Deliverable-2.2

PRISTINE Reference Framework

Deliverable Editor: Eduard Grasa, Fundacio i2CAT

Publication date:	30-June-2014
Deliverable Nature:	Report
Dissemination level (Confidentiality):	PU (Public)
Project acronym:	PRISTINE
Project full title:	Programmability In RINA for European supremacy of virTualised NEtworks
Website:	www.ict-pristine.eu
Keywords:	RINA reference model, specifications, policies, reference architecture
Synopsis:	D2.2 is divided into i) a detailed report on the state of the art of the current RINA specifications relevant to the technological areas addressed by PRISTINE, identifying its limitations and categorizing its policies; and ii) a detailed overview of each one of PRISTINE's research areas, identifying the policies being targeted.

The research leading to these results has received funding from the European Community's Seventh Framework Programme for research, technological development and demonstration under Grant Agreement No. 619305.

Disclaimer

Executive Summary

D22 reports about the work carried out in T2.2. This task has performed a review of the RINA specifications that are relevant to the technological areas addressed in PRISTINE (congestion control, routing, resource allocation, authentication, access control, encryption, security coordination, configuration management and performance management) in order to spot any limitation towards the project's objectives. Successively, it has defined and categorized the variable behaviors in each RINA component involved in the project, for the various research areas taken into consideration, in order to identify the policies that are likely to be in the scope of PRISTINE work. The RINA specifications and reference model documents are not open since they are not yet ready for public release, but access to the documents can be granted upon request on a case by case basis.

The second part of the deliverable discusses the different research areas that PRISTINE is targeting:

- Congestion avoidance/control (WP3, T3.1).
- Distributed resource allocation (WP3, T3.2).
- Addressing and routing (WP3, T3.3).
- Authentication, authorization and confidentiality (WP4, T4.1).
- Security coordination within a DIF (WP4, T4.2).
- Reliability and high availability (WP4, T4.3).
- Network Management (WP5).

For each research area an overview of the problem space from the perspective of the RINA architecture is presented, followed by an initial discussion of the ideas that PRISTINE will work on and the identification of the relevant IPC Process components and policies that will be affected by the project's research and development activities.

Table of Contents

Glossary

1. List of definitions

AP or DAP

Application Process or (Distributed Application Process). The instantiation of a program executing in a processing system intended to accomplish some purpose. An Application Process contains one or more tasks or Application-Entities, as well as functions for managing the resources (processor, storage, and IPC) allocated to this AP.

CACEP

Common Application Connection Establishment Phase. CACEP provides the means to establish an application connection between DAPs, allowing them to agree on all the required schemes and conventions to be able to exchange information, optionally authenticating each other.

CDAP

Common Distributed Application Protocol. CDAP enables distributed applications to deal with communications at an object level, rather than forcing applications to explicitly deal with serialization and input/output operations. CDAP provides the 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. CDAP provides a straightforward and unifying approach to sharing data over a network without having to create specialized protocols.

CEP-id

Connection-endpoint id. A Data Transfer AE-Instance-Identifier unique within the Data Transfer AE where it is generated. This is combined with the destination's CEP-id and the QoS-id to form the connection-id.

DAF

Distributed Application Facility. A collection of two or more cooperating DAPs in one or more processing systems, which exchange information using IPC and maintain shared state. In some Distributed Applications, all members will be the same, i.e. a homogeneous DAF, or may be different, a heterogeneous DAF.

DFT

Directory Forwarding Table. Sometimes referred to as search rules. Maintains a set of entries that map application naming information to IPC process addresses. The returned IPC process address is the address of where to look for the requested

application. If the returned address is the address of this IPC Process, then the requested application is here; otherwise, the search continues. In other words, either this is the IPC process through which the application process is reachable, or may be the next IPC process in the chain to forward the request. The Directory Forwarding table should always return at least a default IPC process address to continue looking for the application process, even if there are no entries for a particular application process naming information.

DIF

Distributed IPC Facility. A collection of two or more Application Processes cooperating to provide Interprocess Communication (IPC). 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.

DTCP

Data Transfer Control Protocol. The optional part of data transfer that provide the loosely-bound mechanisms. 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.

DTP

Data Transfer Protocol. The required Data Transfer Protocol consisting of tightly bound mechanisms found in all DIFs, roughly equivalent to IP and UDP. 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.

DTSV

Data Transfer State Vector. The DTSV (sometimes called the transmission control block) provides shared state information for the flow and is maintained by the DTP and the DTCP.

EFCP

Error and Flow Control Protocol. The data transfer protocol required to maintain an instance of IPC within a DIF. The functions of this protocol ensure reliability, order, and flow control as required. It consists of a separate instances of DTP and optionally DTCP, which coordinate through a state vector.

FA

Flow Allocator. The component of the IPC Process that responds to Allocation Requests from Application Processes.

FAI

Flow Allocator Instance. An instance of a 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) selects the policies to be used on the flow, 4) monitors the flow, and 5) manages the flow for its duration.

PCI

Protocol Control Information. The string of octets in a PDU that is understood by the protocol machine which interprets and processes the octets. These are usually the leading bits and sometimes leading and trailing bits.

PDU

Protocol Data Unit. The string of octets exchanged among the Protocol Machines (PM). PDUs contain two parts: the PCI, which is understood and interpreted by the DIF, and User-Data, that is incomprehensible to this PM and is passed to its user.

RA

Resource Allocator. A 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.

RIB

Resource Information Base. For the DAF, the RIB is the logical representation of the local repository of the objects. 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. From the point of view of the OS model, this is storage.

RMT

Relaying and Multiplexing Task. This task is an element of the data transfer function of a DIF. Logically, it sits between the EFCP and SDU Protection. RMT performs the real time scheduling of sending PDUs on the appropriate (N-1)-ports of the (N-1)-DIFs available to the RMT.

SDU

Service Data Unit. The unit of data passed across the (N)-DIF interface to be transferred to the destination application process. 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.

2. List of acronyms

ABAC

Attribute-Based Access Control.

ACC

Aggregate Congestion Control.

ACL
Access Control List.

AE
Application Entity.

AP
Application Process.

API
Application Programming Interface.

ASN.1
Abstract Syntax Notation One.

CACEP
Common Application Connection Establishment Phase.

CCA
Congestion Control Aggregate.

CDAP
Common Distributed Application Protocol.

CMIP
Common Management Information Protocol.

CRC
Cyclic Redundancy Code.

DAF
Distributed Application Facility.

DAP
Distributed Application Process.

DNS
Domain Name Server.

DHCP
Dynamic Host Configuration Protocol.

DHT
Distributed Hash Table.

DFT
Directory Forwarding Table.

DIF
Distributed IPC Facility.

DRF
Data Run Flag.

DTAE
Data Transfer Application Entity.

DTCP
Data Transfer Control Protocol.

DTP
Data Transfer Protocol.

DTSV
Data Transfer State Vector.

ECN
Explicit Congestion Notification.

EFCP
Error and Flow Control Protocol.

FA
Flow Allocator.

FAI
Flow Allocator Instance.

FST
Fisheye State Routing.

GPB
Google Protocol Buffers.

HIDS
Host Intrusion Detection System.

HTTP
Hyper Text Transfer Protocol.

IPC
Inter Process Communication.

IRM
IPC Resource Manager.

IS-IS
Intermediate System to Intermediate System.

JSON
Java Script Object Notation.

KMIP
Key Management Interoperability Protocol.

MA
Management Agent.

MANET
Mobile Ad-hoc NETwork.

MLS
Multi Level Security.

MPL
Maximum Packet(PDU) Lifetime.

MPLS
Multi-Protocol Label Switching.

MTBR
Mean Time Between Failures.

MTTR
Mean Time To Recover.

NM-DMS
Network Management Distributed Management System.

NSM
Name Space Manager.

NIDS
Network Intrusion Detection System.

OSPF-TE
Open Shortest Path First with Traffic Engineering.

PCI
Protocol Control Information.

PDU
Protocol Data Unit.

PKI
Public Key Infrastructure.

PM
Protocol Machine.

QoS
Quality of Service.

RA
Resource Allocator.

RBAC
Role-Based Access Control.

RIB
Resource Information Base.

RINA
Recursive InterNetwork Architecture.

RMT
Relaying and Multiplexing Task.

RSVP-TE
Resource Reservation Protocol with Traffic Engineering.

RTT
Round Trip Time.

SDU
Service Data Unit.

TCP
Transmission Control Protocol.

TLS
Transport Layer Security.

TTL
Time to Live.

UDP
User Datagram Protocol.

VLAN
Virtual Local Area Network.

WFQ
Weighted Fair Queuing.

XML
eXtensible Markup Language.

1. Overview of the RINA specifications

Figure 1 illustrates the components of the RINA architecture at the macro scale (IPC Processes forming DIFs, that provide IPC services to distributed application processes), and at the micro-scale (common components of an application process and of an IPC Process). The IPC Process components are naturally organized in i) groups of functions that are simple, but performed more often (data transfer); ii) groups of functions that are more complex, but performed less often (data transfer control) and iii) groups of functions that are the most complex, but also performed less frequently (layer management).

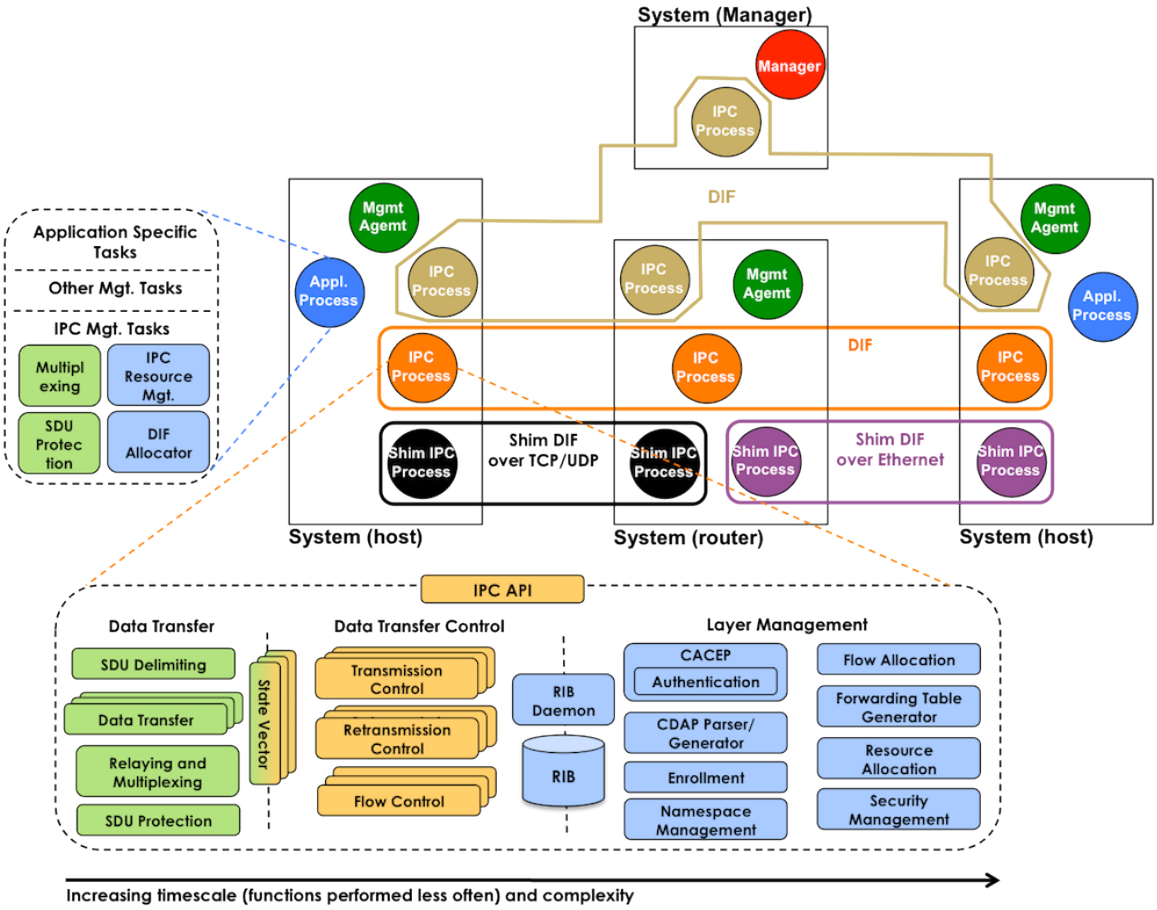


Figure 1. RINA architecture reference model and components

1.1. Application Process - IPC Management components

The IPC Management component of the application process manages the use of underlying DIFs. It consists of SDU Protection, the Multiplexing Task, the DIF Allocator, and the IPC Resource Manager (IRM).

1.1.1. Multiplexing Task

The Multiplexing Task performs the real time management of the use of DIFs by the application process. While traditional multiplexing is expected to be rare (requiring sufficient machinery to make the distributed application a DIF), it may occur. In this case, flow control such as it is will be exerted on the tasks using IPC by blocking. It is more likely that this task will support “inverse multiplexing” or combining of multiple DIF flows of different QoS into a single flow for the application.

1.1.2. SDU Protection

This module provides any required protection for the Service Data Units (SDUs) this distributed application may pass to an IPC facility, including: bit error detection and correction, enforcement of maximum SDU lifetime, encryption and compression.

1.1.3. IPC Resource Manager (IRM)

This component manages the IPC components. It moderates participation in DIF creation should that arise. The IRM is involved when the requested application is supported by another DIF. It will spawn a new IPC Process with appropriate initial policies with instructions to use an existing (N-1)-DIF to communicate with another IPC process to either join an existing DIF or to create a new one.

1.1.4. DIF Allocator (DA)

The DIF Allocator arises naturally from the property that at the IPC boundary, an Application merely requests that resources be allocated with the Destination by name. The Requesting Application does not need to know where the Requested Destination Application is. When an Application submits an Allocate Request to the IRM with an Application-Process-Name, the IRM determines if the application is reachable on any of the DIFs available to this application and if so, one is selected and the Allocate Request is processed normally.

But if the desired application is not on one of the available DIFs, it is perfectly reasonable that this function query its peer IRMs (IRMs of the application processes that have an N-1 DIF in common with the source application process), which may have different DIFs with other applications on them. If it is found, then a new DIF can be created to enable the communication. This function is called the DIF-Allocator. It may span a set of DIFs and has two major functions [trouva]:

- to find the DIF that supports the requested DIFs or DAFs, that is not available to this IRM, and
- to make available a DIF by either joining existing DIFs or creating new DIFs such that both the requested and requesting Applications can allocate IPC. This may require the DIF Allocator to moderate (or negotiate) the creation of DIFs.

1.2. IPC Process Components

1.2.1. IPC Service API

This is the only externally visible API for application processes using the IPC Process services. This API allows applications to make themselves available through a DIF and to request and use IPC services to other applications. The abstract API has six operations (implementations may have more operations for convenience of use and to adapt to the specifics of each operating system, but still logically providing the same operations):

- **portId _allocateFlow(destAppName, List<qosParams>)**. This operation enables an application to allocate a flow to a destination application (identified by destAppName), specifying a list of desired QoS parameters. The operation returns a handle to the flow, the portId, used in other operations to read/write SDUs (Service Data Units, the user data) from/to the flow.
- **void write(portId, sdu). Sends an SDU through the flow identified by _portId**. SDUs are buffers of user data with a certain length. SDUs are delivered to the destination application as they where written by the source application.
- **sdu read(portId). Read an SDU from the flow identified by _portId**.
- **void deallocate(portId). Release all the resources used by the flow identified by _portId**.
- **void _registerApplication(appName, List<DIFName>)**. Register the application identified by appName to the DIFs identified in the list of DIF names. This operation advertises the application within a DIF, so that flows can be allocated to it (it will be always up to the application to take the final decision refusing or accepting them).
- **void _unregisterApplication(appName, List<DIFName>)**. Unregister an application from a set of DIFs or all the DIFs (if the second argument is not present).

1.2.2. Delimiting

The first step in the user-data processing path is to delimit the SDUs posted by the application; since the data transfer protocol may implement concatenation and/or fragmentation of the SDUs in order to achieve a better data transport efficiency and/or to better adapt to the DIF characteristics.

1.2.3. The Error and Flow Control Protocol (EFCP)

The Error and Flow Control Protocol (EFCP) is split into two parts: the data transfer protocol (DTP) and the Data Transfer Control Protocol (DTCP), loosely coupled through the use of a state vector. DTP performs the mechanisms that are tightly coupled to the transported SDU, such as fragmentation, reassembly, sequencing or addressing.

DTCP performs the mechanisms that are loosely coupled to the transported SDU, such as transmission control, retransmission control and flow control. When a flow is allocated an instance of DTP and its associated state vector are created. The flows that require flow control, transmission control or retransmission control will have a companion DTCP instance allocated. The string of octets exchanged between two protocol machines is referred to as Protocol Data Unit (PDU). PDUs comprise of two parts, Protocol Control Information (PCI) and user data. PCI is the part understood by the DIF, while the user data is incomprehensible to the DIF and is passed to its user. The PDUs generated by EFCP are passed to the relaying and multiplexing components. RINA's EFCP is designed based on delta-t, designed by Richard Watson in 1981 [9]. Watson proved that the necessary and sufficient conditions for reliable synchronization is to bound 3 timers: Maximum Packet Lifetime (MPL), Maximum time to acknowledge and Maximum time to keep retransmitting. In other words: SYNs and FINs in TCP are unnecessary, allowing for a simpler and more secure data transfer protocol.

1.2.4. Relaying and Multiplexing Task (RMT)

The role of the Relaying task is to forward the PDUs passing through the IPC Process to the destination EFCP Protocol Machine (PM) by checking the destination address in the PCI. The decision on forwarding is based on the routing information and the Quality of Service agreed. The Multiplexing task multiplexes PDUs from different EFCP instances onto the points of attachment of lower ranking (N-1) DIFs. There are several policies that decide when and where the PDU are forwarded (management of queues, scheduling, length of queues). These policies affect the delivered Quality of Service.

1.2.5. SDU Protection

SDU Protection includes all the checks necessary to determine whether or not a PDU should be processed further (for incoming PDUs) or to protect the contents of the PDU while in transit to another IPC Process that is a member of the DIF (for outgoing PDUs). It may include but is not limited to checksums, CRCs, encryption, Hop Count/Time To Live mechanisms. The SDU Protection mechanisms to be applied may change hop by hop (since they depend on the characteristics of the underlying DIFs). In RINA, Deep Packet Inspection is unnecessary and often impossible.

1.2.6. The Resource Information Base (RIB)

The Resource Information Base (RIB) is the logical representation of the objects that capture the information that define an application state. It can be viewed logically as a partially replicated distributed database. All state information maintained by the IPC Tasks, the Flow Allocator, Resource Allocator, etc. is maintained by and available through the RIB Daemon. This includes all local information on the operational state of the DIF, performance, load, routing update, directory caches, etc.

1.2.7. The RIB Daemon

The RIB Daemon is the task that controls the access to the RIB, and also optimizes the operations on the RIB performed by other components of the IPC Processes. Besides making this information available to the tasks of the IPC Process, it is also the task of the RIB Daemon to efficiently update this information with other members of the DIF and the Network Management System periodically or on certain important events. The information exchanged is necessary to coordinate the distributed IPC.

1.2.8. The Common Application Connection Establishment Phase (CACEP)

CACEP allows two Application Processes to establish an application connection. During the application connection establishment phase, the APs exchange naming information, optionally authenticate each other, and agree on the abstract and concrete syntaxes of CDAP to be used in the connection, as well as on the version of the RIBs. It is also possible to use CACEP connection establishment with another protocol in the data transfer phase (for example, HTTP).

1.2.9. The Common Distributed Application Protocol (CDAP)

CDAP enables distributed applications to deal with communications at an object level, rather than forcing applications to explicitly deal with serialization and input/ output operations. CDAP provides the 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. CDAP provides a straightforward and unifying approach to sharing data over a network without having to create specialized protocols.

1.2.10. The Enrollment Task

All communication goes through three phases: Enrollment, Allocation (Establishment), and Data Transfer. RINA is no exception. Enrollment is the procedure by which an IPC Process joins an existing DIF and obtains enough information to start operating as a member of this DIF. Enrollment starts when the joining IPC Process establishes an application connection with another IPC Process that is already a member of the DIF. During the application connection establishment, the IPC Process that is a DIF member may want to authenticate the joining process, depending on the DIF security requirements. The CACE component (Common Application Connection Establishment) is the one in charge of establishing and releasing application connections. Several authentication modules can be plugged into CACEP, to implement different authentication policies. Once the application connection has been established, the joining IPC Process needs to acquire the DIF static information: what QoS classes are supported and what are its characteristics, what are the policies that the DIF supports, and other parameters such as the DIF's MPL or maximum PDU size.

1.2.11. The Flow Allocator (FA)

Flow allocation is the component responsible for managing a flow's lifecycle: allocation, monitoring and deallocation. Unlike with TCP, RINA port allocation and data transfer are separate functions, meaning that a single flow can be supported by one or more data transport connections (in TCP a port number is mapped to one and only one TCP connection because the port numbers identify the TCP connection). The Flow Allocator (FA) component handles the flow allocation/deallocation requests. Among its tasks it has to: i) find the IPC Process through which the destination application is accessible; ii) map the requested QoS to policies that will be associated with the flow, iii) negotiate the flow allocation with the destination IPC Process FA (access control permissions, policies associated with the flow), iv) create one or more DTP and

optionally DTCP instances to support the flow, v) monitor the DTP/DTCP instances to ensure the requested QoS is maintained during the flow lifetime, and take specific actions to correct any misbehaviours and vi) deallocate the resources associated to the flow once the flow is terminated.

1.2.12. The NameSpace Manager (NSM)

The NSM is a Namespace Management DAF embedded within a DIF. It is the responsible for assigning synonyms (addresses) to the IPC Processes in a DIF, and to resolve registered application names to the addresses of the IPC Processes where these applications are executing. In order to carry out the second task, the NSM has to keep track of application registrations and maintain information in the RIB - including the forwarding tables that allow the name resolution to happen in a distributed way through the DIF.

1.2.13. Routing

Routing is the IPC Process component that exchanges connectivity and other state information with other IPC processes of the DIF and applies an algorithm to generate the forwarding table used by the Relaying and Multiplexing Task (connectivity as well as QoS and resource allocation information is used to generate the forwarding table). The algorithms and information required to generate the forwarding table may be multiple, depending on the QoS classes supported by the DIF.

1.2.14. The Resource Allocator (RA)

The Resource Allocator is the component that decides how the resources in the IPC Process are allocated (dimensioning of the queues, creation/suspension/deletion of queues, creation/deletion of N-1 flows, and others).

1.2.15. Security Management

This component coordinates the three main security functions of an IPC Process:

- Authentication to ensure that an IPC-Process wishing to join the DIF is who it says it is and is an allowable member of the DIF (Enrollment);
- Confidentiality and integrity of all PDUs; and
- Access control to determine whether application processes requesting an IPC flow with a remote application has the necessary permissions to establish communication.

Security management can also collaborate with the Network Management System to perform other security-management related tasks such as credential management or intrusion detection and prevention functions.

1.3. Shim IPC Processes

Shim IPC Processes are used to transition from DIFs to a legacy technology or a physical medium. Shim DIFs wrap the legacy technology or physical medium with the RINA API, in order to allow DIFs on top to operate seamlessly. The goal of the shim DIFs is not to improve the characteristics of the wrapped environment; this has to be achieved by using a full-fledged DIF on top of the shim DIF. There are currently a number of shim DIFs under development.

1.3.1. Shim IPC Process over UDP/TCP

This IPC Process wraps a TCP/UDP layer and presents it with the IPC API, enabling the creation of DIFs as network overlays on top of existing IP infrastructures.

1.3.2. Shim IPC Process over 802.1q

This IPC Process wraps an Ethernet segment and presents it with the IPC API, allowing "normal" IPC Processes to be overlaid on 8021.q layers (VLANs), or over plain Ethernet (untagged frames are supported as well).

1.3.3. Shim IPC Process for Hypervisors

This IPC Process enables direct communication between a guest (Virtual Machine) and the environment that controls the guest execution (e.g. Domain 0 for Xen) using shared memory (thus avoiding the use of the networking stack and Ethernet bridging). XEN and QEMU-KVM are the currently supported hypervisors.

1.4. The Network Management - Distributed Management System (NM-DMS)

A NM-DMS will perform the traditional functions of monitor and repair, deploying new configurations, monitoring performance, etc. The DAF model can be applied to network management to represent the whole range from distributed (autonomic) to centralized (traditional). In the traditional centralized network management architecture, an NM-DMS would be a heterogeneous DAF consisting of one or more DAPs performing management functions, with other DAPs providing telemetry. The management

DAPs might be subdividing roles or tasks within network management or providing management for sub-domains and redundancy for each other. A typical DMS will have the usual tasks of event management, configuration management, fault management, resource management, etc.

1.4.1. The Management Agent (MA)

The NM-DMS DAPs in the traditional agent role (see Figure 1) function like the sensory nervous system collecting and pre-processing data. The Agent will have access to the DAF Management task of all IPC Processes (and associated DAPs) in the processing systems that are in its domain. While there is no constraint, it is likely that an NM-DAF would have one "Agent DAP" for monitoring in each processing system with a DIF or DAF in its domain. The DAF Management task of each DAF or DIF in the NM-DMS domain is a kind of "sub-agent". A Management Agent may be designed to seek out its DMS or alternate DMSs in the event of failures.

1.4.2. Manager DAPs

Manager DAPs perform network management related tasks within an administrative domain, managing a number of systems via the Management Agents. A typical Manager will have the usual tasks of event management, configuration management, fault management, resource management, performance management and security management.

2. Analysis of RINA specifications: Limitations and Policies

This section provides the results of the analysis of the RINA specifications that are relevant for the PRISTINE work. PRISTINE's analysis has focused on assessing the current state of development of each specification, identifying potential limitations and categorizing and describing the different types of policies. PRISTINE's areas of interest have been divided in three categories, each category covered in a different Work Package (WP):

- Congestion control, resource allocation, addressing and routing. Addressed by WP3.
- Security and resiliency. Addressed by WP4.
- Multi-layer network management. Addressed by WP5.

Specifically, D2.2 has analyzed the following specifications:

- **Delimiting**. Implements concatenation/reassembly and concatenation/ separation.
- **Error and Flow Control Protocol**. The error and flow control protocol. Provides transmission control, retransmission control and flow control between the endpoints of an EFCP connection.
- **Relaying and Multiplexing Task**. Multiplexes outgoing PDUs from multiple EFCP protocol machines to one or more N-1 flows, and relays incoming EFCP PDUs to the right EFCP protocol machine or N-1 flow.
- **SDU Protection**. Implements the mechanisms required to protect an SDU during its way through an N-1 DIF, including: forward error correction, time to leaves and encryption.
- **Common Application Connection Establishment Phase**. The protocol that two application processes use to establish an application connection (exchange enough information so that the two applications can start communicating to each other).
- **Common Distributed Application Protocol**. Used by Network and Layer Management related activities, provides six operations to operate the objects externally exposed by a remote process.
- **RIB Managed Object Model**. The external representation of the state of an IPC Process.

- **RIB Daemon**. The entity in charge of providing the interface to the RIB distributed programming model.
- **Flow Allocator**. Manages the lifetime of a flow, including allocation, deallocation and monitoring of the service provided to the flow users.
- **Enrollment**. Captures the exchange of information between an IPC Process that wants to join a DIF and the DIF it wants to join.
- **Resource Allocator**. Monitors and manages all the activities related to the allocation of resources in an IPC Process, exchanging information with its neighbors. It oversees the flow allocation, relaying, multiplexing and PDU Forwarding Table Generation tasks.
- **Namespace Management**. This component is in charge of keeping a mapping of the names of the applications registered to the DIF with the addresses of the IPC Processes those applications are registered at. Namespace management is also responsible for managing the allocation of addresses to the IPC Processes in a DIF.
- **Security Management**. Coordination and support of the security-related functions within an IPC Process (authentication, access control, SDU Protection, credential management).

In order to properly compare the degree of maturity of the different specifications, D22 has followed a criteria that focuses on three main indicators: i) whether it is clear that the IPC Process component being specified is a standalone component in the reference model of the IPC Process or its functions may be merged with those of other components; ii) the degree of completeness in the specification of the component and iii) the availability of implementations of that particular component. The following tables show the different grades of achievement of these criteria.

Table 1. Is it a standalone component

Mark	Description
1	Goals of the component are not clear
2	Goals of the component are clear, but may partially overlap with those of other components
3	Goals of the component are clear, do not overlap with other components, but can be partitioned in independent subsets

Mark	Description
4	Goals of the component are clear, do not overlap with other components and are related between them
5	Condition 4 is met, plus no new goals/ functions have been added to the component in the last 3 years

The first criterion measures the chances that the component being specified stays as a standalone component in the future. For this to happen:

- the goals of the component must be well understood
- the goals of the component must not overlap with the goals of other components; otherwise part or all of the functionalities of the component may be realized by another IPC Process component
- the goals of the component should be interrelated; otherwise the component may be further partitioned into subcomponents

Table 2. Degree of completeness in the specification

Mark	Description
1	Goals and high-level overview are partially described
2	Goals and high-level overview are completely described
3	The functions required to realize the component goals are partially specified, and coarse-grained policies have been identified
4	The functions required to realize the component goals are fully specified, and fine-grained policies have been identified
5	Condition 4 is met, plus there have been a number of different policy specifications that comply with the component specification

The second criterion measures the degree of completeness and consistency of the specification, ranging from a partial understanding of the component goals to a full

understanding of the component goals and a complete specification of the functions to achieve these goals.

Table 3. Availability of component implementations

Mark	Description
1	No implementations available
2	One or more partial implementations available
3	One or more complete, non-interoperable implementations available
4	Two or more complete, interoperable implementations available (with default policies)
5	Two or more complete, interoperable implementations available with a rich set of different policies (not just the default ones)

Finally the last criterion measures how robust the component specification may be, using as an indicator the availability of implementations of that component. The availability of two or more complete, interoperable implementations of the component is considered to be the highest indicator of the specification stability. Currently there are the three RINA prototypes under development (none of them is completely finished, they are at various degrees of maturity).

- **TRIA Network Systems**. C-based Linux implementation, single OS process, all in user-space. Works over TCP and UDP. More details are available at [triaprot].
- **ProtoRINA (BU)**. Java-based implementation, single OS process, all in user-space. Works over TCP and UDP. More details are available at [protorina].
- **IRATI**. C/C++ Linux-based implementation, multiple OS processes, targeting the kernel and user spaces. Works over TCP, UDP, Ethernet (plain and VLANs) and shared memory to allow inter-VM communication (XEN, KVM). This prototype - which will be open-source by the end of Q3 2014 - also integrates the experience from a former Java prototype developed by i2CAT and the TSSG. PRISTINE will be using the IRATI prototype as a basis. More information is available at [irati].

2.1. Delimiting

2.1.1. Overview of Delimiting

The goal of the delimiting module is to encode SDUs in PDUs, to allow for other options than just one complete SDU transported in each PDU. Even though the Delimiting module can be though as of all policy, the current Delimiting specification makes an effort to propose a general, customizable solution that allows for concatenation and fragmentation of SDUs, as well as partial and incomplete delivery of SDUs as explained in the following paragraph.

The current delimiting module defines a mechanism for encoding SDUs within PDUs. It is unnecessary to use this module if each PDU carries precisely one SDU. The delimiting module produces on input, one or more User-Data fields for EFCP to create PDUs; and on output, depending on QoS parameters, complete, incomplete, or partial SDUs for the user of the flow. (Partial delivery refers to whole SDUs that are delivered incrementally, while incomplete delivery refers to an SDU that may have pieces missing. This implies that partial delivery of incomplete SDUs is possible). The described mechanism covers a range of policy options, allowing fragmentation of SDUs, concatenation of SDUs, and both fragmentation and concatenation of SDUs simultaneously. It also provides additional information needed for delivery despite gaps in the SDU stream, if delivery across a gap is permitted by the QoS-cube for the flow. The delimiting policy may be different between QoS-cubes. This module could be used employing several QoS-cubes with different constraints on what flags might appear, or with different concatenation and fragmentation policies.

2.1.2. State of specification

- **Standalone component: 5**. Delimiting has a well-defined responsibility: i) in the outgoing direction, it generates zero or more user data fields from data written by the user of the flow (each user data field is the payload of a DTP PDU); ii) in the incoming direction, it parses the user data fields from DTP PDUs delivered by EFCP to the delimiting module, and extracts SDUs to be delivered to the user of the flow (these SDUs may be incomplete if allowed by the user of the flow).
- **Completeness of the specification: 3**. The specification only describes the mechanism for encoding SDUs within PDUs, but does not describe the incoming and outgoing processing steps followed by a generic Delimiting module that can provide Concatenation and Fragmentation functions. As a consequence, no policies are identified.

- **Availability of implementations: 1**. RINA prototypes don't implement concatenation nor fragmentation, and do not use the SDU delimiting scheme as proposed in this specification.

2.1.3. Major issues/Limitations

As identified in the previous section the major limitation of the current general delimiting specification is that it only focuses on the description of the mechanism for encoding SDUs into PDUs, but does not provide a description of all the steps in the incoming and outgoing processing chains of this module. An example of this processing chain is provided in the Figure below.

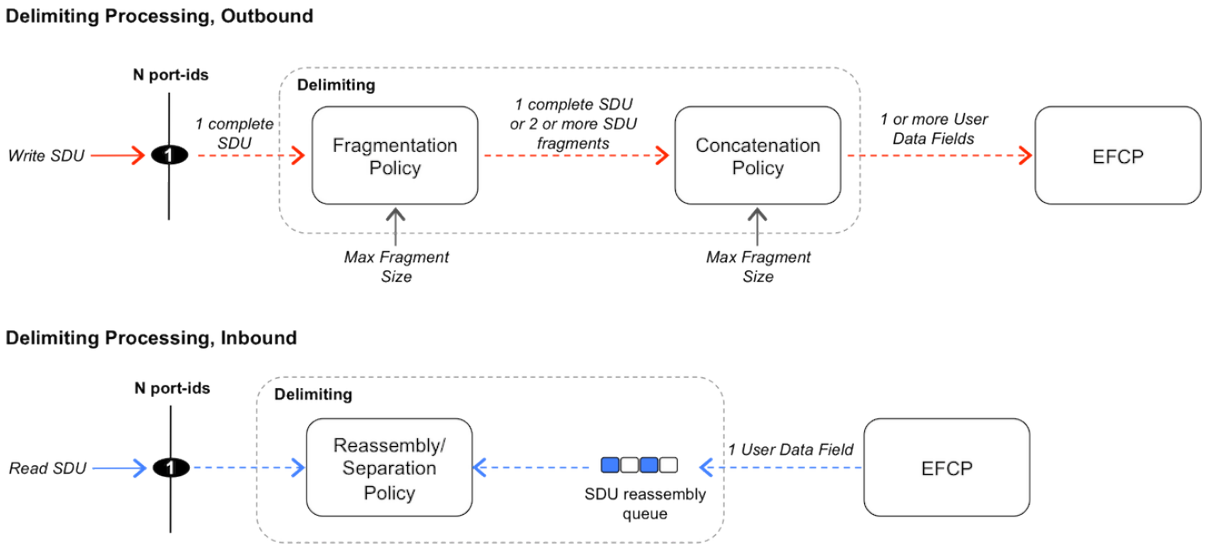


Figure 2. Example of generic delimiting module incoming and outgoing processing chains

In the outgoing direction, the user of the flow invokes the "write" API call, passing a complete SDU to the IPC Process. The delimiting module is given a complete SDU, which is handled to the *Fragmentation Policy*. If this policy is enabled, it will check if the SDU size is larger than the maximum fragment size (which is computed by subtracting the length of the DTP PCI and the length of the delimiting fields introduced by the delimiting mechanism). If so, it will partition the incoming SDU into two or more fragments; otherwise it will let the SDU pass untouched. Some variations are possible over this basic schema.

Then either a complete SDU or multiple SDU Fragments will be delivered to the concatenation policy. Concatenation typically involves queuing incoming SDUs and processing them periodically. When a timer fires the element implementing concatenation retrieves a number of complete SDUs and/or SDU fragments from the queue and assembles one or more User Data Fields that are given to the EFCP module.

Different algorithms with zero or more parameters can perform this task differently depending on the environment.

In the incoming direction there is not as much processing flexibility as in the outgoing direction, since SDUs are already fragmented and/or concatenated. Therefore all it is left to do is to order the incoming user data fields and parse them in order to retrieve complete SDUs and SDU fragments. However, since a priori there could be multiple algorithms for carrying out this procedure, it seems appropriate to define a Separation and Reassembly Policy to carry out this task.

2.1.4. Policies in Delimiting

- **Name**: FragmentationPolicy
  - **Description**: Policy that may partition a complete SDU into multiple fragments, in order to comply with the limitation of the maximum PDU size in a given layer.
  - **Default Action**: Do not fragment (leave the complete SDU untouched).
- **Name**: ConcatenationPolicy
  - **Description**: Policy that creates a complete user data field (used as the payload of a DTP PDU) out of SDU fragments and complete SDUs. It generates SDU sequence numbers as described in the General Delimiting Module specification.
  - **Default Action**: Do not concatenate (create a complete user data field for every single SDU fragment and/or complete SDU).
- **Name**: ReassemblyAndSeparationPolicy
  - **Description**: Processes the elements in the SDU reassembly queue in order to generate SDUs to be consumed by the user of the flow. The SDUs may not be complete if incomplete delivery is allowed.
  - **Default Action**: Parse user data fields in the SDU reassembly queue and generate the corresponding SDUs.

2.2. Error and Flow Control Protocol (EFCP)

2.2.1. Overview of EFCP

The EFCP specification defines the two component protocols that comprise the Error and Flow Control Protocol: the Data Transfer Protocol (DTP), providing tightly bound

mechanisms, and the Data Transfer Control Protocol (DTCP), providing loosely bound mechanisms. Every flow instantiated by a Flow-Allocator-Instance (FAI) creates an instance of DTP and its associated state vector. Those flows that require flow control or retransmission control will have a companion DTCP instance allocated as well. Alone DTP is stateless (although it records state) and corresponds roughly to what has been known as a "unit-data" protocol. With a companion DTCP, the EFCP is able to provide other services, such as reliable transmission with quality of service.

The DTP uses a single PDU that carries addresses, a connection-id, a sequence number, and various flags to signal congestion, etc. As a matter of practicality, the instantiation of a flow of any form requires bounding maximum packet lifetime, MPL. This bound is imposed by SDU Protection, which is outside the scope of EFCP.

While there are 10 operation codes defined for DTCP, in reality there are only three PDU types: (Ack/Nack/Flow, Selective Ack/Nack/Flow, and 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. Not only is Ack and Flow Control information sent to update the sender, but the receiver's left and right window edges are sent as a check on the state. Required fields for these PDUs can be extended by defining policies. While the DTCP controls the flow of DTP PDUs, it never inspects their contents. In essence, DTP writes to the state vector and DTCP reads the DTP state from it.

Once instantiated by the Flow Allocator Instance (FAI), the DTP State Machine accepts Send primitives and generates Deliver primitives to the binding identified by the Port-id translation. DTP accepts PDUs from the Multiplexing Task and delivers PDUs to the Multiplexing Task.

Narrative description of the DTP task

A Data Transfer Task is always created when a flow is allocated. The DTP performs all mechanisms associated with the Transfer PDU, such as sequencing, invoking SDU Protections and Delimiting to do fragmentation/reassembly, etc.

Note that while the DTCP-State Vector can be discarded during long periods of no traffic, the DTP-State Vector cannot be. The DTP-State Vector is only discarded after an explicit release by the AP or by the System (if the AP crashes), i.e. the port-ids are released. The DIF must have procedures to ensure that when traffic resumes the CEP-id is associated with the correct port-id. Even so, after long periods of no traffic it is recommended that the AP re-authenticate its apposite.

The DTP processing is quite simple. Depending on the policies in force on the flow, the Delimiting will provide fragmentation/reassembly functions (breaking SDUs into more than one PDU) and concatenation/separation functions (combining multiple SDUs into one PDU). DTP creates PDUs and processes them upon arrival and imposes sequencing.

When PDUs arrive for DTP, SDU Protection determines if they can be processed, if so they are ordered. If Retransmission Control is in use, i.e. DTCP is present and a Transfer PDU is received, if the Sequence Number of the PDU is less than the last sequence number acknowledged, then this PDU is a duplicate and is discarded. Otherwise, the PDU is put on the PDU Reassembly Queue and Delimiting is invoked to create SDUs and deliver them to the using application process.

Since EFCP may allow gaps in the data stream, the concept of the left window edge exists regardless of whether DTCP is present. With DTCP present, it is assumed that all PDUs less than the LeftWindowEdge have been acked. This means that there will be no retransmissions of PDUs with sequence numbers less than the Left Window Edge. All gaps have been resolved one way or another. When DTCP is not present, this implies that all PDUs with sequence numbers less than the LeftWindowEdge have been processed for delivery to the process above.

While Transmission Control should be a DTCP function and DTCP may set the conditions for it, the action itself must occur with DTP, where the PDUs are actually posted to the multiplexing task. Transmission Control provides the means to control the sending or not sending of PDUs beyond what is indicated by the feedback from the apposite DTCP. For example, there may be conditions under which the destination indicates PDUs may be sent, but other conditions, e.g. congestion, indicate that PDUs should not the rate should be reduced. Transmission Control also responds to the detection of Lost Control PDUs.

Narrative description of the DTCP task

If the service requested for a flow requires the use of the DTCP, it is instantiated when the DTP is created. The DTCP-SV can be discarded during long periods of no traffic. However, the state associated with the FAI for this DTAE-instance must be explicitly released.

The DTCP processing performs more complex policies for the flow control, transmission control and re-transmission control. The DTCP controls whether DTP PDUs can be posted to the RMT. It may do this based on feedback information from the receiver or based on its own estimators. Sending PDUs placed in its queues

for flow control and retransmission control at the appropriate time become DTCP's responsibility.

DTCP requires a degree of synchronization with its opposite to avoid deadlocks and to ensure progress. The DTCP uses the sequence numbers in the DTP PDUs and bounding 3 timers to ensure synchronization. Data Transfer and the feedback mechanisms are used to achieve this shared state. The three timers must be bounded and are: Maximum Packet Lifetime, MPL; maximum time to ack, A; and the time to attempt the maximum number of retries to deliver a PDU, R.

Retransmission control

The Retransmission Control requires Sequencing in the associated DTP instance. Based on the sequence numbers seen by the receiver, an AckPDU is sent to the sender. The Ack/Nack class of PDUs supports what has been traditionally called positive (Ack) and negative acknowledgements (Nack).

There are two modes in which the Retransmission Mechanism can be used: Positive Ack, or Negative Ack. This mechanism applies to all PDUs generated for this flow. The sender of Transfer PDUs will maintain a retransmission queue, i.e. each sender maintains such a queue. When a Transfer PDU is sent by the DTP, a copy is placed on this Retransmission Queue and a timer is started. The timer has a value TR, and is generally a function of round-trip time. (Calculation of the timer value is calculated by the RTT Estimator Policy.) If the timer expires before an AckPDU is received, all PDUs on the Retransmission Queue with sequence numbers less than or equal to the sequence number of the PDU associated with this time are re-sent.

If an Ack is received then, the sender will delete all PDUs from the Retransmission Queue with Sequence Numbers less than or equal to the contents of the Ack/Nack field and discontinue any Retransmission Timers associated with these PDUs. If a Nack is received the sender will retransmit all PDUs on the Retransmission Queue with Sequence Numbers greater than or equal to the contents of the Ack/Nack field.

If a Selective Ack is received the PDUs designated by the ranges of Sequence Numbers indicated by the Ack/Nack List are deleted from the Retransmission Queue and any timers associated with these PDUs are discontinued.

If the originating side receives a Selective Nack, the PDUs with Sequence Numbers from those ranges are retransmitted. Note that since one can never know whether a retransmitted PDU arrives, a Negative Ack can never cause PDUs to be removed from the Retransmission Queue.

Flow Control

Flow control policies may utilize both a rate based strategy or the classic sliding window mechanism. Both may be used without Retransmission Control. With either approach, the receiver of PDUs provides an indication to the sender when it can send. With the window mechanism, the sender is told the Right Window Edge and can send PDUs with sequence numbers less than that; and rate-based, in which the sender is old at what rate it may emit PDUs. The flow control is always expressed as the number of PDUs or as the number of PDUs per unit time, or more precisely, how many PDUs may be sent in a unit of time.

Each approach has its advantages depending on the traffic characteristics of the connection. By having two independent methods, a policy may actually use both. Allowing one to dominate the other, except under extreme conditions when the other may exert an effect. For example, one might allow rate-based flow control to dominate by keeping the right window edge sufficiently high that the receiving transport protocol machine is delivering data at about the same rate. However, the credit level is maintained at the something close to maximum buffer allocation for this connection. If the protocol machine should get behind and begin to exceed its maximum buffer allocation, the credit-based flow control would dominate and temporarily stop the flow on the connection, until the buffer pool was restored to an appropriate level.

2.2.2. State of the specification

- **Standalone component: 5**. EFCP has a clear role in the IPC Process that does not overlap with any other specification: to provide the data transfer and data transfer control functions. Furthermore, the interactions of EFCP with other IPC Process components (IPC API, Delimiting, the Relaying and Multiplexing Task and the Flow Allocator) are well defined and understood.
- **Completeness of the specification: 4**. All of the functions that EFCP carries out have been clearly identified, separated in three groups: data transfer (done by DTP), flow control and retransmission control (done by DTCP). These functions have been completely specified, including the definition of the different fine-grained policies enumerated in the last section of this chapter. The EFCP specification defines default behaviors for all the policies.
- **Availability of implementations: 2**. Three partial implementations of EFCP are currently under development, covering: data transfer, window-based flow control and part of retransmission control. These implementations are TRIA, IRATI and

ProtoRINA. At the date of this writing the three EFCP implementations are still under development and not interoperable.

2.2.3. Major issues/Limitations

EFCP is not a single data transfer protocol, since a complete specification only comes with the definition of a concrete set of policies. EFCP provides a framework for specifying many different data transfer protocols that can be used in different environments under different boundary conditions. As such, a key milestone for EFCP is to make sure that the core specification is correct, robust and allows for the customization of the framework with many different policies. The only way to reach this milestone is through experimentation with a wide range of policies that cover different use cases. PRISTINE will be contributing to this work through WP3 mainly.

2.2.4. Parameters and Policies in EFCP

DIF static parameters

Concrete syntax parameters

These parameters define the concrete syntax of EFCP. One would expect some combinations to be used in many DIFs.

- **AddressLength**: *Unsigned Integer* - The length of an address in bits: (4, 8, 16, 32, 64?)
- **QoSidLength**: *Unsigned Integer* - The length of a QoS-id in bits: (4, 8)
- **PortIdLength**: *Unsigned Integer* - The length of a Port-id in bits. (4, 8, 12, 16)
- **CEPIdLength**: *Unsigned Integer* - The length of a CEP-id in bits.(4, 8, 12, 16)
- **SequenceNumberLength**: *Unsigned Integer* - The length of a SequenceNumber in bits. (4, 8, 16, 32, 64)
- **LengthLength**: *Unsigned Integer* - The length of a PDU in bits. (4,8,16,32?)

DIF-wide parameters

These parameters are defined for each DIF.

- **QoS-Cube-identifiers**: *Unsigned Integer* - While it might be nice if QoS-Cube-ids were globally unambiguous, there is no reason that they have to be. There is more reason that the names of them be.

- **Max SDU Size**: *Unsigned Integer* - The maximum size allowed for a SDU written to/ read from this DIF. This parameter may be restricted by specific QoS cubes or even specific flows to a smaller value but not to a larger value.
- **Max-PDU-Size**: *Unsigned Integer* - The maximum size allowed for a PDU in this DIF.

DIF-wide policies

- **Policy name**: Unknown flow Policy.
  - **Description**: When a PDU arrives for a Data Transfer Flow terminating in this IPC-Process and there is no active DTSV, this policy consults the ResourceAllocator to determine what to do.
  - **Default action**: Discard the PDU.

QoS-cube specific parameters

These parameters are set on a per QoS-cube basis. Each parameter can be defined as a range, thus defining a QoS cube in the QoS space. The following parameters are an initial list of potential candidates that will be refined as more experience is gained. QoS-cube parameters do not need to be standard, but it will be useful that the most widely used parameters can be understood by many DIFs.

- **Average bandwidth**: *Unsigned Integer* - measured at the application in bits/sec
- **Average SDU bandwidth**: *Unsigned Integer* - measured in SDUs/sec
- **Peak bandwidth-duration**: *Unsigned Integer* - measured in bits/sec
- **Peak SDU bandwidth-duration**: *Unsigned Integer* - measured in SDUs/sec
- **Burst period**: *Unsigned Integer* - measured in seconds
- **Burst duration**: *Unsigned Integer* - measured in fraction of Burst Period
- **Undetected bit error rate**: *Real Number* - measured as a probability
- **MaxSDUSize**: *Unsigned Integer* - measured in bytes
- **Partial Delivery**: *Boolean* - Can SDUs be delivered in pieces rather than all at once?
- **Incomplete Delivery**: *Boolean* – Can SDUs with missing pieces be delivered?
- **Order**: *Boolean* - Must SDUs be delivered in order?
- **Max allowable gap in SDUs**: *Unsigned Integer* - a gap of N SDUs is considered the same as all SDUs delivered, i.e. a gap of N is a "don’t care.")

- **MaxDelay**: *Unsigned Integer* - in secs
- **Jitter**: *Unsigned Integer* - in secs

DTP Policies and Parameters

Parameters:

- **DTCPpresent**: *Boolean* – Indicates whether this connection is using DTCP.
- **Initial A-Timer**: *Unsigned Integer* – Assigned per flow that indicates the maximum time that a receiver will wait before sending an ACK. Some DIFs may wish to set a maximum value for the DIF. If 0 means immediate acking.
- **SequenceNumberRollOverThreshold**: *Unsigned Integer* – When the sequence number is increasing beyond this value, the sequence number space is close to rolling over, a new connection should be instantiated and bound to the same port-ids, so that new PDUs can be sent on the new connection.

Policies:

- **Policy name**: RcvrTimerInactivity Policy
  - **Description**: If no PDUs arrive in this time period, the receiver should expect a DRF (Data Run Flag) in the next Transfer PDU. If not, something is very wrong. The timeout value should generally be set to 3(MPL+R+A).
  - **Default action**: Set the DRFFlag to True, select an initial Sequence Number, send a Transfer PDU with zero length data. If DTCP is present, discard all PDUs on the retransmission queue, discard any PDUs on the ClosedWindowQueue and send a Control AckPDU. Notify the user there has been no activity for the period.
- **Policy name**: SenderInactivityTimer Policy
  - **Description**: This timer is used to detect long periods of no traffic, indicating that a DRF should be sent. If not, something is very wrong. The timeout value should generally be set to 2(MPL+R+A).
  - **Default action**: Set the DRFFlag to True, select an initial Sequence Number, send a Transfer PDU with zero length data. If DTCP is present, discard all PDUs on the retransmission queue, discard any PDUs on the ClosedWindowQueue and send a Control AckPDU. Notify the user there has been no activity for the period.
- **Policy name**: InitialSequenceNumber Policy.
  - **Description**: This policy allows some discretion in selecting the initial sequence number, when DRF is going to be sent.

- **Default action**: Select a Sequence Number at Random.

DTCP Policies and Parameters

Parameters:

- **Flow Control**: *Boolean* – Indicates whether Flow Control is in use. The equivalent of the relation (Window-based OR Rate-based).
- **Retransmission Present**: *Boolean* – Indicates whether Retransmission Control (potentially with gaps) is in use.

Policies:

- **Policy name**: Lost Control PDU Policy
  - **Description**: This policy determines what action to take when the PM detects that a control PDU (Ack or Flow Control) may have been lost. If this procedure returns True, then the PM will send a Control Ack and an empty Transfer PDU. If it returns False, then any action is determined by the policy.
  - **Default action**: Send ControlAck PDU indicating last ControlAck received.
- **Policy name**: RTT Estimator Policy
  - **Description**: This policy is executed by the sender to estimate the duration of the retransmission timer. This policy will be based on an estimate of round-trip time and the Ack or Ack List policy in use.
  - **Default action**: A standard algorithm for computing a running average of round trip time.

Retransmission control policies

Parameters:

- **MaximumTimeToRetry**: *Unsigned Integer* - Maximum time to attempt the retransmission of a packet, this is **R**.
- **DataRexmsnMax**: *Unsigned Integer* – Indicates the number of times the retransmission of a PDU will be attempted before some other action must be taken.
- **InitialRtxTime**: *Unsigned Integer* - Indicates the time to wait before retransmitting a PDU (tr).

Policies:

- **Policy name**: Retransmission Timer Expiry Policy
  - **Description**: This policy is executed by the sender when a Retransmission Timer Expires. If this policy returns True, then all PDUs with sequence number less than or equal to the sequence number of the PDU associated with this timeout are retransmitted; otherwise the procedure must determine what action to take. This policy must be executed in less than the maximum time to Ack.
  - **Default action**: Check the number of retransmission and if greater than the maximum, then declare an error; Otherwise, retransmit all PDUs on the Retransmission queue with a sequence number greater than or equal to the Ack field.
- **Policy name**: Sender Ack Policy
  - **Description**: This policy is executed by the Sender and provides the Sender with some discretion on when PDUs may be deleted from the ReTransmissionQ. This is useful for multicast and similar situations where one might want to delay discarding PDUs from the retransmission queue.
  - **Default action**: For all PDUs on the Retranmission queue with Sequence Number less than or equal to this Ack, remove them from the Queue and cancel the A-Timers associated with them.
- **Policy name**: Receiving Ack List Policy
  - **Description**: This policy is executed by the Sender and provides the Sender with some discretion on when PDUs may be deleted from the ReTransmissionQ. This policy is used in conjunction with the selective acknowledgement aspects of the mechanism and may be useful for multicast and similar situations where there may be a requirement to delay discarding PDUs from the retransmission queue.
  - **Default action**: If this is a Selective Ack delete the PDUs from the Retransmission queue and cancel the associated Timers. If Nack immediately retransmit the indicated PDUs.
- **Policy name**: RcvrAck Policy
  - **Description**: This policy is executed by the receiver of the PDU and provides some discretion in the action taken. The default action is to either Ack immediately or to start the A-Timer and Ack the LeftWindowEdge when it expires.
  - **Default action**: If Acking Immediately Then Update the Left Window Edge, Send an Ack/Flow Control PDU, Cancel any A-Timers associated with Acked PDUs; Otherwise Set the A-Timer.

- **Policy name**: SendingAck Policy
  - **Description**: This policy allows an alternate action when the A-Timer expires when DTCP is present.
  - **Default action**: Left Window Edge is updated, Delimiting is invoked to create SDUs, Ack/Flow Control PDU is sent.
- **Policy name**: RcvrControlAck Policy
  - **Description**: This policy allows an alternate action when a Control Ack PDU is received.
  - **Default action**: Check Consistency of Sender's Window values against what the Control Ack PDU indicates they should be. Adjust as necessary and Send an Ack/Flow Control PDU and an empty Transfer PDU.

Flow control policies

Parameters:

- **Window-based**: *Boolean* – Indicates whether window-based flow control is in use.
- **RcvBytesThreshold**: *Unsigned Integer* – The number of free bytes below which flow control does not move or decreases the amount the Right Window Edge is moved.
- **RcvBytesPercentThreshold**: *Unsigned Integer* – The percent of free bytes below which flow control does not move or decreases the amount the Right Window Edge is moved.
- **RcvBuffersThreshold**: *Unsigned Integer* – The number of free buffers at which flow control does not advance or decreases the amount the Right Window Edge is moved.
- **RcvBufferPercentThreshold**: *Unsigned Integer* – The percent of free buffers below which flow control should not advance or decreases the amount the Right Window Edge is moved.
- **Rate-based**: *Boolean* – Indicates whether rate-based flow control is in use.
- **SendBytesThreshold**: *Unsigned Integer* – The number of free bytes below which flow control should slow or block the user from doing any more Writes.
- **SendBytesPercentThreshold**: *Unsigned Integer* – The percent of free bytes below, which flow control should slow or block the user from doing any more Writes.

- **SendBuffersThreshold**: *Unsigned Integer* – The number of free buffers below which flow control should slow or block the user from doing any more Writes.
- **SendBuffersPercentThreshold**: *Unsigned Integer* – The percent of free buffers below which flow control should slow or block the user from doing any more Writes.

Policies:

- **Policy name**: Closed Window Policy
  - **Description**: This policy is used with flow control to determine the action to be taken when the receiver has not extended more credit to allow the sender to send more PDUs. Typically, the action will be to queue the PDUs until credit is extended. This action is taken by DTCP, not DTP
  - **Default action**: Put the PDU on the ClosedWindowQueue and Block further Write API calls on this port-id.
- **Policy name**: Flow Control Overrun Policy
  - **Description**: This policy determines what action to take if the receiver receives PDUs but the credit or rate has been exceeded.
  - **Default action**: The PDU is discarded.
- **Policy name**: Reconcile Flow Conflict Policy
  - **Description**: This policy is invoked when both Credit and Rate based flow control are in use and they disagree on whether the PM can send or receive data. If it returns True, then the PM can send or receive; if False, it cannot
  - **Default action**: None.
- **Policy name**: Receiving Flow Control Policy
  - **Description**: This policy allows some discretion in when to send a Flow Control PDU when there is no Retransmission Control.
  - **Default action**: Send a Flow Control PDU with receiver's current Right Window Edge

Sliding window flow control policies

Parameters:

- **MaxClosedWindowQueueLength**: *Unsigned Integer* – The number PDUs that can be put on the ClosedWindowQueue before something must be done.

- **InitialCredit**: *Unsigned Integer* – Added to the initial sequence number to get right window edge.

Policies:

- **Policy name**: Transmission Control Policy
  - **Description**: This policy is used when there are conditions that warrant sending fewer PDUs than allowed by the sliding window flow control, e.g. the ECN bit is set.
  - **Default action**: Add as many PDUs to PostablePDUs as Window allows, closing it if necessary. Setting ClosedWindow Flag as appropriate.
- **Policy name**: Rcvr Flow Control Policy
  - **Description**: This policy is invoked when a Transfer PDU is received to give the receiving PM an opportunity to update the flow control allocations.
  - **Default action**: If there are enough free buffers move the Right Window Edge, otherwise don't

Rate-based flow control policies

Parameters:

- **SendingRate**: *Unsigned Integer* – Indicates the number of PDUs that may be sent in a TimePeriod. Used with rate-based flow control.
- **TimePeriod**: *Unsigned Integer* – Indicates the length of time in microseconds for pacing rate-based flow control.

Policies:

- **Policy name**: NoRate-SlowDown Policy
  - **Description**: This policy is used to momentarily lower the send rate below the rate allowed.
  - **Default action**: Add the PDU to PostablePDUs and increment the number of PDUs sent in this time period. In other words, normal behavior.
- **Policy name**: NoOverrideDefaultPeak Policy
  - **Description**: This policy allows rate-based flow control to exceed its nominal rate. Presumably this would be for short periods and policies should enforce

this. Like all policies, if this returns True it creates the default action which is no override.

- **Default action**: Set RateFulfilled indicating all the PDUs that can be sent in this time period have been sent and put the PDU on PostablePDUs. Normal behavior.

- **Policy name**: RateReduction Policy
  - **Description**: This policy allows an alternate action when using rate-based flow control and the number of free buffers is getting low.
  - **Default action**: If the number of free buffers are low, reduce the rate by 10%, Otherwise, if plenty of buffers check to see if the current rate is lower than the configured rate and if it is set it to the Configured Rate.

2.3. Relaying and Multiplexing Task (RMT)

2.3.1. Overview of the RMT

Strictly speaking the specification of the RMT is not needed as a RINA standard because it has no externally visible conformance requirements. However, it is worth describing the tasks of the RMT in order to identify the different policies that can influence its behaviour. Logically the RMT sits between the Data Transfer Protocol and the SDU Protection Module. The purpose of the RMT is twofold: 1) it multiplexes SDUs from higher ranking (N+1)-DIFs onto port-ids of the (N-1)-DIFs and 2) relays incoming PDUs from (N-1)-port-ids to either the appropriate EFCP instance, i.e. flow terminating in this IPC Process or to the appropriate outgoing (N-1)-port-id.

PDUs are delivered to the RMT ready to be forwarded. No further processing is required. For outgoing PDUs, this implies SDU Protection has already been applied. For incoming PDUs, SDU Protection has been applied and any PDUs that do not pass have been discarded. Assume that RMT maintains a queue for each QoS cube for each (N-1)-DIF it can access.

The DT-SV includes the identifier of the RMT queue that the PDUs generated by an EFCP-instance are delivered to. This information is provided to the Flow Allocator by the Resource Allocator at initialization and changed by the Resource Allocator as required. Changing the RMT queue does not affect active flows.

The primary job of the RMT in its multiplexing role is to pass PDUs from DTP instances to the appropriate (N-1)-port. This decision is based on two criteria: routing and QoS. In its relaying role, PDUs that arrive on an (N-1)-port with a destination address that does not belong to this IPC-Process are usually forwarded based on its destination address,

and QoS-cube-id fields. The Figure below illustrates the flow of the traffic through the RMT as it enters and leaves the IPC Process.

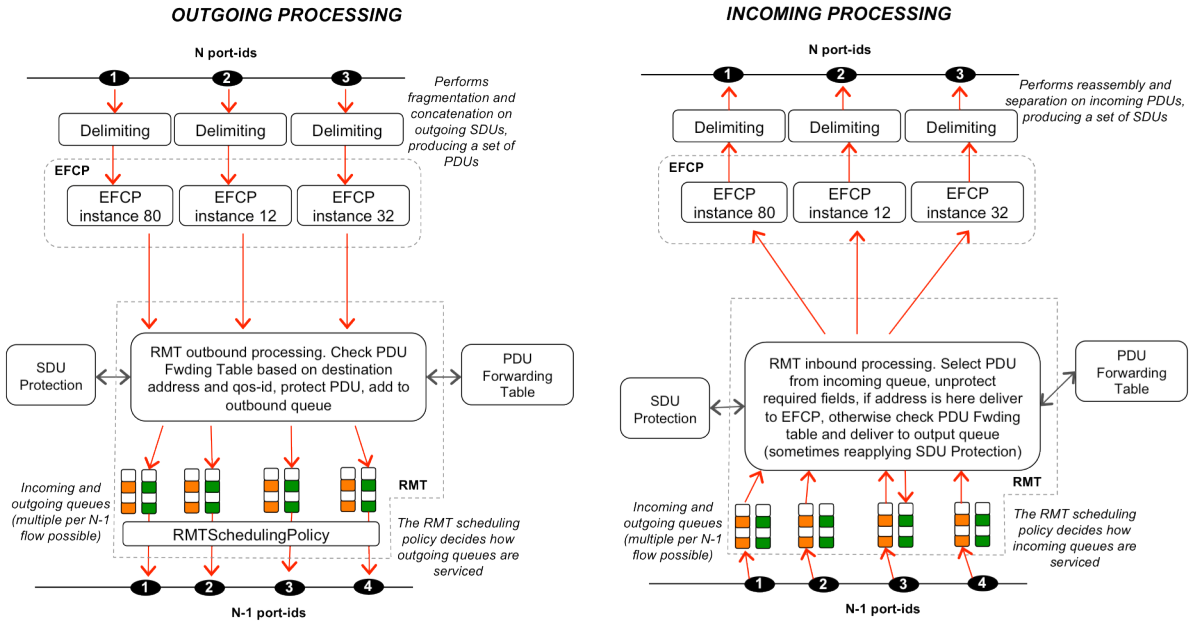


Figure 3. Processing of incoming and outgoing traffic by the RMT

Each flow created by the (N)-DIF has requested a particular destination and QoS. This DIF supports some number of QoS cubes and is supported by one or more (N-1)-DIFs and ports (interfaces) that may each support other QoS cubes. The task of this DIF, using the facilities of the EFCP, SDU Protection, and RMT is to match the QoS requested by a flow to a QoS cube supported by this DIF. Note that the (N-1)-port may not support the QoS requested and the task of EFCP, SDU Protection, and RMT is to improve the QoS within the range acceptable to the flow or in some cases, not to make it worse.

RMT's role in this task is controlled by two policies: 1) to determine the scheduling of the queues, and 2) to control the length of the queues and what action to take if they become too long.

2.3.2. State of specification

- **Standalone component: 5**. Both the role of the RMT and its relationship with other IPC Process components are very clear. In the outgoing direction, the RMT multiplexes PDUs from EFCP into one or more N-1 flows, with the aid of the PDU Forwarding Table. It may also call the SDU Protection Module to protect outgoing SDUs. In the incoming direction, the RMT reads PDUs from N-1 flows, unprotects them and - based on the output of the PDU Forwarding Table - either passes the PDU to an EFCP instance or sends it to another N-1 flow. The Resource Allocator manages the RMT.

- **Completeness of the specification: 3**. Even though the specification of the RMT functions is complete, the vast majority of the RMT work goes into its policies. The current specification does not provide default policies, therefore it is not possible to implement a working RMT only with the information in the specification document.
- **Availability of implementations: 3**. The three RINA implementations under development implement some sort of Relaying and Multiplexing Task component, following the schema described in this document with more or less accuracy. However, the three implementations use very simple policies only sufficient for simple experiments and demo DIFs.

2.3.3. Major issues/Limitations

The big challenge for the RMT both in terms of research and implementation is the design of sophisticated policies that can effectively multiplex and relay traffic of different characteristics in the same DIF. To date only very simple approaches for RMT scheduling have been adopted, therefore it is important to get more experience in this topic; also to gain a deeper understanding of how the RMT behavior affects the level of service provided by a DIF. Part of this work will be addressed in PRISTINE's WP3.

2.3.4. Policies in the RMT

- **Name**: RMTQMonitorPolicy.
  - **Description**: This policy can be invoked whenever a PDU is placed in a queue and may keep additional variables that may be of use to the decision process of the RMT-Scheduling Policy and the MaxQPolicy.
- **Name**: RMT-SchedulingPolicy.
  - **Description**: This is the meat of the RMT. This is the scheduling algorithm that determines the order input and output queues are serviced. We have not distinguished inbound from outbound. That is left to the policy. To do otherwise, would impose a policy. This policy may implement any of the standard scheduling algorithms, FCFS, LIFO, longestQfirst, priorities, etc.
- **Name**: MaxQPolicy.
  - **Description**: This policy is invoked when a queue reaches or crosses the threshold or maximum queue lengths allowed for this queue. Note that maximum length may be exceeded.
- **Name**: PDUForwardingPolicy.

- **Description**: This policy is invoked per PDU, in order to obtain the N-1 port(s) through which the PDU has to be forwarded. A forwarding table constructed by routing that maps the *destination address* and *qos-id* fields of the PDU to the list of N-1 ports will be one of the most common ways of implementing this policy, but need not to be the only one (such as in the case of topological routing, as explained in section 3.3).

2.4. SDU Protection

2.4.1. Overview of SDU Protection

SDU Protection includes all checks necessary to determine whether or not a PDU should be processed further or to protect the contents of the PDU while in transit to another IPC Process that is a member of the DIF. It may include but is not limited to checksums, CRCs, Hop Count/TTL, encryption, etc. SDU Protection is a stateless function that is not specific of a DIF, any DAF can choose to protect its SDUs if it doesn't trust the underlying DIF, as illustrated in the following Figure.

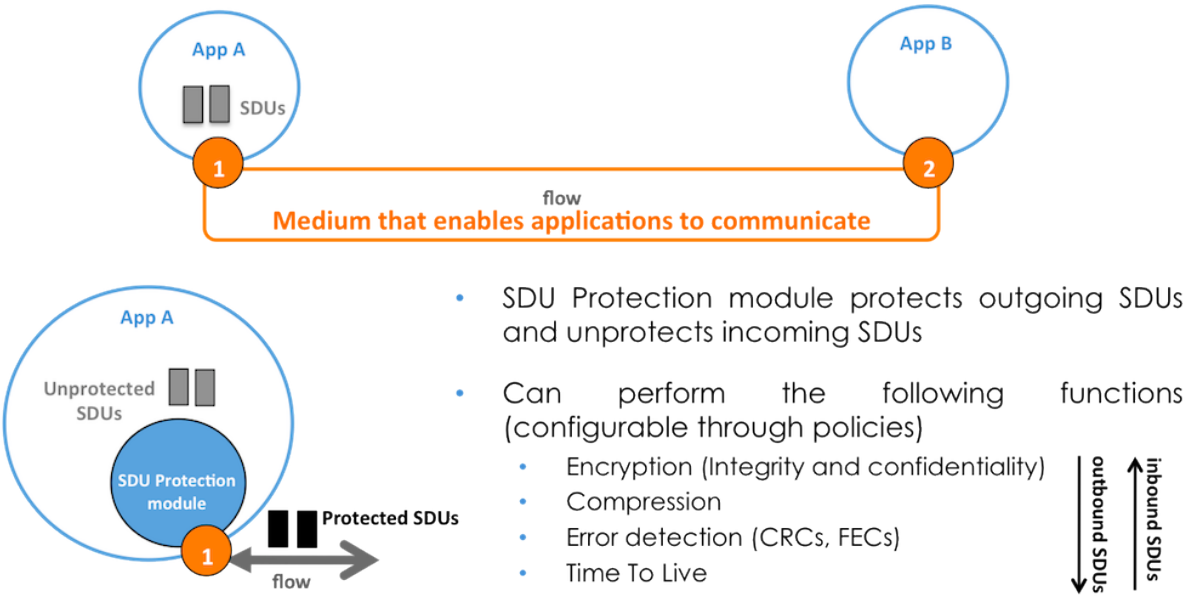


Figure 4. Schematic view of SDU Protection functionality

Within a DIF, SDU protection is performed on an N-1 flow basis; meaning that SDU Protection policies can vary for each N-1 flow the DIF is using (since the level of trust in underlying DIFs may be different). All the functionality of the SDU Protection module is policy; however the current RINA specifications characterizing SDU Protection in DIFs define a template that tries to categorize all the potential SDU protection functionalities into four categories:

- **Encryption**. Encoding SDUs so that N-1 DIFs cannot parse its content or modify it without being detected by the application that receives the SDU.
- **Compression**. Although it is not exactly a protection mechanism, this is the place in the architecture where compression goes.
- **Error detection**. A procedure by which extra bits are added to the PDU that allow the receiver to detect and correct bit errors.
- **TTL (Time To Live)**. Mechanisms that limit the lifetime of a PDU within a DIF, ensuring that Maximum Packet Lifetime (MPL) is enforced.

2.4.2. State of specification

- **Standalone component: 5**. The goals of this component are clear and identified in the previous section. Relationship of SDU Protection with the RMT and EFCP are also well understood: in the *outgoing path* the EFCP (if the IPC Process does not have different SDU Protection policies for different N-1 flows) or the RMT passes an unencrypted PDU to the module, which returns it encrypted; in the incoming path SDU Protection takes an encrypted PDU and decrypts it (fully or partially).
- **Completeness of the specification: 3**. There has been an effort to categorize the different types of functions in the scope of this component, resulting in the definition of four types of policies. However no specific policies for SDU Protection have been defined up to date.
- **Availability of implementations: 1**. None of the available RINA prototypes implements any SDU Protection policy.

2.4.3. Major issues/Limitations

The major limitation of SDU Protection is that up to date no policies for this component have been defined - which is specially important in a component that is almost completely policy. The definition of multiple policies for very different operational environments should facilitate the improvement and consolidation of the specification of the generic SDU Protection Module.

2.4.4. Policies in SDU Protection

The following abstract policies were derived from the definition of the SDU protection functionality. They cover the intended functionality of the module, without providing any particular details, which need to be further refined.

- **Name**: EncryptionPolicy.
  - **Description**: All the procedures that need to be applied to encrypt and decrypt EFCP PDUs passed to the SDU Protection Module. In addition to the choice of cryptographic algorithms and key sizes to perform this functions, the policy should also dictate what parts of the PDU should be encrypted (all the PDU, just the PCI, just the payload, both the payload and PCI but separated) and under which circumstances (do the same for all PDUs, do different actions depending on the PDU type, qos-id, etc.).
- **Name**: CompressionPolicy.
  - **Description**: Procedures to compress and compress SDUs passed to the SDU Protection Module. As in the case of encryption, the policy should define how to compress, what to compress (PCI, payload, both at once, both separate) and when.
- **Name**: ErrorCheckPolicy.
  - **Description**: Procedures to add redundancy to a PDU so that the receiver can detect and correct errors in the bits of that PDU. Mechanisms such as parity bits, checksums, cyclic redundancy codes (CRCs), cryptographic hash functions, etc fall into this category. As with the two previous policies, the error check can be applied to all the PDU or just some part of it, and can be homogeneous across all PDUS or behave differently depending on specific values of the PCI.
- **Name**: TTLPolicy.
  - **Description**: Limits the lifetime of a PDU within a DIF - this type of SDU Protection policies are DIF-wide and do not vary on a per N-1 DIF basis. Example policies could be hop counts (approximating MPL with a number of hops in the DIF) or timestamps.

2.5. Common Application Establishment Phase (CACEP)

2.5.1. Overview of CACEP

CACEP allows two AEs in different Application Processes (APs) to establish an application connection. During the application connection establishment phase, the AEs 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 must be

possible to allow multiple versions of the RIB to be used, to allow for incremental network management upgrades.

As illustrated by the Figure below, CACEP operates the following way. The Initiating Process first allocates a (N-1)-flow with a destination application. When this is complete, it sends a *M_Connect Request* with the appropriate parameters (mainly source and destination application naming information, identification of the authentication mechanism to be used - if any -, ids of the abstract and concrete syntaxes, and version of the RIB) and initiates the authentication policy. Depending on the complexity of the authentication policy, zero or more CDAP messages will be exchanged between the communicating Application Processes. When the authentication policy completes, a positive or negative *M_Connect Response* is returned by the destination application process and the connection is established or terminated respectively.

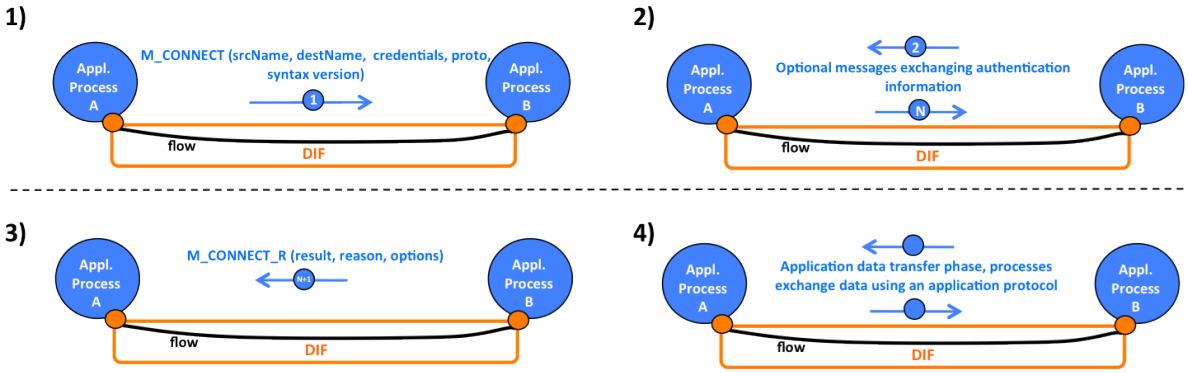


Figure 5. Operation of CACEP

When any of the two APs wishes to terminate the application connection, it sends a *M_release* Request message to its neighbour, which may or may not require an associated *M_release Response* message. After that the application connection is over. The two APs may choose to deallocate the flow that was supporting the application connection, or to re-use the same flow for a new application connection. It is also possible to use CACEP connection establishment with another protocol in the data transfer phase (for example, HTTP).

2.5.2. State of specification

- **Standalone component: 5**. The goals of CACEP are very focused and clearly defined: to allow to application entities to establish an application connection.
- **Completeness of the specification: 4**. The CACEP specification completely describes the functions carried out by the protocol, including the definition of an optional authentication policy. Up to date no specific authentication policies for CACEP have been defined, something that PRISTINE will address in its WP4.

- **Availability of implementations: 4**. There are currently two interoperable implementations of CACEP: the TRIA and IRATI ones; although both of them only use the default authentication policy (which is to not use authentication).

2.5.3. Major issues/Limitations

No major limitations have been identified in the CACEP specification.

2.5.4. Policies in CACEP

- **Name**: AuthenticationPolicy
  - **Description**: Policy that the application processes use to authenticate each other. It can range from *none*, to user/password, Public Key Infrastructure (PKI) - based authentication, etc.

2.6. Common Distributed Application Protocol (CDAP)

2.6.1. Overview of CDAP

The Common Distributed Application Protocol (CDAP) is used by communicating RINA applications to exchange structured application-specific data. The RINA architecture uses CDAP to construct specialized distributed applications that cooperate to create a Distributed IPC (Inter-Process Communication) Facility, which provides network transport to other applications. However, it can be used by any application that needs to share information or initiate state changes with another application over a network. The protocol itself is not application-specific.

CDAP enables distributed applications to deal with communications at an object level, rather than forcing applications to explicitly deal with serialization and input/output operations. CDAP provides the 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. CDAP provides a straightforward and unifying approach to sharing data over a network without having to create specialized protocols.

CDAP is an object-oriented protocol modelled after CMIP (the Common Management Information Protocol) [cmip] that allows two AEs to perform six operations on the objects exposed by their Resource Information Bases (RIBs). These fundamental remote operations on objects are: create, delete, read, write, start and stop. Since in RINA there is only one application protocol (CDAP), the different AEs in the same application process do not identify different application protocols, but the subsets

of the RIB available through a particular application connection. That is, different AEs provide different levels of privileged access into an Application Process RIB - as illustrated in the Figure below.

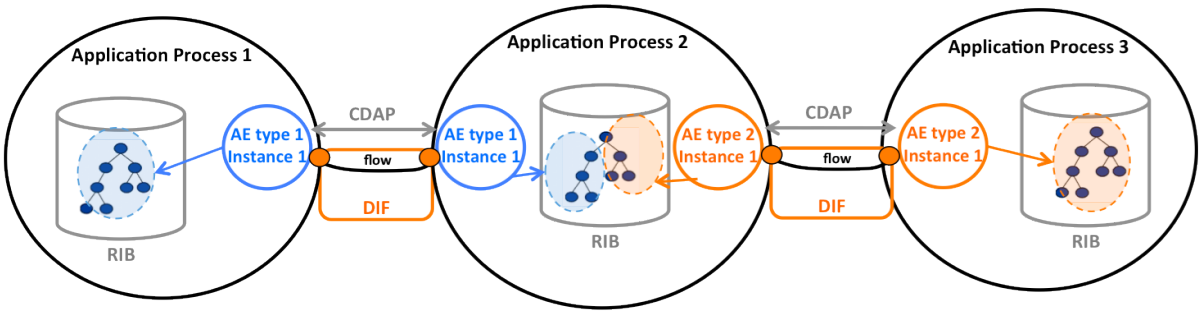


Figure 6. Graphical example illustrating CDAP operation

Operating on objects

CDAP allows applications to send and receive data using structured information that we refer to as Objects. We refer to the set of objects stored within an application that is available via CDAP as its local Resource Information Base (RIB). All modern programming languages have an object concept, though the component types and aggregation capabilities are richer in some languages than others. In the CDAP model, the application AE's that are communicating with one another create a shared object space, providing access to the portion of the application's RIB that is relevant to the AE's purpose, and allowing a distributed application to selectively create and share distributed objects.

Provided that an application can encode an object into a sequence of bytes, and that the application it is communicating with can decode them properly, CDAP places no restrictions on application object values, however, for its own operation, CDAP uses a simple and easily-represented definition of the general object concept, consisting of entities that are i)a limited set of scalar types (a single scalar is a degenerate object); ii) aggregations of objects of the same type (arrays); or iii) aggregations of potentially-dissimilar objects (structures).

The objects in an application have four properties of concern to CDAP:

- A Class, or type, that recursively captures all of the types of the outermost object and those of any contained objects,
- a name that is unique among other objects of the same class,
- a value,
- an object identifier, an integer that may be assigned to a specific instance of an object by its owner as an alias to its class and name.

Scope and filter

Scope and filter allow the user of CDAP to operate on multiple objects with a single CDAP message. Scoping selects objects to be operated upon within the managed object containment tree. As shown in the Figure below, the scope of an operation is defined relative to a base managed object:

- Operation applies to the base object only
- Operation applies to the Nth level subordinate objects only
- Operation applies to the base object plus all of its subordinates (entire sub-tree)

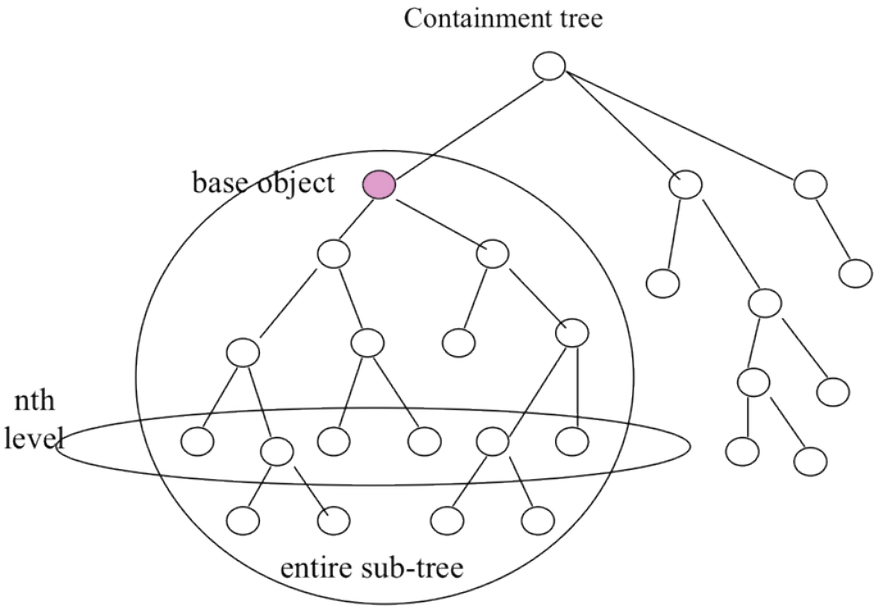


Figure 7. Graphical representation of scope in a CDAP operation

Filtering permits objects within scope to be selected according to test criteria, allowing the CDAP user to select a subset of all the objects in the scope of an operation. The operation is then applied to all selected objects.

Concrete syntaxes

The encoding of the message type (*opCode*) and other values in the message into a form used for communication on a *wire* is referred to as the concrete syntax of the message. Agreement between communicating applications on the concrete syntax to use is a fundamental requirement for communication. Once this is established, it becomes possible to discuss other aspects of the communication, such as the version of the message declarations (the abstract syntax) and object definitions and values at the application level (usually summarized as a *version*) in use.

Google Protocol Buffers (GPB) [gpb] has been the first concrete syntax defined to encode and decode the CDAP protocol messages. The reasons for choosing GPB are that it provides an efficient encoding (both in terms of parsing/generation time and bit efficiency), it is being used in production in massive scale distributed systems by Google and there are free, open tools for developers in many programming languages. Other encoding schemes such as the JavaScript Object Notation (JSON) [json], eXtensible Markup Language (XML) [xml] or Abstract Syntax Notation One (ASN.1) [asn1] concrete encodings may be used in some environments and will be studied in the future.

2.6.2. State of the CDAP specification

- **Standalone component: 5**. The role of CDAP is very well defined: it is the only application protocol in the RINA architecture. IPC Processes use CDAP to exchange structured information, allowing them to operate on each other RIBs (Resource Information Bases). The interactions between IPC Processes and the other components are also well understood, being the RIB Daemon the only direct user of CDAP in the IPC Process.
- **Completeness of the specification: 3-4**. The current CDAP specification is mostly complete with the exception of the definition of the filter mechanism, as explained in the "major limitations" section. To date only one concrete syntax has been specified: Google Protocol Buffers, although others are planned for the near future.
- **Availability of implementations: 3-4**. The TRIA and IRATI prototypes have almost complete interoperable CDAP implementations. All CDAP features are addressed by both prototypes except for scope and filter.

2.6.3. Major issues/Limitations

The biggest pending feature in the CDAP specification is the definition of the filter mechanism. The current document mentions that the same approach that CMIP uses could be adapted to RINA. In CMIP the filter is a boolean expression that can be applied to all CMIP request operations and evaluated on the attributes of the objects in its scope. The types of rules that can be applied to the request include equality, greater than or equal, less than or equal, and presence of an attribute. Other set and string based tests can be asserted, with combinations of the filters being grouped with the standard boolean terms of AND, OR, and NOT.

CDAP could take this filter definition and then get experience with it to understand the issues and limitations when applied in the RINA context. With this knowledge it would

be possible to either consider this filter definition as adequate or to propose a better specification that overcomes the limitations of filters as defined in CMIP. PRISTINE will be contributing to this research under the scope of WP5.

2.6.4. Policies in CDAP

There are no policies in CDAP strictly speaking. The only aspect of the protocol that allows for multiple options is the **concrete syntax**.

2.7. RIB Object Model

2.7.1. Overview of the current RIB Object Model

The only specification available for a RIB object model is an incomplete demo specification just build for the purpose of RINA experimentation activities (basically to allow early experimentation between the TRIA, BU and IRATI prototypes). This RIB schema is incomplete and designed bottom up: objects were added as required during the design and implementation of the prototypes, but not much though was put into what would be an optimum RIB schema design.

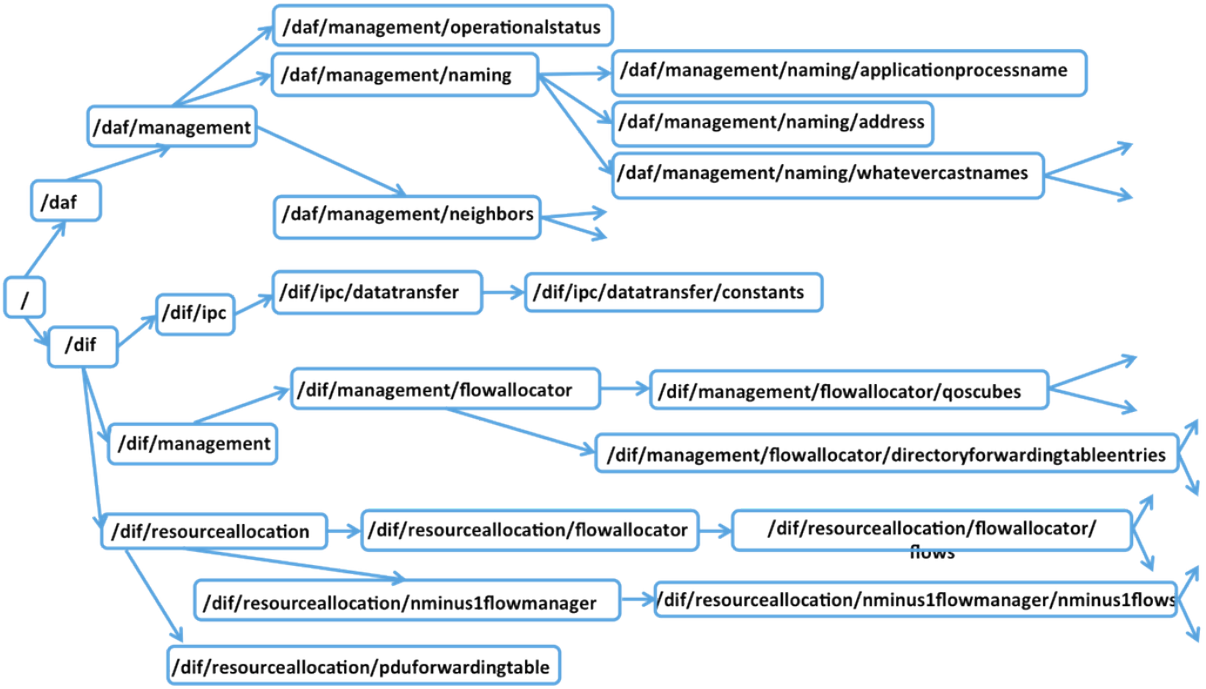


Figure 8. Graphical illustration of relationships between object instances in the current RIB model

The Figure above shows a graphical illustration of the relationships between the different object instances in the current RIB managed object model specification. This experimental RIB schema does not support object class inheritance or notifications,

and uses Google Protocol Buffers [gpb] as its concrete syntax. The specification defines the following objects:

- **/daf/management/operationalstatus**: Operational state of the IPC Process.
- **/daf/management/naming/applicationprocessname**: Naming information (process name and instance) of the IPC Process.
- **/daf/management/address**: The current address of the IPC Process.
- **/daf/management/naming/whatevercastnames**: Placeholder object that contains all the whatevercast names known by the IPC Process.
- **/daf/management/naming/whatevercastname/<name>**: A whatevercast name, associated to a rule that resolves to one or more IPC Process names.
- **/daf/management/naming/neighbors**: Placeholder object that contains information on all the nearest neighbors of the IPC Process (IPC Process it shares an N-1 flow with).
- **/daf/management/naming/neighbors/<neighbor_process_name>**: Information about a neighbor of the IPC Process, including naming, address, port-id of the N-1 flow in common and supporting DIFs that can be used to allocate a flow to it.
- **/dif/ipc/datatransfer/constants**: DIF-wide data-transfer parameters that are constant across EFCP connections, such as: the length of the fields of the headers of EFCP PDUs, the Maximum Packet Lifetime, the DIF's Maximum PDU size, etc.
- **/dif/management/flowallocator/qoscubes**: Placeholder object that contains all the qos cubes supported by the DIF.
- **/dif/management/flowallocator/qoscubes/<qoscube-id>**: Information about a qos-cube supported by the DIF, including: ranges of performance supported (capacity, PDU loss, delay, in-order-delivery of PDUs) and policies required to provide the advertised service.
- **/dif/resourceallocation/flowallocator/flows**: Placeholder object that contains all the flows of which the IPC Process is source or destination.
- **/dif/resourceallocation/flowallocator/flows/<port-id>**: Information about a flow: source and destination application process names, id of the qos-cube the flow belongs to, source and destination port-ids, ids and policies of all the EFCP connections supporting the flow, etc.
- **/dif/management/flowallocator/directoryforwardingtableentries**: Placeholder object that contains all the entries of the directory forwarding table, which maps the destination application process name to an address of an IPC Process (the next IPC Process that will process a Flow Allocation request).

- **/dif/management/flowallocator/directoryforwardingtableentries/<destination_app_name>**: An entry in the directory forwarding table, which contains the naming information of an application process registered in the DIF, and the address of the IPC Process to whom the Flow Allocation request has to be forwarded.

2.7.2. State of specification

- **Standalone component: 5**: The RIB Object model provides the external view of the state of the IPC Process, as well as the means of manipulating it via operations on the objects exposed (via Layer and Network Management activities). The goals of this component and the interaction with other IPC Process components are clear.
- **Completeness of the specification: 2**: The current specification is just a tool for facilitating early RINA prototypes, but its goal was never to become a serious candidate for a RINA Managed object model.
- **Status of implementations: 4**: The three RINA prototypes available implement this managed object model. Interoperability experimentations have successfully been carried out.

2.7.3. Major issues/Limitations

The current managed object model specification is just a facilitator of early RINA prototyping efforts, it should not be considered a work to be extended. Any serious attempt to define a RINA managed object model must start with a top-down approach, capturing the main entities identified in the RINA reference model as well as the relationships between them. The definition of a managed object model following this approach will be one of the initial tasks of PRISTINE's WP5, reported in deliverable D5.1.

2.7.4. Policies in RIB Object Model

The whole specification can be considered policy, since each DIF could define its own model. However, the goal of RINA is to maximize commonality and minimize invariances whenever possible, and DIFs with common managed object model definitions are the key to facilitate Network Management.

If we focus on a single managed object model specification, it is likely that it will also contain policies; at the very minimum to allow for different concrete syntaxes (such as Google Protocol Buffers, YANG, the multiple ASN.1 concrete syntaxes, etc).

2.8. RIB Daemon

2.8.1. Overview of the RIB Daemon

The RIB Daemon is the logical repository for information relating to the state of the DIF and the IPC Process in particular. As such it can be viewed logically as a partially replicated distributed database. All state information maintained by the IPC Tasks, the Flow Allocator, Resource Allocator, etc. is maintained by and available through the RIB Daemon. This includes all local information on the operational state of the DIF, performance, load, routing update, directory caches, etc. Besides making this information available to the tasks of the IPC Process, it is also the task of the RIB Daemon to efficiently update this information with other members of the DIF and the Network Management System periodically or on certain important events. The information exchanged is necessary to coordinate the distributed IPC. There is no requirement that the same update strategy needs to be used for all information in the DIF. It will be advantageous to use different strategies for different kinds of information and among different subsets of DIF members. Events can cause the RIB to be queried or to update its peers.

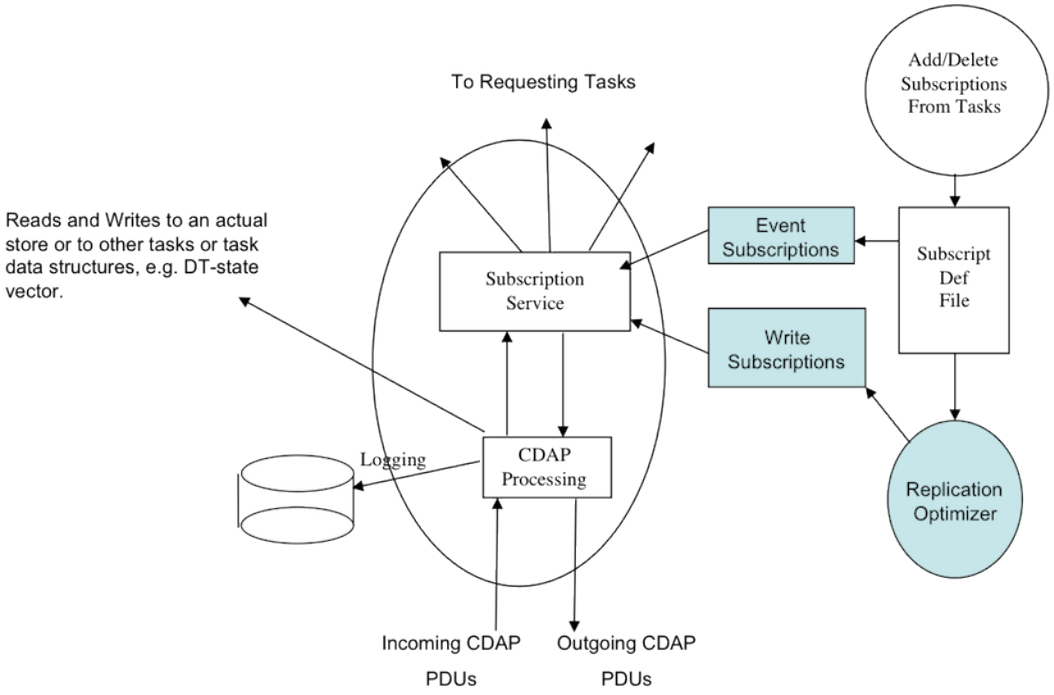


Figure 9. Model of the RIB Daemon operation

It is important to remark that the RIB Daemon is the only user of the CDAP protocol in an IPC Process, as illustrated in the Figure above. The CDAP protocol and one or more AE's (Application Entities) can be used to create a Distributed Application, in which the objects stored in the RIB of an application instance represent its view of,

and/or its portion of, the complete distributed RIB of the distributed application. The local view of the distributed objects' coherency, consistency, and timeliness are under the explicit control of the application by manipulating those properties of the object in the RIB, rather than by programming using explicit communication operations in the application. In this programming model, the CDAP and AE are not explicitly visible to the application, and the application will not use those communication API's directly – its view of the distributed RIB is solely through its own local RIB objects. The layer management tasks of an IPC Process use this programming model, abstracted by the RIB Daemon (who provides the RIB API to layer management tasks and generates the required CDAP PDUs in response to API invocations).

When a Layer Management task needs to perform remote operations information with other IPC Processes in the DIF, it notifies the RIB Daemon the information that needs to be shared (subtree of objects in the RIB), the IPC Processes the information needs to be shared with and the timing requirements for this information sharing. For example, the Flow Allocator can tell the RIB it needs to invoke a *create Flow* operation over an a particular IPC Process as soon as possible; or the PDU Forwarding Table Generator needs to communicate updates of a particular subtree of objects in the RIB to all the direct neighbors of the IPC Process periodically. By centralizing all the operations over remote objects requested by the IPC Process in the RIB Daemon, there is the opportunity to minimize the CDAP traffic caused by these remote operations.

When the RMT receives a layer management PDU for the IPC Process, it passes the payload of the PDU (which contains an encoded CDAP message) to the RIB Daemon. The RIB Daemon decodes the CDAP message, validates that the requested operation can be performed on the requested object (by checking a) the access rights of the requestor in collaboration with the Security Manager, and b) that the object exists), invokes operations on other IPC Process components or an actual data store and informs any tasks interested in the operation over the remote object.

2.8.2. State of the RIB Daemon specification

Although the RIB Daemon is not an externally visible component of the IPC Process - and therefore documenting its behavior is not required for achieving interoperability between different implementations - its key role in layer management makes it worth to explore its internal structure, to document its functions and its relationship with other components.

- **Standalone component: 5**. The responsibilities of the RIB Daemon are very clear: i) interfacing the layer management tasks of the IPC Processes to the RIB

(providing the distributed programming model described in the overview section), ii) optimizing the generation of CDAP traffic in the DIF and iii) notifying the interested components about relevant operations invoked in one or more objects by a remote IPC Process. The interaction of the RIB Daemon with the other IPC Process components is also well understood.

- **Completeness of the specification: 3**. Goals and functions of the RIB Daemon have been identified, including some coarse-grained policies presented below. However, more experience is required with different specifications and implementations of these policies in order to completely understand the challenges and limitations of the RIB Daemon.
- **Availability of implementations: 2**. The three prototypes mentioned in this Deliverable implement the RIB Daemon, with a more or less rich degree of functionality. Implementations have not focused in the optimization of CDAP message generation yet, and the API provided to the layer management tasks needs to be improved once more experience with the RIB programming model is acquired.

2.8.3. Major issues/Limitations

No major limitations have been identified, except for the fact that the component needs to be refined both at the specification and implementation levels. In the former area concrete examples of RIB Daemon policies should be investigated and specified, while in the latter experimentation with these policies is required. PRISTINE will indirectly work in this area, since all the layer management tasks involved in the different research areas of the project will have different update and replication needs for the RIB objects these tasks manage.

2.8.4. Policies in the RIB Daemon

- **Name**: UpdatePolicy.
  - **Description**: Defines how often a set of objects in the RIB need to be updated, performing what remote operations and on which IPC Processes.
- **Name**: ReplicationPolicy.
  - **Description**: Defines how a set of objects in the RIB are replicated (example: fully replicated, partially replicated, not replicated), and over which IPC Processes these set of objects are replicated.
- **Name**: SubscriptionPolicy.
  - **Description**: Links a series of remote operations on one or more objects in the RIB to layer management tasks that want to be informed when these remote

operations occur (for example, the Flow Allocator being informed about *Create* operations on *Flow* objects).

- **Name**: LoggingPolicy.
  - **Description**: Defines what events (operations on remote RIB objects) should be logged, how (what information of the event should be stored) and where.

2.9. Enrollment Task (Enrollment)

2.9.1. Overview of the Enrollment Task

Enrollment is the procedure by which an IPC-process joins an existing DIF. Enrollment occurs after an IPC-Process establishes an application connection (using an (N-1)-DIF) with another IPC-Process, which is a member of a DIF. Once the CDAP Connection is established (see the CDAP specification or the Common Application Connection Establishment Phase specification), this enrollment procedure may proceed.

This procedure, depicted in the Figure below, is intended to minimize the amount of data to send to the “New Member” on the assumption that it had been a member and lost contact with the DIF, either through a crash or failure of the physical media. Other enrollment procedures are possible.

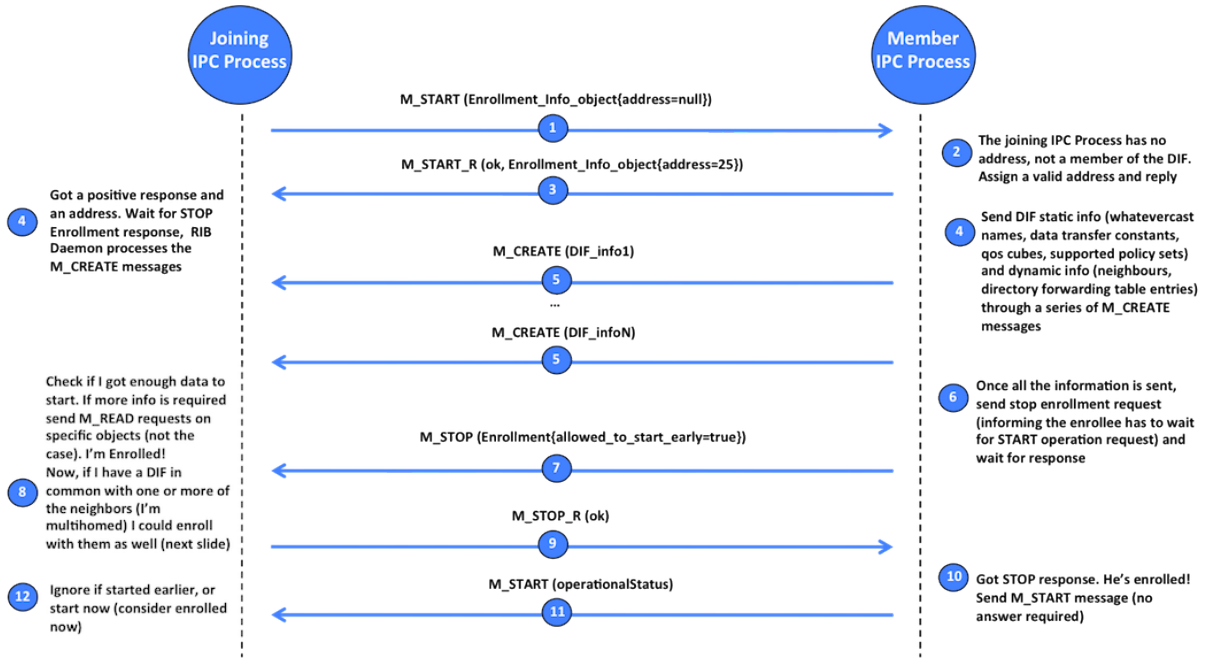


Figure 10. Illustration of the enrollment procedure in the current specification

The Member reads the New Member’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.

2.9.2. State of specification

- **Standalone component: 4**. Enrollment is a well-defined procedure by which an IPC Process joins an existing DIF (or, in case the IPC Process was already a member of the DIF, gets a new neighbor). Relationships with other components - specially with the Namespace Manager and the RIB Daemon - but some details about the relationship of Enrollment with the Flow Allocator (for example, to allocate a reliable flow to simplify the enrollment program in some cases) need to be worked out in more detail.
- **State of the specification: 2**. The current enrollment specification defines a complete enrollment program that is useful in a number of particular DIF environments, but does not completely follow the spirit of other RINA specifications: the document doesn't make an attempt to come up with generic mechanisms that are universally applicable across all possible DIFs, nor does it identify coarse or fine grained policies that allow for the customization of these generic enrollment actions.
- **Availability of implementations: 4**. Both the TRIA and IRATI prototypes implement this enrollment specification, and successful interoperability tests have been carried out.

2.9.3. Major issues/Limitations

The main limitation of this specification is that although it defines an enrollment program that makes sense, is probably applicable to many situations but is not a *real* RINA specification, in the sense that it does not separate generic enrollment mechanisms from policies. Therefore the current specification needs to be reworked following this path, wrapping many of the actions described in the current enrollment document as policies of a generic DIF enrollment framework.

2.9.4. Policies in the Enrollment Task

Although the current specification doesn't separate the proposed enrollment sequence between mechanism and policy, there are a number of steps that can vary from DIF to

DIF, and are therefore candidates to become policies once the enrollment specification is updated. Among these ones there are:

- Different alternatives for ensuring that the enrollment program can complete successfully even if the N-1 DIF providing the flow over which enrollment happens can just provide unreliable flows. Some alternatives could be that the enrollment program implements a *stop and wait* behavior with timeouts for each of the request-response steps; or that the enrollment tasks first requests the allocation of a reliable flow and use it internally to carry out the enrollment information exchange.
- The static or near-static DIF information that the existing DIF member makes available to the new member.
- The checks that the joining IPC Process makes over the information it receives from the existing member, including deciding to request for more information.
- How to deal with other IPC Processes of the DIF that can also be potentially neighbors of the joining IPC Process (because they have an N-1 DIF in common with them). The joining IPC Process could learn about the candidate members and start enrollment with them; or the existing member IPC Process that enrolled the new member could inform the potential new neighbors and instruct them to start enrollment with the new neighbor - just to illustrate two potential solutions for this problem.

2.10. Flow Allocator (FA)

2.10.1. Overview of the Flow Allocator

The Flow Allocator is responsible for creating and managing an instance of IPC, i.e. a flow. It allocates the port-ids and the EFCP-instance. It does not participate in synchronization, which is purely a DTCP function. Logically, there is an instance of a Flow Allocator for each allocated flow (called Flow Allocator Instance or FAI). A Name Space Management function is incorporated in the Flow Allocator to manage the assignment of synonyms, i.e. addresses and to resolve the mapping of (N+1)-names/addresses to (N)-names/addresses of an IPC Process. This function is also accessed during enrollment of a new member of the DIF to obtain a synonym (address) for use internal to the DIF. The policies of the NSM may also delegate blocks of addresses to members of the DIF.

The IPC-API communicates requests from the DAP 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 DAP (the process that has requested the flow) 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 instantiates 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. (This avoids a race condition with between the CDAP Create Flow Response and the first data transfer PDU on the requested flow.)

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 Name Space Management component of the Flow Allocator, the FAI must find the IPC-Process in this 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; and
- instantiating the requested application process, if necessary, and allocating a FAI and port-id in the destination IPC-Process.

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. The whole procedure is illustrated in the Figure below.

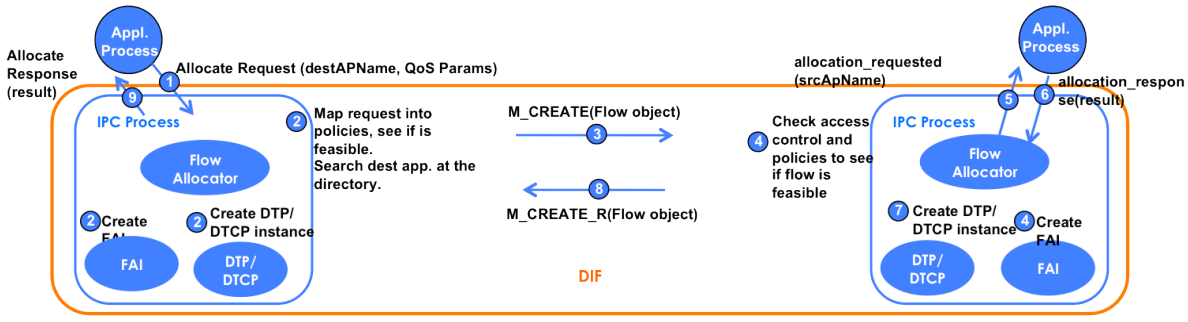


Figure 11. Operation of the Flow Allocator when allocating a flow

2.10.2. State of specification

- **Standalone component: 4**. The goals of the Flow Allocator is to manage the lifecycle of individual flows: creation, monitoring and deletion. The component has been in the reference model since its first versions, but the relationship with other layer management components has been evolving during the last years. In the current reference model and specifications, the FA is managed by the Resource Allocator and interacts with the Namespace Manager and EFCP.
- **Completeness of the specification: 3-4**. The flow creation and deletion functions of the FA have been completely specified, less work has been done on flow monitoring. As with other components, some of the most complex tasks of the FA are performed by its policies, of which to date there are no realistic examples - other than very simple cases to enable demo and experimentation activities.
- **Availability of implementations: 3-4**. The three RINA implementations under development -TRIA, ProtoRINA, IRATI- have a working FA component. The FAs from the TRIA and the IRATI implementations are interoperable. However these implementations feature very simple policies, specially for choosing the specific policies associated to a flow request.

2.10.3. Major issues/Limitations

Work on the FA should be concentrated in two areas: i) the development of more sophisticated procedures for mapping incoming flow requests to specific policies for the EFCP connection(s) supporting the flow; ii) achieving a deeper understanding of the flow monitoring functions - what parameters of the connections implementing the flow should be monitored, what actions can be taken if one or more of these parameters get out of a given range; and iii) understanding more the relationship of the Flow Allocator with other layer management components, specially with the Resource Allocator.

2.10.4. Policies in the FA

- **Name**: AllocateNotifyPolicy.
  - **Description**: This policy determines when the requesting application is given an Allocate_Response primitive. In general, the choices are once the request is determined to be well-formed and a create_flow request has been sent, or withheld until a create_flow response has been received and MaxCreateRetires has been exhausted.
- **Name**: AllocateRetryPolicy.

- **Description**: This policy is used when the destination has refused the create_flow request, and the FAI can overcome the cause for refusal and try again. This policy should re-formulate the request. This policy should formulate the contents of the reply.

- **Name**: NewFlowRequestPolicy.
  - **Description**: This policy is used when converting an Allocate Request into a create_flow request. Its primary task is to translate the request into the proper QoSclass-set, flow set, and access control capabilities.

- **Name**: SeqRollOverPolicy.
  - **Description**: This policy is used when the SeqRollOverThres event occurs and action may be required by the Flow Allocator to modify the bindings between connection-endpoint-ids and port-ids.

2.11. NameSpace Manager (NSM)

2.11.1. Overview of the Name Space Manager (NSM)

Managing a name space in a distributed environment requires coordination to ensure that the names remain unambiguous and can be resolved efficiently. The 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. 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.

When an IPC Process joins a DIF, one of the first things the enrollment procedure must do is to check if the new member has an address that is valid within the DIF and, if required, assign a valid address to the new member. These two tasks are carried out by the NSM, which keeps track of the addresses in use in the DIF and implements the DIF's address allocation policy - which can follow any approach from centralized to hierarchical to totally distributed.

For an application process to be the target of a flow allocation request, it must be known or knowable to IPC. This procedure is called *application registration*; which can be done explicitly - the name of the application is registered in the DIF - or implicitly - the application is given the rights to access the DIF and the IPC Process will search for it when it receives a flow allocation request. Either way the NSM must keep a mapping between the application name and the IPC Process address.

During flow allocation an application process requests a flow to another application process by name. The first task of the DIF, performed by the flow allocator, is to locate the destination application process - which is performed by resolving the application name with the assistance of the NSM. In order to carry out this mapping, the NSM generates and maintains a Directory Forwarding Table (DFT), which returns an IPC Process address given application process name. The returned address belongs to the next IPC Process to look at. The search finishes when the returned address is the same as the one of the IPC Process the DFT resides - which means that the destination application is registered there.

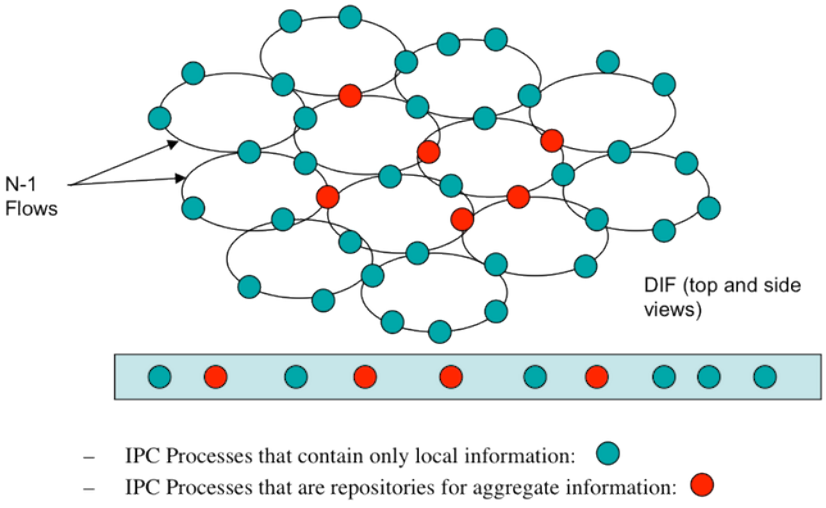


Figure 12. Example organization of a the NSM within a DIF

The Figure above shows an example of the organization of an NSM within a DIF. Circles are IPC Processes, all of them maintaining a DFT with local entries (mappings of the applications that are locally registered) and entries that will cause the *Allocate Request*

to be forwarded to one of the IPC Processes that have been specialized to maintain a repository of non-local mappings (red circles in the Figure). Therefore, if an Allocate Request cannot be resolved locally, it is forwarded to a *repository IPC Process* who will either forward it to the system where the destination application process is located or to another *repository IPC Process* (depending on the strategy followed for organizing the information on the repositories, which can be caches, hierarchical, DHTs, complete replications, etc).

2.11.2. State of specification

- **Standalone component: 4**. The reference model provides a clear description of the goals of this component, as well as its interaction with other IPC Process Components such as the Enrollment Task and the Flow Allocator.
- **Completeness of the specification: 3**. The major functions of the component are described in the RINA reference model, and a few coarse-grained policies can be inferred from this description.
- **Availability of implementations: 3**. The TRIA and IRATI prototypes implement a NSM that features static address assignment, and fully replicates the distribution of the DFT (so that all IPC Processes have all the application-process name to IPC Process address locally available - only doable in small DIFs).

2.11.3. Major issues/Limitations

Although not specified in a separate document, the reference model provides a description that is sufficient to understand the goals, main functions and coarse-grained policies of the NSM. In order to consolidate the specification, define a more detailed description and probably identify finer-grained policies is is necessary to work out all the details a number of complete examples in very different environments (such as a mobile ad-hoc DIF and, a top-level ISP DIF or a backbone DIF).

2.11.4. Policies in the NSM

Since the DIF NSM is not yet completely specified, some of the policies described below could be modified or broken down into finer-grained policies once research and development activities focused on the NSM advance.

- **Name**: AddressValidationPolicy.
  - **Description**: Procedures used to validate the validity of an address within a DIF. A myriad of options from centralizing this functionality in a single IPC Process to totally distributed [weniger] are available.

- **Name**: AddressAssignmentPolicy.
  - **Description**: Procedures used to assign a valid address (unique within the DIF) to an IPC Process. Policies can cover the range from totally centralized (with an IPC Process centralizing address allocation - approach somewhat equivalent to DHCP - Dynamic Host Configuration Protocol), to hierarchical (a number of IPC Processes are assigned different address ranges, which they use to assign addresses to IPC Processes in their *area*), to totally distributed (such as the approach presented in [thopian]).
- **Name**: DFTReplicationPolicy.
  - **Description**: Describes how the information of the DFT has to be replicated: what parts and to what other IPC Processes in the DIF. Choices could go from not replicating the information at all (so that each IPC Process only has local knowledge and therefore a search strategy has to be followed), to fully replicated (using some sort of controlled broadcast such as in link-state routing), to Distributed Hash Tables (DHTs) to hierarchical databases (such as DNS, the Domain Name System, in which some *repository* IPC Processes are responsible for keeping the mappings of a subset of the hierarchy).
- **Name**: DFTGenerationPolicy.
  - **Description**: Uses the DFT information received from neighbor IPC Processes and some *search rules* in order to generate the DFT of an IPC Process.

2.12. Routing

2.12.1. Overview of 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 table. To support flows with different QoS will in current terminology require using different metrics to optimize the routing. However, this must be done while balancing the conflicting requirements for resources. Current approaches can be used but new approaches to routing will be required to take full advantage of this environment. The choice of routing algorithms in a particular DIF is a matter of policy.

Routing in RINA is all policy; that is, each DIF is free to decide on the best approach to generate the PDU forwarding table for its environment. Particular approaches will have its own, separate policy specification (for example, the FP7 IRATI project is investigating a routing policy based on Link-State routing [irati-ls]). In some DIFs

there may not be routing at all, if the forwarding strategy doesn't require the use of this function (for example, in the case of "pure" topological addresses with a distance function). The maximum that a generic routing specification could do is to define the environment in which the routing component operates, identifying the main events that may influence its behavior.

2.12.2. State of specification

- **Standalone component: 4**. Goals of routing are clear, but more investigation is required to try different routing policies and, for each one, analyze which is the relationship between Resource Allocation and Routing. This research would allow a clearer understanding of the role of routing with respect to resource allocation, since there appears to be some overlap for certain routing policies (for instance, link state routing).
- **Completeness of the specification: 2**. The reference model identifies the goals of routing, and provides a brief overview of its application in the DIF environment. However, a specification that provided more details of such an environment and discussed a number of alternative policies describing its potential target environments would help clarifying the dimension of this IPC Process component.
- **Availability of implementations: 2-3**. The IRATI prototype has implemented a Link-State routing policy based on exchanging the state of N-1 flows with the nearest neighbors, using this information to construct a link-state database and applying Dijkstra to compute a routing table for a single metric. ProtoRINA is experimenting with both link-state and distance-vector routing policies [wang].

2.12.3. Major issues/Limitations

As of today only LinkState Routing has been specified as a routing policy for a DIF. Multiple other options are possible and its adaptation to the DIF environment should be investigated.

2.12.4. Policies in Routing

Multiple routing policies are possible within a DIF, each of which provides also a set of choices in order to further adapt the specific strategy to the DIF operational environment. For example, a routing policy based on LinkStateRouting may define multiple options that: i) control the connectivity information exchanged between IPC Processes; ii) specify under what events this information is exchanged/disseminated or iii) identify the metrics and the algorithm used in the route computation procedure.

2.13. Resource Allocator (RA)

2.13.1. Overview of the Resource Allocator (RA)

The Resource Allocator gathers 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 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 DIF Management System (DMS), 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 Resource Allocator is responsible for ensuring that the operation of data transfer within the IPC Process remains within (and meets) its operational requirements. The current RA notes (they do not provide enough detail to be considered a full specification) consider the meters and dials available, but will not consider how it might actually be achieved. Policies might run the gamut from coarse, such as avoid congestion, to much finer such as provide several distinct QoS classes.

The meters

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 neighbors or some other subset - all two or three hop neighbors - 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 dials

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 doesn’t 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 neighbors, depending on the traffic load offered to the IPC Process and other conditions in the DIF.
- **Assignment of RMT Queues to (N-1)-flows**. The RA implements this mapping with input from the PDU Forwarding Table Generator.
- **Forwarding Table Generator Output**. The RA takes input from other aspects of management to generate the forwarding table. This is commonly thought of as

the output of “routing.” It may well be here, but we want to leave open approaches to generating the forwarding table not based on graph theory.

2.13.2. State of specification

- **Standalone component: 4**. The goals of the resource allocator are clear, as the main responsible of activities related to resource allocation in an IPC Process. As such, the resource allocator interacts with other layer management tasks like the Flow Allocator, the RMT or the PDU Forwarding Table Generator. However, it is yet not completely clear which are the boundaries of those tasks and the resource allocator, which requires further exploration of the RA under different use cases.
- **Completeness of the specification: 2**. Since right now the RA component is defined as almost all being policy, the specification just provides a light framework to characterize the operation of the Resource Allocator. It is yet to be seen if there is more commonality that can be imposed amongst all the possible uses of the RA and therefore be incorporated in the specification (such as common events of interest to the RA, base RIB objects applicable to any DIF, etc.).
- **Availability of implementations: 1**. The current RINA prototypes do not provide an implementation of the functions under the scope of the Resource Allocator.

2.13.3. Major issues/Limitations

More research into the functions of this component is required in order to refine the description of the Resource Allocator. In particular, the interactions of the RA with the Flow Allocator, the RMT, routing and EFCP need to be investigated in more detail, to better understand possible overlaps between the functions carried out by the different components; as well as fine-grained policies were specific behavior can be plugged in. PRISTINE will be contributing to this goal by investigating this component under the congestion control and IPC Process resource dimensioning research areas.

2.13.4. Policies in the RA

It remains to be seen if there can be common, well-defined, fine-grained policies that are generic enough to be applicable to all the possible DIFs that can exist. At the coarsest level, this component as a whole can be considered policy.

2.14. Security Manager

2.14.1. Overview of Security Management

Security Management is the IPC Process component responsible for implementing a consistent security profile for the IPC Process, managing all the security-related functions and also executing some of them.

- **Authentication**. Procedure by which the identity of an IPC Process is validated. Executed by CACEP in conjunction with the Security Manager.
- **Access control**. Decide whether external entities are allowed to access IPC Process resources. There are three main types of access control decisions that an IPC Process has to perform: i) decide if an IPC Process is allowed to join a DIF; ii) decide if an application is allowed to request a flow to another application and iii) decide if other IPC Processes (or the DIF Manager via the Management Agent) are allowed to invoke an operation over an object in the RIB.
- **Confidentiality and integrity**. Ensure that PDUs exchanged with other IPC Processes are resistant to overhearing and tampering by any of the N-1 DIFs that are supporting the operation of the IPC Process.
- **Auditing**. Procedure by which a representative number of the IPC Process activities are monitored with the goal of detecting abnormal behaviours and triggering the appropriate actions in case that happens.
- **Credential Management**. Comprises all the functions to manage any credentials required to perform the former security-related procedures. Credential management includes the following functions: credential registration, generation, verification, distribution and maintenance (including revocation).

2.14.2. State of specification

- **Standalone component: 4**. Goals of the security manager are clear, but the details on the interaction with other IPC Process components (specially CACEP, SDU Protection, the RIB Daemon and the Flow Allocator) need to be worked out in more detail.
- **Completeness of the specification: 1**. Only the reference model briefly describe the Security Management component of an IPC Process, but to date there is no specification available yet.
- **Availability of implementations: 1**. The current prototypes don't implement any of the security functions of an IPC Process.

2.14.3. Major issues/Limitations

The security manager of an IPC Process has not been specified yet, it is just briefly introduced in the reference model. Therefore this is an area where a significant amount of work has to be carried out, both in describing the generic behavior of this component as well as in identifying fine-grained policies.

2.14.4. Policies in Security Management

Although the security management component has not been properly specified yet, the following policies are likely to belong to this area. Further research on the topic will stabilize this list, possibly breaking down some of the current definitions into finer-grained policies.

- **Name**: NewMemberAccessControlPolicy.
  - **Description**: Decides whether an authenticated IPC Process can join the DIF.
- **Name**: NewFlowAccessControlPolicy.
  - **Description**: Decides whether a new flow to an application process registered in the IPC Process is accepted.
- **Name**: RIBAccessControlPolicy.
  - **Description**: Decides whether an operation on an object in the RIB is accepted.
- **Name**: AuditingPolicy.
  - **Description**: Identifies what are the main sources of information that the Security Manager will analyze for internally auditing the operation of the IPC Process (EFCP traffic, RIB Daemon events, etc). It also defines how the information is processed to decide if abnormal behaviours are going on, and the actions to be taken in case these behaviors are detected.
- **Name**: CredentialManagementPolicy.
  - **Description**: Actions that control how credentials are managed in a distributed fashion through the DIF (including validation, assignment, revocation, etc). Options range from a policy that delegates all the work to a central authority (such as the Network Manager), to more distributed approaches in which part of all of this functions are executed by IPC Process autonomously.

2.15. Summary of the specifications analysis

The following Table summarizes the analysis of the RINA specifications, highlighting the numerical marks given to each one of the criteria defined for the analysis

(standalone component, completeness of the specification and availability of implementations).

Table 4. RINA specifications analysis summary table

Specification	Standalone component	Completeness of the specification	Availability of implementations
Delimiting	5	3	1
EFCP	5	4	2
RMT	5	3	3
SDU Protection	5	3	1
CACEP	5	4	4
CDAP	5	3	3
RIB Daemon	5	3	2
RIB Object Model	5	2	4
Flow Allocator	4	3	4
Enrollment	4	2	4
NameSpace Manager	4	3	3
Routing	4	2	2
Resource Allocator	4	2	1
Security Management	4	1	1

The first criterion (*Standalone component*) tried to measure the maturity of the IPC Process structure, analyzing the degree to which the separation of the IPC Process tasks into the different components described in the specifications is consolidated. All the components achieve a high mark in this criterion (4-5), highlighting that the IPC Process components are well-identified and the relationships between them are properly understood in most cases. However, a more detailed knowledge of some components such as the Resource Allocator or Security Management and its interactions with other parts of the IPC Process is still required.

The marks of the second criterion (*completeness of the specification*) show that there is a lot of work to do in this topic. EFCP, CACEP and CDAP - the core protocols of the

IPC Process - are the specifications that are more mature; pending the specification of different policies in the case of EFCP. A second group of specifications comprising Delimiting, the RMT, the RIB Daemon, the Flow Allocator or the Name Space Manager have a lower level of maturity. This is due to the fact that their goals are clear, but the specification of their functions is not complete and only coarse-grained policies have been identified. Finally, the less developed specifications are those of the Resource Allocator, the Managed Object Model, Enrollment, Routing or Security Management.

Finally, as it was expected, the lower marks belong to the last criterion (*availability of implementations*), since this is the area that requires more work (implementation requires a detailed enough specification in the first place). The current prototypes have focused in the implementation of the RMT, DTP (the Data Transfer Protocol of EFCP), CACEP, CDAP, the RIB Daemon, the Enrollment Task and the Flow Allocator, with IRATI and protoRINA also providing initial implementations of routing.

Section 3 is going to identify how PRISTINE will address some of the shortcomings identified in this analysis from the specification, simulation and implementation points of view.

3. PRISTINE Research and Development Areas

Section 3 provides an overview of the work that will be carried out by each of the PRISTINE research and development areas. As part of the overview, the targeted components of the RINA architecture and their policies for each research area are identified. A summary is provided in the table below.

Table 5. RINA components targeted by each research area

Process	Component	Research/ Development area(s)
IPC Process		
	EFCP	Congestion Control
	RMT	Congestion Control, Resource Allocation
	SDU Protection	Authentication, access control and confidentiality
	RIB object model (layer management)	Congestion Control, Resource Allocation, Routing and Addressing
	RIB object model (network management)	Network management
	CACEP	Authentication, access control and confidentiality
	CDAP	Network management
	Enrollment	Resiliency and High availability
	Flow Allocator	Congestion Control, Resource Allocation
	Namespace Manager	Routing and Addressing
	Routing	Routing and Addressing
	Resource Allocator	Congestion Control, Resource Allocation, Resiliency and High availability

Process	Component	Research/ Development area(s)
	Security Manager	Security Coordination
Management Agent		Network management
Manager		Network management

3.1. Congestion Control

3.1.1. Overview

PRISTINE's RINA congestion control operates on traffic aggregates between edges of DIFs (we abbreviate congestion control with "CC" and aggregate congestion control with "ACC"). This makes it possible to control congestion inside the network, where it occurs, rather than requiring a control loop that spans across a potentially large amount of heterogeneous infrastructure (as with today's TCP-based Internet congestion control).

The most basic way of dealing with CC in RINA is the following. Each DIF supports a number of flows, which provide IPC services to the users on top (users of flows can be end user applications or IPC Process of other DIFs). Each flow in the DIF is implemented via one or more EFCP connections, which support flow control (window-based or rate-based). Processing of EFCP connections is only performed at the IPC processes that are the endpoint of these EFCP connections, as shown in the next Figure. IPC Processes in the route followed by EFCP PDUs just rely them to the destination IPC Process. In its way to destination EFCP PDUs go through a number of queues (buffers), used by the RMT to relay those PDUs over multiple N-1 flows. It may happen that those buffers grow bigger than certain threshold(s), indicating a congestion condition. If this is the case, the RMT will mark the EFCP PDUs going through these buffers with an Explicit Congestion Notification (ECN) mark, consisting in one or more bits (depending on the 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. The sending EFCP protocol machine will reduce the sending rate, also causing the user of the flow to push-back if required (this push-back can be done by blocking the user of the flow or by signaling an *error* condition in the *write to flow* API calls).

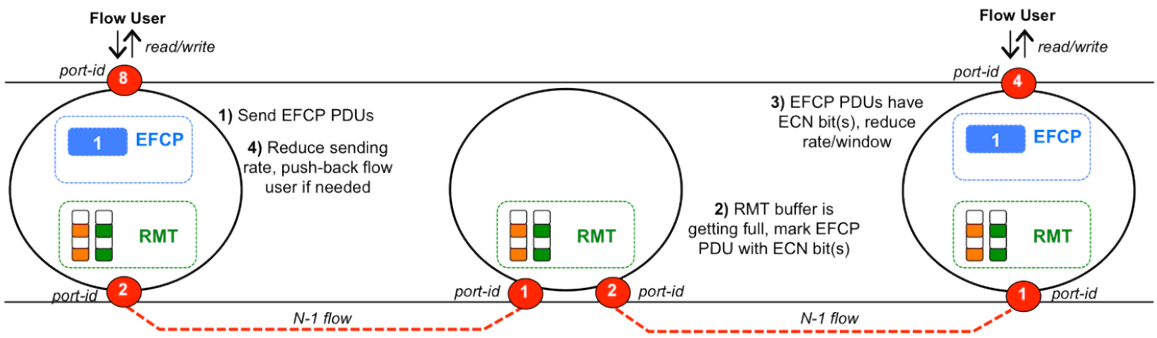


Figure 13. Dealing with congestion in a DIF

This solution involves a separate congestion control loop per EFCP connection in the DIF. Since the behavior of the individual control loops is not coordinated, the combined response to congestion of all the individual control loops may not be optimal, requiring more time to recover from congestion and causing some connections to reduce the sending rate more than strictly necessary. Task 3.1 will explore a way to improve this behavior, by defining aggregate loops that react to congestion in a coordinated fashion. Therefore, instead of each individual "EFCP connection receiver" telling the "sender" which should be its sending rate, the aggregate will compute the rates for a number of senders and receivers.

Task 3.1 will specify a set of policies that affect different DIF components, and that work in conjunction to achieve the desired congestion control effects. These set of policies can vary from DIF to DIF, effectively allowing per-DIF congestion control schemes that optimally adapt to the DIF operating requirements (as specified by the analysis of the PRISTINE use cases in Work Package 2). The interactions between the multiple congestion control-loops will be analyzed and studied using the simulator provided by Task 2.4. Comparison against the base case will show the benefits/drawbacks of the proposed aggregate-based congestion control strategy.

3.1.2. RINA components and policies in scope of congestion avoidance research

- **Component**: Error and Flow Control Protocol (EFCP)
  - **Policy**: *Transmission Control Policy*, an algorithm to compute the new sending rate (or window size) based on the information on the ECN bit(s) - in the basic case.
  - **Policy**: *Flow control* and *retransmission control* policies in general will also be reviewed in order to better understand their relationship with congestion control/congestion avoidance.

- **Component**: Relaying and Multiplexing Task (RMT)
  - **Policy**: *RMTQMonitorPolicy* and *MaxQPolicy*. It is not yet completely clear to which policies the required behavior belongs, but the RMT at least needs to detect one or more congestion conditions (for example multiple thresholds of buffer sizes) and then set the ECN bit(s) accordingly in the affected EFCP PDUs. If the buffer grows after a certain limit, the RMT needs to start dropping PDUs using a certain algorithm.
- **Component**: Resource Allocator
  - **Policy**: Fine-grained policies of the resource allocator are not identified yet, but this is the component in charge of the creation, maintenance and deletion of congestion control aggregates (CCA)s. It has to decide when to create a new CCA, what connections have to be part of the aggregate, exchange information with other IPC Processes in the aggregate to maintain the distributed aggregate state and compute sending rates for the connections that are part of the aggregate under congestion conditions.
- **Component**: Flow Allocator
  - **Policy**: *NewFlowRequestPolicy*. Each new EFCP connection must be mapped to a congestion control aggregate.

3.2. Resource Allocation

3.2.1. Overview

Introduction to resource allocation in a DIF

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 an efficient use of the resources of the DIF. DIFs provide a unified model for resource allocation that does not need to distinguish between a *connection-oriented* and a *connectionless* mode of operation: both are the extreme cases of a continuum space. To understand how this works, we will start by reviewing the flow allocation procedure.

When application instances request a flow to a DIF, they provide a set of requirements the flow must support in order to be useful for the application - the call to allocate a flow can be thought as *allocateFlow(destination_app_name, flow_characteristics)*. These requirements are expressed by providing a set of values and/or ranges for a number of

parameters such as minimum rates, maximum loss, maximum delay, reliable delivery of SDUs, etc. Flows are requested by application instances. The destination application name identifies the application instance(s) that are the recipient(s) of the flow request. The name can resolve to:

- A particular instance of a particular application
- Any instance of a particular application
- A number of instances of a particular application (according to some rule - could be all the members of a distributed application, or a certain subset)

The flow is just the service provided by the DIF, an abstraction that provides a communication channel with certain characteristics to an application instance. Flows are "implemented" within a DIF by EFCP connections. A single flow may be supported by one or more EFCP connections. An EFCP connection is an association between two EFCP protocol machines (at two IPC Processes) implementing the Error and Flow Control Protocol. The Relaying and Multiplexing task is in charge of multiplexing PDUs from multiple EFCP protocol machines into one or more N-1 flows.

When an IPC Process receives the flow allocation request, it has to classify to which of the existing QoS cubes (classes) the flow belongs to - in case there is at least one QoS cube that supports the application requirements. Each QoS cube has a predefined range of policies (for the EFCP and the RMT, basically) designed to guarantee certain outcomes for the flows belonging that class. Some of the policies in the QoS cube can be further customized via a number of policy-specific parameters, allowing for a certain differentiation in the outcomes provided by flows belonging to the same cube. For example, if one of the policies of the QoS cube dictates that connections have to use sliding-window flow-control, two different connections belonging to the same class could have different values for the initial and/or maximum window sizes.

Each EFCP connection is configured with a number of policies provided by the QoS cube, and assigned the QoS cube's qos-id. The qos-id is one of the fields that are present in the PCIs of all the data transfer EFCP PDUs. Once the flow has been allocated, EFCP PDUs carrying user data are delivered to the RMT in order to be relayed to the IPC process at the other "end" of the EFCP connection. In order to do so, the RMT looks up the PDU forwarding table in order to find out which is the N-1 flow the PDU has to be written to - the PDU Forwarding Table provides a mapping of <*destination address, qos-id*> to *list of N-1 port-ids the PDU has to be written to*. This decision is based on the destination address and qos-id associated to the PDU. Once the N-1 flow has been identified, the EFCP PDU is put in a queue before being written to the flow. There can

be multiple queues for each N-1 flow (the most common case being either one or one per qos cube), to allow for differential treatment of several classes of traffic.

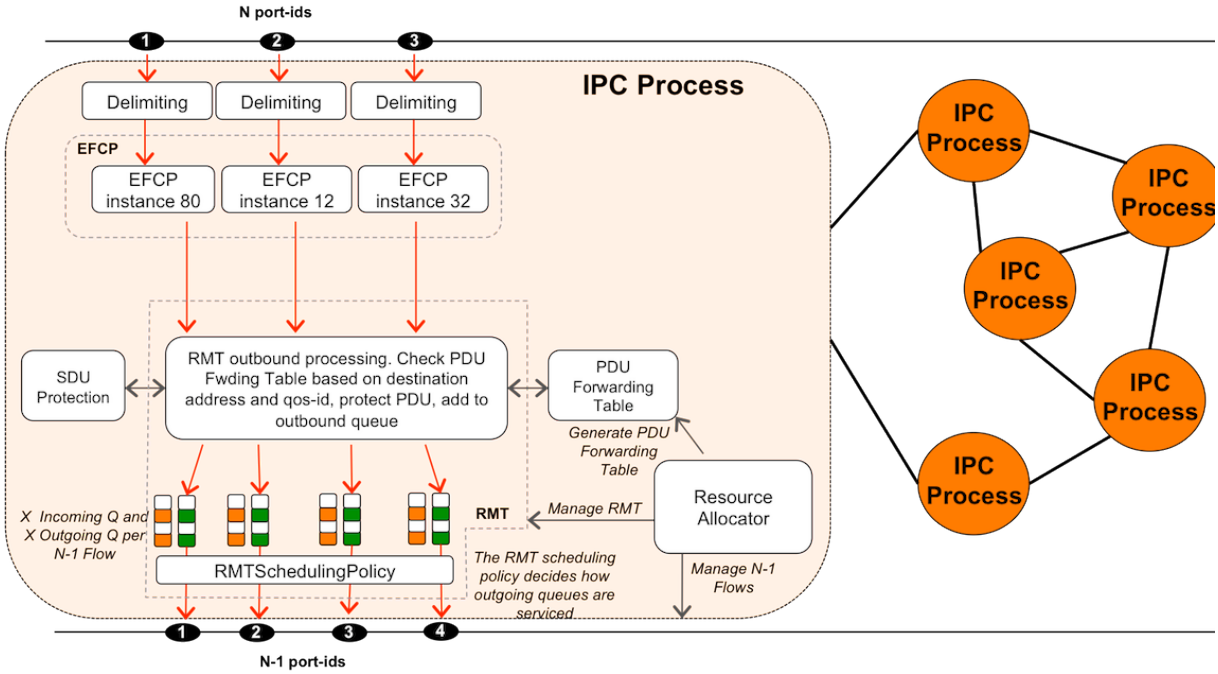


Figure 14. IPC Process, components relevant to resource allocation

Resource allocation controls the number of QoS cubes available in each IPC Process, the assignment of flows to QoS cubes (via the Flow Allocator), the generation of the PDU forwarding table (with the assistance of the PDU forwarding table generator), the number of RMT input/output queues per N-1 flow and the policies servicing these queues. Therefore, the **key question** that influences the outcomes perceived by the applications and the efficiency in DIF resource utilization is: **how are the resources of the DIF distributed among the different flows?**

Distributed resource allocation model

In addition to the resources required for EFCP processing at the sending and receiving IPC Processes, the resources of an IPC Process that need to be allocated to the different flows are the ones of the Relaying and Multiplexing Task: buffer space and scheduling capacity. The RMTs of the different IPC Processes in a DIF have two prerequisites in order to be able to successfully process the PDUs of the different flows supported by the DIF: i) find out which is the next hop for a PDU and ii) have enough resources (buffer space, scheduling capacity) allocated to be able to process the PDU. Therefore, both connectivity and resource allocation information has to be effectively distributed among DIF members. Keeping in mind our goal of characterizing "connectionless" and "connection-oriented" operation as just the **two extremes** in the distributed resource allocation model, we can see that:

- The "pure connectionless" mode does not differentiate among the flows supported by the DIF and therefore does not require differential allocation of resources in each IPC Process: the overall buffer space and scheduling capacity is **equally shared** among all the PDUs, since there is a single class of service. Allocation of resources is performed once at the beginning and ignored afterwards, therefore there is no need to exchange resource allocation information during the operation of the DIF. Routing information is distributed to all the IPC Processes in the DIF, and routing is performed in each IPC Process. If one IPC Process fails, any other IPC Process can perform its functions since they all have the knowledge to process all the PDUs that are travelling through the DIF.
- In the "pure connection oriented mode" resources are dynamically allocated for each new flow at setup time, but only at the IPC Processes that are at the in the path transversed by PDUs belonging to that flow. Routing is only performed at the edges during connection setup. In this mode of operation each flow is allocated its own resources, but only in the IPC Processes of a single path between source and destination (there can be variants of this "pure approach" if protection options are provided). If one of the IPC Processes in the path of the connection fails, recovery is complex since resource allocation has to be re-done.

Between these two extremes there are a wide range of resource allocation configurations that can be applied to different scenarios, supporting their operation requirements. Analyzing these two extremes it is clear that both routing and resource allocation have to be treated as a distributed problem, which means that:

- Routing and resource allocation information has to be distributed among the members of the DIF.
- Based on the information learned from other members of the DIF, IPC Processes can dynamically modify forwarding and resource allocation to adjust to the DIF's operating conditions.

Current approaches to distributed routing and resource allocation have focused on distributing routing and resource allocation information, for example using link-state routing protocols such as Intermediate System to Intermediate System (IS-IS) [is-is] or Open Shortest Path First with Traffic Engineering extensions OSPF-TE [rfc3630]. The information disseminated by these type of protocols (which usually includes information on resource usage/availability per class of service) is then used to recompute the forwarding table. However, resource allocation is only modified when new connections are setup or torn down (such as LSPs using the Resource reSerVation Protocol with Traffic Engineering Extensions (RSPV-TE) [rfc3209] in Multi-Protocol

Label Switching (MPLS) networks [rfc3031]). Therefore, another strategy that can be explored is that IPC Processes modify the resources allocated to the different traffic classes dynamically based on the DIF's current utilization of resources.

A way of modelling distributed resource allocation in order to answer the question **how are the resources of the DIF distributed among the different flows?** is the following. Each IPC Process has resources to allocate, which will be used by a number of EFCP connections with certain attributes. Each IPC Process in the DIF computes the probability that is on the path of the connection and allocates resources accordingly. Resource allocation can be made at various degrees of granularity, depending on the degree of traffic differentiation that the DIF may perform:

- **No differentiation**. All traffic is assumed to be from the same type and treated the same. Resource allocation is done uniformly, without differentiating between different QoS classes or flows. This is the equivalent of having a single FIFO queue at the RMT for each N-1 flow.
- **Resource Allocation at the class level**. Each DIF provides a number of QoS Cubes (classes), and all flows are mapped to a single class. Resource allocation is performed at the granularity of the QoS cube, and PDUs classified and processed based on their QoS id.
- **Resource Allocation at the flow level**. Each flow can be treated differently, and resource allocation is performed at the granularity of EFCP connections. RMTs process PDUs based on their connection-ids.

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.

Task 3.2 will explore the machinery required to apply this approach to resource allocation, focusing on the dimensioning of the RMT resources using different multiplexing models, as well as modelling the information exchanged by IPC Processes in order to perform efficient online distributed resource allocation. Task 3.2 will deal with resource allocation at the class level, since this approach allows to support applications with different requirements in the same DIF, and has less overhead than resource allocation at the flow level.

RMT multiplexing models

The RMT multiplexing models - queuing disciplines and scheduling algorithms - have a strong influence on the outcomes (throughput, loss, delay, jitter) provided by the flows supported by the DIF. These multiplexing models also influence the isolation among the different flows, the differential treatment that each flow can receive, the behaviour of the system under saturation and the efficiency in overall resource utilization. Within Task 3.2 PRISTINE plans to investigate the following models: finite FIFO queues, weighted fair queueing and urgent/cherishing multiplexing (based on the quality attenuation concepts [reeve]).

Finite FIFO queues

Finite FIFO queues provide a basic model for multiplexing traffic over an N-1 flow, since it does not allow for traffic differentiation. All the traffic is processed in the order that they enter the queue. Once the queue is full, packets are dropped. This multiplexing model does not provide isolation between flows and its behavior under saturation conditions (when the offered load is higher than what the system can process) is not predictable.

Weighted Fair Queuing

The Weighted Fair Queuing (WFQ) family of algorithms provides some degree of isolation and differential treatment to packet flows using it. In WFQ the incoming traffic is allocated to one of several queues that, in the worst case, allow each traffic category to get a share of the outgoing link capacity. Each queue in the WFQ system can be viewed as a FIFO with an associated minimum and maximum service rate (the maximum being typically of 100%). WFQ manages bandwidth and as such provides some degree of flow isolation and differential treatment - each flow gets its own share of the outgoing link capacity-, but it cannot guarantee bounds on the packet loss and delay experienced by the flows using the mechanism under saturation conditions [davies].

Urgent/cherishing multiplexing (delta-q)

The urgent/cherishing multiplexing model [holyer] is originated from trying to understand how to deliver predictable quality in networks working close to or at saturation conditions (when the offered traffic load is close to or even higher than the network's traffic processing capacity). This work originated the delta-Q model for reasoning about network quality, in which quality attenuation is defined as the quality degradation (in terms of loss and delay) that each packet flow perceives in transversing the network.

The delta-q model recognizes that in any queueing system there is a relationship between loss rate, throughput and delay; meaning that these systems only have two degrees of freedom:

- For a fixed loss rate, reducing delay means that throughput must decrease.
- For a fixed throughput, reducing delay means an increase in loss rate.
- For a fixed delay, reducing loss rate will require a reduction in throughput.

Each network element in the path of a packet introduces a certain quality attenuation, which cannot be *undone*. Part of this quality attenuation is due to the fact that packets contend for shared resources with other packets (queues and scheduling capacity). Quality attenuation cannot be undone, but it can be differentially allocated to the different flows transversing a network element. Therefore, for a fixed throughput, loss and delay can be differentially allocated to different traffic classes. This differential allocation is achieved by the urgent/cherishing multiplexing model, which defines two explicit orderings: loss and delay. Both are combined to provide an overall quality partial order.

Table 6. Example of urgent/cherishing multiplexing with 9 categories and Best Effort traffic.

A1	A2	A3	-
B1	B3	B3	-
C1	C2	C3	-
-	-	-	BE

The Table above shows an example of the urgent/cherishing multiplexing with 9 categories of quality traffic and Best Effort traffic. The different columns define an order for delay requirements, keeping the same loss: traffic of the category A1 will be served with strict priority over traffic of the category of A2, A2 over A3, etc. In contrast, different rows define different levels of loss, with a common delay: traffic of category A1 will experience less loss than traffic of category B1; B1 less loss than C1, etc. This generalizes to N*M categories (plus best effort). The full details on how to mathematically model the system and calculate the quality attenuation that traffic flows will experience are provide in [reeve].

3.2.2. RINA components and policies in scope of resource allocation research

- **Component**: Relaying and Multiplexing Task (RMT)

- **Policy**: *RMTQMonitorPolicy* and *MaxQPolicy*. Depending on the RMT multiplexing model used, different actions have to be taken when new PDUs are placed in the queues and/or the queues reach certain thresholds.
- **Policy**: *RMT-SchedulingPolicy*. Different policies will be tried, depending on the multiplexing model used.

- **Component**: Flow Allocator (FA).
  - **Policy**: *NewFlowRequestPolicy*. The new flow has to be mapped to one of the QoS cubes defined. Some interaction with the resource allocator may be required in order to decide how the connections of the flow will be spread among different paths through the DIF.
  - **Policy**: *AllocateRetryPolicy*. If the parameters requested for a flow are feasible by the IPC Processes at the *source* but are not acceptable by the destination *IPC Process*, the source Flow Allocator may reformulate the request and try again.
- **Component**: Resource Allocator (RA).
  - **Policy**. The current RA specification has not defined fine-grained policies for this component yet. Task 3.2 will investigate what resource allocation information has to be exchanged between IPC Processes to be consistent with the distributed resource allocation model described above, under what events, and what actions have to be taken when an IPC Process receives updated resource allocation information.

3.3. Routing and Addressing

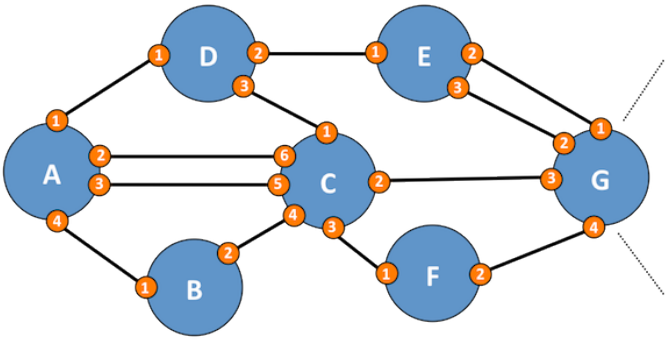
3.3.1. Overview

Introduction to addressing and routing in RINA

Addressing is the operation by which an IPC Process that is a member of a DIF is assigned one or more synonyms for the IPC Process to use in the DIF. These synonyms, designed to facilitate the efficient operation of the IPC Process, are called addresses. The *Namespace Manager* is the IPC Process component that is in charge for maintaining and operating (assigning, revoking and recovering addresses) the address namespace in a given DIF. An IPC Process gets assigned an address when it first joins the DIF, but it can get new ones during the lifetime of the IPC Process within the DIF for various reasons (mobility or DIF partitions, for example).

Routing is an optional procedure that supports the forwarding mechanism. Each IPC Process has to solve the **forwarding problem**: given a set of EFCP PDUs and a

number of N-1 flows, to which flow should each PDU be forwarded? Routing is a way of solving this problem, in which the routing procedure exchanges information with other IPC Processes in the DIF in order to generate a next-hop table for each PDU (usually based on the destination address and the id of the QoS class the PDU belongs to). The next-hop table is then converted into the PDU Forwarding Table with input from the resource allocator, by selecting an N-1 flow for each "next-hop". However forwarding doesn't always require routing, and can be sometimes solved with the data at the PCI of EFCP PDUs and local information.



Destination @	Next hop @
A	C
B	C
C	C
D	E
E	E
F	F

IPC Process G next hop table

Destination @	Forward to
A	3
B	3
C	3
D	2
E	1
F	4

IPC Process G forwarding table

Figure 15. Example showing a simple next-hop table and PDU Forwarding table based on destination address only

The PDU Forwarding Table Generator is the IPC Process component responsible for carrying out the routing functionality. One example of a PDU Forwarding Table Generation strategy is Link-State Routing, depicted in the Figure below. Using this routing approach, information about the state of N-1 flows is disseminated through the DIF via CDAP messages to/from nearest neighbors. Each IPC Process keeps an up-to-date snapshot of the state of the N-1 flows of all the DIF, propagating the state of the N-1 flows to neighbors when there is a modification in the state of such flows. This information is then used to compute the PDU forwarding table using some sort of constrained-based shortest path first algorithm.

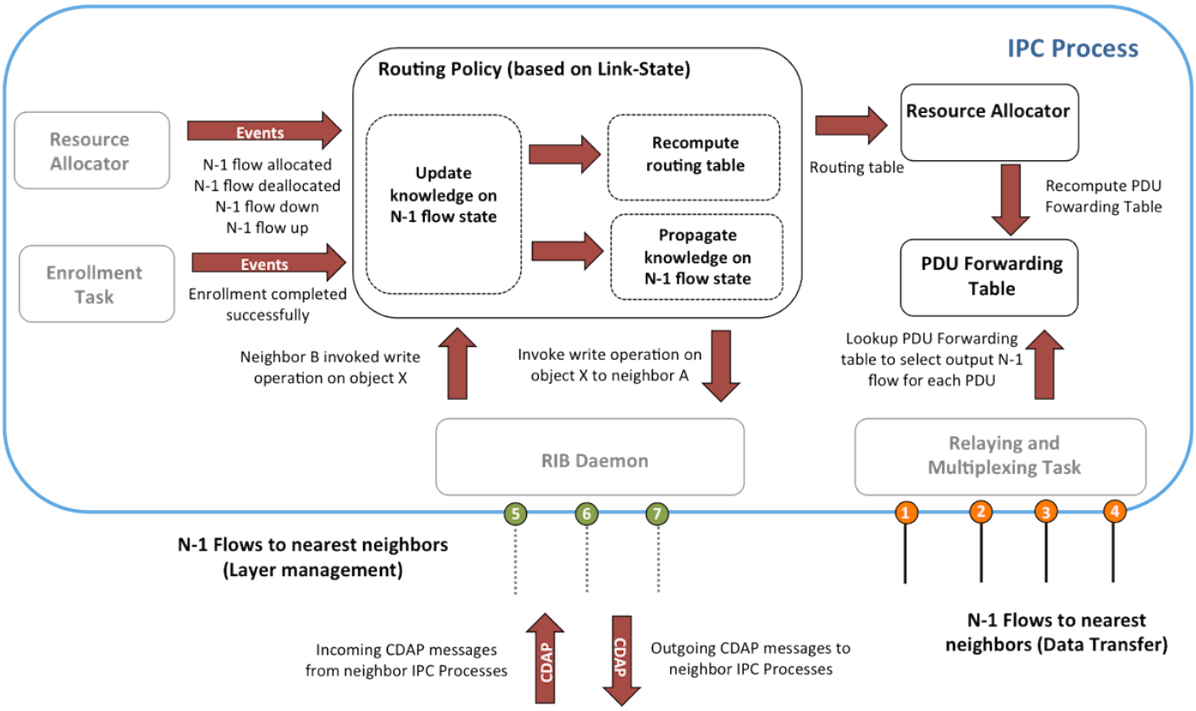


Figure 16.

Both addressing and routing are internal DIF operations; how they are carried out is entirely a policy of the DIF. Therefore there is no single approach that provides the optimal behavior in all operating environments: multiple strategies are possible. PRISTINE will choose a set of routing and addressing mechanisms proposed and used in computer networking, investigate how they can be applied to the RINA model and under what operational conditions these solutions have a good performance.

Addressing and routing approaches in PRISTINE's scope

Hierarchical/Topological routing

The explosion of routing table sizes is one of the main problems of the current Internet. Part of the problem is due to the fact that IP does not support multi-homing, and also due to the fact that the scope of the network layer is too big (routing is just done for the full Internet, instead of breaking the problem in different layers). RINA already solves these two issues, but the size of routing tables can still be further minimized if topological addressing is used. Networks do routing because they need to compute a forwarding table to generate the next hop; routing is not and end goal but a way to be able to take PDU forwarding decisions. Therefore, if the forwarding decision is derived from the address in the PDU and the address of the router the PDU is passing through, the forwarding decision can be made directly from the address in the PDU. Routing information need to be exchanged only when there were distortions in the topology (due to transient failures or "short-cuts"). But this information only needs to be known in

the vicinity of the distortion, not everywhere. If a PDU is not going near the deviation, it does not need to know about it. Routing exchanges are only needed in the neighborhood of a distortion. Some routers would store no routes at all, and those that did would store tens or hundreds of routes (orders of magnitude less than today). More important, routing table size can be bound.

In order to be able to exploit this scenario, the network connectivity graph at a given layer (the layer where routing is applied) has to reasonably coincide with the topology of its address space. Topology is an abstraction of a network graph, i.e. imposes invariants in the graph structure that individual graphs that follow the topology will always conform to (e.g. ring topology, star topology, hierarchical topology, etc). Furthermore, the topology has to be metrizable; that is, impose a distance function in the address space, so that it is possible to compare how ”far” or ”near” are two addresses in that address space.

T3.3 will investigate is what topologies for address spaces make sense, are easily maintained, scale, have nice properties for routing, etc for the three scenarios of the PRISTINE project. The challenge will be to find useful and effective algorithms for creating and configuring topologies of address spaces based on the abstractions and aggregation and the topologies of subnets without tying it to the physical topology of the network, but at the same time providing a convergence to that physical graph. The physical graph should be guided as much by the topology of the address space as by other considerations.

T3.3 work will result in a set of policies that : i)control automatic address assignment within a DIF, ii) describe distance functions that allow an IPC Process to compute the next hop given a PDU’s destination address and the addresses of directly connected IPC Processes, iii) model the routing information exchanged and identify the events that would trigger this exchange of information.

Optimized routing techniques from MAnets

Mobile Ad-hoc NETworks are multihop wireless networks composed of a set of autonomous nodes that move freely without any prior planning or coordination among themselves. MANET networks size ranges from a few nodes to several hundred nodes for different applications such as disaster recovery management or military tactical networks. MANETs are characterized by a highly dynamic topology due to their members’ mobility. Collaboration between nodes insures the network self configuration and self organization. In fact by incorporating routing functionality into mobile nodes, they are able to discover and track topology changes and react to calculate

updated routes accordingly. Data forwarding needs the cooperation of intermediate nodes to deliver packets from sources to destinations.

Due to the limited available bandwidth routing protocols designed for the MANETs must be efficient with low overhead consumption. Several routing protocols have been proposed in the literature trying to optimize the radio resources consumption and to support scalability as well.

Routing protocols for MANETs are classified into three classes, Proactive Routing protocols, Reactive Routing protocols, and Hybrid Routing protocols. In the Proactive Routing protocols family, topology information is exchanged among the nodes periodically, hence they can calculate routes to all the destinations in the network. The major advantage is the immediate availability of a route when needed, which comes with a price of continuous bandwidth consumption. With the Reactive Routing protocols family the routes are searched on demand only. It is based on a flooding of a route request message to the entire network. The destination or an intermediate node with an available route will reply by a route reply message indicating the route to the requested node. This means that nodes do not need to store unused routes to destinations. But they suffer from high bandwidth consumption when a route is searched, especially when the network size increases. Optimized flooding techniques and connected dominated set techniques are used to limit the bandwidth consumption. Hybrid Routing protocols tries to cope with the drawbacks of the two previous categories, save bandwidth and avoid storing useless information on intermediate nodes. Hybrid Routing protocols behave in a proactive manner in the vicinity of each node (few hops away) to limit the periodic control overhead propagation. Beyond this area they behave in a reactive manner to search routes to distant destination. There exists another optimized technique trying to limit the effect of bandwidth consumption for large networks called Fisheye State Routing (FSR). The principle is similar to the Link State routing protocols where link state information is diffused to the entire network. In FSR periodic broadcast is linked to the number of hops the topology information is propagated. Link state information is propagated more frequently to nodes located in the neighbourhood (like in hybrid protocols family) than the ones located far away. As a result, FSR keeps precise routes to neighbouring nodes, but less updated and precise ones to farther nodes. This is not problematic since the path will be more precise while approaching the destination.

Clustering techniques are used for hybrid routing protocols where proactive routing used inside the clusters and reactive protocol is used to look for a destination located outside the local cluster. The Hybrid Routing protocol family is more adapted to large networks. Organizing the network in a hierarchical way is more efficient when the

number of nodes increases. The most popular way of building hierarchy is to group nodes close to each other into explicit clusters. A clusterhead is elected for each cluster which is in charge of communicating with its peers in order to find routes to destinations located outside its cluster. Several communication levels can be built on top of this organization for more performing routing and reduced routing table size.

In PRISTINE, we will apply MANET hierarchical routing techniques to the use cases sharing a similar characteristic as the MANETs: high dynamic topology.

Compact Routing

Hierarchical/topological routing can lead to important routing table size reduction through the aggregation of nodes into sub-networks, sub-networks into super-networks, and so on, so that nodes in one super-network only have to store one routing table entry to reach any node in another super-network. It has been well studied in the literature that the efficacy of a hierarchical routing strongly depends on the characteristics of the underlying network connectivity graph, which make addresses to be aggregated in a more or less effective way. Specifically, hierarchical routing performs best over connectivity graphs with abundance of remote nodes or regular tree structures. However, its performance significantly worsens, e.g., over scale-free connectivity graphs, such as the one that the Internet currently describes.

The good news about RINA, compared to the current Internet, is that a DIF administrator has the capacity, up to a certain point, to deploy and configure a connectivity graph of N-1 flows among IPC processes facilitating scalable hierarchical/ topological routing inside the DIF. This could be foreseen as feasible in backbone/ regional/metro service provider network environments, which experience limited dynamicity over time and IPC processes can effectively be grouped in sub-networks according to their geographic location. In other situations, however, the connectivity graph among IPC processes in a DIF can be much more random (like in the PRISTINE distributed cloud use case), which can prevent an effective hierarchical addressing.

Therefore, alternative scalable routing policies for RINA DIFs will also be studied in PRISTINE. In particular, compact routing has been considered as possible alternative to overcome the scalability limitations of the current Internet. Briefly speaking, compact routing schemes aim to address the fundamental trade-off between the “stretch” of the resulting end-to-end routes (i.e., ratio between the length of the resulting routes over that of the shortest paths) and the size of the routing tables. In other words, they aim at reducing the amount of entries in the routing tables, while keeping acceptable route lengths (the lack of full routing information tend

to produce longer sub-optimal routes). A large variety of compact routing schemes have been proposed in the literature for universal as well as specific network graphs (e.g.,[TZ2001][BC2006][Abraham2008]). Furthermore, they are classified as name-dependent or name-independent, depending on whether some topological information is encoded in node addresses or not, respectively. While name-independent compact routing schemes are preferred, which do not require label changes are required upon network topology changes, their complexity and stretch is typically much higher than those of the name-dependent ones.

The basic principle of many of the proposed compact routing schemes is as follows. For each network node, a local neighborhood (also named as cluster or vicinity) is defined. Nodes know the shortest path to all nodes in their local neighborhood. Moreover, some of the nodes in the network act as “landmarks”, reachable through the shortest paths by all the remainder network nodes. When a source node has to send packets to a destination, these are sent through the shortest path, provided that the destination is within its local neighborhood. Otherwise, packets are sent to the landmark closest to the destination, which subsequently routes the packet to the destination. To offload landmark nodes, if a packet sent to a landmark arrives at a node that knows the shortest path to the destination, this one directly routes the packets over the shortest path (shortcutting improvement). It can be easily demonstrated (using the triangular inequality) that a stretch bounded by 3 can be achieved with this procedure, being lower in many realistic network conditions.

In PRISTINE, dynamic compact routing schemes building upon this principle will be proposed and their performance evaluated in the different use cases identified in the project. The specific target here will be to provide routing scalability in those DIFs where hierarchical routing becomes ineffective due to its connectivity graph characteristics.

Greedy routing

In addition to the compact routing approach, greedy routing has recently appeared as an attractive routing solution to drastically overcome the scalability problems of the Internet. In greedy routing, a coordinate (in a certain metric space) is assigned to every node in the network. These coordinates are used during the packet forwarding, and can be based on the physical location of the nodes (referred to as *greedy geographic routing*) or on virtual coordinates embedded in a different metric space (*greedy geometric routing*). That is, when a node has to send packets to a certain destination, the node greedily forwards them to its neighboring node that is closer to the destination, according to the assigned coordinates. Such a solution dramatically improves the

scalability of the network routing, as routing tables only have to store the coordinates of the adjacent neighbors. However, greedy routing schemes can get stuck in a local minimum (a node that does not have any neighbor closer to the destination than itself), leading to unsuccessful packet delivery. In this regard, Kleinberg, R. demonstrated in [Kleinberg05] that virtual coordinates can be embedded in a hyperbolic space for any kind of graph, so that the greedy routing is always successful.

In PRISTINE, the appropriateness of scalable greedy routing schemes will be investigated for large-scale RINA DIFs. A main limitation of greedy routing schemes is their poor performance upon topology changes, as a single change in the topology (e.g., addition/removal of new nodes/links or changes upon link/node failures) can invalidate the greedy embedding and require re-calculation of the coordinates of many network nodes. Looking at the literature, some proposals exist to smoothly achieve incremental greedy embeddings (to add new nodes) or to avoid communication voids resulting from failures. The benefit-complexity trade-off of incorporating these proposals in greedy routing schemes for RINA DIFs will be evaluated, deciding on their adoption when interesting. In any case, greedy routing should be restricted to RINA DIFs showing limited dynamicity in terms of topology changes.

3.3.2. RINA components and policies in scope of routing and addressing research

- **Component**: NamespaceManager.
  - **Policies**: AddressAssignmentPolicy, AddressValidationPolicy.The address assignment and validation policies that will be considered by PRISTINE's T3.3 can be of three types:
    - *Centralized*. The NSM is structured in a centralized way, in which one or more IPC Process (or the Network Management System) maintain the full state of the address namespace within a DIF and manage address assignment in a centralized way. This approach may be considered for the Greedy routing policy.
    - *Hierarchical*. The DIF is structured in a hierarchy, breaking the DIF in subsets of IPC Processes which form clusters, clusters of clusters, etc. Each DIF subset is assigned a partition of the address space. One or more IPC Processes in each subset (for example, those IPC Processes belonging to edge routers) manage the address namespace for all the IPC Processes that belong to the subset. This approach will be studied for topological/hierarchical routing, compact routing and MANet-inspired proactive and hybrid routing.

        - *Fully distributed*. Address assignment and validation is performed locally at each IPC Process, using local information and procedures to ensure that the chosen address is unique within the DIF. This strategy will be studied for MAnet-inspired reactive and hybrid routing.

- **Component**: Routing.
    - **Policy**: All the routing component can be considered policy. As described in the former section, the routing policies considered by PRISTINE will be
        - *Topological/hierarchical*. Default routes need not to be disseminated, since the only knowledge required to compute next hops are the neighbor's addresses. Only routing information for short-cuts (special routes to accomodate traffic with a particular level of service) and failures needs to be disseminated - maybe using link-state routing-like techniques.
        - *Compact*. There is the need to disseminate the state of N-1 links within the cluster the IPC Process belongs to, and to compute shortest paths to the IPC Processes in the cluster. IPC Processes also need to maintain connectivity information to all the landmarks in the DIF. In case the IPC Process is a landmark, it has to compute routes to other landmarks and to all the IPC Processes for which it is the closest landmark.
        - *Greedy*. The same considerations as in the *topological/hierarchical* routing policy apply.
        - *MANet-inspired*. In case of reactive routing policies, they have to perform controlled flooding of route requests. Proactive or hybrid routing policies disseminate all or part of the N-1 link state information to a subset of IPC Processes in the DIF. The subset can range from direct neighbors to all IPC Processes in the DIF, and the amount of information distributed can vary depending on the distance (in number of hops) to the IPC Process performing the routing update.
- **Component**: Relaying and Multiplexing Task (RMT).
    - **Policy**: PDUForwardingPolicy.
        - *Hierarchical/Topological* or *Greedy*. The policy would maintain non-default next-hops in a PDU Forwarding Table, and compute default next-hops on the fly. First the policy would check for an entry in the non-default next-hop table, if there were no entries for the PDU's destination address, it would then compute the next hop applying the distance function to the addresses of neighbor IPC Processes.

- *Compact* . The policy will return either the next hop to destination (if the address of the destination IPC Process is in the cluster of the IPC Process) or the next hop to the landmark that is closer to the destination IPC Process.
- *MANet-inspired*. The behavior of the policy will depend on the choice of the routing policy (proactive, reactive or routing). A generic description could be to search for the next hop to the destination address in a table (proactive); if no entries are found, ask the routing policy to compute the next hop by exchanging information with neighbor IPC Processes (reactive).

3.4. Authentication, Access Control and Confidentiality

3.4.1. Overview

Introduction to security in RINA

The recursive model of the RINA architecture provides a clear security model in which the trust relationships between layers (DAFs or DIFs) and between members of a single layer are well identified. The Figure below illustrates these trust boundaries, which facilitate the placement of the different security functions in the RINA architecture - in contrast to the Internet where security is designed in every protocol instead of at the system-level, making security complex and brittle [reeve].

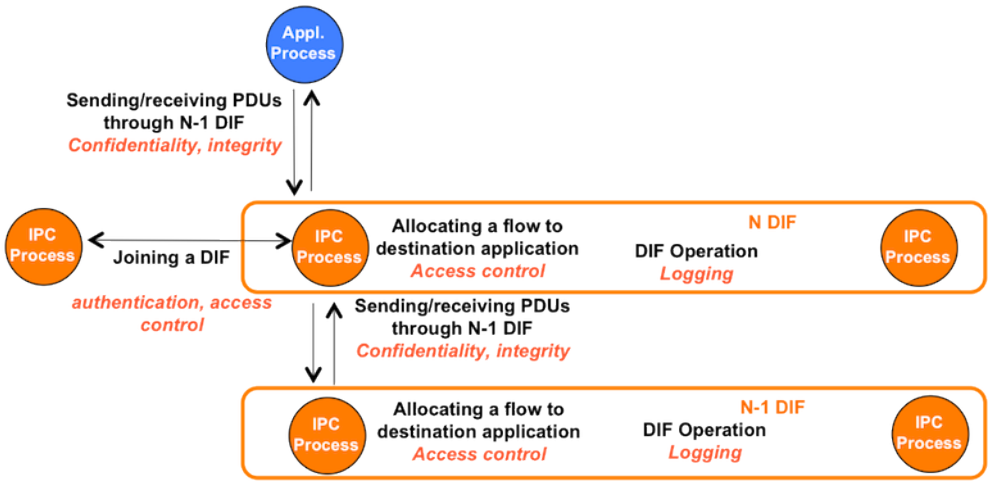


Figure 17. Placement of security functions in the RINA architecture

Users of a DIF - which can be a distributed application (a DAF) or another DIF - need to have very little trust on the DIF they are using: only that the DIF will attempt to deliver SDUs to a destination process. Applications (or IPC Processes) using a DIF are ultimately responsible for ensuring the **confidentiality** and **integrity** of the SDUs they pass to the DIF. Therefore, if they use a DIF that is not trusted, proper **SDU**

protection mechanisms have to be put in place. For example, an encryption policy that encrypts the application SDUs before passing them to the DIF (and decrypts them at the other side, right after reading the SDUs from the DIF).

Looking at the trust relationships within the components of a single layer - focussing on the DIF case - the placement of security functions is clear. When a new IPC Process wants to join a DIF, it first needs to establish a flow to one of the DIF members. The new member and the DIF member need to share an N-1 DIF in common, through which the flow will be established. Here an initial **access control** decision can be made by the N-1 DIF: the flow to the existing member may be rejected (because the N-1 DIF doesn't have enough resources to allocate the flow or because the N-1 DIF considers that the new member does not have permissions to allocate the flow to the existing member).

If the flow to the existing member is accepted, now the joining IPC Process has to establish an application connection with the existing DIF member via CACEP. The establishment of an application connection involves going through **authentication** (the strength of which can range from none to very sophisticated schemes). If the application connection is successfully established, the existing member will decide whether the new member is admitted to the DIF or not (**access control**). In case of a positive answer, the enrolment procedure will be initiated.

Remote operations on the IPC Process RIBs are another are where the **access control** function is of key importance. These remote operations (communicated via CDAP messages targeting one or more objects of the IPC Process RIB) may be invoked by other IPC Processes - in the case of layer management related operations - or by the Network Management System. At the finest granularity, it is possible to take an access control decision to authorize the access to each individual object in the RIB for each of the six CDAP operations (create, delete, read, write, start, stop).

All the security functions of an IPC Process are overseen by the security management component, which also performs other security management functions required for the security measures such as credential management. Network Management provides a higher-level of security coordination, having a centralized view of the state of multiple IPC Processes belonging to one or more DIFs. All in all, it can be said that DIFs in the RINA architecture are *securable containers*, since when proper security tools are used, a DIF is a structure used to hold or transport something that can be made to be not subject to threat [small].

Authentication

Each IPC process must either join an existing DIF or create a new one. In any case the IPC process will have to successfully pass an authentication process that is performed by either another IPC process that already belongs in the DIF or by the special management application that will initiate the DIF. PRISTINE will explore the architectural aspects of authentication and how it links to SDU protection policies. In particular, it will investigate how an IPC Process authenticates when joining a DIF (or AP joining a DAF in the general case). It will focus on how authentication fits with the RINA protocols, e.g. CACEP; how it is related to SDU Protection; and how it is extended to other IPC Processes in the DIF.

A plethora of symmetric key and asymmetric key based authentication protocols exist. The precise authentication mechanism to be used when joining a DIF will be selectable by policy; the architecture must allow additional policies to be added as required. PRISTINE will select two representative authentication approaches to provide test cases for the solution proposed. Approaches that may be used as test cases in WP4 are described below.

Password-based

WP4 could investigate how password authentication would be supported in RINA. The password would need to be transported over a secure channel (defined as a way of transmitting data that is resistant to overhearing or tampering) to prevent it being eavesdropped. Server authentication, i.e. authenticating the IPC Process already enrolled in the DIF, is likely to be required in the secure channel to prevent man in the middle attacks. WP4 would need to determine the order in which the various steps are performed when an IPC Process joins a DIF; how the Authentication Module is used when using password authentication; and how authentication fits with SDU Protection and the secure channel.

Cryptography-based

A typical method is Diffie-Hellman key exchange, which uses certificates to authenticate users and, by exchanging keys, persists the authentication, ensuring that the same users are communicating all the time. WP4 would need to investigate how the Authentication Module would support the authentication, e.g. certificate checking; where in the process of joining a DIF the key exchange would happen; and which RINA components perform the key agreement.

Access Control

Applications and IPC processes need to be able to specify which external applications and IPC processes are allowed to communicate with them, respectively, and what level of access they should be given. The way in which such access control can be supported within the RINA framework could vary from quite simple mechanisms such as Access Control Lists, to much more fine grained and expressive mechanisms such as Attribute Based Access Control (ABAC) or even to formal Multi-Level Security (MLS) models based on labeled security levels. PRISTINE will select representative authorisation approaches to provide test cases to investigate how access control models would be supported in RINA. The most likely candidates are Access Control Lists and Capability-based Access Control. PRISTINE will also investigate approaches for supporting MLS in RINA. These three access control models are described below.

Access Control Lists (ACLs)

An ACL can be described as a list attached to a system object that enumerates the subjects (users, OS processes, etc) that can access the objects, as well as the permissions granted to each subject (to execute different operations on the resources, for example) [rfc4949]. Examples of ACLs applied to networking today are the *network ACLs* that can be attached to router interfaces, or stateless firewalls based on packet filters. An example of the application of this model to the DIF would imply the setup of a number of ACLs based on the *application name of processes*, to control:

- The permissions that remote IPC Processes have on the objects of an IPC Process RIB (each RIB object could define its ACL).
- The IPC Processes that are allowed to join a DIF.
- The application processes that are allowed to allocate a flow to a specific target application process registered in a DIF (the IPC Process could define an ACL per registered application process).

ACLs are simple to implement but inflexible and hard to maintain, since permissions are bounded to the identity of individual subjects. If there are a lot of subjects in the system or the system is very dynamic (in that new subjects are usually entering and exiting the system), ACLs do not provide a proper authorization mechanism. A way to ameliorate these issues is to define ACLs based on groups of subjects instead of individual ones. That is, individual subjects belong to one or more groups; and objects in the system provide different levels of permissions to the different groups (the mapping of an individual subject to a group can be performed during the subject's

authentication phase). Group-only ACLs are equivalent to simple Role-based access control models, as shown by [barkley].

Capability-based Access Control

A capability can be defined as a *communicable, unforgeable token, ticket or key that gives the possessor permission to access an entity or object in a computer system* [dennis]. Each capability defines a series of access rights over an object that the holder of the capability can exercise (these access rights constrain the operations that the capability holder can perform over the object). Each subject has a list of capabilities that determines the objects that the subject can access in the system. Capabilities can be exchanged between subjects, effectively delegating access rights: this feature can cause problems if access to an object needs to be restricted after the distribution of capabilities; to solve this issue, a capability revocation mechanism is required.

In distributed systems the capability-based access control model externalizes the authorization decision from the manager of the object being accessed, since it can be considered that a subject is authorized to use a resource when it receives the appropriate capability. Therefore, a key element in any capability-based system is the authority or authorities in charge of assigning and revoking capabilities to subjects. Capabilities must be made unforgeable, therefore subjects must not be able to fabricate *artificial* capabilities granting them access rights they have not been entitled. Another key design aspects in a capability-based system is to model the information contained by each capability, that is: i) the identification of the object whose access is being granted by the capability and ii) the permissions over the object. A brief overview of an example of a capability-based access control applied to RINA (assuming a single management domain, which is the scope of PRISTINE) is exposed in the following lines.

- The Manager process of the Network Management-DMS (NM-DMS) is the capability manager, responsible for issuing and revoking capability tokens.
- IPC Processes request capability tokens to the NM-DMS (directly or via the system's IPC Manager component), granting them permissions to perform different actions:
  - Allocate flows to another IPC Process.
  - Join a particular DIF.
  - Execute operations on objects of the RIBs of other IPC Processes.
- In order to make capabilities unforgeable, they must be digitally signed using cryptographic techniques by the capability manager (so that non-repudiation and integrity of the capability token can be ensured).

Different types of capability tokens may be used for the different types of actions, resulting in slightly different design of the capability tokens. For example, the capability to join a DIF could have the structure depicted below. In this case the capability token would have to be part of the M_CONNECT CDAP message issued by the IPC Process that wants to join the DIF.

- **object_id**. The name of the DIF that the IPC Process has permission to join.
- **permissions**. Joining a DIF could be considered a binary action (an IPC Process is either allowed or not), in which case this field would be empty. However, it could happen that some DIFs wanted to provide a finer grained access control: for example, some IPC Processes would always be allowed to join the DIF, while others only on certain circumstances.

In the case of the capability that modeled the access rights to perform an operation over a certain RIB object, the token could have the structure depicted below. The capability token would have to be embedded in the CDAP message requesting the operation over the RIB object.

- **object_id**. The name of the RIB object instance the IPC Process wants to operate on.
- **Permissions**. The CDAP operations allowed in the object: CREATE, DELETE, READ, WRITE, START and STOP.

Multi-Level Security (MLS)

Multi-Level Security (MLS) refers to protecting data (or "objects" more generally) that can be classified at various levels, from users or processes ("subjects" more generally) who may be cleared at various levels. The levels refer to data that in some sense is more "valuable" or "sensitive" the higher the level it is in, and users who are in some sense more "trusted" to access the data the higher the level they are in. In its widest interpretation, the whole of information security could be viewed as MLS, as it is concerned with protecting sensitive data (as opposed to non-sensitive data at a lower level) from unauthorised users (as opposed to more trusted authorised users at a higher level).

An MLS system is a computer system that processes data at multiple different classification levels, and allows simultaneous access by subjects at multiple different clearance levels while preventing unauthorised access of these subjects to the data. In order to prevent unauthorised access, such systems will implement a security model, by

far the most common of which is the Bell-La Padula model [gollman]. This is commonly referred to as the "no read up, no write down" model. The "no read up" property is fairly self-explanatory, as it prevents users from reading data at a higher classification level than their clearance. The "no write down" property means that users cannot write data at a lower classification level than their clearance, which prevents users accidentally or deliberately labeling data at a lower classification level than its true classification level.

We will investigate approaches for supporting MLS within RINA. One approach is to allow data at multiple levels within a DIF and ensure that an IPC Process within the DIF can only access data appropriate to its clearance. Alternatively, different DIFs could be used for each level, so that all IPC Processes within a DIF are cleared to access all of the data within the DIF. These and/or other approaches may be considered.

Confidentiality

The IPC processes of a given N-DIF may or may not trust the lower level DIFs. The cryptographic mechanisms required to create a secure channel between two IPC processes of the same DIF will be investigated. Depending on the level of security that has to be achieved in the different use cases, several choices have to be taken, ranging from the type of cryptographic algorithms, the proper placement of the security mechanisms, the size of the secret keys to the lifetime of the key - as explained in the authentication section, credential management will be studied within the *security coordination* research area.

In Task 4.1, PRISTINE will research how a secure channel is set up, used and managed in RINA, particularly when an IPC Process joins a DIF, and how it is linked to authentication. It will also investigate what secure channels are needed and what components they protect. An existing protocol will be selected for implementation to provide a test case for the solutions proposed. A likely candidate is Transport Layer Security (TLS) [rfc5246], which would provide a fairly complete example of setting up and using a secure channel, since it involves authentication (with the use of digital certificates as a common option), generation of secret keys and policies for protecting SDUs (including compression, encryption, etc). The relevant parts of the TLS handshake protocol would be defined as authentication policies, while the TLS record protocol would result in policies for the SDU Protection module.

3.4.2. RINA components and policies in scope of authentication, access control and confidentiality

- **Component**: Common Application Connection Establishment (CACEP)

- **Policy**: AuthenticationPolicy. Two main categories of authentication policies are in the scope of PRISTINE: password-based authentication and some form of cryptographic-based authentication (the relevant parts of the TLS handshake protocol are a likely candidate for exploration).

- **Component**: SDU Protection.
  - **Policy**: EncryptionPolicy. The TLS record protocol will be examined in order to identify suitable policies for SDU Protection, in particular for encryption.

- **Component**: Security Management.
  - **Policies**: NewMemberAccessControlPolicy, NewFlowAccessControlPolicy, RIBAccessControlPolicy. The applicability of the three access control models described in section 3.4.1 (ACLs, capabilities and MLS) will be investigated for each of the DIF's access control policies.

3.5. Security Coordination

3.5.1. Overview

In order to make a DIF a secure container, the security mechanisms described in the former section (authentication, access control, SDU Protection) have to be properly coordinated. PRISTINE will investigate and implement security coordination strategies within a DIF, focusing on distributed credential management, logging of events and auditing and identification of attacks and counter-measures. Although the different topics could be addressed from a central management system, T4.2 will focus on DIFs that can autonomously perform these functions in a distributed fashion, at least partially. These autonomous management functionalities embedded in the DIF increase the reliability of the system.

Credential management

Key distribution for authentication, and access control and cryptographic purposes in a distributed system is a complex issue. Although some form of central management approach, such as a Certification Authority or Identity Provider, may be necessary to at least bootstrap the system, PRISTINE will investigate ways in which keys, identities and other attributes can be created, distributed, authenticated, and revoked in as a decentralised way as possible. Amongst the main functions of a key management system there are: entity registration, key generation, and retrieval, verification, key distribution, key maintenance (revocation).

Task 4.2 will describe a key management system for authentication, access control and cryptographic purposes in achieving both confidentiality and integrity of information. It will assess current architectures against the use case requirements and select an architecture to be implemented. Key management approaches that could be explored are described below.

KMIP

Key Management Interoperability Protocol (KMIP) [kmip] is an OASIS (Organisation for the Advancement of Structured Information Standards) standard that allows applications to securely communicate with key management servers to obtain cryptographic objects, e.g. a key or certificate. It supports a wide range of key management lifecycle operations, including create, locate or destroy a cryptographic object.

Task 4.2 could investigate how KMIP could be used to distribute keys within a DIF for SDU protection. It could also be used to manage cryptographic objects needed for authentication when joining a DIF.

Cloud Key Distribution

Task 4.2 could investigate how to reuse a cloud key management system, e.g. from SlapOS or re6st, to manage keys in RINA. It could investigate how to allocate and revoke keys to IPC processes for use in authentication.

Logging of events and auditing

Identifying the right events and information to be monitored is a key part of detecting potential attacks from misbehaving DIF members. Detecting anomalous behavior is usually the function of an system external to the normal operation of the network. A *Network Intrusion Detection System* (NIDS) is one example, in which one or more special devices are inserted at strategic places of the network to monitor a significant part of the traffic entering and exiting the network. In RINA, internal functions in the IPC Processes and the Management Agents can also be used to detect abnormal behavior, playing the role of a *Host Intrusion Detection System* (HIDS).

One of the key questions that PRISTINE has to answer is which subset of all the information known by the IPC Process is more valuable for detecting possible attacks against the DIF. Some initial targets to consider are:

- **Layer management events logged by the RIB Daemon**. This can potentially include all the operations requested over the objects in the RIB of IPC Process.

The analysis of this data could reveal, for instance: an excessive number of flow allocation requests (suggesting the DIF may be suffering a Denial of Service - DoS - attack), resource allocation requests inconsistent with the DIF policies, an excessive number of failed enrollment attempts by the same IPC Process, too much routing updates, IPC Processes with the same address in the same DIF, etc.

- **EFCP data transfer traffic going through the RMT**. Capturing samples of the different connections to be able to characterize the traffic distribution would be helpful for later comparing it with traffic distributions of well-known attacks.
- **EFCP data transfer traffic processed by the EFCP protocol machine**. This data would enable the identification of IPC Processes that did not comply with the policies of a particular EFCP connection; for example an IPC Process that ignores flow control information and systematically transmits more data than the maximum is allowed to do.

The other fundamental question in this topic is how to analyze the information that has been gathered. Two of the main strategies in use today are **statistical analysis** and **signature-based** analysis. The former method analyzes the monitored information and compares it with a "baseline" that represents normal DIF operating conditions (usual traffic patterns, usual events logged by the RIB Daemon, etc); while the latter compares the monitored information with signatures of well-known attacks. Both methods have their pros and their cons: statistical analysis may give false positives identifying a legitimate use of the DIF as a potential attack; signature-based analysis is more accurate but can only detect the attacks whose signatures are known a priori.

Task 4.2 will also work in close collaboration with WP5, in order to identify how the addition of a central Network Manager process can improve the chances of detecting and protecting against abnormal behavior. In essence, if the IPC Processes (via the MA) or the MA itself inform the Network Manager process about the detection of abnormal behaviors, the Network Manager process can have an overall view of abnormal situations in a DIF and set of DIFs, and use this greater visibility to better identify a potential attack (and optionally react to it) or dismiss the alerts.

Identification of attacks and counter-measures

Assuming a proper logging infrastructure in place, every IPC Process in the DIF has enough information to detect attacks or other abnormal behavior. PRISTINE will categorize the different types of attacks that a DIF can suffer, what are their symptoms and what counter-measures can be taken against them. An initial analysis has already been carried out, identifying the scenarios in the Figure below.

In the first case we assume that an application is attempting to overwhelm a target system with numerous allocate requests. Both the source and destination IPC Process have the capability to return an error and deny the requests if they come to the conclusion that the source application has exceeded a certain rate of allocated resources. A second case is depicted in the top-right figure, where we assume that a DIF has been compromised and a rogue member has been allowed to join the DIF. The consequences of this depend on the information available to the rogue member (access to the RIB). The permission is also important (read, write or both). Possible actions of a rogue member of the DIF can be change the flow control rate to perform a DoS attack or circulate the encryption keys. A third case we need to study is when the underlying DIF is considered untrusted. SDU protection mechanisms proposed in T4.1 can ensure that the PDU will not be corrupted while in transit.

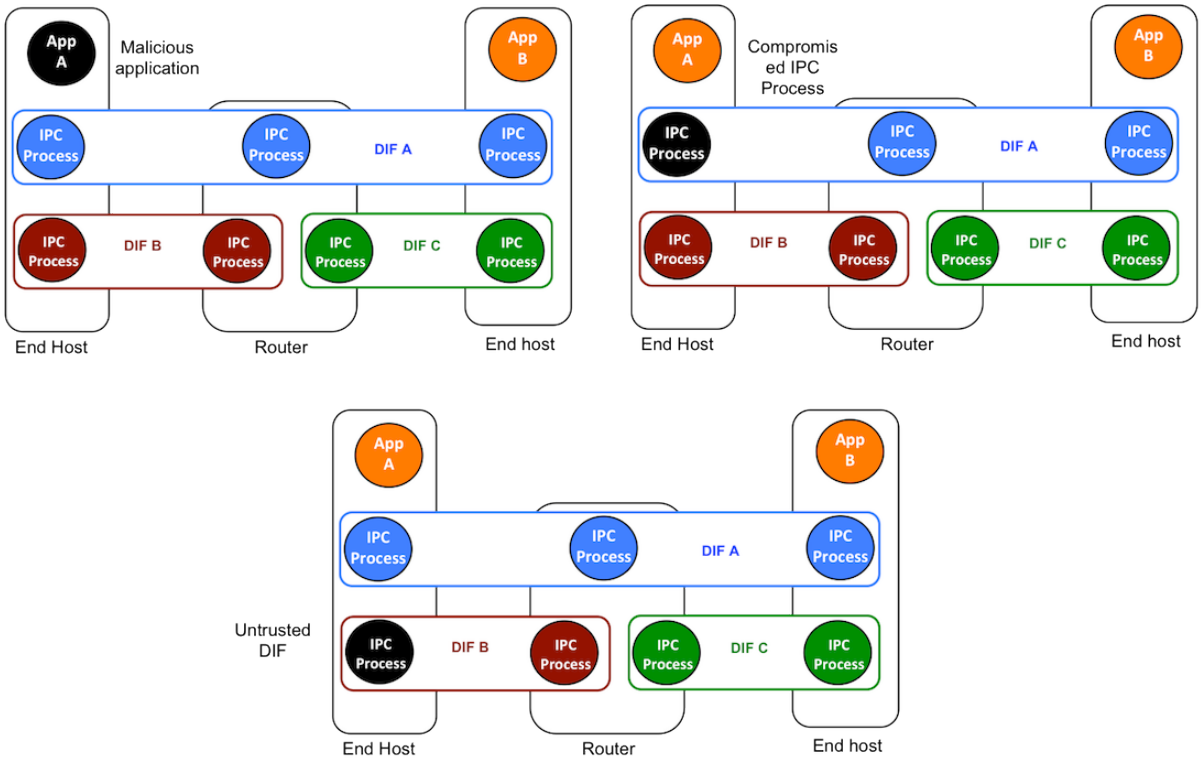


Figure 18. Different roles an attacker can play in a DIF

An initial analysis of the possible attacks to RINA networks is carried out in [small], where an initial categorization of the different attacks a DIF can suffer is provided, along with the information required to perform such attacks successfully and indications of how an attacker could get access to that information. [boddapati] focuses on the analysis of transport-level attacks, evaluating the robustness of RINA against them and comparing the results with those of TCP/IP. T4.2 will extend the initial work on this area by making deeper thread analysis and also proposing counter-measures to

improve the robustness of RINA against the different attacks that can potentially affect the architecture.

3.5.2. RINA components and policies in scope of security coordination

- **Component** : Security Manager.
  - **Policy** : CredentialManagementPolicy. The KMIP standard is a potential target for investigation for the extraction of suitable credential management procedures. The procedures identified will be adapted for their use within a DIF (or a set of DIFs). Another area of exploration are credential management systems for cloud applications.
  - **Policy** : AuditingPolicy. Both statistical and signature-based analysis will be initially considered as suitable auditing policies. D4.1 will further study both approaches and their pros and cons on their applicability to different DIF environments.
- **Component** : RIB Daemon.
  - **Policy** : LoggingPolicy. Several logging profiles will be studied, considering DIFs with different trust levels ranging from trusting all IPC Processes to trusting no other DIF members. As the trust level decreases the information that needs to be logged will increase, becoming more detailed. The different information that can be logged will be classified by its relevance, so that it can be progressively recorded as the trust in neighbor IPC Processes increases or decreases (for example, because the IPC Process has detected or has been told that the DIF is under attack).

3.6. Resiliency and High Availability

3.6.1. Overview

Introduction

RINA networks consist of a variety of DIFs owned by one or more organizations. As illustrated in the following figure, the different DIFs are stacked on top of each other, with DIFs of lower rank (N-1) carrying multiplexed traffic belonging to several flows provided by DIFs of higher rank (N). DIFs of lower rank tend to have a smaller scope than DIFs of higher rank (there are exceptions such as DIFs providing VPN-style

services) and the nature of the traffic they are carrying also tends to be more predictable - since aggregated traffic is less bursty than the traffic from individual flows.

The goals of T4.3 can be summed up in one question: "How do we ensure that DAFs can maintain a certain service-level specifically relating to availability (a maximum downtime in a specific period of time)?" The mechanism to achieve the availability is to provide resiliency to failures.

The industry generally measures availability by using different metrics, such as: a) the Mean Time To Recover (MTTR) b) the Mean Time Between Failures (MTBF) and c) the *number of nines* (or *class of nines*), which is expressed as percentage of time a system is up over a given period of time (usually a year). The following table shows the translation from a given availability percentage to the corresponding amount of downtime a system would be allowed per year, presuming that the system is required to operate continuously.

Table 7. Availability and downtime per year

Availability	Downtime per year
90 (one nine / class 1)	36.5 days
99 (two nines / class 2)	3.65 days
99.9 (three nines / class 3)	8.76 hours
99.99 (four nines / class 4)	52.56 minutes
99.999 (five nines / class 5)	5.256 minutes
99.9999 (six nines / class 6)	31.536 seconds
99.99999 (seven nines / class 7)	3.1536 seconds

Another critical quantity is the maximum allowable interruption time(s). For instance, in telephony this is 50ms (else the call is dropped). This metric sets an upper bound on the time that recovery actions can take to complete when a failure has occurred.

T4.3 will look at appropriate DAFs for each of the defined use cases, setting targets for these metrics to be achieved for these DAFs.

In order to provide resiliency, two actions are absolutely required: failure detection and failure recovery.

- failure detection. This is the most basic premise, we need to become aware of a failure before we can take any action to mitigate its impact on the overall system's performance.

- failure recovery. After detecting a failure, we can take actions to restore the performance of the system to an acceptable level. This may require an additional action: failure localisation.
- failure localisation. Some recovery mechanisms require to be aware of the location of the failure before they can efficiently perform recovery.

In order to return the system to its original state a fourth action is needed:

- fault repair.

In this task, we will only investigate failure detection, failure recovery and failure localisation. In a multilayer environment, race conditions can occur for each of these aspects.

- race conditions in failure detection. Intuitively, we would find that the smaller the scope of a DIF and the closer it is to the hardware (i.e. the lower its rank), the faster it will detect failures. This is true for the most commonly used failure detection mechanism: loss of (N>1) consecutive "hello" packets. There is a natural tradeoff between the interval of sending these "hello" packets (and the overhead incurred) and the failure detection time.
- race conditions in failure recovery. If there is no direct notification of failure detection between DIFs, the necessary and sufficient condition to avoid race conditions is that, if an N-DIF makes use of the services provided by an (N-1)-DIF, the failure detection time of the N-DIF is higher than the sum of the failure detection time and the recovery time of the (N-1)-DIF. This can be (almost) guaranteed in practice by a implementing a hold-off timer, with its inherent inefficiencies (how long is long enough?). Another option is that, upon detecting a failure, a DIF will send a notification to DIFs of higher rank that it has detected a failure and is initiating recovery. If there are no race conditions in failure detection, this is sufficient. The mechanism to do this coordination needs to be investigated in this task.
- race conditions in failure localisation are not an issue. If localisation is required for the recovery actions, it should be taken into account in the recovery time.

The outcomes of this task will be: 1) a set of DIFs tailored to support the selected DAF based on the use cases. 2) a set of algorithms for the DAF (most critically the ones associated with the RIB daemon) and a set of policies for each of the DIFs. Such policies include routing, graceful restart, etc. 3) possible updates to the reference model if needed.

Initial Observations

The responsibility entity for detecting, recovering from, and locating failures lies at first with the DIF in which the failure occurs. Only if an (N)-DIF is unable to sufficiently recover from a failure, an (N+m)-DIF (m>0) should take over responsibility for recovery and further propagate the recovery if it cannot resolve the failure. In most cases m=1. (Exceptions are, for instance, if the N+1-DIF has too limited scope or has no failure recovery policies - both would be sub-optimal network designs).

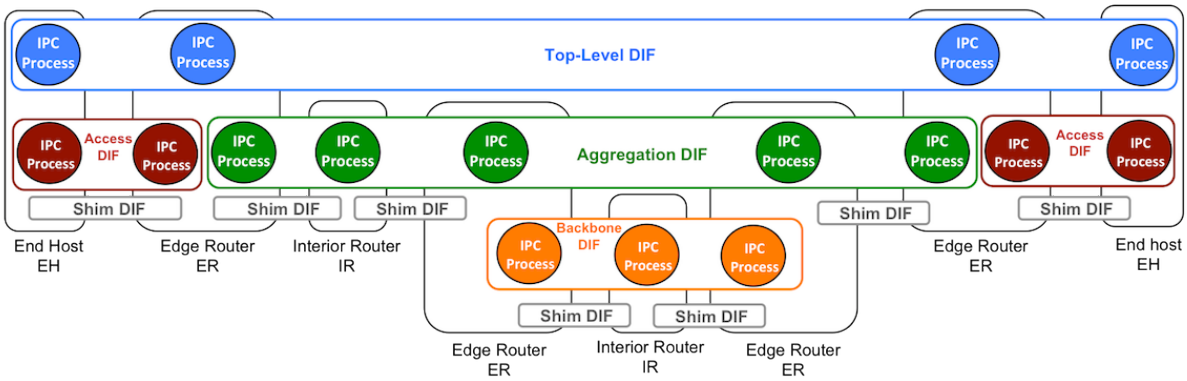


Figure 19. Example of a provider network designed with RINA, to illustrate RINA's multi-layer environment

There are two types of failures to include: link failures and IPC process failures.

Link failures

In the RINA model, link failures are failures that occur between two IPC processes in an N-DIF (N>0) (or two Application Processes in an N-DAF). These failures will ALWAYS correspond to a failure in the supporting (N-1)-DIF. Hence, given the recursion, all link failures will fall into two cases:

a) a failure of a (physical) link in the 0-DIF, i.e. the path between two 0-IPC-processes. This can also be a failing non-RINA forwarding device in case of a Shim DIF over a legacy protocol (e.g. an Ethernet switch in case of the Shim DIF over Ethernet). These failures should be either resolved within the 0-DIF (if the legacy technology supports it) or in the lowest-ranked m-DIF with sufficient capabilities (usually m=1).

b) a failure of an IPC process in an (N-n)-DIF (1<n#N). This resolves into a failure of an IPC process (see below).

Failure of an N-IPC process

(Note 1: a complete RINA-router failure will have the same effect as the failure of the 0-IPC processes on all physical interfaces. Note 2: The following also applies to DAP

failures, however, we restrict ourselves to IPC process failures and assume the reader to apply the same reasoning to DAP failures).

Failure of an N-IPC process should be resolved within the N-DIF. The RIB daemon (forwarding table generator) will have to converge the state of the DIF related to forwarding (i.e. the forwarding tables) to cope with the failure. If this is insufficient, the recovery action should propagate to the lowest-ranked (N+m)-DIF with sufficient capabilities (usually m=1). The IPC processes of the (N+1)-DIF (or AP's of the N +1-DAF) cannot recover the traffic and must gracefully terminate the flows and roll back their registration with the failed (N)-IPC process. Similar for all higher-ranked processes that share a dependency on the failing IPC process. They can register with another IPC process (belonging to either the same or a different DIF) to attempt re-establishing the flow.

3.6.2. RINA components and policies in scope of resiliency and high availability

- **Component**: ResourceAllocator.
  - **Policy**: Monitoring N-1 flows. Resource allocators in adjacent IPC Processes exchange PDUs over the N-1 flow(s) they have in common to actively monitor its characteristics (delay, throughput, loss rate, etc). If the N-1 flow characteristics fall below certain thresholds the flow is considered down and a corrective action is triggered.
  - **Policy**: Action to take when an N-1 flow is considered "down". An example of this policy could be to immediately trigger a routing update and a recomputation of the PDU Forwarding Table.
- **Component**: FlowAllocator.
  - **Policy**: FlowMonitoringPolicy (as an alternative to the Resource Allocator actively monitoring N-1 Flows). Flow allocators in the IPC Processes of the N-1 DIF can monitor the characteristics of the N-1 flow provided to the upper layer. They can either inject special-purpose PDUs to the flow and infer its characteristics, or use the real traffic on the flow to extract this information (for example: PDU loss probability can be easily computed thanks to the sequence numbers in the DTP PDUs; delay could be calculated if timestamps are included in the PCI, etc). When the N-1 flow is considered to be failing, a notification is delivered to the user of the N-1 flow (which will be the IPC Process at the N-layer).

3.7. Network Management

3.7.1. Overview

PRISTINE will initially focus on traditional centralized Network Management configurations, assuming a single Management Domain. As the project evolves and Network Management tools mature (specially the definition of the RIB managed object model), PRISTINE may also consider configurations in which Management Agents have more autonomy, as well as interactions among domains.

The Network Management work that will be carried out in PRISTINE can be divided into four categories, as further explained in the sections below:

- **Common elements in the Network Management System**. These provide the architectural framework and common structure for the elements in the Network Management DAF, upon which specific Network Management Tasks can be built. Amongst these base elements there are: the application protocol used in the DAF (CDAP), the RIB managed object model, the Management Agents and the Manager.
- **Configuration Management**. Tasks associated to the setup and maintenance of the DIF configurations at each system.
- **Performance Management**. Tasks associated to the detection of problems in DIF operation that affect its capacity towards achieving their terms of service (i.e. complying with the service level agreements of the flows the DIF is providing).
- **Security Management**. Tasks associated to the coordination of security functions between a single or multiple DIFs, including: centralized monitoring, auditing, etc.

Introduction to Network Management in RINA

A high-level overview of the RINA architecture is provided by the Figure below. 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 - Distributed Management System (or NM-DMS in short).

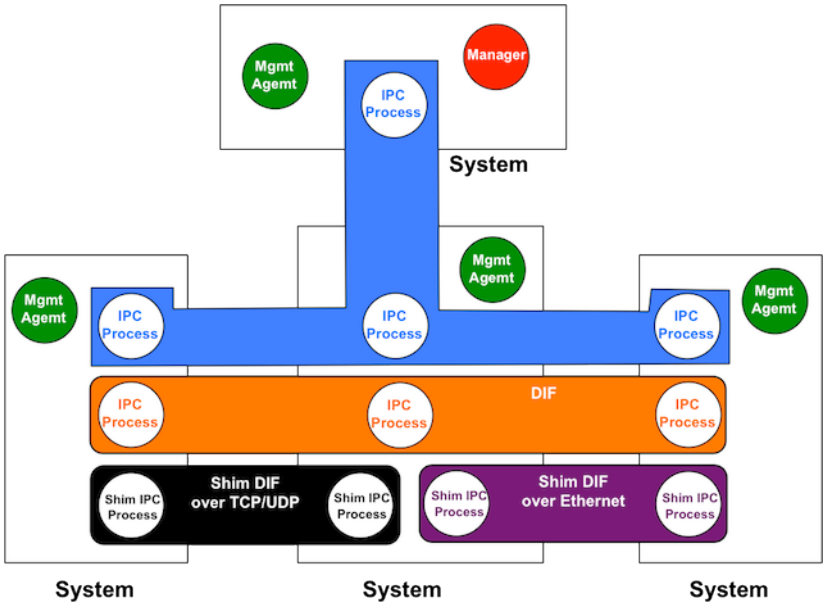


Figure 20. Graphical model of the RINA architecture

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 must 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 must be possible for distributed systems to fail-safe without central control. A NM-DMS will perform the traditional functions of monitor and repair, deploying new configurations, monitoring performance, etc. The DAF model can be applied to network management to represent the whole range from distributed (autonomic) to centralized (traditional).

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

The NM-DMS DAPs in the traditional agent role function like the sensory nervous system collecting and pre-processing data. The Agent will have access to the DAF Management task of all IPC Processes (and associated DAPs) in the processing systems

that are in its domain. While there is no constraint, it is likely that an NM-DAF would have one "Agent DAP" for monitoring in each processing system with a DIF or DAF in its domain. The DAF Management task of each DAF or DIF in the NM-DMS domain is a kind of "sub-agent." A Management Agent may be designed to seek out its DMS or alternate DMSs in the event of failures.

In order to interact with each system, the manager process needs 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 DMS and the management agent can communicate by performing remote operations on the RIBs of IPC Processes via CDAP - the Common Distributed Application Protocol.

Common elements in the NM-DMS

The following Figure illustrates the base elements of the NM-DMS architecture considered by the PRISTINE project - limited to a single management domain. A central manager process uses one or more DIFs to allocate flows to the management agents of the system under his domain. After establishing application connections with each management agent via CACEP, the Manager interacts with the management agents by operating over the objects in their RIBs via CDAP operations. Management agents can also send notifications back to the Manager using CDAP operations.

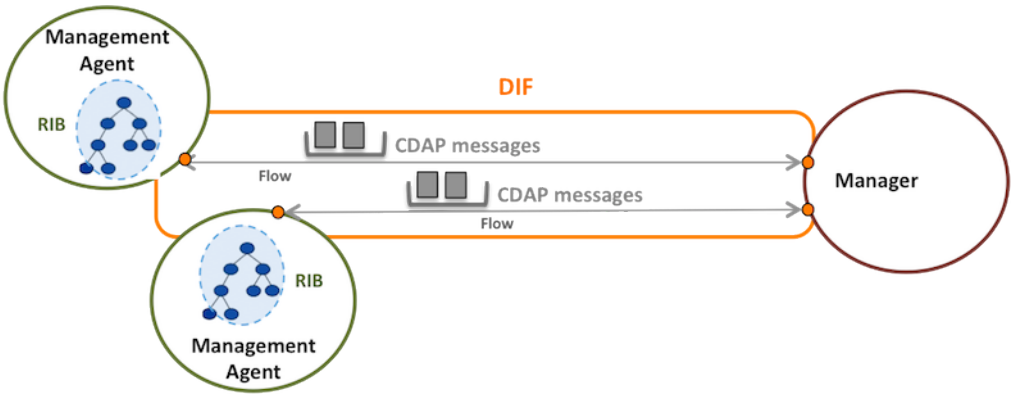


Figure 21. Common element of the management framework

CDAP and CACEP

CACEP is the protocol used to establish application connections between application entities (AEs, communicating entities on different application processes). Application establishment is required in order for the two AEs to have enough information

to understand each other and to optionally authenticate each other. Amongst this information there is: the version of the CDAP protocol message declarations, the concrete syntax used to encode protocol messages on a *wire format*, the version of the RIB and object set to use, the source and destination names of the communicating AEs or optional credentials used to authenticate both parties.

Once the application connection is in place, the application entities use CDAP to exchange information. CDAP is an object-oriented protocol modeled after CMIP (the Common Management Information Protocol) [cmip] that allows two AEs to perform six operations on the objects exposed by their Resource Information Bases (RIBs). These fundamental remote operations on objects are: create, delete, read, write, start and stop. Since in RINA there is only one application protocol (CDAP), the different AEs in the same application process do not identify different application protocols, but the subsets of the RIB available through a particular application connection. That is, different AEs provide different levels of privileged access into an Application Process RIB. CDAP incorporates two mechanisms in the protocol to allow for operating over multiple objects at once (scope and filter).

Managed object model (RIB object definition)

The managed object model of RINA specifies the characteristics of RINA managed objects, their relationships and how they are named. It also provides tools to describe the attributes and behavior of these objects, as well as discusses different options for object definition languages and its encodings. All instances of managed objects of the same type are described by a *managed object class* definition. A managed object class definition defines, for an instance of the class:

- The properties or characteristics visible at the managed object boundary - these are called *attributes* and each property has a value.
- The *management operations* that may be applied (which are the same for all the objects: create, delete, read, write, start and stop).
- The *behavior* the object exhibits in response to management operations, including any potential side-effects.
- The *notifications* the object can produce; describing in what conditions they may be produced and the information each notification provides.
    - CDAP does not explicitly define notification operations, since - as opposed to CMIP - CDAP is intended to be the only required application protocol, and not all applications required notifications. However notifications are an essential property of the managed object model and can be modeled using

the 6 operations available in CDAP (PRISTINE's D5.1 describes a proposal for modeling notifications).

- Its *position in the inheritance hierarchy* of the managed object class. The inheritance hierarchy defines which classes inherit the attributes of other classes, creating a parent-child relationship between managed object classes.
- All the possible *name bindings* of the managed object class. A *name binding* provides the information to name an object in the context of the RIB *containment tree*. The naming of managed object instances is based on the idea that managed objects are contained within other managed objects. For example, an IPC Process is executing in a system, in order to refer to the IPC Process Managed object corresponding to the IPC Process is natural to say that it is contained in a system managed object. A managed object has only one immediately containing object, and so the containment structure of managed objects is a tree.

Management Agent (MA)

The Management Agent (MA) is a functional entity mainly in charge of managing RINA related resources in a processing system. It maintains the dialogues with the DMS by reacting to its requests - for control, administration and maintenance - while hiding/ abstracting the processing system details - e.g. such as those OS related - in order to reduce the overall system complexity - i.e. the complexity of the DMS, of its related MAs and their inter-communications.

The main functionalities exposed by the MA can be further categorized in the following areas:

NM-DMS related:

- IPC provisioning (e.g. control the creation and destruction of IPC processes).
- IPC configuration (e.g. such as policies configuration).
- Monitoring and fault management (e.g. monitoring the state of an IPC process or the state of N-1 flows)
- Inter-layer management (i.e. inter-layer optimizations and configurations -such as Access Control Lists between DIF-N and DIF-(N-1)).

OS-DMS related:

- Status retrieval:
  - Hardware resources, such as:

        - CPU (e.g. architecture, available capacity, current load).
        - Memory (e.g. available memory, used memory).
        - Storage (e.g. available storage, used storage).
        - Network (e.g. available NICs, NICs used by shim IPC Processes).
    - Software "resources", such as:
        - Management Agent information (e.g. name, description, localization, agent software version).
        - PRISTINE SDK information (e.g. SDK version).
        - Policies catalogue (i.e. policies already available in the processing system, see. next section).
        - OS related information (e.g. OS name, version)

DIF Manager

A DIF Manager provides a (logically) centralised configuration, fault, security and performance management functions for the DMS. In other words, it performs network management related tasks within an administrative domain. There is at least one DIF Manager in an administrative domain, and it manages one or more Management Agents, corresponding to one or more nodes within the network. The DIF Manager can be replicated to improve the resiliency of the DMS, or act as a peer in multi-administrative domain deployments.

Configuration Management

T5.2 aims to develop, evaluate and provide a configuration management framework for the PRISTINE scenarios. It will extend the common elements of the NM-DMS architecture defined in T5.1 with functionality to support the following tasks:

- DIF instantiation and destruction: The management framework shall provide a convenient way to instantiate and destroy the DIFs (IPC processes), at the appropriate scale.
- DIF configuration: The management framework should provide means for configuring the DIF parameters such as name, scope, addressing schema, etc. It will define a declarative contract based specification for DIF configuration, and provide a mechanism to verify DIF configuration, avoiding inconsistencies between DIFs within a system and in the overall domain, for example, it should allow CDAP instructions to be generated from a declarative specification, for configuring the

RIB within a domain. The configuration management framework should support the declarative contract based reconfiguration of DIF's policies, at the appropriate scale. This implies the discovery, inventory and (re)configuration of policies in a DIF from the DIF Manager, as well as ensuring consistency on the policies across the different systems.

Performance Management

The objective of T5.3 is to develop and evaluate a multi-layer performance management framework for the PRISTINE scenarios. This task will extend the common elements of the NM-DMS architecture defined in T5.1, and build on the declarative contract based configuration abstractions defined in Task 5.2. This task will investigate appropriate complex event processing techniques, develop and evaluate them within the scenarios identified by PRISTINE. Specifically, this task will:

- Identify alarm triggers from the declarative DIF configuration specification, and ensure collection points for these alarms are established via the CDAP primitives and RIB managed object model defined in T5.1.
- Correlate generated alarms with their associated DIF, and calculate the effective management "ripple" caused by the alarm. For example, calculating the set of (layer N+1) DIF's that depend on the DIF at layer N.
- Analyse application impact of the correlated alarms, for example, analysis on the actual usage of network resources as opposed to the declared expected usage. This allows performance tuning, where the allocated resources are optimised to the aggregate needs, increasing resources to over-loaded DIF's and reducing resources to underutilised ones. This will involve adjusting the declarative contract based configurations and application expectations to a more optimised form. The analysis will include addressing if there a historical basis to these alarms (e.g. greater than 20% under utilisation, greater than 10% admission failures, etc.).
- Investigate ways automated policy responses can be included in the declarative configuration specification. Are there "patterns" in how these alarms are handled and can these be include in a "best-practice" configuration guide.

Security Management

There is a need for policy-controlled data access and communication between DIFs. Generally, there will be a security classification associated with every DIF. The IPC processes within a DIF have the same security classifications. The Multi-Level Security (MLS) concept will be applied for achieving secure communication between the DIFs.

MLS generally refers to protecting objects (processes or data) that can be classified at various sensitivity levels, from users, applications or processes (subjects) who may be cleared at various clearance levels. A DIF should be able to decide, according to its own classification label and the classification level of the DIF with which it wants to communicate, whether to protect the data (e.g., content-based encryption, IPsec/SSL) or not before passing it on. Data will only be protected when it is passed from a DIF to another DIF with lower security classification. In general, encryption is better to be done at lower DIFs as more part of data block/packet is protected. It is envisaged that security controls will be carried out as part of multi-layer management solution as specified below:

- Security configuration: for access control and MLS in order for establishing/ agreeing security classification requirements for communication with lower DIFs.
- Security control enforcement: to achieve MLS across the PRISTINE platform.
- Security monitoring: to perform data collection from lower DIFs for Security Incident and Event Monitoring (SIEM), and perhaps for forensics as well.

3.7.2. RINA components and policies in scope of network management

- **Component**: Common Distributed Application Protocol
  - **Policy**: CDAP does not consider different policies, but different concrete syntaxes - although PRISTINE will continue to use the current one (Google Protocol Buffers). However the project will review the current scope/filter approach of the protocol in order to operate over multiple objects with a single operation.
- **Component**: RIB object model
  - **Policy**: Although all the RIB model may be considered policy (since each DIF may implement its own), PRISTINE will try to define a RIB managed object model that can be used by all the DIFs - capturing the common objects that model the RINA elements present in the reference model.
- **Component**: Management agent. The current RINA specifications do not provide a detailed description of the internal functions of the Management Agent, nor identifies fine-grained policies that are specific to this component. PRISTINE will design and implement a MA process, and learn from this experience to characterize generic aspects of the MA. However, as a member of the Network Management DAF, there are some DAF-generic policies that also apply to the MA, such as policies

for: enrollment to the DAF, managing the DAF namespace, protecting SDUs or multiplexing network management traffic.

- **Component**: Manager The current RINA specifications do not provide a detailed description of the internal functions of the DMS Manager, nor identifies fine-grained policies that are specific to this DAF. PRISTINE will design and implement DMS manager process, and use this to provide feedback on the following policies, within the "management" DAF, which is used to communicate between DMS manager and Managed Agents. It is expected the following policies may be affected:
  - **Policy**: UpdatingPolicy. This is a policy that determines when, what and how to update the objects of the RIB. The RIB Daemon is used in the DMS Manager and the DMS agent, so there may be policies specific to the DMS manager, for example, sync a certain DIF with a DMS manager for another administrative domain or used when determining to update all agents within a specified DIF.
  - **Policy**: Management Policies: It is foreseen that there is a need of "management" recipes with in the DMS Manager. It is reasonable to expect some of these recipes are "selection" policies, whose sole purpose is to diagnose/detect network conditions, gather additional information, and come to a decision as to which of the available (management) recipes should apply. The results of these decision actions, will result in CDAP management operations being applied on the RIB, altering the "running configuration" and result in the corresponding RINA policies being applied.

For example, a node failure can be detected by multiple QoS alarms from Flows in different DIFs all passing through the failed node. Depending on the configuration the DIFs will reroute traffic where possible. In order to completely respond to such a failure, a decision needs to be made on a higher plane on what action to take, for example, notify operator *joe.smith* via SMS on weekends, or via IM during the working week.

4. Conclusions

In addition to the conclusions exposed in section 3.2.15, D2.2 concludes that:

- The major aspects of the RINA reference model - detailing the main elements of the RINA architecture and their interactions - are well consolidated.
- In spite of the limitations in the different IPC Process components identified in section 2, the base framework of the RINA architecture is solid. However more work needs to be carried out to better define the detailed interactions between IPC Process components. This is reflected in the degree of maturity of the different specifications; of which EFCP, CDAP and CACEP (the core DIF protocols) are the most well-defined specifications.
- A key task in RINA research is now to consider many different networks with a variety of operational requirements, in order to design DIFs with different roles and different policies for its components with the goal of understanding what policy-sets are better for different boundary conditions. One of the goals of RINA research is to define a catalogue of different policy-sets that allow different DIFs to optimally operate in a variety of environments providing a specific level of service.
- The DIF components and policies in the scope of PRISTINE research and development activities have been identified. Most of the components define fine-grained policies, but there are policies belonging to some components - specially related to resource allocation, security and network management - that are yet too coarse-grained. PRISTINE research activities will help identifying finder-grained policies for these components, as well as how these policies interact with each other.
- The analysis of the RINA research areas in the scope of PRISTINE has shown that congestion control, resource allocation and routing have a strong relationship. These tasks will start their research individually, but as PRISTINE evolves it is likely that the commonality of the three tasks will be exploited to research and develop more integrated solutions.

Bibliography

[Abraham2008] I.Abraham, C.Gavoille, D.Malkhi, N.Nisan, and M.Thorup. Compact name-independent routing with minimum stretch. *ACM TALG*, 2008.

[asn1] ITU-T. *X.680-X.693 : Information Technology - Abstract Syntax Notation One (ASN.1) & ASN.1 encoding rules*. November 2008. Available online[1].

[barkley] J. Barkley. *Comparing simple role based access control models and access control lists*. Proceedings of the second ACM workshop on Role-based access control pages 127-132 (1997).

[BC2006] A. Brandy, and L. Cowen. "Compact routing on power-law graphs with additive stretch", Proceedings of *the 8th Workshop on Algorithm Engineering and Experiments (ALENEX 2006)*.

[boddapati] G. Boddapati, J. Day, I. Matta, L. Chitkushev. *Assessing the security of a clean-slate Internet architecture*. 7th Workshop on Secure Network Protocols (NPsec), 2012.

[buprot] BU RINA team. *ProtoRINA github site*. Available online[2]

[coyne] E. Coyne, T. R. Weil. *ABAC and RBAC: Scalable, Flexible and Auditable Access Management*. IEEE IT Professional Magazine, May/June 2013. Available online[3].

[cmip] ITU-T. *X.711 : Information technology - Open Systems Interconnection - Common Management Information Protocol: Specification*. OCtober 1997. Available online[4].

[davies] N. Davies. *Delivering Predictable Quality in Saturated Networks*. Predictable Network Solutions Technical report. Available online[5].

[dennis] J. B. Dennies and C. Van Horn. *Programming Semantics for Multiprogrammed Computations*. Communications of the ACM 9(3), March 1966.

[1] http://www.itu.int/rec/T-REC-X.680-X.693-200811-I/en
[2] https://github.com/ProtoRINA/users/wiki
[3] http://csrc.nist.gov/groups/SNS/rbac/documents/coyne-weil-13.pdf
[4] https://www.itu.int/rec/T-REC-X.711-199710-I/en
[5] http://www.pnsol.com/public/TP-PNS-2003-09.pdf

[ferraiolo] D.F. Ferraiolo, D. R. Kuhn. *Role-Based Access Controls*. 15th National Computer Security Conference, 1992. Available online[6]

[gpb] Google. *Google Protocol Buffers developer guide*. Available online[7].

[gollman] D. Gollmann. "Computer Security", Second Edition, John Wiley & Sons, November 2005.

[holyer] N. Davies, J. Holyer and P. Thompson. *An operational model to control loss and delay of traffic in a network switch*. Third IFIP Workshop on Traffic Management and Design of ATM Networks, 1999.

[hu] V. C. Hu, D. Ferraiolo, R. Khun, A. Schnitzer, K. Sandlin, R. Miller, K. Scarfone. *Guide to Attribute Based Access Control (ABAC). Definition and Considerations*. NIST special publication 800-162, January 2014. Available online[8].

[irati] S. Vrijders, D. Staessens, D. Colle, F. Salvestrini, E. Grasa, M. Tarzan and L. Bergesio, *Prototyping the Recursive Internetwork Architecture: the IRATI approach*. IEEE Network, vol. 28 no. 2, March 2014. Available [online]

[irati-ls] IRATI project. *Specification for a PDU Forwarding Table Generator Policy based on Link-State Roting*. Deliverable D2.3, February 2014. Available online[9]

[is-is] *Intermediate System to Intermediate System intra-domain routeing information exchange protocol for use in conjunction with the protocol for providing the connectionless-mode network service (ISO 8473)*. ISO/IEC International Standard 10589, November 2012.

[json] Ecma International. *The JSON Data Interchange Format*. Standard ECMA-404, October 2013. Available online[10]

[Kleinberg05] R. Kleinberg, "Geographic routing using hyperbolic space," in *INFOCOM 2007*.

[6] http://csrc.nist.gov/rbac/ferraiolo-kuhn-92.pdf
[7] https://developers.google.com/protocol-buffers/
[8] http://nvlpubs.nist.gov/nistpubs/specialpublications/NIST.sp.800-162.pdf
[9] http://irati.eu/deliverables-2/
[10] http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf

[kmip] K. Thota, K. Burgin.*Key Management Interoperability Protocol Specification Version 1.2*. OASIS Committee Specification Draft 01 / Public Review Draft 01, 09 January 2014. Available online[11].

[protorina] Y. Wang, I. Matta, F. Esposito, J. Day. *Introducing ProtoRINA: A Prototype for Programming Recursive-Networking Policies*. Editorial Note at ACM SIGCOMM Computer Communication Review (July 2014 Issue of CCR). Available online[12].

[reeve] Reeve, D.C. *A new blueprint for Network QoS*. PhD thesis, University of Kent at Canterbury, UK, 2003.

[rfc1287] D. Clark, L. Chapin, V. Cerf, R. Braden, R. Hobby. *Towards the Future Internet Architecture*, Network Working Group, RFC 1287, December 1991. Available online[13]

[rfc3031] E. Rosen, A. Viswanathan, R. Callon. *Multiprotocol Label Switching Architecture*. RFC 3031, January 2001.

[rfc3209] D. Awdunche, L. Berger, D. Gan, T. Li, V. Srinivasan, G. Swallow. *RSVP-TE: Extensions to RSVP for LSP Tunnels*, RFC 3209, December 2001.

[rfc3630] D. Katz, K. Kompella, D. Yeung. "Traffic Engineering (TE) Extensions to OSPF Version 2", RFC 3630. September 2003.

[rfc4949] R. Shirey. "Internet Security Glossary, Version 2", RFC 4949. August 2007. Available online[14]

[rfc5246] T. Dierks, E. Rescorla. "Transport Layer Security (TLS) Protocol, Version 1.2". RFC 5246, August 2008. Available online[15]

[small] J. Small, J. Day, L. Chitkushev. *Threat Analysis of Recursive Internetwork Architectures Distributed InterProcess Communication Facilities*. In press.

[thopian] Thopian M. R., Prakash, R. *A distributed protocol for dynamic address assignment in mobile ad-hoc networks*. IEEE Transactions on Mobile Computing, Volume 5, Issue 1, January 2006.

[11] http://docs.oasis-open.org/kmip/spec/v1.2/csprd01/kmip-spec-v1.2-csprd01.html
[12] http://csr.bu.edu/rina/papers/CCR2014.pdf
[13] http://www.ietf.org/rfc/rfc1287.txt
[14] http://tools.ietf.org/html/rfc4949
[15] http://tools.ietf.org/html/rfc5246

[triaprot] TRIA Network Systems. *RINA Implementation Overview*. First RINA Workshop, Barcelona, January 2013. Available online[16]

[trouva] E. Trouva, E. Grasa, J. Day, S. Bunch. *Layer Discovery in RINA networks*. Proceedings of IEEE CAMAD 2012, pages 368-372. Available online[17]

[TZ2001] M. Thorup, U. Zwick, "Compact routing schemes", Proceedings of *the Thirteenth annual ACM symposium on Parallel algorithms and architectures (SPAA '01)*, July 2001.

[xml] World Wide Web Consortium (W3c). *Extensible Markup Language 1.0, Fifth Edition*. W3C Recommendation, November 2008.

[wang] Y. Wang, I. Matta and N. Akhtar. *Experimenting with Routing Policies Using ProtoRINA over GENI*. The Third GENI Research and Educational Experiment Workshop (GREE2014), March 19-20, 2014, Atlanta, Georgia. Available online[18]

[weniger] K. Weniger. *Passive duplicate address detection in mobile ad-hoc networks*. Proceedings of IEEE WCNC 2003. Available online[19]

[16] http://irati.eu/wp-content/uploads/2013/01/ImplementationOverview.pdf
[17] http://ieeexplore.ieee.org/xpl/articleDetails.jsp?tp=&arnumber=6335369
[18] http://csr.bu.edu/rina/papers/GREE2014.pdf
[19] http://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=1200609&url=http%3A%2F%2Fieeexplore.ieee.org%2Fxpls%2Fabs_all.jsp%3Farnumber%3D1200609