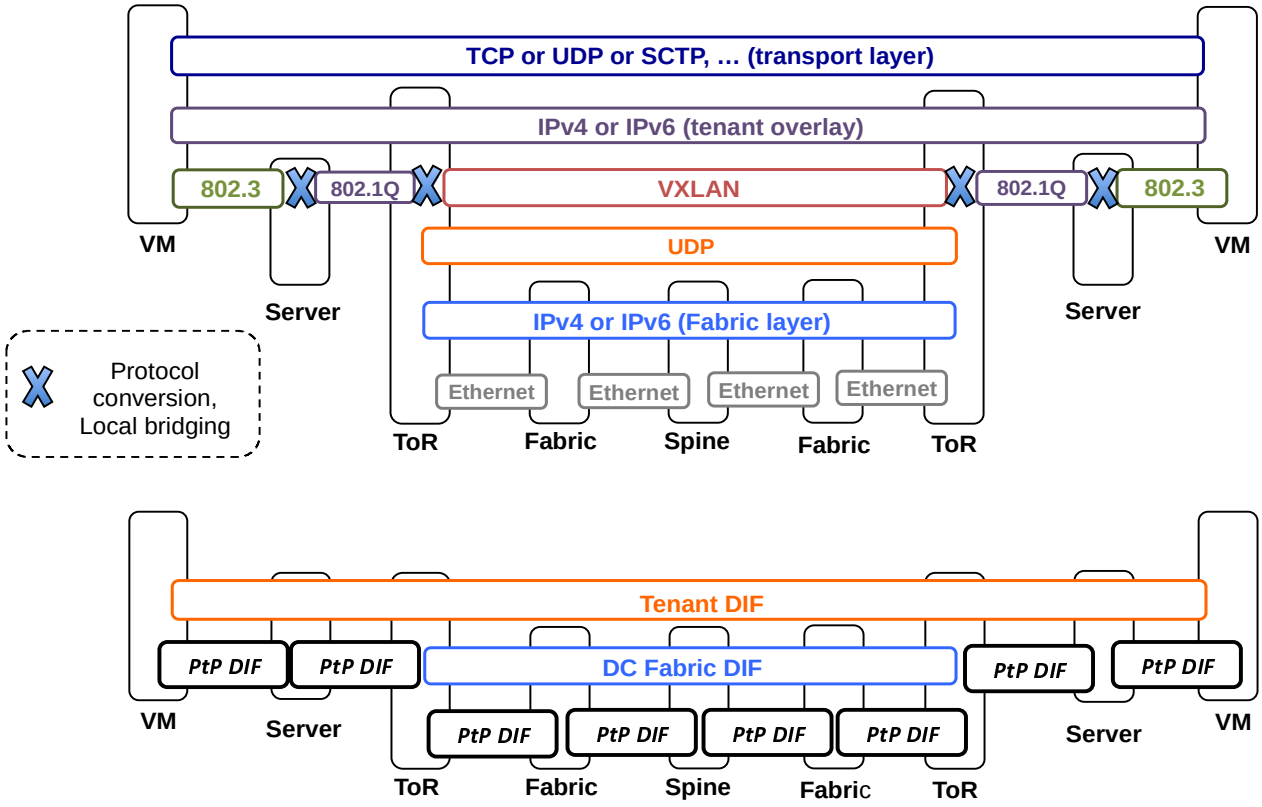


Figure 9: Multi-tenant data-centre: current protocol architecture (top), RINA (bottom)

5.5 Summary of RINA structural properties

This clause summarizes how RINA addresses the issues identified in clause 4.3.1.

- **Different layers, same functions but different policies.** In RINA all layers perform the functions required to provide distributed IPC over a certain scope and range of QoS. *Layers are units of resource allocation, not units of modularity*. Each layer manages a set of resources (buffers, scheduling capacity, bandwidth) using a common invariant structure, regardless if the layer manages a point to point medium, performs aggregation over a metropolitan network or provides a VPN service to a certain application. Different policies allow the common layer infrastructure to optimally operate over different scopes and ranges of QoS.

- **Strict layering: layers are black boxes.** Each RINA layer is independent of each other, a DIF can only use the services of lower layer DIFs by invoking the operations in the lower DIF API - hence no layer violations are allowed. Within a layer all the functions (data transfer, data transfer control and layer management) have different degrees of dependence or coupling to each other. The design of the different protocol frameworks within a RINA DIF makes sure that all the DIF functions have enough information available for its correct operation.

- **Variable number of layers, decided at design time or even operational time.** RINA - the architecture -does not bound the number of layers that can be found in a network. *Each network should use the number of layers that optimally solve its resource allocation and security requirements: no more and no less*. Therefore, the network architect has considerable freedom and flexibility in designing the network, while still bounding the overall complexity due to the use of the immutable common infrastructure at each layer - regardless of its ranking/positioning. This environment contrasts with the "fixed and static" number of layers in current network architectures, where concepts such as network Virtualization, tunnels and overlays have to be introduced to work around this limitation at the cost of increasing the overall system complexity.

- **Consistent API across layers.** Since all layers provide the same IPC service, the API offered by all layers is the same. This API allows instances of APs to allocate and use communication flows to other applications, defining a set of requirements for the quality of the communication flow. Since IPCPs are just a specialized form of application processes, DIFs use underlying DIFs as any other distributed application would do.

6 Generic protocol frameworks

6.1 Internal structure of an IPC Process

One of the key RINA design principles has been to maximize invariance and minimize discontinuities. In other words, extract as much commonality as possible without creating special cases. Applying the concept from operating systems of separating mechanism and policy [i.7], first to the data transfer protocols and then to the layer management machinery (usually referred to the control plane), it turns out that only two protocols are required within a DIF [i.8]:

- A single data transport protocol framework that supports multiple policies and that allows for different concrete syntaxes (length of fields in the protocol PDUs). This protocol framework is called EFCP - the *Error and Flow Control Protocol.*

- A common application protocol that operates on remote objects used by all the layer management functions (enrolment, namespace management, flow allocation, resource allocation, security coordination). This protocol is called CDAP - the Common Distributed Application Protocol. CDAP is accompanied by a common layer management machinery that simplifies the specification and development of specific layer management functions.

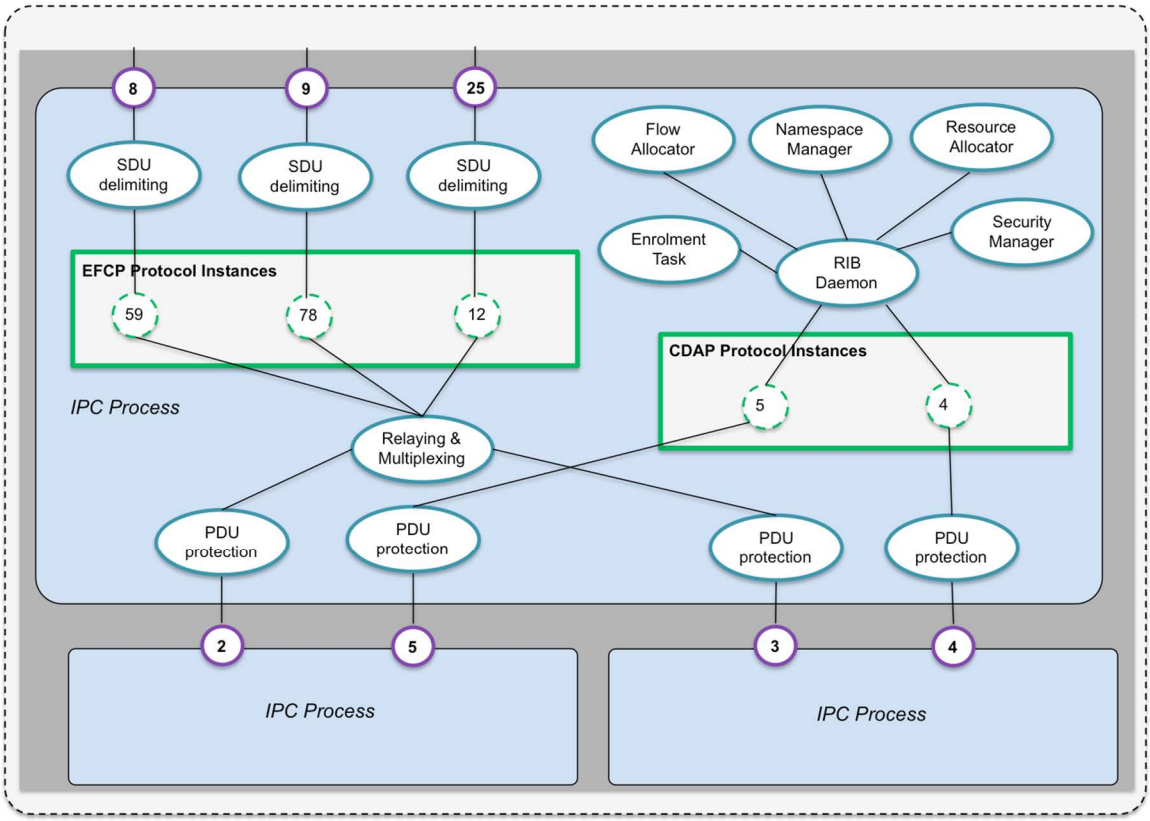


Figure 10: Internals of an IPC Process, described using the protocol model in ETSI GS NGP 007 [i.4]

Separation of mechanism and policy also provided new insights about the structure of those functions within the IPC Process. The primary components of an IPCP are shown in Figure 10 and can be divided into three categories:

a) Data Transfer, decouple through a state vector from.

b) Data Transfer Control, decoupled through a Resource Information Base from.

c) Layer Management. These three loci of processing are characterized by decreasing cycle time and increasing computational complexity (simpler functions execute more often than complex ones):

- *SDU Delimiting*. The integrity of the SDU written to the flow is preserved by the DIF via a delimiting function. Delimiting also adapts the SDU to the maximum PDU size. To do so, delimiting comprises the mechanisms of fragmentation, reassembly, concatenation and separation.

- *EFCP, the Error and Flow Control Protocol*. This protocol is based on Richard Watson's work [i.9] and separates mechanism and policy. There is one instance of the protocol state for each flow originating or terminating at this IPC Process. The protocol naturally cleaves into Data Transfer (sequencing, lost and duplicate detection, identification of parallel connections), which updates a state vector; and Data Transfer Control, consisting of retransmission control (ack) and flow control.

- *RMT, the Relaying and Multiplexing Task*. It makes forwarding decision on incoming PDUs and multiplexes multiple flows of outgoing PDUs onto one or more (N-1) flows. There is one RMT per IPC Process.

- *SDU Protection*. It does integrity/error detection, e.g. CRC, encryption, compression, etc. Potentially there can be a different SDU Protection policy for each (N-1) flow.

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

6.2 Data transfer: functions, protocols and procedures

6.2.1 Introduction

EFCP, the Error and Flow Control Protocol, is the single data transfer protocol of a DIF. In order to allow for its adaptation to different operating environments, EFCP supports multiple policies and multiple specific syntaxes. To do so, the EFCP specification defines the hooks where the different policies can be plugged in - also describing the behaviour of default policies - as well as the abstract EFCP syntax (PDU types and its fields, without describing its encoding). EFCP leverages the results published by Richard Watson (and later implemented in the delta-t protocol). Watson proved that bounding three timers is a necessary and sufficient condition for reliable transport connection management; in other words: SYNs and FINs are unnecessary. This not only simplifies the protocol implementation, but it also makes it more reliable against harsh network environments [i.10] or transport-level attacks [i.11].

EFCP has two parts: DTP (Data Transfer Protocol), which deals with the mechanisms tightly coupled to data transfer PDUs (such as addressing or sequencing) and DTCP (Data Transfer Control Protocol), which deals with the loosely coupled mechanisms such as flow control or retransmission control. DTP and DTCP are fairly independent and operate with their own PDUs, being just loosely coupled via a state vector.

6.2.2 DTP PDU abstract syntax

Figure 11 illustrates the abstract syntax of EFCP DTP PDUs. Note that the length of a*ddress, qos-id, CEPID, length* and *sequence number fields* depends on the DIF environment. For example, no source or destination address fields are required for DIFs in point-to-point links.

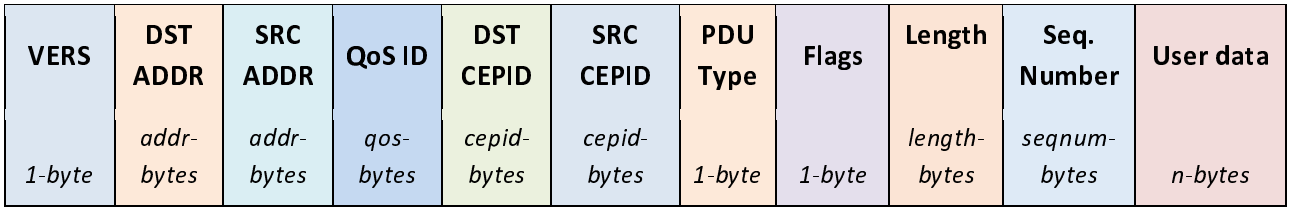


Figure 11: Abstract syntax of DTP PDUs

- **Version:** EFCP version.
- **Src/destination address:** Addresses of the IPC Processes that host the endpoints of this EFCP connection.
- **QoS-id:** Id of the QoS cube where this EFCP connection belongs (see clause 8.1).
- **Src/destination cepids:** The identifiers of the EFCP instances that are the endpoints of this EFCP connection.
- **PDU type:** Code indicating the type of PDU (in this case it is a DTP PDU).
- **Flags:** Indicate conditions that can affect the processing of the PDU and can change from one PDU to another.
- **Length:** The total length of the PDU in bytes.
- **Sequence number:** Sequence number of the PDU.
- **User data:** Contains one or more SDU fragments and/or one or more complete SDUs.

6.2.3 DTCP PDU Formats

Depending on the policies associated to a particular EFCP connection, the DTCP instance may be configured to perform flow and/or retransmission control functions. While the EFCP specification defines 10 operation codes defined for DTCP, in reality there are only three PDU types:

a) Ack/Nack/Flow;

b) Selective Ack/Nack/Flow; and

c) Control Ack.

Each of these control PDUs carries addresses, a connection-id, a sequence number, and retransmission control and/or flow control information, etc. The opcodes indicate which fields in the PDU are valid. Required fields for these PDUs can be extended by defining policies.

6.2.4 Overview of data-transfer procedures

A high-level overview of the data-transfer procedures is provided by Figure 12. Note that this is an ***example scenario*** showing logical functions, in any case is suggesting a particular implementation strategy.

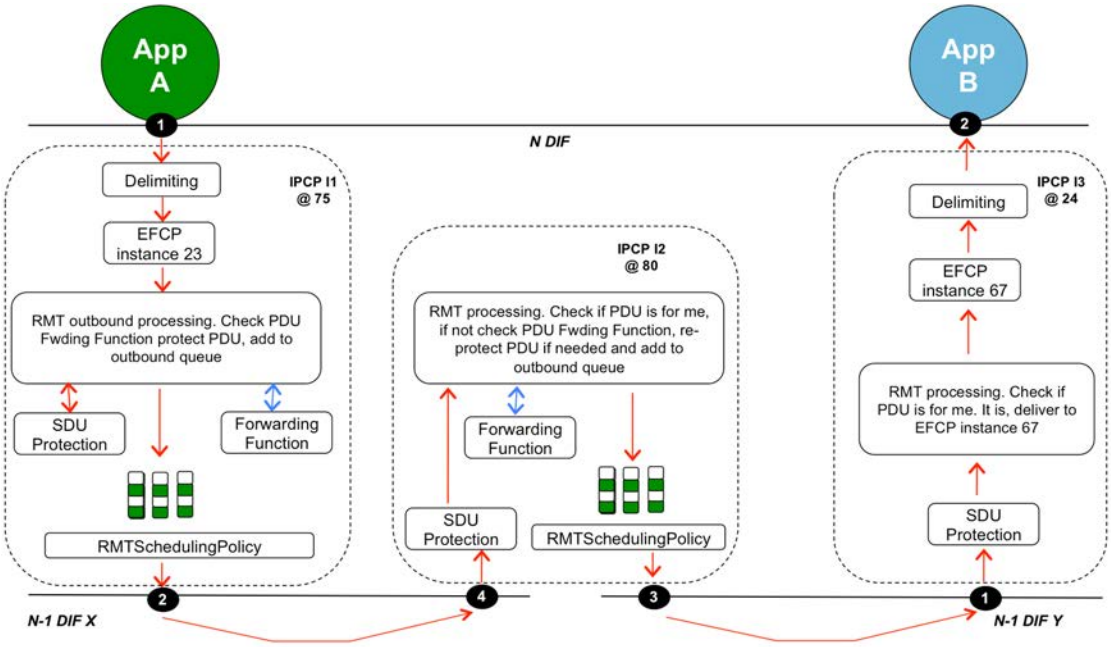


Figure 12: Example of data transfer procedures

In this example scenario, the *DIF N* provides a flow identified by port-id 1 between applications *A* and *B*. *Application A* writes an SDU to the port, invoking the DIF API. The SDU is then processed by the delimiting function of *IPCP I1*, which will create one or more EFCP user-data fields from the SDU, according to the delimiting policy. EFCP user-data fields are delivered to the *EFCP instance 23* - which is currently bound to *port-id 1* - which creates one or more EFCP data transfer PDUs and hands them over to the Relaying and Multiplexing Task (RMT).

The RMT checks the forwarding function (another policy), which returns the port-ids of one or more N-1 flows through which the PDU needs to be forwarded to reach the next hop (in this case the *IPCP with address 80*). In general, there will be one or more queues in front of each N-1 port, and a scheduling policy will sort out outgoing PDUs for transmission according to different criteria. Once the N-1 port through which the PDU will be forwarded is known, the associated SDU protection policy can be applied to the PDU (or it can be applied when EFCP creates the PDU if there is a common SDU protection policy for all N-1 ports).

Eventually *IPCP I2* reads the PDU from *N-1 port 4*. It removes SDU protection required to process the PDU header, and the RMT decides if it is the final destination of the PDU (depending on the DIF environment; for example, checking the destination address field in this example). In this case the IPCP is not the final destination, so the RMT checks the forwarding function, which returns one or more N-1 ports through which the PDU will be forwarded. The RMT re-applies protection if needed (SDU protection policy may be different), and handles the PDU for transmission to the scheduling policy, which eventually writes the PDU to the N-1 port.

Finally, the PDU reaches *IPCP I3* through *N-1 port 3*. SDU protection is removed, the RMT checks if it is the final destination of the PDU and, since in this case it is, it delivers the RMT to the destination EFCP instance (EFCP instance 87 in the example) for further processing. The EFCP instance updates its internal state and may generate zero or more control PDUs. EFCP recovers the PDU's user data field, and works with the delimiting function according to the configured policies in order to recover full SDUs. Finally, SDUs are read from *port 2* by *application B.*

6.3 Layer management: protocol, functions and procedures

6.3.1 Introduction

The different layer management functions of an IPC Process leverage a common machinery to exchange information with their peers. All the IPC Process externally visible state is modelled as objects that follow a logical schema called RIB, Resource Information Base. The RIB specification defines the object naming, relationships between objects (inheritance, containment, etc.), the object attributes and the CDAP operations that can be applied on them. Access to the RIB is mediated by the RIB Daemon. The RIB Daemon of an IPCP exchanges CDAP PDUs with RIB Daemons of neighbour IPCPs. These PDUs communicate remote operations on objects. When a layer management task wants to communicate an action to a peer (e.g. a routing update), it requests the RIB Daemon to perform an action on one or more objects of one or more neighbour IPCPs. The RIB Daemon generates the required CDAP PDUs and sends them over the required N-1 flows to communicate the action to its neighbours. When the RIB Daemon receives a CDAP PDU, it decodes it, analyses what objects are involved and notifies the relevant layer management functions (who have previously subscribed to objects of their interest).

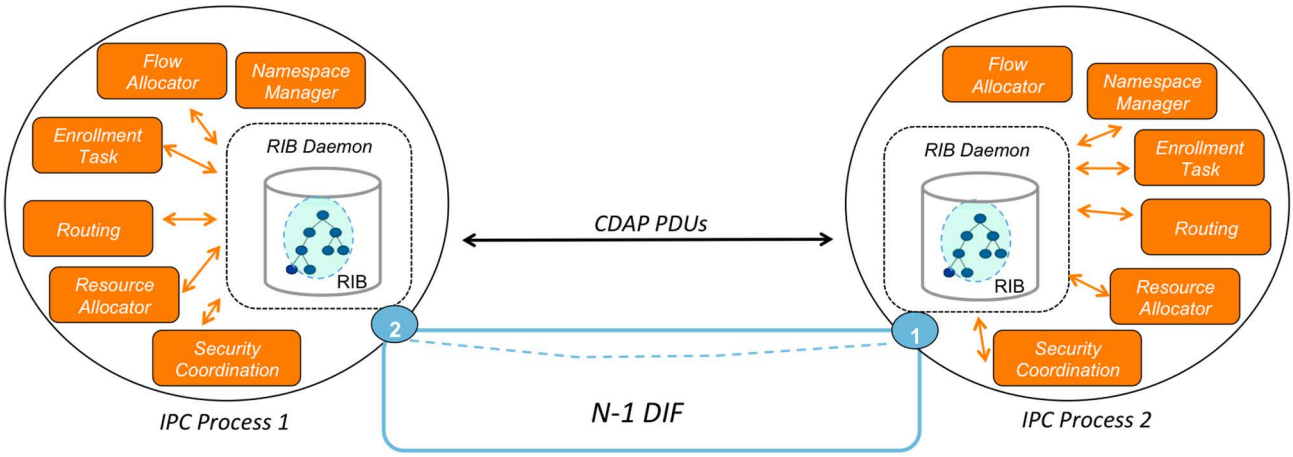


Figure 13: Common layer management machinery: RIB, RIB Daemon and CDAP

The whole process is illustrated in Figure 13. This design allows the layer management tasks to just focus in the functions they provide and delegate the rutinary tasks of generating and parsing protocol PDUs to the RIB Daemon (in fact, layer management tasks are not even aware of CDAP). If required, new layer management functions could be added without the need of defining new protocols. Moreover, the RIB Daemon can coordinate and optimize the generation of protocol PDUs from different layer management tasks; thus minimizing the layer management traffic between peer IPCPs. The CDAP specification defines an abstract syntax that describes the different types of CDAP PDUs and their fields. Multiple concrete encodings can be supported (it is just a DIF policy), such as the various ASN.1 encodings, Google™ Protocol Buffers, etc.

Before being able to exchange any information, two peer IPCPs establish an association between them. This association is called ***application connection*** in RINA terms. During the application connection establishment phase, the IPCPs exchange naming information, optionally authenticate each other, and agree in the abstract and concrete syntaxes of CDAP/RIB to be used in the connection, as well as in the version of the RIB. This version information is important, as RIB model upgrades may not be uniformly applied to the entire network at once. Therefore, it should be possible to allow multiple versions of the RIB to be used, to allow for incremental upgrades.

6.3.2 Layer management functions: enrollment

Enrollment is the procedure by which an IPCP joins an existing DIF and is initialized with enough information to become a fully operational DIF member. Enrollment occurs after an IPC-Process establishes an application connection with another IPCP, which is a member of a DIF. Once the application connection is established this enrolment procedure may proceed. The specific enrollment procedure is a policy of each DIF, but in general it involves operations similar to the ones of the example policy described in the following paragraph.

The Member IPCP reads the New Member IPCP's address. If null or expired, it assigns a new address; otherwise, assumes the New Member was very recently a member. The New Member then reads the information it does not have taking into account how "new" it is. These parameters characterize the operation of this DIF and might include parameters such as max PDU size, various time-out ranges, ranges of policies, etc. Once complete, the New Member is now a member and this triggers a normal RIB update (to get the latest up to date information on routing, directory, resource allocation, etc.).

6.3.3 Layer management functions: namespace management

Managing a name space in a distributed environment requires coordination to ensure that the names remain unambiguous in the right scope and can be resolved efficiently. The Name Space Manager (NSM) embedded in the DIF is responsible for mapping application names to IPC Process addresses - the latter being the name space managed by the DIF NSM. Specific ways of achieving this mapping are policy-based and will vary from DIF to DIF. For small, distributed environments, this management may be fairly decentralized and name resolution may be achieved by exhaustive search. Once found the location of the information that resolved the name may be cached locally in order to shorten future searches. It is easy to see how as the distributed environment grows that these caches would be further organized often using hints in the name itself, such as hierarchical assignment, to shorten search times. For larger environments, distributed databases may be organized with full or partial replication and naming conventions, i.e. topological structure, and search rules to shorten the search, requiring more management of the name space.

The two main functions of the DIF NSM are to assign valid addresses to IPC Processes for its operation within the DIF and to resolve in which IPC Process a specific application is registered. In other words, the NSM maintains a mapping between external application names and IPC Process addresses where there is the potential for a binding within the same processing system. Therefore enrollment, application registration and flow allocation require the services of the NSM.

6.3.4 Layer management functions: flow allocation

The Flow Allocator is responsible for creating and managing an instance of IPC, i.e. a flow. The IPC-API communicates requests from the application to the DIF. An Allocate-Request causes an instance of the Flow Allocator to be created. The Flow Allocator-Instance (FAI) determines what policies will be utilized to provide the characteristics requested in the Allocate. It is important that how these characteristics are communicated by the application is decoupled from the selection of policies. This gives the DIF important flexibility in using different policies, but also allows new policies to be incorporated. The FAI creates the EFCP instance for the requested flow before sending the CDAP Create Flow Request to find the destination application and determine whether the requestor has access to it.

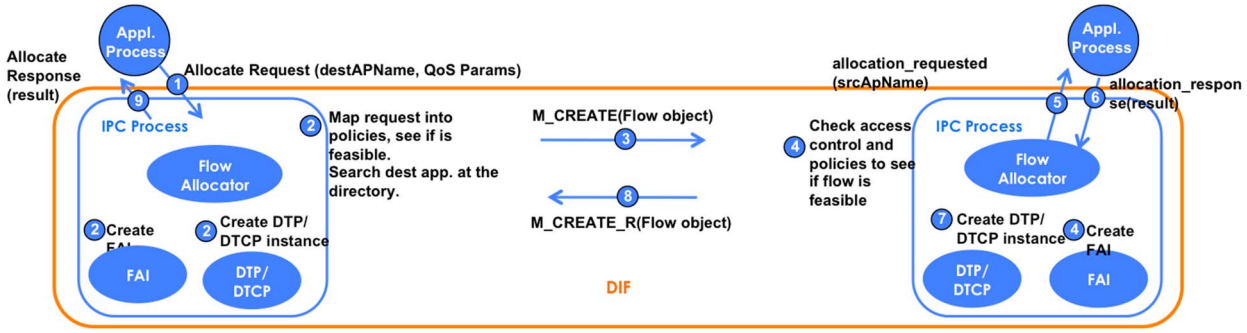


Figure 14: Illustration of the flow allocation procedure

A create request is sent with the source and destination application names, quality of service information, and policy choices, as well as the necessary access control information. Using the NSM component, the FAI searches the IPCP in the DIF that resides on the processing system that has access to the requested application. This exchange accomplishes three functions:

- Following the search rules using the Name Space Management function to find the address of an IPC-Process with access to the destination application.
- Determining whether the requesting application process has access to the requested application process and whether or not the destination IPC-Process can support the requested communication.

- Instantiating the requested application process, if necessary, and allocating a FAI and port-id in the destination IPCP.

The create response will return an indication of success or failure. If successful, destination address and connection-id information will also be returned along with suggested policy choices. This gives the IPC-Processes sufficient information to then bind the port-ids to an EFCP-instance, i.e. a connection, so that data transfer may proceed.

6.3.5 Layer management functions: resource allocation

The Resource Allocator (RA) is the core intelligence of the IPC Process. It monitors the operation of the IPC Process and makes adjustments to its operation to keep it within the specified operational range. The functions of the RA are further explained in clause 8.2.

6.3.6 Layer management functions: routing

A major input to the Resource Allocator is Routing. Routing performs the analysis of the information maintained by the RIB to provide connectivity input to the creation of a forwarding function. Clause 8 discusses this topic in more detail. **The choice of routing algorithms in a particular DIF is a matter of policy.**

6.3.7 Layer management functions: security coordination

Security coordination is the IPC Process component responsible for implementing a consistent security profile for the IPC Process, coordinating all the security-related functions (authentication, access control, confidentiality, integrity) and also executing some of them (auditing, credential management). The sophistication of this layer management function is a matter of policy.

6.4 Summary of RINA protocol framework design principles

This clause summarizes how RINA addresses the issues identified in clause 4.3.2.

- **Clear division of the functions of each layer.** Each DIF performs functions to provide IPC over a certain scope and range of QoS (data transfer and data transfer control functions), as well as the functions to autonomically manage this IPC service within the DIF (layer management functions). Data transfer functions are loosely coupled to data transfer control functions via a state vector, and both data transfer and data transfer control functions are loosely coupled to layer management functions via the RINA Information Base (RIB). There is no need for concepts such as "separate data and control planes" with different protocols, all the functions of a DIF are cleanly split between the aforementioned three categories. This approach is also usually followed by IEEE protocols (such as IEEE 802.11™ [i.56]), RINA just generalizes it to all layers.

- **Two protocol frameworks per DIF.** Within a DIF data transfer and data transfer control functions are realized via the Error and Flow Control Protocol (EFCP). EFCP defines an abstract syntax (definition of fields in data transfer and control PDUs) and common programmable functionality to realize the specific data transfer protocol of each DIF. To do so, a concrete syntax (encoding for the header in data transfer PDU) and a set of policies for flow control, retransmission control, congestion management, scheduling, forwarding and SDU Protection has to be defined. Layer management functions (enrolment, flow allocation, resource allocation, namespace management, routing, authentication, access control, security management) leverage the common layer management infrastructure, which provides:

  i) an application protocol to remotely operate on objects (CDAP);

  ii) a schema to model the state of an IPC Process; and

  iii) a set of policies to distribute the IPCP state information as required by each layer management function (on demand, hierarchically, on certain events, etc.).

  Layer management functions just need to define its object information model and specify what happens when CDAP operations are invoked over the defined objects.

- **Common layer functions are invariant, only policies can change**. Another benefit of separating the immutable parts (the protocol frameworks identified in the former bullet point) from the variable part at each DIF is that policies are not designed in isolation. Each policy has a clear "plugging point" in the infrastructure of each DIF, with a well-defined function and API. Therefore, the number of interactions between individual policies and the common, immutable DIF infrastructure is bounded by design, limiting the complexity of a DIF. Moreover, since each DIF has the same immutable infrastructure, the same policy can be utilized in any DIF, thus maximizing re-usability and applicability of policies. This situation facilitates reasoning about the behaviour of the network and troubleshooting it.

7 Naming and addressing

7.1 Names in RINA and their properties

ETSI GS NGP 007 [i.4] provided an initial description of the names that identify the different components of the generic protocol model. This clause applies such scheme to the RINA architecture described in clauses 5 and 6.

Figure 15 illustrates the names required in RINA, from the perspective of the N-DIF. Since the protocol architecture is recursive, the same naming scheme is applied to each DIF. IPCPs provide flows to applications at the DIF above, which in turn may be other IPCPs. These applications serviced by the IPCP are identified with an *application name*, which has the following properties:

- ***Application name:*** Assigned to applications, it is location independent (does not change when the application attaches to multiple networks or DIFs). It uniquely identifies the application within an application namespace. In general it names a set, which can have a single member. Therefore, in general names can identify a collection of instances of application processes, which may be "all instances of a distributed application", a "subset of the instances of a distributed application", "any instance of a distributed application" a "specific instance of a distributed application", etc.

Since IPCPs are also applications, they are identified by an application name. However, this application name is usually assigned from a namespace that has a larger scope than the DIF where the IPCP belongs, and is not designed to facilitate forwarding and routing of packets between IPCPs within that DIF. Therefore, it is useful to assign one or more temporary synonyms to the IPCP, the *address(es)*:

- **Address:** Location-dependent synonym of an application name, assigned to an IPCP. An address reflects the location of the IPCP within its DIF (location defined as an abstraction of the DIF connectivity graph). An address is temporary, its scope is the DIF and should be changed when it no longer properly reflects the location of the IPCP in the DIF (i.e. is no longer aggregatable).

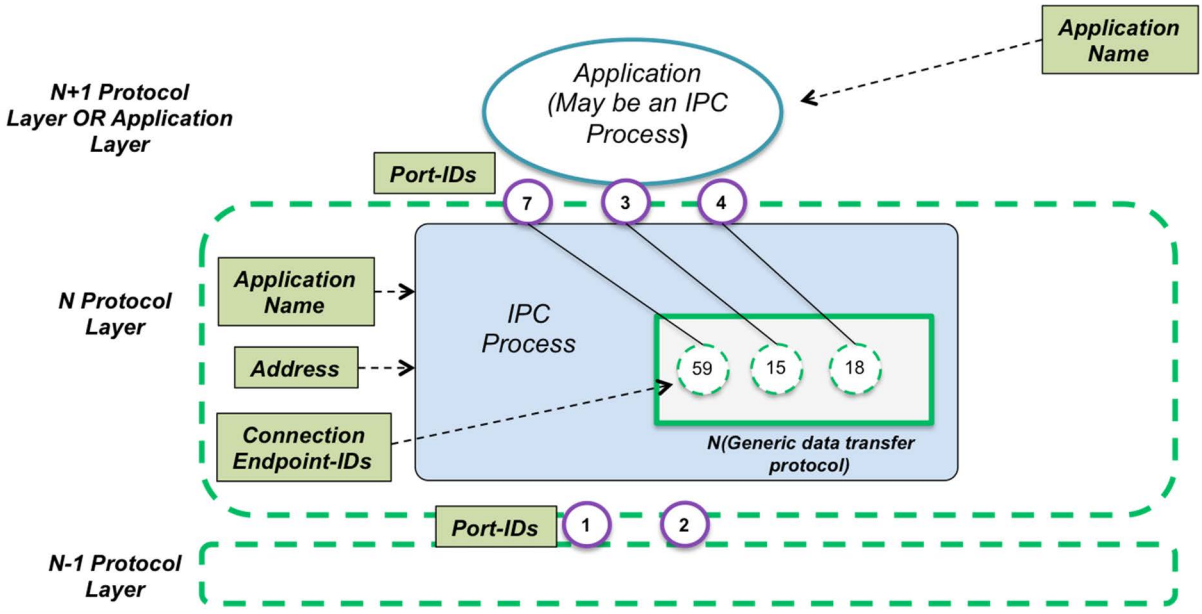


Figure 15: Names in the RINA architecture, from the perspective of the N-DIF

Flows (communication services provided by IPCPs to applications) provide an IPC service between two ports. These ports are used by the application to send and receive data, allowing the application to treat the serving DIF as a black box. As stated in ETSI GS NGP 007 [i.4], each port in a Network Equipment is identified by a port-ID:

- **Port-IDs:** Identify the end of a flow within a network equipment (the scope of this name is local to the network equipment).

IPCPs support the operation of a flow via a data transfer connection. All the IPCPs involved in providing a flow (can be two or more) create an instance of a data transfer protocol to support the exchange of data written and read to such flow. Instances of the data transfer protocol are identified by connection-endpoint-IDs (or CEPIDs). CEPIDs are the identifiers carried in data packets, used to demultiplex the packets belonging to a specific connection once they reach the destination IPCP. IPCPs create a local binding between the port-ID and its supporting CEPID. This decoupling allows greater security and flexibility at the DIF [i.12]:

- **Connection-endpoint-IDs:** Identify the data transfer protocol instances that are the endpoints of a connection. Its scope is local to the IPCP, which also stores a local binding to a port-ID.

Figure 16 shows how these names are used to operate a DIF and what relationships exist between them, from the perspective of a DIF (since the architecture recurses, the same type of relationships and concepts are valid for every single DIF). Applications that wish to be reachable via a certain DIF register to that DIF via its application name. The IPCP at that system creates a mapping between the application name and the IPCP address. The collection of such mappings for a DIF is called ***directory.*** Distributed DIF directories can be designed and implemented in multiple ways (fully replicated, hierarchical, via distributed hash tables, partially replicated, centralized, search-based, etc.), and the specific choice for each DIF will depend on the scope of the DIF and its operational requirements.

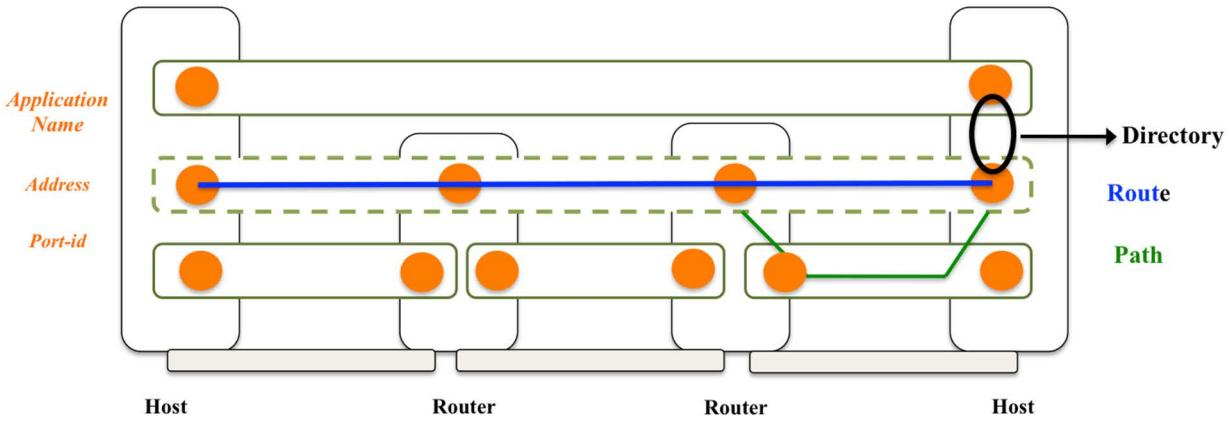


Figure 16: Directories, routes and paths from the perspective of an N-DIF (dotted rectangle)

The use of directories allows DIFs to hide the addresses of IPCPs that are members of the DIF from the applications using it. Applications just request flows to a destination application name (which may resolve to a single instance or multiple instances of an application process), and it is the DIF the one that internally uses the directory to perform the mapping of the destination application name to the addresses(s) of the IPCP(s) via which the target application instance(s) are reachable.

Routes are sequences of IPCP addresses. Routing algorithms compute a set of next-hop addresses for each destination address (it is important to note that the routing algorithm may return more than one next-hop for the same destination address, with equal or different costs). This procedure generates the next-hop address table. Each IPCP then locally maps the next-hop address to the port-ID of an N-1 flow through which will forward the packets that travel towards that destination address (the ***path*** towards the next hop). This procedure generates the forwarding table. Notice that the two procedures (calculating next-hops and then deriving N-1 flows) are decoupled and can execute at different timescales.

7.2 Implications for multi-homing

Given the naming scheme presented in clause 7.1 and the properties exhibited by its names, multi-homing is achieved without the addition of any dedicated protocol. Multi-homing is the ability for a network node to exploit connectivity via multiple points of attachment (that is, data can leave and enter the node via multiple underlying interfaces), which may or may not be attached to the same network. Translating the problem into the RINA architecture environment, a multi-homed IPC Process should be able to send and receive data via multiple N-1 ports.

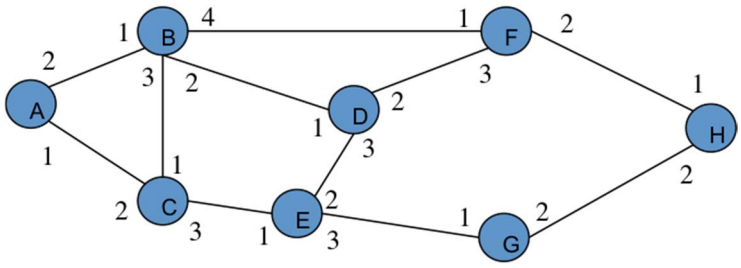


Figure 17: Connectivity graph of a DIF, formed by generic IPC network entities

Figure 17 shows the connectivity graph of an example DIF. The blue circles represent IPCPs, interconnected via N-1 flows (the solid lines). Each blue circle is labelled with the address of the IPCP (A, B, C, etc.), and each solid line with the local port-ID of the N-1 flow. Figure 18 shows the next-hop table computed by each IPCP, assuming that a simple shortest-path algorithm is used (to keep the example simple). Each column shows the next-hop table for a specific generic IPC network entity (the first column shows the next hop table for "A", the second one the next hop table for "B", etc.)

Using Next Hop

Destination Address	A	B	C	D	E	F	G	H
A	-	A	A	B	C	D	E	F
B	B	-	B	B	C	D	E	F
C	C	C	-	B	C	D	E	F
D	B	D	B	-	D	D	E	F
E	B	D	E	E	-	D	E	F
F	B	D	B	F	D	-	E	F
G	B	D	E	E	E	D	-	G
H	B	D	B	F	D	H	H	-

Figure 18: Next-hop table for each generic IPC network entity

If the IPCP "A" is sending packets to "H", the packets will contain "H" as the destination address. The IPCP "H" has 2 N-1 flows, one to "F" (with port-ID "1"), and one to "G" (with port-ID "2"). If the N-1 flow between "F" and "H" fails, "F" and "H" can detect it, and issue a routing update that discards the N-1 flow between "F" and "H". Then each IPCP will recompute the next hop table and PDU forwarding table taking into account that this N-1 flow is no longer available. Packets that were in transit can be re-routed because they are addressed to "H", irrespective of the path they follow to reach "H". IPCP A does not need to change the destination address of the packet, nor do anything special to re-route around the failure (just process the regular routing updates).

Had addresses been assigned to the N-1 ports instead of generic IPC network entities (equivalent situation to that advocated by the IPv6 addressing architecture [i.13]) the situation would be more complex, as illustrated in Figure 19. Here the labels assigned to the IPC Processes ("A", "B", etc.) are just for identifying them in the columns of the next-hop table, but not used for routing or forwarding. N-1 ports are assigned addresses that are unique in the DIF, which are used to compute the next-hop table shown in Figure 20.

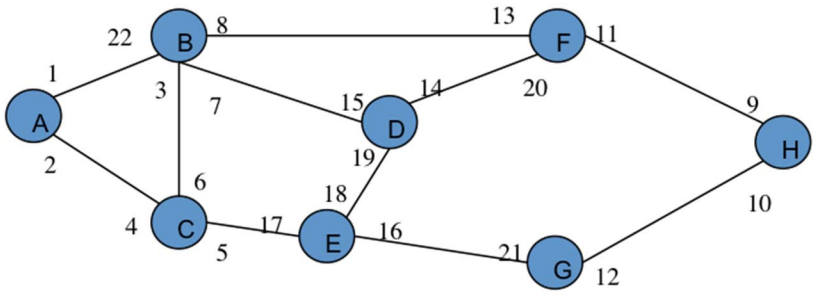


Figure 19: Connectivity graph of a DIF, with addresses assigned to N-1 ports

The first noticeable difference is that N-hop tables are much larger, since there are much more addresses (one for each N-1 port instead of one per IPCP). This in itself is a problem, but not the biggest one. If "A" sends packets to "H", the packets in the network are not reflecting this situation anymore. Source and destination addresses in packets will carry the address of one of the N-1 ports of the source/destination IPCPs; this example assumes it is "1" and "9". If the link between "1" and "9" fails (as before), "F" and "H" will detect this situation and issue a routing update, which will cause next-hop and forwarding tables to be recomputed. But packets that were in transit are addressed to "9", which is no longer available. Next hop and forwarding tables do not have enough information to know that address "10" brings packets to the same place as address "9", therefore packets will be dropped even though there is a viable path to IPCP "H". What is more, node "A" has to be informed that destination address "9" is no longer reachable, and that it should be using address "10" as a destination address instead.

It is important to notice all the problems that such a small change has created, requiring 3 to 4 times more addresses, larger next-hop and forwarding tables, extra protocols to detect failed addresses of multi-homed IPCPs and use alternative ones, and larger packet loss rates when failures in multi-homed IPCPs occur.

	Next Hop							
Destination Address	A	B	C	D	E	F	G	H
1	-	1	3	7	19	14	16	11
2	-	1	2	18	17	14	16	12
3	22	-	3	18	17	14	16	12
4	4	1	-	7	19	14	10	11
5	4	6	-	18	17	14	16	12
6	22	6	-	18	7	14	10	11
7	4	-	17	7	7	14	16	11
8	4	-	3	14	19	13	10	11
9	22	15	17	14	19	9	16	-
10	4	6	17	18	21	14	10	-
11	4	6	17	18	21	-	10	11
12	22	6	17	14	19	9	-	12
13	22	15	17	7	17	-	16	12
14	22	13	17	-	21	14	10	11
15	22	15	3	-	17	14	16	11
16	4	15	17	14	-	9	16	12
17	4	6	17	7	-	14	10	11
18	4	15	2	18	-	9	10	11
19	4	6	17	-	19	9	16	12
20	22	15	17	20	19	-	16	12
21	4	6	17	18	21	14	-	11
22	22	-	2	18	17	14	16	12

Figure 20: Next-hop table (addresses assigned to N-1 ports)

7.3 Implications for renumbering

Most real networks have to be eventually renumbered [i.14]: a subset or all the addresses assigned to network entities need to be updated. It may happen that the network has grown to the point that its current addressing plan is no longer effective or does not scale. Or perhaps the network is changing upstream providers and needs to get new addresses from the new provider (provider-based addresses are the norm in the current Internet), or there has been a corporate merger of two network providers (and hence their networks) Whatever the reason, renumbering in IP networks is a complex procedure involving a number of steps: IP addresses need to be assigned to interfaces on switches and routers, routing information needs to be propagated, ingress and egress filters are to be updated - as well as firewalls and access control lists, hosts get new addresses and DNS entries have to be updated.

An overview of the problems associated to renumbering of IP networks is provided in [i.15]. Since TCP and UDP connections are tightly bound to a pair of IP addresses, changing any of them will destroy the flow. Since DNS is an external directory - not part of the network layers - the renumbering process usually leads to stale DNS entries pointing to deprecated addresses. Even worst, applications may operate through the direct use of IP addresses, which will require an update in the application code, its configuration or both. Router renumbering usually requires an exhaustive manual and error-prone procedure for updating control plane Access Control Lists or firewall rules. Moreover, static IP addresses are usually embedded in numerous configuration files and network management databases [i.16].

Most if not all of the issues described are rooted in the incomplete naming and addressing architecture of IP networks. In contrast to the scheme presented in clause 7.1, in the IP world there are no application names - domain names are synonyms for IP addresses resolved outside of the network layer - and end to end communication flows are created between transport layer endpoints identified by an IP address and a transport port number. If flow requests to the network were based on application names and the network internally resolved those names to network addresses then renumbering would be completely transparent to applications.

The issues for ACLs and firewalls, which use rules based on IP addresses, have a similar origin. Since the IP address is both the identity of protocol machines (nodes) in the IP layer and also the name used for forwarding IP packets, there is an issue if the network is renumbered: ACL and firewall rules have to be updated to reflect the new address assignment. Obviating the fact that in well-formed architectures firewalls are not necessary [i.17], the problem can be generalized to that of setting access control rules in the IP layer. There is clearly a need for a stable, location-independent name that identifies the node (protocol machine) in the layer, and another location-dependent name that is used for forwarding packets between protocol machines (the addresses). In this scenario access control rules can be written in terms of the location- independent names; if addresses change access control rules do not need to be updated. Similar considerations apply to management-related problems: if the system being managed is identified by a location independent (management) application name, all problems related to stale addresses in configuration files and network management databases are just avoided.

The following paragraphs analyse if renumbering can cause issues in a network that conforms to the RINA architecture. Figure 21 illustrates the application registration and flow allocation procedures carried out by a DIF formed by IPC Processes (orange circles).

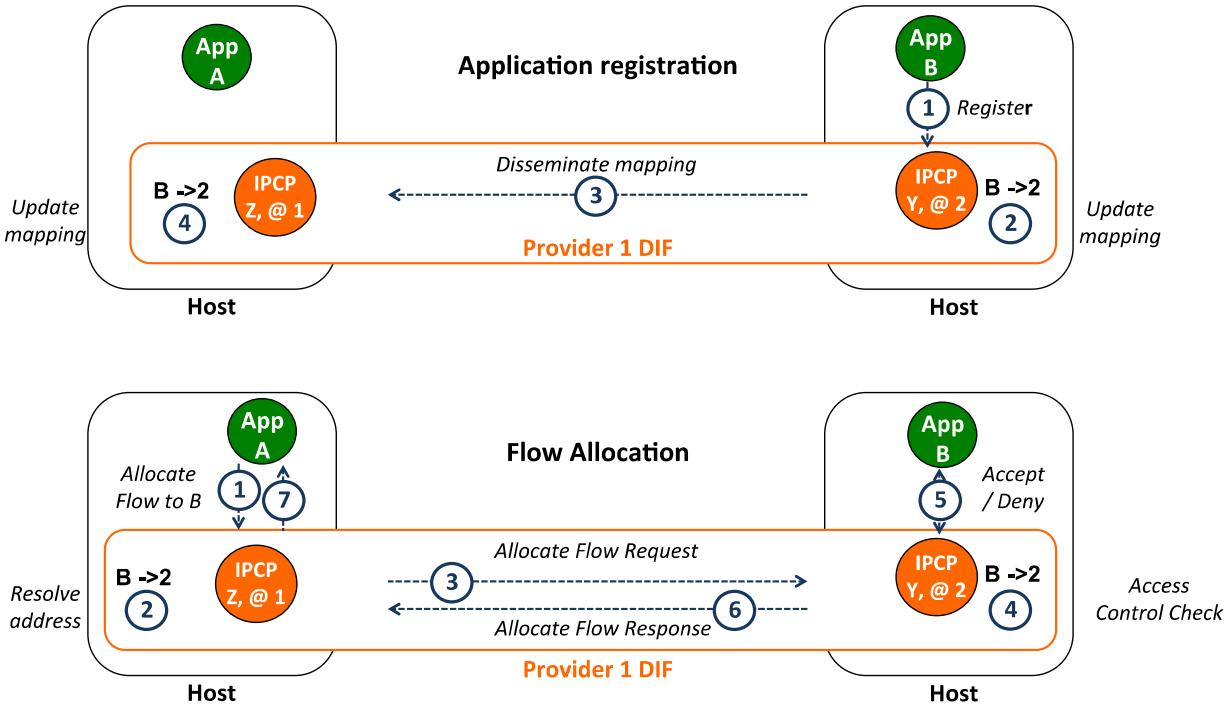


Figure 21: Application registration and flow allocation procedures

Application process (AP) named "B" registers to the orange DIF. IPC Process "Y" is the DIF representative at the system where AP "B" is executing. Upon receiving the registration request the IPCP updates its internal directory map and disseminates the new registration information through the DIF. Eventually IPCP "Z" learns this information and updates its internal directory map. Note that the procedures for maintaining the distributed directory in the DIF are a policy: they change from DIF to DIF, depending on its operational environment (the one illustrated in this example is kept very simple for the sake of clarity).

When an AP wants to communicate to another one, it requests the allocation of a communication flow to the destination AP, just providing its name. The local IPCP that processes the request queries the DIF directory to resolve the address of the IPCP through which the destination AP is reachable, and forwards the flow allocation request to the resolved IPCP. The destination IPCP does an access control check and notifies the local AP, who has the last say in accepting or rejecting the flow. An example of this procedure is shown in the bottom part of Figure 21.

Notice that in all this procedure the addresses of the IPCP are never exposed outside of the DIF, therefore if addresses change the ability to create new flows is not compromised and the identity of old flows is not lost. Moreover, since DIFs are also distributed applications - and IPCPs are just application processes - the naming problems of IPCPs at each DIF are also solved. IPCPs have a stable location-independent AP name that use as their identity, while they also have one or more temporary location-dependent synonyms (addresses) that are used for forwarding the packets of the DIF's generic data transfer protocol. Addresses are temporary by design, therefore renumbering is just part of the normal lifecycle of the DIF.

Network management is a distributed application, with some application process instances playing the role of "Managers" and other application instances playing the role of "Management Agents". As such, they all have location-independent application names, which are the ones used for establishing flows between Managers and Management Agents (or between Management Agents). Therefore, since addresses of IPCPs are never used to contact a managed system, renumbering cannot cause issues. [i.18] discusses the renumbering procedure in RINA in depth, providing some experimental results on renumbering multiple DIFs at the same time.

7.4 Implications for mobility

What is the fundamental problem with mobility management? Applications in mobile devices need to be able to keep sending and receiving data through the network, even if they keep changing their points of attachment to the network as they move (the service continuity requirement). The degradation in the quality of service (packet loss, delay) perceived by the application while the mobile device changes its physical point of attachment (the handover procedure) should also be minimized.

A good mobility management solution requires at least two sets of identifiers:

a) application names that do not change as the mobile host moves, so that communication endpoints have stable identities regardless of their location; and

b) addresses that change as the host moves, to reflect the position of the device in the network and allow routing to scale.

The fundamental problem of mobility management in the Internet is that the only identifier assigned to an entity in a Mobile Host (MH) that has scope greater than the system itself is the IP address. Hence there is a conflict: a single identifier cannot satisfy both properties at the same time.

There are no location-independent application names in the current Internet architecture. Communication endpoints are identified by the concatenation of the IP address and a local transport port. If the IP address changes, the identity of the communication endpoint is lost and the communication flow breaks. Domain names are macros for IP addresses. Domain names are not used for routing, and do not satisfy the properties stated in the former paragraph as changes to the DNS name do not (and cannot in general) propagate to users of an IP address if the DNS mapping is changed to refer to a different IP address.

Consequently, the IP address cannot change but the location of the device does; therefore, the routing infrastructure needs to be cognisant of IP addresses that are mobile. If nothing was done, then every router that might see a MH address would have to have a separate routing table entry for that address, since it would not be aggregatable once the MH leaves the home area where the address was originally assigned. In addition, the routing updates would have to be often enough and propagated through the entire IP routing infrastructure faster than the rate of change of points of attachment. This does not scale and several solutions have been proposed to mitigate this problem. Such mobility management solutions have been classified into two groups:

a) solutions that work at the IP layer;

b) solutions that hide mobility from the IP layer.

Type II solutions manage mobility at lower layers between the mobile device and a mobile gateway, essentially creating a large IP subnet that looks like a fixed layer from the IP layer point of view. Cellular systems such as LTE (Long Term Evolution), manage mobility by setting up and tearing down tunnels between the MH (called User Equipment in cellular terms) and mobile gateways that provide connectivity to the Internet or private IP networks. Tunnels have to be re-created every time the MH attaches to a different access point, incurring significant overhead. If the MH roams from a mobile network provider to another one, service continuity is lost (since the MH gets assigned an IP address from a different provider).

Type I solutions, labelled IP mobility solutions, provide different degree of support to mobility via protocols that operate at the IP layer. There are three primary approaches to doing IP Mobility: Mobile IPv4 (MIPv4) [i.19], Mobile IPv6 (MIPv6) [i.20], and Proxy Mobile IPv6 (MIPv6) [i.21]. LISP [i.22], the Locator-Identifier Separation Protocol, has also been recently proposed as an IP mobility solution. An analysis of these protocols can be found in [i.23].

Almost all IP mobility management solutions require the use of tunnels to handle mobility. Tunnels require the addition of dedicated forwarding table entries per mobile host, as well as a protocol to signal their creation, modification and destruction. Centralized mobility anchors as in PMIPv6 cause single points of failures and require significant resources at anchor nodes, since a lot of traffic has to be forwarded through them. LISP is not capable of dealing with mobility of hosts at the overlay IP layer, and incurs larger overheads to deal with mobility than networks based on RINA do [i.24].

The one solution analysed in [i.23] that does not require tunnels to deal with mobility (fully routed via BGP) has two main problems:

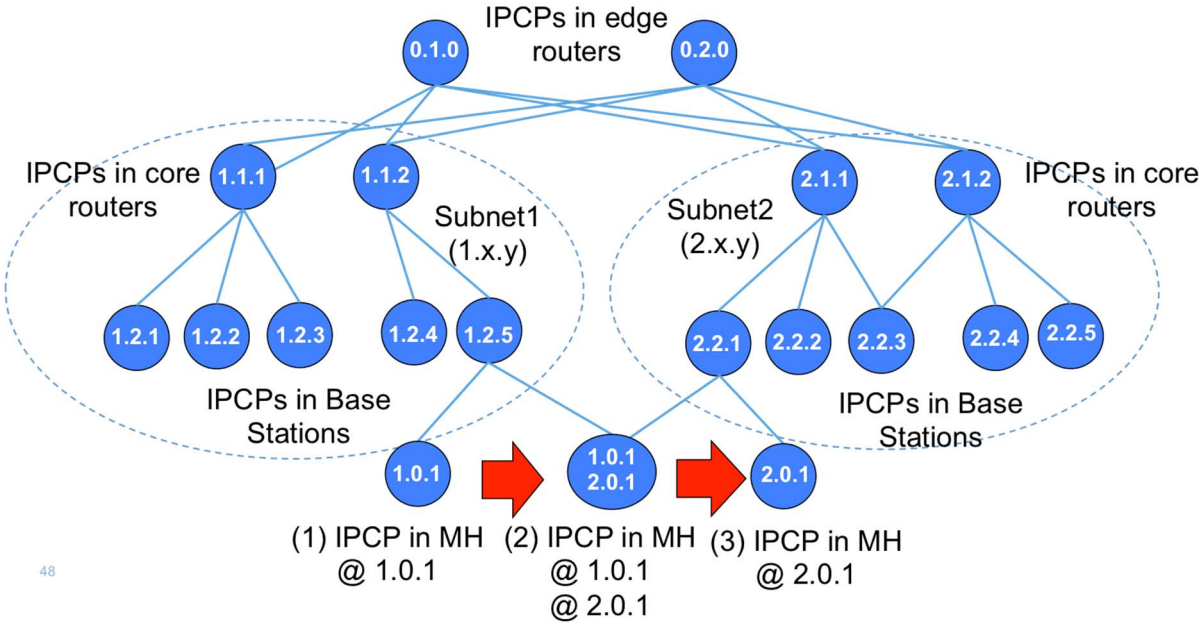
a) since the scope of the network layer is global routing convergence is too slow; and

b) since the Mobile Host IP address does not change, routing overhead is high and contributes to further increases in the size of forwarding tables tracking mobile nodes.

The fundamental reason mobility management is so complex is the lack of a full addressing complement. There is nothing to hold on to but the IP address. An added problem with all of these proposals is that they just address mobility for specific use cases. They require considerable additional mechanisms and protocols, which require additional security mechanisms, and have no other benefits.

In contrast, for a network that uses the naming and addressing scheme proposed in clause 7.1, mobility is nothing more than multihoming where the points of attachment change a bit more frequently. Clause 7.2 described how support for multihoming does not require any special protocols, it is realized as a consequence of the naming structure. The following paragraphs provide a short analysis of how mobility is achieved in networks built to the RINA architecture.

Addresses locate the Mobile Host (MH). Within a DIF, each IPCP gets assigned an address that is location-dependent, structured to reflect the location of the IPCP with respect to the other entities in its DIF. If the MH, which contains this IPCP, moves too far (e.g. attaches to a base station in another subnetwork) the address will no longer be aggregatable causing an increase in router table size and potentially less efficient routing. This means that the IPCP address in the MH needs to change to keep router table size manageable and efficient (these address changes will not happen with every base station change, only when the MH enters a different subnetwork within the DIF).



NOTE: The MH IPCP gets a new address when attaches to the BS IPCP. For a brief period of time the MH IPCP is multihomed to both BS IPCPs.

Figure 22: IPCPs in a MH doing a handover between base stations (BS) of different subnets

The procedure for updating addresses in RINA DIFs is discussed in the previous clause, showing that it can be done in real time without impacting the flows provided by the DIF (service continuity and QoS are preserved). Therefore, when a IPCP in a MH attaches to an entity of a new Base Station in a different subnetwork - as depicted in Figure 22, it will get a new address - and the old one will stop being advertised and disappear. The new address will be disseminated via the routing system, and directory updates will modify the required higher-layer application names to address bindings. All these procedures are already part of the architecture and do not require any additions to handle mobility scenarios.

Responsiveness to location change. Since address changes trigger routing and sometimes directory updates (only if the IPCP that changed addresses has server applications registered), these updates need to be fast enough (with respect to the rate and number of handover events) to allow the DIF to converge and reach a stable operational state. This puts a limit on the scope and size of the DIF, which should be properly designed to operate effectively. This is a similar problem to the one experienced by the Distributed Mobility Management (DMM) fully routed solution, with the key difference that in RINA network designers can decide the number and scope of DIFs in the network.

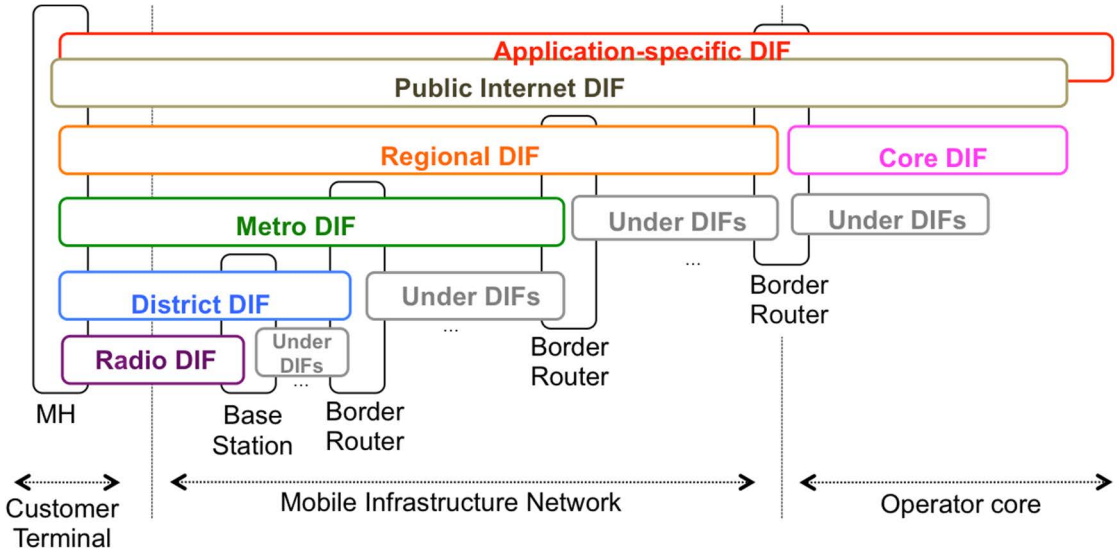


Figure 23: Mobile network partitioned into multiple DIFS of increasing scope

As seen in Figure 23, mobile networks can have multiple DIFs of increasing scope. Lower DIFs will manage frequent mobility events for a lower number of mobile hosts, over a reduced scope (e.g. a neighbourhood in a city). Higher DIFs will only be aware of handovers between access points belonging to different lower DIFs, therefore they will have to deal with less frequent mobility events and can have greater scope. This structure can be leveraged by the network designer to scale up the network and bound the size of routing tables and rate of mobility events at each DIF.

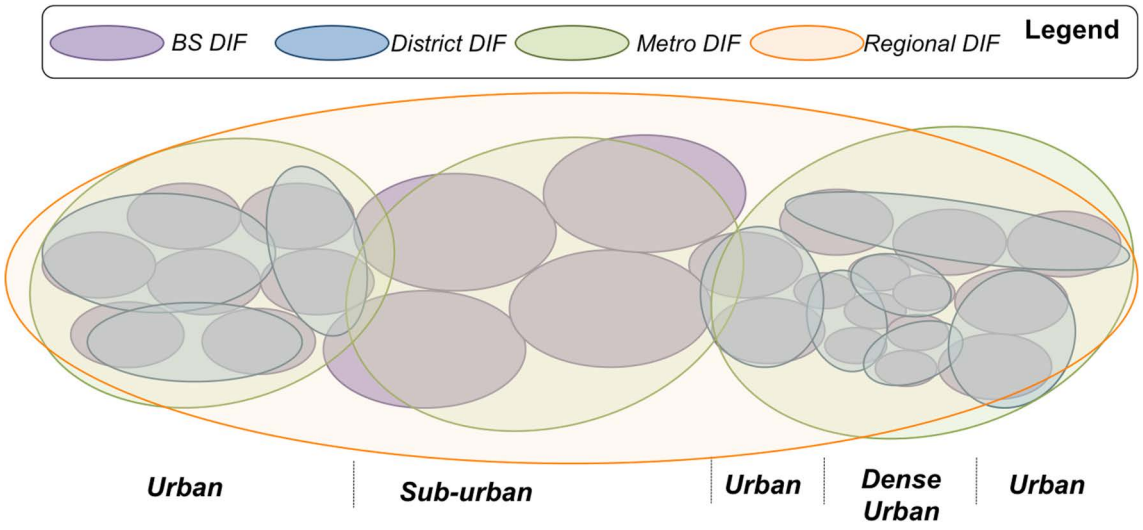


Figure 24: Mobile network viewed from above, with DIFs represented as circles

Moreover, the number of DIFs needs not to be the same in the whole network. For example, urban areas with a high density of users will benefit from a higher number of DIFs than rural ones, as shown in Figure 24. Last but not least, the operator can dynamically create more or less DIFs in different regions of the network according to its needs and demand for network resources: e.g. it may create an ad-hoc structure of DIFs to cluster all the mobile hosts of people attending a concert, to provide enough capacity and hide their mobility events from other parts of the network.

Adding more DIFs increases the header overhead, but since every DIF has the same protocols and the presence and length of protocol header fields (e.g. addresses) can be customized to the requirements of each DIF, the overhead of adding a DIF can be minimized compared to today's situation in which all layers are different. Moreover, scaling up a single layer horizontally also introduces overhead (length of protocol header fields will be larger, protocol mechanisms become more complex) and makes it harder to scale (as shown by the DMM fully routed solution example).

Service continuity and QoS degradation. The same flows are in place through the lifetime of the application connection, mobility events do not disrupt existing flows since application names are stable and location independent. There are no tunnels to set up and tear down. The QoS experienced by applications is only degraded by the handover delay, packet loss can only occur if there was no physical communication with the network.

Manageability. The application names never change. The mapping of application name to (N)-address and of (N)-addresses to (N-1)-addresses is ensured by engineering the scope of the DIFs to ensure the update time is small compared to the rate of change of (N-1)-addresses. Mobility management does not require extra procedures that are not already present in other types of networks (enrolment, routing, directories, address updates).

Scalability. Network designers are not limited to the use of a single DIF, they can scale its network both horizontally (creating larger DIFs) and vertically (creating higher DIFs on top of each other). The number of DIFs in a mobile network becomes a matter of network design - not decided and frozen in standards - therefore it can be adapted and optimized dynamically while the network is running.

7.5 Summary of RINA architectural properties relevant to naming, addressing and routing

This clause summarizes how RINA addresses the issues identified in clauses 4.3.3 and 4.3.4:

- **Complete naming and addressing architecture.** RINA uses a full naming and addressing complement. Application Processes have location-independent names, that uniquely identify them within a certain namespace. APs can also have synonyms that facilitate certain tasks within the distributed applications they are members of. Within a DIF, IPC Processes (which are just a form of APs) get assigned addresses, which are location-dependent names that facilitate forwarding PDUs. Addresses are never exposed outside of a DIF, its management is kept completely internal to the DIF. When an application registers to a DIF, a directory maps its application name to the address of the IPC Process it is attached to. If the application changes its point of attachment to the DIF, the mapping (which is internal to the DIF) is updated, but the application name does not change. Routing within a DIF is a two-step process: first a route in terms of IPC Process addresses is computed; then, for each next hop address, a suitable path to the next hop is chosen (the path is identified by the port-id of the N-1 flow towards the IPC Process). The choice of the path can be driven by resource allocation (e.g. load balancing), resiliency or other requirements.

- **Addresses assigned to nodes at each DIF.** The node in a layer is the protocol machine that strips off the header of that protocol in that layer. IPC Processes are the nodes of the DIF, and therefore the entities being addressed (in contrast to "Points of Attachment to lower level layers as recommended by the IPv6 addressing architecture [i.13]). This simple rule enables an IPC Process to have a single address regardless how to reach it (addresses are route independent), making multi-homing trivial and minimizing the number of addresses present in the forwarding table.

- **Networks can be seamlessly renumbered.** Since addresses are just temporary IPC Process synonyms internal to each DIF, IPC Processes can be dynamically and automatically renumbered without impacting the services they provide to upper DIFs. Seamless dynamic renumbering can be achieved because the address of an IPC Process does not capture its identity, and also because addresses are not exposed outside of a DIF. This feature not only facilitates the lifecycle management of a network (the addressing plan of each DIF can evolve as the needs of the DIF change), but also minimizes the size of forwarding tables in networks where IPC Processes move.

- **Naming DIFs enables application discovery across DIFs.** Not only individual Application Processes have names, but also the full distributed application. Since DIFs are just distributed applications, they have names assigned also. Therefore, it is possible to track which applications are available through which DIFs over a set of interworking RINA networks. The function that dynamically manages this mapping is called DIF Allocator, and enables an autonomic DIF discovery process. Since applications may be available through multiple DIFs, the DIF Allocator enables the dynamic discovery of the optimal DIF to reach a specific destination application (where optimal may be a function of latency, reliability, etc.), on a case by case basis.

- **Multi-homing achieved as a consequence of the structure.** Multi-homing is supported at the IPC Process level without the need of specialized protocols, just as a consequence of the naming an addressing schema. Since IPCPs are assigned location-dependent and route-independent addresses, the binding of destination IPC Process address to N-1 flow is a forwarding decision that is taken by the nearest neighbours of the IPCP. If a certain path (N-1 flow) to an IPCP fails, this mapping needs to be recomputed in nearest neighbours (either by re-routing, or by switching to an alternate path if multiples are available, etc.) The destination address of the IPCP with the failed path does not change, therefore in-flight PDUs can be re-routed and are not lost.

- **Mobility achieved as a consequence of the structure.** Mobility in RINA can be seen as dynamic multi-homing with expected failures. When an IPC Process in a DIF moves, it obtains new Points of Attachment and drops no-longer valid ones. As long as the IPCP address is still valid (properly represents the location of the IPCP in an abstraction of the graph of the DIF), mobility just triggers routing updates: no tunnels have to be setup. If the IPCP has moved to far away, it is renumbered following the normal procedures. All in all results in a much simpler and generic approach towards mobility, which keeps consistent regardless of the layer and underlying technology used.

8 QoS, Resource Allocation and Congestion Control

8.1 Consistent QoS model across layers

Even though all DIFs provide the same type of service, the characteristics of the offered service will vary between DIFs. Each DIF defines a set of supported QoS Cubes, i.e. QoS classes for flows providing statistical bounds on metrics like data rate, latency, losses, etc. Given these QoS Cubes, applications and upper DIFs can request flows with specific requirements, and the DIF then provides a flow with the QoS Cube that matches these requirements, using the policies (scheduling, routing, etc.) best suited for its purpose. Since all DIFs offer the same service interface, RINA features a consistent QoS model from the application down to the physical layers, in which QoS requirements are part of the meta-data passed to the DIF in each flow allocation request, as illustrated in Figure 25. Higher-level DIFs request flows with a certain QoS to lower-level DIFs, until the physical media is reached.

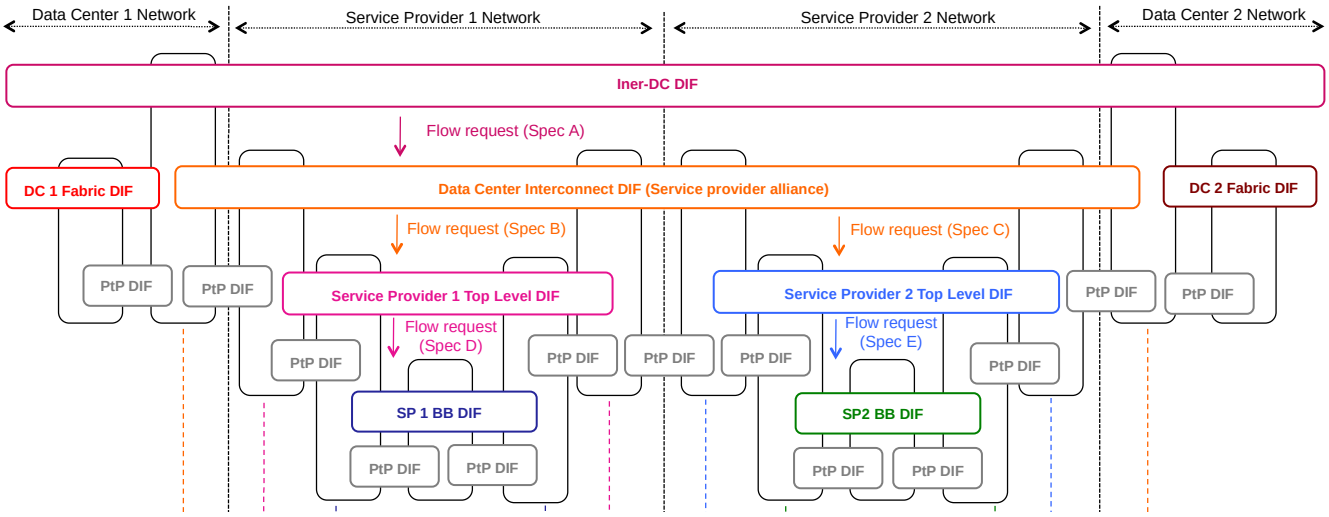


Figure 25: Consistent QoS model from the application down to the physical layers

Therefore, one of the key tasks when designing a DIF is to decide what QoS cubes it will be supporting. The term QoS cube comes from modelling the performance space as a set of N independent parameters whose values can vary between certain ranges. By using multiple axis, An N-dimensional performance space can be defined. QoS cubes define a region in the performance space, supporting a specific subset of parameters with a limited range. Some examples of QoS cubes could be the following:

- No loss with in-order delivery of data, no bounds on delay, no bounds on accepted load (equivalent to TCP in the public Internet).
- No bounds on delay, loss and accepted load (equivalent to UDP in the public Internet).
- 5 ms delay and 0,00001 loss probability for 95 % of packets, 15 ms delay and 0,0001 loss probability for 99,999 % of packets, for a maximum accepted load of 10 Mbps.

When a DIF receives the flow allocation request, it inspect the QoS parameters in the allocate flow request and maps the flow to one of the QoS cubes available in the DIF. Each QoS cube has a set of associated policies that have been designed to comply with the ranges supported by the cube. Such policies deal mainly with flow control, routing, forwarding, scheduling and congestion management.

All the EFCP PDUs that carry data from the flow are marked with the QoS cube identifier, so that intermediate IPC Processes can properly identify and process the traffic of each QoS cube. QoS cubes are standards within a DIF: all DIF participants support the same QoS cube identifiers and execute a coherent set of policies to provide a consistent end-to-end (within the DIF) QoS experience.

NOTE: In general flows at lower layer DIFs (N-1 DIFs) aggregate multiple flows from higher-layer DIFs (N-DIFs), hence one of the tasks of the DIF is to map the QoS class supported by the N-flow into one of the QoS classes offered by N-1 flows.

This QoS model applies recursively to each DIF. When an IPCP in a DIF requests a flow allocation to another IPCP, it uses the same abstract IPC service API as any other application, hence it tells a lower level DIF what is the name of the application it wants to communicate with and what are the quality requirements for the flow (loss, delay, in order delivery, etc.). The lower level DIF is not aware whether the application requesting the flow is a DIF or any other type of application. It will just map the flow request to one of the QoS cubes of the N-1 DIF, and allocate the flow. Hence QoS cubes definitions are internal to a DIF, but the semantics of the parameters passed to the allocate flow request should be standard across DIFs (so that they are all interpreted the same way by different DIFs).

8.2 Resource Allocation

The problem of resource allocation in a DIF consists in allocating resources to the different flows in a way that complies with the applications requirements and also makes efficient use of the resources of the DIF. The main resources of an IPC Process that need to be allocated to the different flows are buffer space and scheduling capacity. The DIF also relies on the resources allocated to the N-1 flows provided by other DIFs.

Multiple resource allocation policies fit within this framework: resources can be allocated to individual EFCP connections at flow setup time (emulating a virtual circuit approach), or can be shared between multiple EFCP connections belonging to the same QoS cube (class-based QoS), or may be even shared between all the EFCP connections equally (no QoS differentiation, a pure connectionless approach).

The model also allows for offline vs. online resource allocation. In offline resource allocation the DIF has the knowledge of the traffic it has to support, and resources are allocated in advance. In online resource allocation resources are allocated on the fly, as new flows are allocated and deallocated by applications using the DIF. A combination of both approaches is also possible - and will probably be the most common case. Using a combination of the approaches the DIF can perform an initial resource allocation offline, and then adapt to the operating real-time conditions using online resource allocation.

The Resource Allocator (RA) is the layer management task in charge of coordinating and adjusting the resource allocation policies within the DIF. It monitors the operation of the IPC Process and makes adjustments to its operation to keep it within the specified operational range. The degree to which the operation of the RA is distributed and performed in collaboration with the other RAs in members of the DIF and the degree to which the RA merely collects and communicates information to a Network Management System (NMS), which determines the response is a matter of DIF design and research. The former case can be termed autonomic, while the latter case is more the traditional network management approach. Both approaches have their use cases and application areas. The traditional approach is suitable when resources in the members of the DIF are tightly constrained, while the autonomic approach is more suitable when fast reaction times are required. The norm will be somewhere in between and there are a number of interesting architectures to explore. The RA is responsible for ensuring that the operation of data transfer within the IPC Process remains within (and meets) its operational requirements. Within an IPCP, the RA has access to the following meters and can adjust the following dials.

There are basically three sets of information available to the IPC Process to make its decisions:

- The traffic characteristics of traffic arriving from user of the DIF, i.e. the application or (N+1)-DIF.
- The traffic characteristics of the traffic arriving and being sent on the (N-1)-flows.
- Information from other members of the DIF on what they are observing (this latter category could be restricted to just nearest neighbours or some other subset - all two or three hop neighbours - or the all members of the DIF).

The first two categories would generally be measures that are easily derived from observing traffic: bandwidth, delay, jitter, damaged PDUs, etc. The shared data might consider internal status of other IPC Processes such as queue length, buffer utilization, and others.

The Resource Allocator has several "levers" and "dials" that it can change to affect how traffic is handled:

- **Creation/Deletion of QoS Classes.** Requests for flow allocations specify the QoS-cube the traffic requires, which is mapped to a QoS-class. The RA may create or delete QoS-classes in response to changing conditions. (It should be recognized that the parameters being manipulated are relatively insensitive and define ranges more than points in space.)
- **Data Transfer QoS Sets.** When an Allocate requests certain QoS parameters, these are translated into a QoS-class which in turn is translated into a set of data transfer policies, a QoS-class-set. The RA may modify the QoS-class-set used. For example, one could imagine a different set of policies for the same QoS-class under different load conditions.
- **Modifying Data Transfer Policy Parameters.** It is assumed that some data transfer policies may allow certain parameters to be modified without actually changing the policy in force. A trivial example might be changing the retransmission control policy from acking every second PDU to acking every third PDU.
- **Creation/Deletion of RMT Queues.** Data Transfer flows are mapped to Relaying and Multiplexing queues for sending to the (N-1)-DIF. The RA can control these queues as well as which QoS classes are mapped to which queues. (The decision does not have to exclusively based on QoS-class, but may also depend on the addresses or current load, etc.).
- **Modify RMT Queue Servicing.** The RA can change the discipline used for servicing the RMT queues.
- **Creation/Deletion of (N-1)-flows.** The RA is responsible for managing distinct flows of different QoS-classes with the (N-1)-DIF. Since multiplexing occurs within a DIF one would not expect the (N)-QoS classes to be precisely the same as the (N-1)-QoS classes. The RA can request the creation and deletion of N-1 flows with nearest neighbours, depending on the traffic load offered to the IPC Process and other conditions in the DIF.

8.3 Congestion control

The way RINA controls congestion is a generalization of how it is done in the Internet: if there is only one DIF doing congestion control in the network, it operates in an end-to-end fashion. If two or more congestion controlled DIFs are concatenated, the end-to-end control loop is broken into shorter loops. Congestion control in RINA "naturally" exhibits properties of various improvements that have been made to (or at least proposed for) the Internet, without inheriting the problems that come from imposing these mechanisms on an architecture that was not made for them (all the problems that Performance Enhancing Proxies (PEPs) have) [i.52].

PEPs usually break end-to-end connections into several closed-loop connections; in each connection, a congestion control scheme which considers local link characteristics can be exploited. Internet PEPs have several disadvantages. In addition to the reliability problem (they break the end-to-end semantics of the transport connection), complexities in using IPsec and TLS arise because security is an end-to-end function in these cases. PEPs also do not scale well with the number of flows and running a separate TCP instance for every flow is expensive in terms of the processing delay.

The layered architecture of RINA does not have these problems of Internet PEPs because:

1) security is a per-DIF function: PDUs are protected as they cross DIF boundaries, and because each DIF has its own congestion control, encrypted connections do not need to be split; and

2) flows towards each next hop are aggregated at the lower DIFs, which means that there is much less state to maintain.

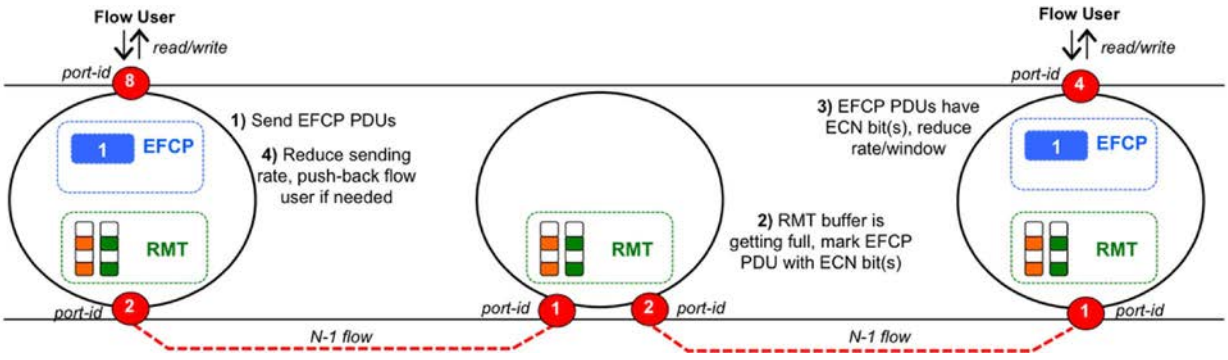


Figure 26: An illustrative example of how congestion is managed in a DIF

When a receiver of an EFCP connection sees an EFCP PDU with the ECN mark, it will compute a new (reduced) rate for the sender, and signal this new rate to the sending EFCP protocol machine using the flow control mechanism of the EFCP connection (alternatively it may echo the information back to the EFCP sender, who will do the calculation of the new rate/window). The sending EFCP protocol machine will reduce the sending rate, also causing push-back on the user of the flow. If required, this push-back can be done by blocking the user of the flow or signalling an error condition when a write API call is made.

An important difference with the Internet is that congestion can be managed on a per QoS-cube basis; the DIF has enough information to adopt different congestion management policies for different QoS clases. For example, in the presence of "low congestion" only best-effort traffic may be pushed back; if congestion continues to raise all traffic except the low-latency one is pushed back; and so on.

8.4 Summary of RINA design principles relevant to QoS, Resource Allocation and Congestion Control

This clause summarizes how RINA addresses the issues identified in clause 4.3.5:

- **Consistent QoS model across layers.** An abstract service model that is technology-independent and a consistent service API across all layers are the key properties that allow RINA to provide a consistent QoS model across DIFs. The role of all DIFs is to provide communication flows with QoS characteristics over a certain scope. These QoS characteristics - the flow Service Level Agreement, SLA - bound the statistical properties of the flow, such as packet loss, delay, capacity as well as other properties like in-order delivery of data. Each DIF maps the flow requirements to one of the service categories (QoS levels) it provides. Each DIF marks the PDUs belonging to a certain flow with a QoS-id, so that the policies at each IPCP can treat PDUs belonging to the same class in a consistent way. Providers can signal QoS requirements to each other, without divulging technological details of the operation of its network, hence eliminating barriers towards a multi-provider QoS market.

- **Congestion control loops using ECN close to where congestion occurs.** Each DIF has the capability to identify and manage congestion. A network designer can use this ability to minimize the response time to congestion and its variance (which is key to react against it at the proper timescales), and confine the congestion effects to the DIF(s) where it is happening and to the flows that are causing it. Use of ECN information in the EFCP PDU headers ensures that control loops are really reacting to congestion and not to other phenomena such as packet loss - "*as it may happen with implicit congestion detection approaches in wide use in the Internet today*".

- **Per-DIF customized congestion control policies for heterogeneous networks.** Since each DIF can manage congestion in its own resources, the congestion management policies can be tailored to the DIF environment. The controller used in a wireless DIF will be different to that of a low-level network DIF or to that of a VPN DIF. Heterogeneous networks can have a more efficient resource management than today by using heterogeneous congestion management policies, instead of relying on a single end-to-end congestion controller which has no visibility of any lower-layer retransmissions or resource contention. Chained controllers allow per-network segment customization, while stacked controllers reduce the number of competing control loops and increase the stability of the overall network behaviour [i.52].

9 Security

9.1 Introduction

Current Internet security is complex, expensive and ineffective. Attacks on the applications using the Internet and on the infrastructure providing it grow every year, in spite of the ever-increasing security-related protocols, systems and devices deployed in the net. One of the usual explanations is that the TCP/IP protocol suite was not designed having security in mind. Security mechanisms have been added as add-ons or separate protocols. However, two decades after recognizing this issue [i.26], Internet security levels are still not adequate for a critical infrastructure. It can be argued that one of the main causes of the Internet's security problems are the fundamental flaws and limitations in the TCP/IP protocol suite design [i.27].

The lack of a structured approach towards design causes a proliferation of protocols, each of which has to be individually protected. Unexpected interactions between the ever-increasing protocol-base, inefficiency in the number of repeated security mechanisms, a flat network layer and exposing addresses to applications make the Internet virtually impossible to secure at an affordable cost [i.28]. In contrast RINA provides an architectural framework with a well-defined building block - the DIF - that recurses as many times as required. Security functions belong to the DIF, not to individual protocols within the DIF. Interactions and the trust model between DIFs are well understood, allowing network architects to reason about the security of a network, understand the threats it is exposed to and design the policies that are more adequate to protect its DIFs.

The following clauses provide further detail on the RINA security properties.

9.2 Securing DIFs instead of individual protocols

DIFs are the composable building block of RINA, the tool available to network designers to build networks. DIFs, and not individual protocols, are the item to be secured in the RINA architecture. The recursive structure of RINA enables a clear security model in which the trust relationships between DIFs and between the members of a single DIF are well understood. Figure 27 illustrates these trust boundaries, which facilitate the placement of the different security functions.

Users of a DIF need to have little trust of the DIF they are using: only that the DIF will attempt to deliver Service Data Units (SDUs) to some process. Applications using a DIF are ultimately responsible for ensuring the *confidentiality* and *integrity* of the SDUs they pass to the DIF. Therefore, proper SDU protection mechanisms (such as encryption) have to be put in place.

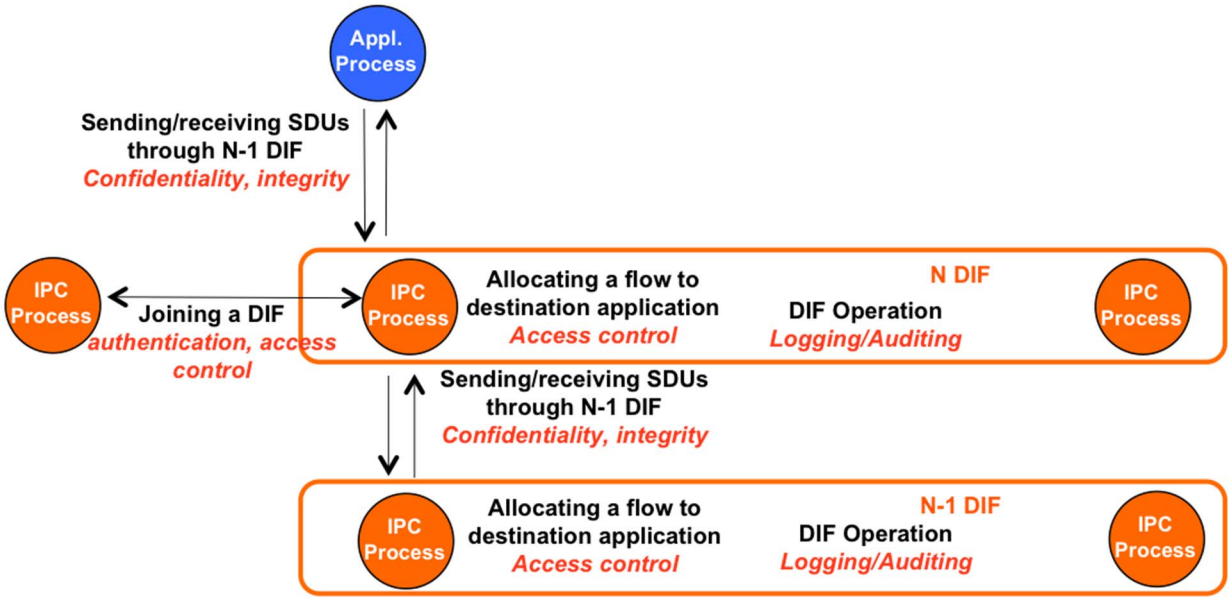


Figure 27: Placement of security functions in RINA

When a new IPCP wants to join a DIF it first needs to allocate a flow to another IPCP that is already a DIF member via an N-1 DIF both processes share in common. Here *access control* is used to determine if the requesting application is allowed to talk to the requested application. If the flow to the existing member is accepted, the next step is to go through an *authentication* phase, the strength of which can range from no authentication to cryptographic schemes. In case of a successful authentication the DIF member will decide whether the new IPCP is admitted to the DIF, executing a specific *access control* policy.

Remote operations on peer IPCPs are another area where *access control* is of key importance. All the layer management functions of an IPCP use a common infrastructure to exchange information with its peers: the Resource Information Base (RIB) and CDAP. CDAP defines a protocol to perform six remote operations over a set of distributed objects, which are used to model the information of each specific layer management task. The RIB imposes a schema (naming and set of relationships) on the DIF objects. At the finest granularity it is possible to take an access control decision to authorize the access to each individual object in the RIB schema for each of the CDAP operations (allowing different tasks to define their own access control restrictions).

Small et al. perform a threat analysis of RINA at the architecture level in [i.17], concluding that when proper authentication, SDU protection and access control policies are put in place, a DIF is a securable container: a structure used to hold or transport data that can be made not subject to threat. In contrast the TCP/IP protocol suite security model is usually based on building security functions for each protocol.

For example, DNSSEC [i.29] provides data integrity and authentication to security-aware resolvers. IPsec [i.30] is a general framework for secure IP communications, supporting confidentiality, integrity, authentication or protection against replay attacks. However, since IPsec works end-to-end within an IP layer, it either only protects the IP payload (transport mode) or makes IP connection-oriented (tunnel mode), encapsulating a protected IP packet into an unprotected IP packet. This makes IPsec a partial solution, not addressing the requirements of IP control plane protocols, which need to define their own security functions, such as OSPF [i.31] or BGP [i.32]. TLS, the Transport Layer Security Protocol [i.33], specifies a set of related security functions to enable secure communications over the transport layer.

All in all, this approach results in more overhead and complexity compared to securing DIFs. In [i.34] Small performs an initial comparison between RINA and the current Internet, measuring the flows, protocols and mechanisms required to secure each architecture. Small concludes that RINA networks can deliver on security requirements with less complexity than is currently possible using the Internet protocol suite.

9.3 Recursion allows for isolation and layers of smaller scope

One of the main challenges in securing the current Internet is that of its scope. The attack surface of a layer increases with its size: the bigger a layer is the larger will be the number of potential attackers and of exploitable vulnerabilities such as missconfigurations, use of weak credentials, etc. The scope of the public Internets IP layer is huge; big enough to exhaust IPv4's 32 bits of address space, and it will continue growing with IPv6.

Not only does this fact make nodes in the public Internet more prone to attacks, but also thwarts the deployment of new protocols due to two reasons: either the new protocols need to assume some level of trust in their peers that cannot be guaranteed in the wild public net; or millions of standard defense mechanisms such as firewalls need to be updated to consider the particularities of the new protocol, which is hard to do in practice (e.g. SCTP deployment [i.35]). Existing protocols are also hard to upgrade, even if the upgrades are critical to the security of the Internet such as the case for BGP security extensions [i.36].

The advent of network virtualization and its large-scale deployment in datacentres (DCs) has provided DC network designers with a tool to create DIFs of smaller scope that provide enhanced isolation, minimize the impact of security threats and enable the customization of security policies to different user profiles [i.37]. RINA's recursive structure generalizes network virtualization, allowing network architects to compose DIFs of arbitrary size and custom policies into a network with a clear security model.

Figure 28 shows an example of the network of a service provider, connected to a customer network (left) and peering with another providers network (right). Provider 1 only shares the access DIF and the multi-provider DIF with other networks, the internal DIFs of the provider - regional and backbone DIFs - are not visible outside of the providers network. This design reduces the networks attack surface, limiting the damage that an external attacker can perform: most of the provider's routing and resource allocation functions are executed in the internal DIFs. Compromising those DIFs requires physically compromising the providers assets, therefore the provider can focus its resources on protecting the perimeter of its network with strong authentication and SDU protection policies.

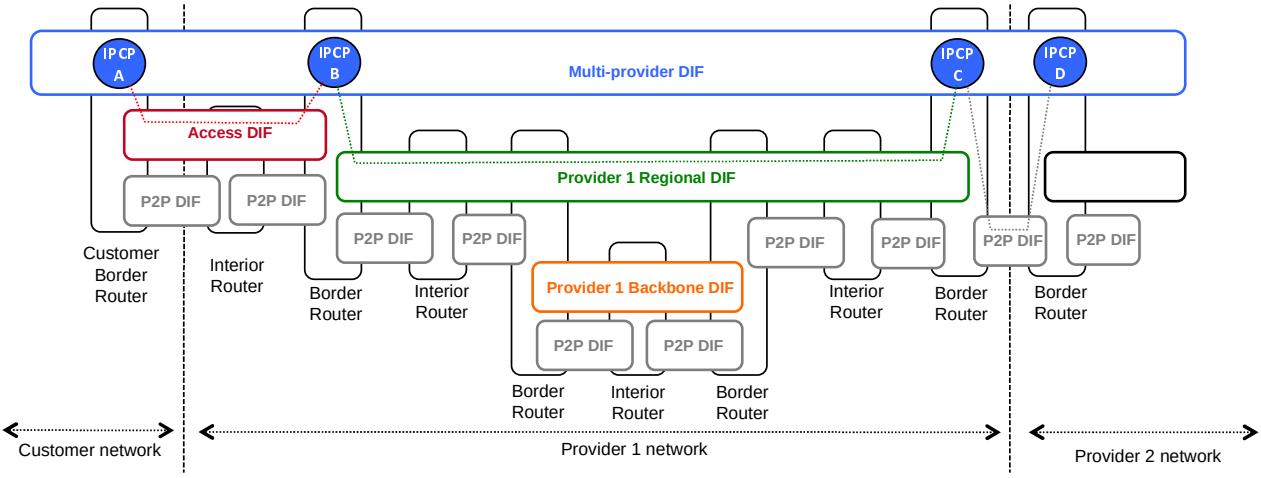


Figure 28: Using RINAs recursive structure to hide the internal DIFs of a service providers network

9.4 Separation of mechanism from policy

In RINA the principle of separating mechanism from policy [i.7] is used to separate the fixed parts of an IPC Process function - which are the same across DIFs - from the variable parts. For example, an acknowledgement is a mechanism, when to acknowledge is policy. This principle enables the same mechanisms to be re-used across DIFs, minimizing the number of different mechanisms present in the network [i.34] while still allowing for the customization of the DIF security policies. Policies written for a DIF can be re-used in other DIFs, maximizing the efficiency of specifications and implementations.

Separating mechanism from policy allows each DIF to adapt to different operating environments while keeping an upper bound to the complexity of the architecture, which is one of the critical metrics when securing a distributed system [i.38]. Network architects do not need to design more protocols, just policies that are tailored to the requirements of different DIFs. For example, in Figure 28 IPCP B in the blue DIF use some form of cryptographic authentication and encryption when exchanging information with IPCP A over the red DIF, since the red DIF is shared between the provider and different customers. However, IPCP B may use no authentication nor encryption when exchanging information with IPCP C over the green DIF , since in this case all systems are under the control of the provider.

9.5 Decoupling of port allocation from synchronization

In the TCP/IP Protocol suite TCP overloads the port-id to be both a local handle (socket) and the connection-endpoint-id (CEPID). Furthermore, the lack of application names overloads port-ids with application semantics: application endpoints are identified by a combination of IP address and a well-known port-id that is assigned when the application binds to an IP layer. Static destination port-id values have to be known by the source application when requesting a transport connection. Therefore, an attacker wanting to intercept a particular TCP connection only needs to guess/spoof the source port-id.

In RINA port-allocation and synchronization are separate functions, applying the results obtained by Watson with the delta-t protocol design [i.9]. The port-allocation procedure is explicitly triggered by an application requesting a flow to a destination application. The source IPCP dynamically assigns a local port-id to the flow and creates an instance of the EFCP protocol that takes care of the feedback synchronization aspects (flow and retransmission control). The EFCP instance is identified by a dynamically generated source CEPID that is mapped to the port-id via a local binding as shown in Figure 29. The destination IPCP does the equivalent steps, resulting in a local port-id and a destination CEPID. The source and destination CEPID are the values seen on PDUs in the wire; port-ids are just of local significance and used by applications to read/write data from flows.

The state of ports and connections is managed with different approaches: port state is explicitly created and removed by applications (hard-state) whereas connection state is created and removed following a timer-based approach (soft-state): after long periods of no traffic the connection state is removed, and created again when new traffic is sent/received on the connection. Bodappati et al. showed in [i.11] how RINA leverages this design to achieve a greater resiliency tan TCP/IP to transport-level attacks such as port scanning, connection opening or data transfer.

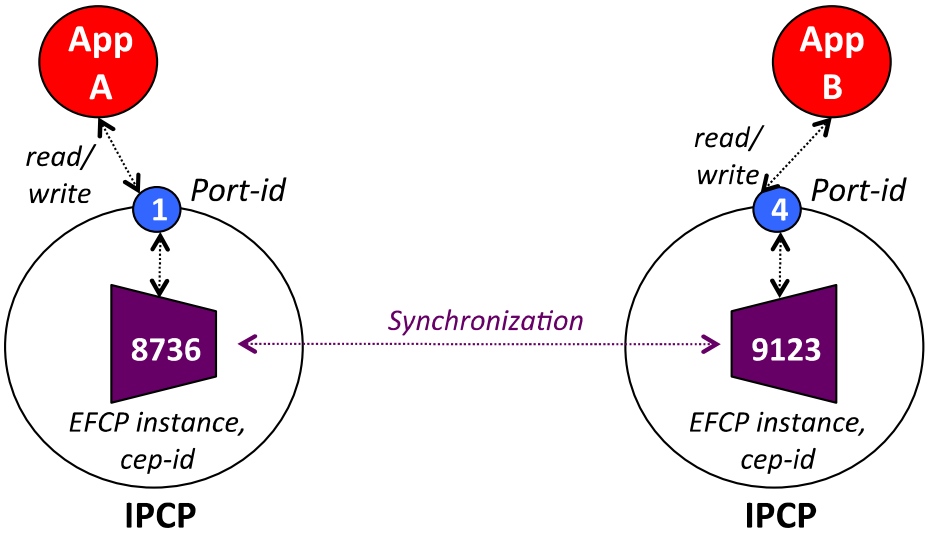


Figure 29: Decoupling of port allocation and synchronization in RINA

9.6 Use of a complete naming and addressing architecture

Naming and addressing design considerations also have a profound impact in the security of network architectures. Since the TCP/IP protocol suite does not have application names (DNS is an external directory), IP layers expose addresses to applications. This disclosure of information facilitates spoofing of IP addresses, and in combination with the use of common monitoring tools such as traceroute or ping allows attackers on end hosts to learn about the addresses of potential targets in a layer - routers or other hosts - as well as the network connectivity graph. Attackers can use this information to setup DDoS attacks by automating the discovery and infection of vulnerable machines [i.39], or to attack the network infrastructure by gaining control over routers.

RINA features a complete network and addressing architecture, with application names and per-DIF directories that perform application to IPC Process address resolution. When an application requests a DIF to allocate a flow to a destination application, it provides the source and destination application process names. The DIF internally resolves the destination application name to the address of the IPC Process where the destination application is registered.

Due to the existence of application names a DIF's addressing information (address format, valid/active addresses) is not divulged outside of the scope of the DIF. An attacker in a host cannot address the IPC Processes of a DIF unless it joins the DIF it wants to attack, which requires authentication.

9.7 Summary of RINA design principles relevant to security

This clause summarizes how RINA addresses the issues identified in clause 4.3.6:

- **Secure DIFs instead of individual protocols.** RINA provides a consistent security model across layers. *DIFs are securable containers*, they can be not subject to threat providing the appropriate authentication, access control and SDU protection policies are put in place. In RINA, *the unit of protection is the DIF and not its individual protocols*. The functional decomposition of a DIF naturally defines the placement of security functions and places the trust boundaries between and within DIFs. Authentication, access control and SDU protection are policies that can be plugged into any DIF - regardless of its ranking and requirements - thus minimizing the cost of specifying and implementing security policies "*compared to the per-protocol security model of the current Internet*".

- **Decouple port allocation from data transfer synchronization - no "well-known ports"**. Each DIF allocates flows based on a destination application name, which is internally resolved to an IPC Process address. When the flow is allocated, the requesting application gets a (OS) local identifier as a local handle to the flow: the port-id. As opposed to the current Internet, port-ids are dynamically generated and of local significance only, "*thus well-known port scanning is not an effective attack for DIFs*". Moreover, the transport state associated to the flow is managed by instances of the EFCP protocol, which have their own separate identifiers (connection-endpoint ids or CEPIDs) locally mapped to port-ids. *CEPIDs are also dynamically generated at flow allocation time*. Decoupling of port-ids and CEPIDs allows the same flow to be supported by multiple EFCP instances during the flow lifetime (for example to avoid sequence numbers to rollover if encryption is in use).

- **Use of application names: addresses are internal to each DIF.** IPC Process addresses are internal to a DIF and never divulged outside of it. All a user application knows about a DIF is its name and the QoS classes it supports. *Therefore, an application using a DIF cannot address any member of the DIF unless it joins the DIF*. If this risk/threat exists, the DIF designer will have configured the DIF to use authentication and access control, therefore a potential attacker needs to circumvent these security controls before being able to communicate to any IPC Process in that DIF. RINA provides a more controlled environment in which association control and the degree of network openness is a matter of design and not "open to everyone by default, but add infrastructure to close it if connectivity has to be limited". Strong layering prevents the DIF from leaking information which is not essential for providing its service and that may be used to attack it.

10 Network Management

10.1 Common elements of a management framework

A high-level overview of the common elements in the RINA network management framework is provided by Figure 30. Each processing system (rectangle boxes) can run one or more IPC processes, implementing one or more DIFs in that system. IPC Processes in a system are managed by the Management Agent, who has read and write permissions to the IPC Processes' Resource Information Base (RIB)s. In some theoretical configurations one or more DIF Managers communicate with the agents in each system of its management domain to provide central configuration, fault, security and performance management. The management agents and the DIF Managers together form a distributed application that manages elements of one or more DIFs, and is called Network Management System - Distributed Application Facility (or NMS-DAF in short).

While the IPC-Processes that comprise the DIF are exchanging information on their operation and the conditions they observe, it is generally necessary to also have an outside window into the operation of DIFs comprising the network. While the members may reach a local optimization, it is often more complex to discover global optimizations without an "outside" perspective. In these systems, control needs to be automatic. Events are happening far too fast and state is changing too rapidly for a centralized system to be effective. Furthermore, the nature of distributed systems always opens the possibility for partitioning. Hence, it should be possible for distributed systems to fail-safe without central control. A NMS-DAF will perform the traditional functions of monitor and repair, deploying new configurations, monitoring performance, etc. The DAF (distribution application facility) model can be applied to network management to represent the whole range from distributed (autonomic) to centralized (traditional).

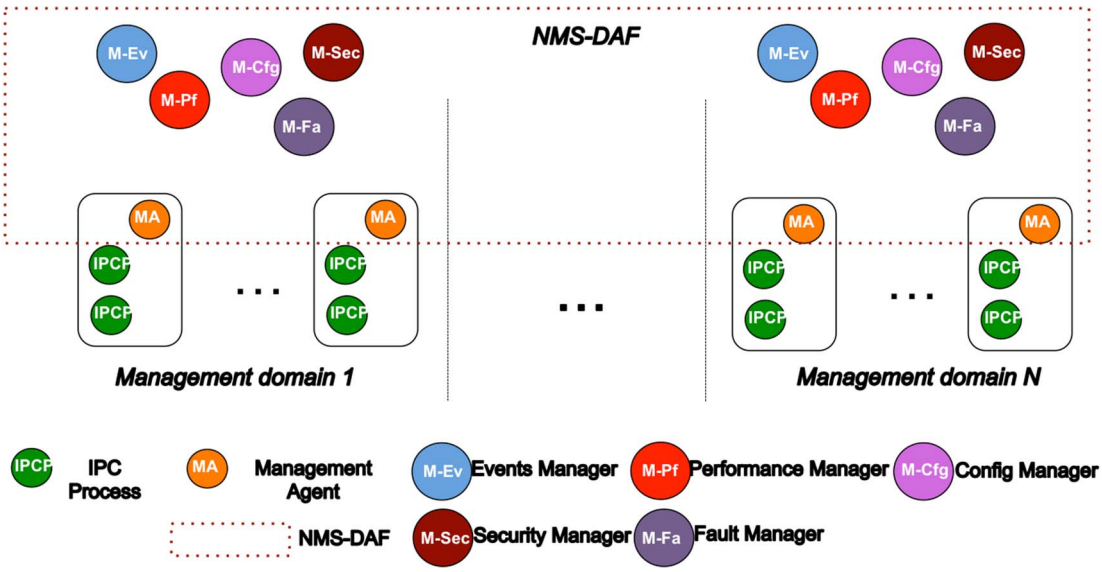


Figure 30: Typical configuration of a centralized NMS-DAF with multiple management domains

In the traditional centralized network management architecture, an NMS-DAF would be a heterogeneous DAF consisting of one or more application processes providing management functions, with other application processes providing telemetry. The management APs might be subdividing roles or tasks within network management or providing management for sub-domains and redundancy for each other. A typical NMS-DAF will have the usual tasks of event management, configuration management, fault management, resource management, etc. This also has the advantage of shifting the focus away from boxes to a distributed system model.

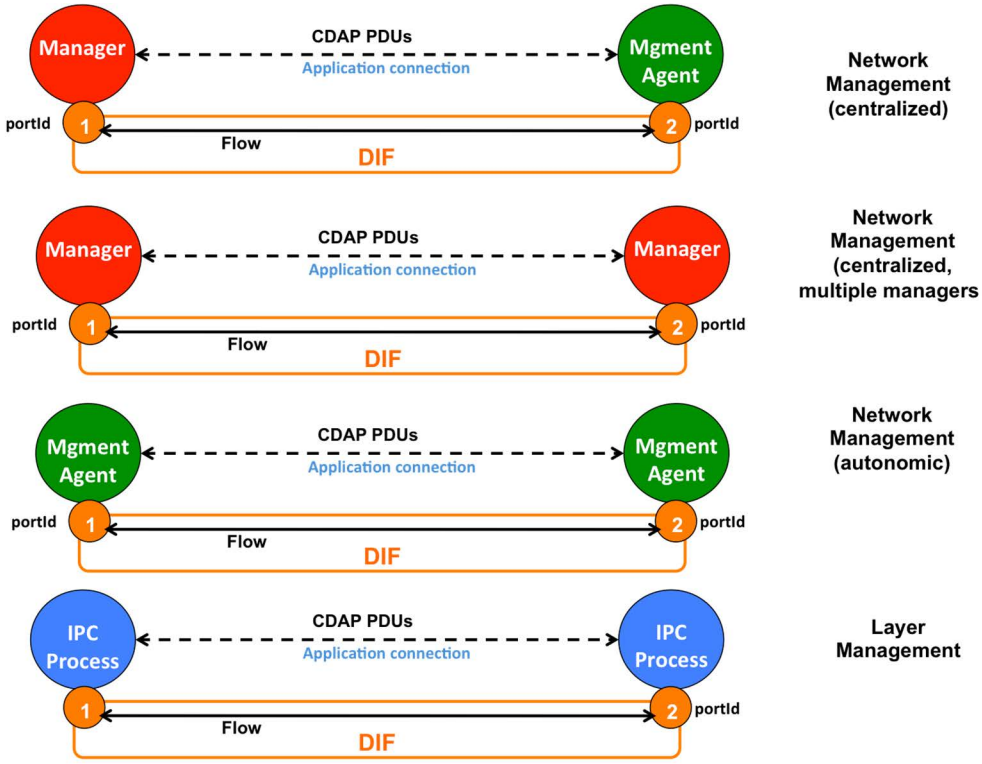


Figure 31: Different interactions between NMS-DAF Application Processes, and IPC Processes for layer management (down)

The NMS-DAF is a distributed application (a DAF), and hence leverages the common elements in the DAF Framework. In order to interact with each system, the manager processes need to have a DIF in common with it. There are several ways of achieving this, ranging from using a single DIF dedicated to interconnect the manager with each Management Agent, to using different DIFs for different systems. Once the manager has allocated a flow to a Management Agent, it establishes an application connection to it via CACEP, which includes optional authentication. Once the application connection is in place, the Managers and the management agent can communicate by performing remote operations on the RIBs of IPC Processes via CDAP - the Common Distributed Application Protocol.

Figure 31 illustrates the different interactions between the APs in the NMS-DAF (with the exception of the last one, which is a layer management interaction between IPCPs in a DIF):

- **Manager-Management Agent interaction.** The most common interaction in the traditional configuration of Network Management Systems, in which a Manager process uses agents in each Computing System in order to monitor those systems and update its configuration when needed.

- **Manager-Manager interaction.** In most cases, like when a DIF is owned by multiple independent entities or for dividing the Manager functions into separate areas (Fault, Configuration, Event, Performance, etc.), it is necessary to partition the management of one or more DIFs into multiple management domains. At least one Manager process is responsible for managing one of those individual domains. Therefore, Manager to Manager interactions are also required.

- **Management Agent - Management Agent interaction.** In this configuration the Management Agents have more autonomy to take certain decisions based on the information learned from other neighbour Management Agents. The degree of autonomy can vary depending on the configuration in use: from using Management Agents to aggregate management information in "sub-domains" to Network Management Systems with no central Managers at all.

- **IPC Process - IPC Process interactions.** While this type of interaction does not purely fall in the category of "Network Management", IPC processes in a DIF use the same tools to exchange information: CACEP for application connection establishment and CDAP to operate on the objects of the neighbouring IPC Processes RIBs. Examples of usage of this interaction are enrolment, routing, flow allocation or resource allocation.

10.2 Managing a repeating structure

The common structure of RINA can be exploited by Network Management in a number of ways.

Simplification in the management of multiple layers, yet with the possibility to configure different policies. Today more and more layers are performing functions that were not traditionally assigned to them (such as link-state routing in layer 2 DC networks [i.40] or in layers implementing the so-called "Virtual Networks"). Layers are transitioning from performing a single function to becoming units of distributed resource allocation, something the RINA model fully predicts. However, since the different traditional layers have evolved independently from different starting points and have been standardized by different committees, the same functions (such as routing, or authentication) have very different semantics.

As a consequence, configuration, performance or fault management become complex, since the NMS have to be aware of the details of every specific technology implemented in different layers. In contrast, RINA networks provide the same structure; configuring for example a link-state routing in the "DC-fabric DIF" or in each of the different "Tenant DIFs" with exactly the same semantics - the same objects with different values can be used everywhere [i.41].

More effective complex network event management. The commonality provided the RINA architecture enables the definition of a performance model that is common to all layers, as explained in clause 8. That is a very powerful tool for the Network Management System, since:

a) the NMS would be receiving less types of events, freeing it from translating to an internal performance model to reach a proper understanding of the overall network status;

b) the correlation of events in multiple layers would become much simpler, facilitating the identification of problems by the NMS and allowing it to roll-up actions to effectively solve and mitigate them.

The same arguments apply to security, perhaps even with a broader impact.

More automation is possible: ***network management should be "monitor and repair" - not "control" [***i.42***]***. With RINA the focus of the NMS can shift from "protocols" to "policies", requiring less human intervention from human operators - and hence less human errors. Since protocols would become a commodity - just part of the common, standard layer infrastructure - research and experimentation could shift towards understanding the behaviour of different policy-sets under different operating conditions. This would enable network administrators to manage the network the following way:

1) At design time, characterize the different conditions that the network will experiment during its day-today operation (e.g. in terms of offered load, failure of links, identified ongoing security attacks, etc.).

2) At design-time, group these conditions into well-defined "operational regions", and choose policy-sets that are effective for each of the different regions.

3) At run-time, let the NMS monitor the network, decide if the measurements belong to one of the "designed" operational regions. If the operational region did not change, continue monitoring. If the operational region transitioned into another "designed" operational region, automatically apply the related policy-set. If the operational region is unknown - as it could be the case with natural catastrophes, terrorist attacks, etc. - notify the human operator.

10.3 Summary of RINA design principles relevant to Network Management

This clause summarizes how RINA addresses the issues identified in clause 4.3.7:

- **Two immutable protocol frameworks and a well-defined set of policies.** Everything in the RINA architecture tries to maximize the invariants and reduce the variability in computer networks, minimizing it to the essential subset of functions that need to change to adapt to different application and physical layer requirements. The resulting immutable, common structure with a single type of layer that features two protocol frameworks and a well-defined set of policies greatly simplifies the complexity of the network management problem. This is especially true compared to the current landscape of redundant, independently designed protocols across layers and within layers. Simplifying the network structure not only reduces complexity, but also enables more sophisticated automation within the Management System.

- **Consistent API and functions across all layers, strict layering.** Management systems of RINA networks never need to worry about cross-layer effects, since strict layering is enforced by design. Interactions between layers are well understood and easy to model as layers of IPC service over different scopes and ranges of QoS. The Management System coordinates multiple layers but does not need to micro-manage them: since each layer is autonomic the NMS can set high-level strategies that each layer will execute via the layer management functions.

- **Single management protocol and common layer object model.** Commonality is the key towards effective network management. Management systems of RINA networks use a single protocol to interact with Management Agents of the systems at different segments of the network. This management protocol, CDAP, is not even specific to Network Management (it is also used for layer management). CDAP is used to operate on the objects exposed by the RIB of Management Agents at each system. All DIFs have a common RIB model (see [i.43] for an initial specification), which further facilitates the task of managing DIFs: the only difference from managing one DIF to the other is in the object models of individual policies.

- **Each layer has a bounded and well-defined set of programmability points.** Separation of mechanism and policy (immutable and mutable infrastructure) defines the set of network functions that can be programmed within each layer, as well as the abstract contract such policies need to honour (what are the input parameters, what is the expected behaviour, what are the output parameters if any). Thus, a layer is as flexible and programmable as necessary, but not more: the programmability offered by the common infrastructure makes sure network designers do not break the RINA architectural principles.

11 Deployment considerations

11.1 General principles

11.1.1 Supporting applications

The vast majority of distributed applications today use the Sockets API to request communication services, either directly or via some higher-level software framework that internally uses Sockets. The Sockets API does not abstract the communication service provided from its implementation very well: network addresses are exposed and the application developer has to explicitly choose the transport protocol that will provide the communication service. This fact makes the introduction of new transport technologies very challenging, since it is not transparent for the application developer [i.6].

Hence, a number of strategies are possible to enable legacy application support:

a) modifying the application code that interacts with the network so that it talks to the IPC service API (or even to another network API that is truly technology agnostic);

b) writing a Sockets emulator that maps the calls to the Sockets API to calls to the IPC service API;

c) modifying application middleware/libraries that work over the Sockets API providing a higher abstraction to applications (this way multiple applications can be supported by just modifying a single library);

d) writing a RINA module for a Transport Services (IETF TAPS) API implementation [i.6], since TAPS provides an API that aims to be transport protocol agnostic.

ARCFIRE's deliverable D3.1 [i.44] describes a design and implementation of an IPC service API for RINA that is very close to the file API defined in the POSIX standard. Flows are modelled as file descriptors that can be allocated, read, written and closed by applications. The server and client-side workflows are very similar to those of the sockets API (as illustrated by Figure 32, thus minimizing the learning curve for application developers familiar with the Sockets API. ARCFIRE researchers have ported two applications from sockets to this POSIX-like IPC service API: the Ngnix web server and the Dropbear SSH server [i.45]. The porting of each application involved rewriting about 300 - 400 lines of code, which is a relatively small effort.

Porting applications to the IPC service API has the advantage that such applications can leverage the features of a generic IPC network architecture (application naming, protocol independence, capability to request QoS options for the communication service). However, the cost of modifying an application to deploy it over a RINA network may be too high in many cases. A sockets emulator is the best option for those scenarios. This piece of software replaces the implementations of the Sockets operations in the standard C library by calls to an IPC service API (such as the RINA POSIX-like API). An initial implementation of a Sockets emulator has been started in [i.46]. A general problem that a sockets emulator needs to solve is how to map information passed to the Sockets API to an *application name*. If the application uses domain names, then it can be directly mapped to an *application name*, but applications that directly pass IP addresses may be more problematic.

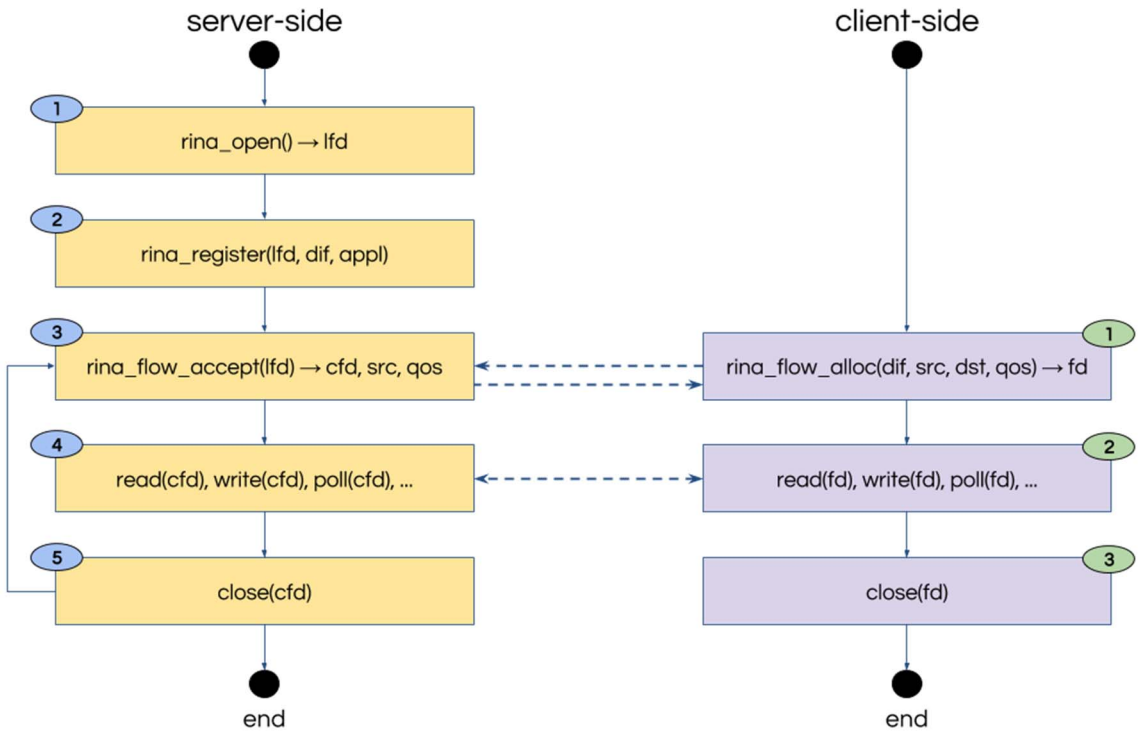


Figure 32: POSIX-like IPC service API designed by the ARCFIRE project

11.1.2 Overlays: shim DIFs

DIFs can be deployed over existing layers of protocols as an overlay. The adaptation of an existing protocol to a DIF is carried out via a shim DIF. The task of a shim DIF is to put as small as possible a veneer over a legacy protocol to allow a DIF to use it unchanged. In other words, this shim DIF allows the DIF above built using the IPC service API to operate over Ethernet without change. The shim DIF wraps the underlying protocol layer with the IPC service interface. The goal is not to make legacy protocols provide full support for the DIF and so the shim DIF should provide no more service or capability than the legacy protocol provides.

The IRATI project defined shim DIFs to overlay RINA on top of Ethernet (with and without VLANs) [i.47], TCP and UDP [i.48] and shared memory for VM-Hypervisor communication [i.49]. ARCFIRE has also specified and implemented a shim DIFs to overlay RINA on top of WiFi. The shim DIF abstraction has proven an effective tool to overlay RINA on top of existing network technologies.

11.1.3 DIFs as a multi-protocol transport (IP, Ethernet, etc.)

DIFs can also be used as an underlay technology that can efficiently transport IP, Ethernet or flows belonging to other protocols with the adequate level of service. Using this approach DIFs can be used as an alternative or a replacement to MPLS in a variety of scenarios such as datacentre network fabrics, BGP-free service provider backbones, Metropolitan Area Networks delivering Carrier Ethernet services, etc. DIFs can provide services such as point-to-point flows, IP VPNs, Ethernet VPLS instances, etc.

11.1.4 Transport layer gateways

Gateways allow the interoperability of applications supported by a network built to the generic network IPC architecture and applications running in systems that run traditional network protocols. Such scenarios are very likely especially in the early days of deployment, where a number of generic IPC network architecture islands will be operating surrounded by an ocean of IP-only systems. For example, a generic IPC network architecture enabled datacentre will need to support customers that may just support IP, without requiring the applications deployed at the datacentre to be aware about the differences.

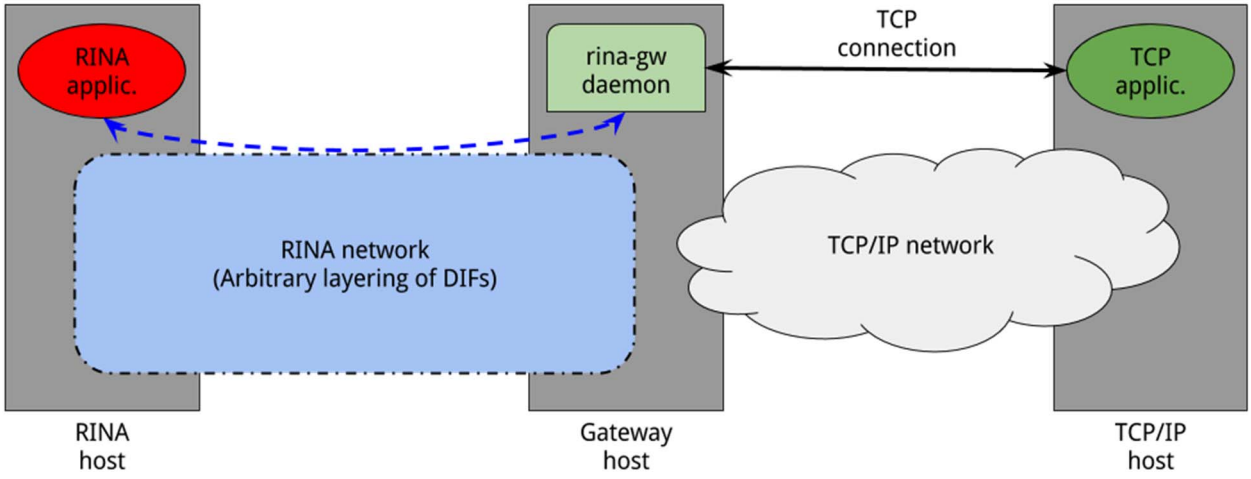


Figure 33: Example configuration of a TCP-RINA gateway

The PRISTINE and ARCFIRE projects have worked in the TCP-RINA gateway, a daemon that acts as a proxy/gateway between a TCP/IP network and a RINA network [i.50] as depicted in Figure 33. On the one side, the gateway accepts TCP connections coming from a TCP/IP network and proxies them by allocating RINA flows towards the proper server applications in the RINA network. On the other side, the gateway accepts flow allocation requests coming from the RINA network and proxies them to a TCP server by means of new TCP connections. The gateway can be easily extended to support the relaying of other transport protocols such as UDP.

Transport layer gateways come with their own drawbacks:

a) since the gateway terminates the transport connection and the RINA flow, the end-to-end semantics of the transport connection and the RINA flow are broken;

b) credentials need to be properly managed in the case that TLS (Transport Layer Security) is used over the transport connection.

However, such gateways are deployed today in production networks for other reasons (Performance Enhancing Proxies or PEPs, for example), therefore they are a viable solution to support interoperability in the appropriate use cases.

11.2 Example interoperability scenarios

11.2.1 Datacentre networking

Datacentre networks are organized to provide an efficient connectivity substrate to applications running in a large pool of compute resources (either directly in physical machines or in Virtual Machines). DCs can support a single type or multiple applications, from a single or multiple users. Some DCs are specialized to provide cloud services, assigning a subset of the DC resources to an application or set of applications, which are isolated from the rest of the DC in terms of performance and security. This type of DC is said to support multi-tenancy.

Networks of multi-tenant DCs usually follow a 3 or 5 stage Clos topology, with ToR (Top of Rack) switches/routers aggregating the traffic of the servers in a single rack, Fabric switches/routers implementing a middle layer of aggregation and Spine switches/routers implementing the backbone part of the interconnect. Figure 34 shows a typical configuration of the protocol layers involved in such configuration. Usually there is an IP fabric layer that provides connectivity between all the ToRs at the DC (depending on the size and requirements of the DC, such DC fabric may be layer 2 based). The DC fabric supports multiple virtual network overlays that provide private IP connectivity between the resources (VMs or containers) of a specific tenant. Virtual network overlays are usually implemented by a layer 2 or layer 3 virtual network solution such as VXLAN or NVGRE.

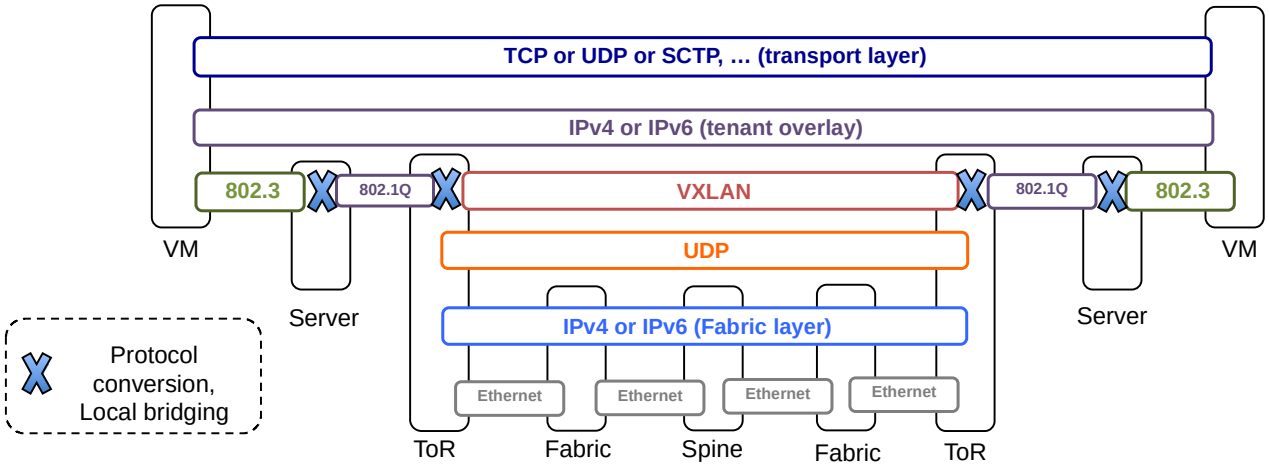


Figure 34: Protocol layers in a typical multi-tenant datacentre network

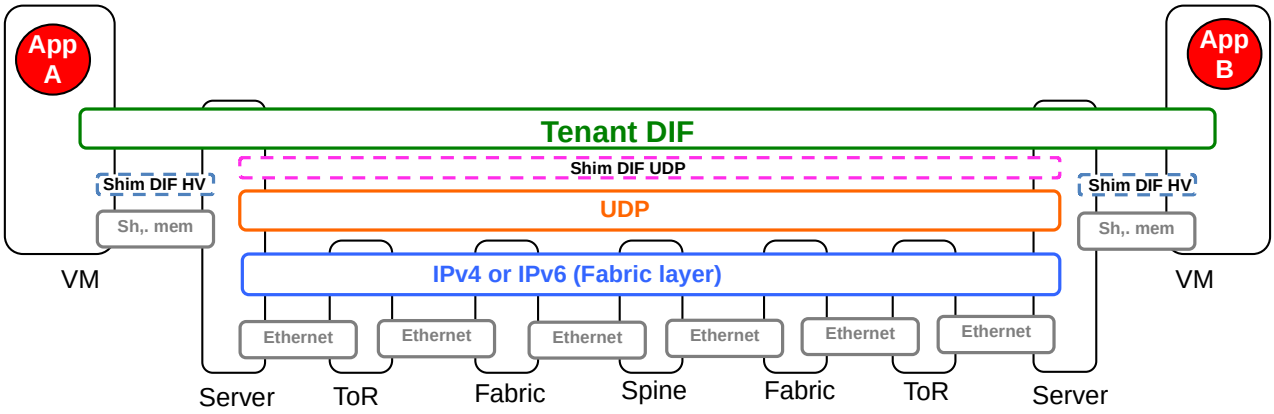


Figure 35: RINA-based virtual network overlay configuration

RINA can support the networking needs of multi-tenant DCs in a number of ways. A software-based solution only (which could be based on an optimized version of the current RINA prototypes) could carry out the work of the virtual network overlays, as illustrate in Figure 35. The connectivity between VMs of the same tenant is provided by a dedicated DIF that sits on top of shared memory (for VM to Hypervisor Host communication) [i.51] or UDP (for Hypervisor Host to Hypervisor Host communication). In this scenario RINA is only deployed at the Hypervisor and VM operating systems. Legacy applications are supported via the sockets emulator, while native RINA applications can leverage the RINA API. A RINA-TCP/UDP gateway can allow customers on the Internet (or other networks) to connect to the applications supported by the tenant DIFs. The DIF Allocator would enable customers with RINA-enabled devices to connect to the appropriate tenant DIF.

Assuming the availability of high-performance RINA implementations for router platforms (based on FPGAs, ASICs or similar), RINA can also replace the IP-based datacentre fabric with another DIF. A RINA-based datacentre fabric would facilitate a better congestion control across the DC [i.52] reduce the forwarding table size of the fabric routers [i.53] and simplify datacentre network management [i.41].

11.2.2 Communication/Internet Service Provider

BGP-free service provider cores are a design pattern followed by some Internet Service Providers to hide their core router from the rest of the Internet. Core routers support the multiplexing and transport of multiple IP (or even Ethernet) VPN services between border routers, as well as flows that support Internet service. This design is featured in Figure 36 and usually implemented via MPLS technology.

Provider Edge (PE) routers implement logical, isolated instances of IP routing and forwarding engines per VPN, called VRF (Virtual Routing and Forwarding). Each VRF exchanges routes with other VRFs belonging to the same VPN, populating the local VPN IP routing table associated to the VRF. VRFs encapsulate traffic going towards the core (P) routers with two MPLS labels: one representing the VPN (inner) and another one (outer) for the pseudo wire to the next hop P router. P routers forward the traffic based on the outer MPLS label until it reaches the destination PE, which processes the inner label to deliver the traffic to the right VRF instance.

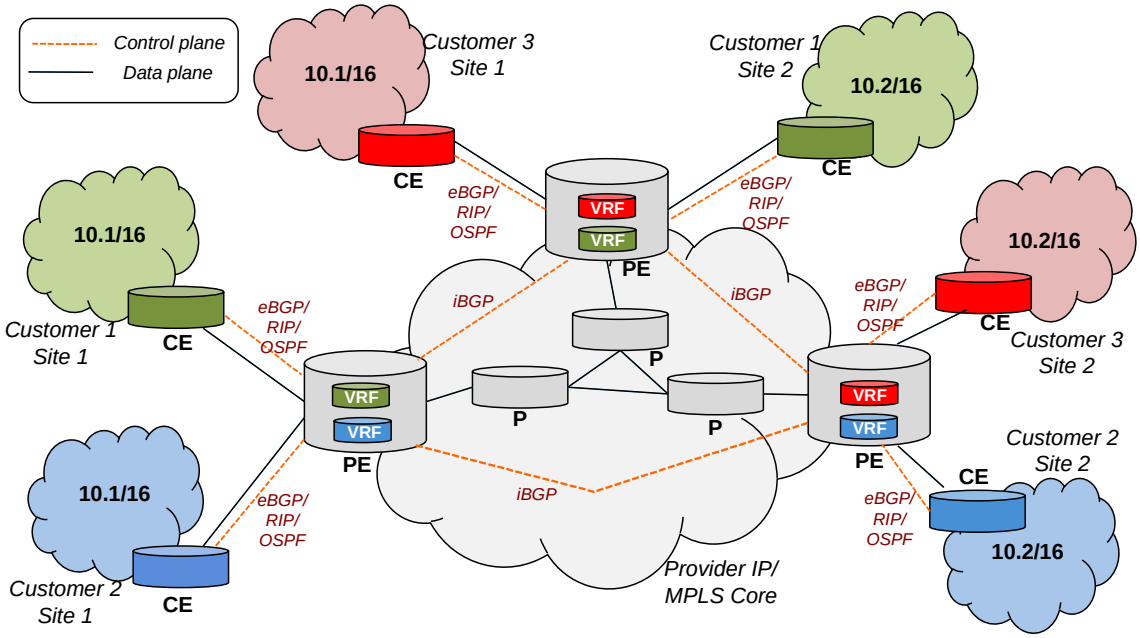


Figure 36: IP VPNs over a BGP-free service provider core

RINA can be deployed in this use case as an alternative to MPLS, replacing MPLS routers by RINA routers at the core network as seen in Figure 36. VRF instances are instances of application processes that register at the *orange DIF*. The application name of VRF instances should facilitate the discovery of all the other VRFs that are part of the same VPN. For example, the Figure uses a naming convention of *VPN id* as the process name, *router id* as the process instance and a label to indicate the nature of the VPN (e.g. *RINAIP*) as the entity name (because the same DIF can transport VPNs of other protocols). Once VRF instances are created, the Network Management System requests the creation of point-to-point flows between VRF instances belonging to the same VPN. Some of the flows can be dedicated to communicate control information (e.g. exchange of routes via iBGP or other means), while other flows will carry the VPN data traffic. For example, all VRF instances could allocate a *control* flow to an application acting as a BGP route reflector, and then allocate *data flows* to all the other VRF instances in the VPN. Such flows would appear as output interfaces for the VRF, being able to forward IP traffic through them. ARCFIRE has carried out a prototype implementation of the RINA-enabled VRF application, called the *iporina daemon* [i.54]. The daemon behaves as explained in this paragraph, with the exception that it does not use BGP to announce routes (it uses a simpler mechanism for demo purposes).

Very large core service provider networks will benefit from partitioning the backbone in multiple layers to facilitate scaling, which can be easily done by adding another DIF below the orange DIF in Figure 37. This scenario is currently under investigation as part of ARCFIRE' s experiment 3 [i.55].

The communication service provider can peer with other CSPs/ISPs at the IP level to exchange Internet routes or inter-domain VPN routes. However, it is also possible to operate common DIFs with other service providers that implement RINA. Those DIFs can also be supported by the RINA-enabled service provider backbone, in parallel to the IP VPNs and other services. This way, a RINA-enabled CSP can keep RINA completely hidden within its core network when it peers with IP-based ISPs, but it can also expose RINA capabilities to other RINA-capable CSPs. The same can be done for customers: those that support IP only will connect to the PE routers via IP; but those customers that are RINA-enabled can be offered a RINA VPN instead of an IP VPN.

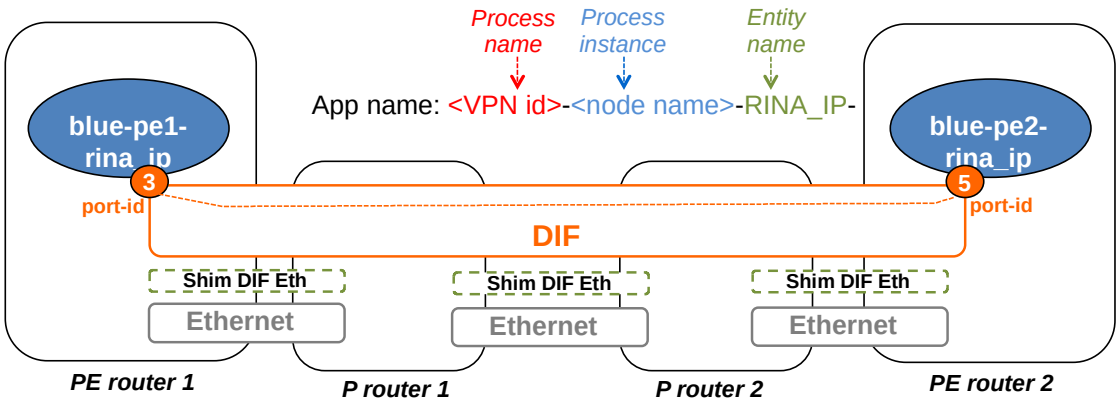


Figure 37: Backbone DIF providing a flow between two VRF instances

11.2.3 Software-Defined WAN (SD-WAN)

Software-Defined Wide Area Networks (SD-WAN)s are emerging as an alternative, cheaper solution to dedicated circuits in the WAN connectivity segment. Use cases such as branch office connectivity to central offices or secure cloud access have been the initial target of this technology. SD-WAN solutions are usually deployed between customer and provider edges (for customer to cloud) or between branch and central office edges. SD-WANs are able to exploit and optimize multiple underlying connectivity services available at customer edges (public Internet, dedicated circuit, wireless access), usually routing flows belonging to a variety of applications through different media depending on customer-defined policies. Management of SD-WAN edge routers is typically carried out from a centralized location, which may be a private datacentre that belongs to the SD-WAN owner or a public cloud in case of a hosted solution. This scenario is shown in Figure 38.

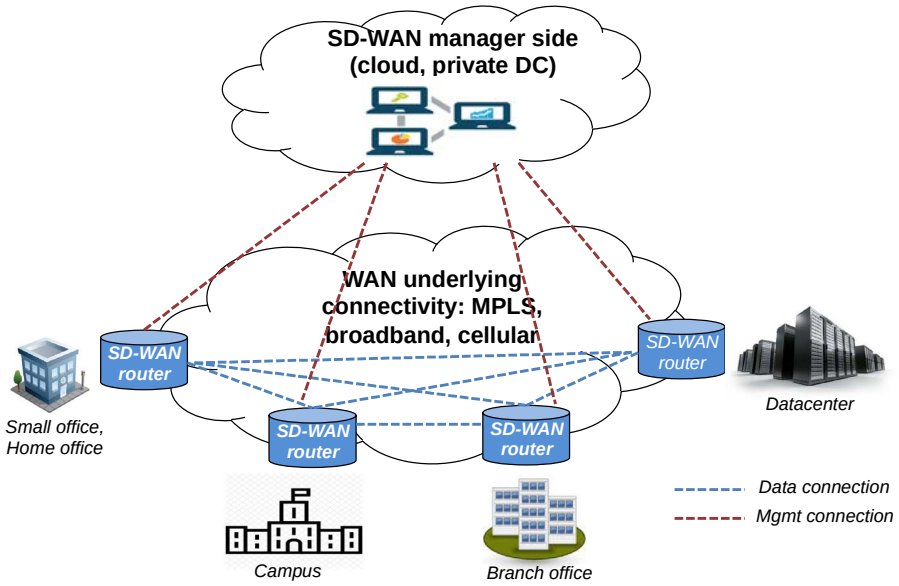


Figure 38: Typical SD-WAN scenario

The SD-WAN software router is usually configured to identify different IP flows (based on the 5-tuple), classify them into different classes, optionally apply some kind of security mechanism and then forward them through one of the underlying interfaces (e.g. broadband or MPLS circuit). The operation of the SD-WAN router may involve performing Network Address Translation (NAT), rewriting the IP header to add extra information or encrypt some fields, etc. RINA can provide a cleaner solution to the SD-WAN problem by leveraging DIFs and their properties. Figure 39 shows a simple scenario, in which a single DIF is used to deliver an SD-WAN service across multiple sites.

Figure 39 shows an example of a RINA-based SD-WAN router. An SD-WAN DIF sits on top the different technologies that provide WAN connectivity to the customer side: in this example, an xDSL broadband connection and an MPLS circuit. The DIF allocates flows to the N-1 DIF over each one of the technologies to all the other SD-WAN routers that are nearest neighbours (the discovery procedure can be autonomous or orchestrated via the Management System). Then, the Network Management system created different flows between the IP protocol machines of costumer IP layers (in red) at each SD-WAN router. These multiple flows provide different QoS options to the IP flows. The IP protocol machines classify the IP packets according to the flows they belong to (specified via the Network Management System), and then forward them through one of the flows provided by the SD-WAN DIF. The DIF will forward the packets to the next hop according to the QoS advertised by the flow, therefore some customer IP flows will be forwarded though the MPLS circuit and other through the broadband link. The DIF can also define different protection strategies depending on the characteristics of the N-1 flow, and may therefore encrypt the data that is forwarded through the broadband link.

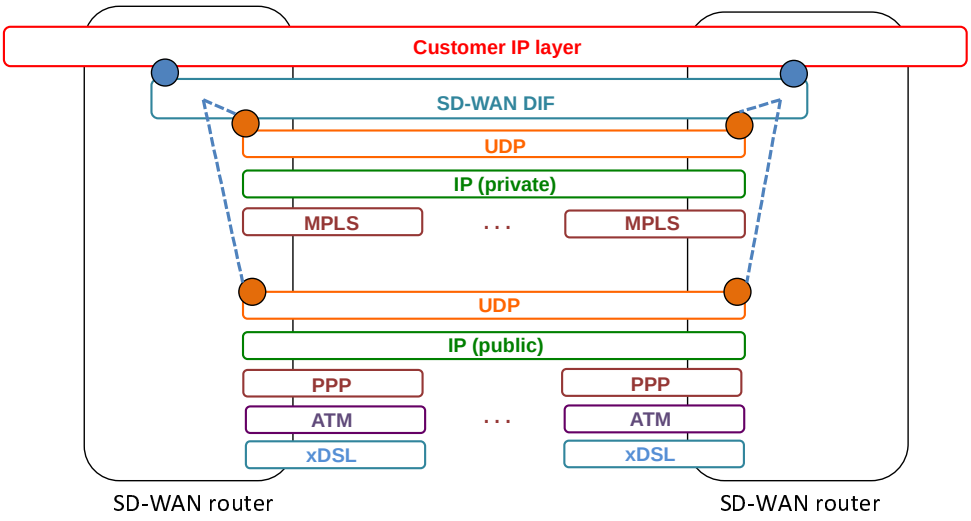


Figure 39: RINA-based SD-WAN router

Annex A:
Authors & contributors

The following people have contributed to the present document:

Rapporteur:

Dr. Eduard Grasa, Fundació i2cat, eduard.grasa@i2cat.net

Other contributors:

Dr John Day, Boston University, jeanjour@comcast.net

Miquel Tarzan, Fundació i2CAT, miquel.tarzan@i2cat.net

Dr. Diego López, Telefónica, diego.r.lopez@telefonica.com

Kevin Smith, Vodafone, kevin.smith@vodafone.com

Annex B:
Change History

Date	Version	Information about changes
May 2017	0.0.1	Table of contents
November 2017	0.0.1	Early draft, added content to clauses 5.3, 7 and 10
August 2018	0.0.2	Complete draft of clauses 4 and 5, fixed typos and some figures in other clauses
September 2018	0.0.3	Changed from GS to GR, adapted title and scope
September 2018	0.0.4	Added clause 6 draft
September 2018	0.0.5	Added clause 8 and 9 draft
September 2018	0.0.6	Stable draft
November 2018	0.0.7	Stable draft, after editHelp cleanup
November 2018	0.0.8	Integrated Vodafone's contributions
December 2018	1.0	Integrated feedback from #NGP13 meeting

History

Document history		
V1.1.1	February 2019	Publication