MASTER THESIS

Title: **A study of RINA (*) a novel Internet architecture**

(*) Recursive InterNetwork Architecture

Submitted by: **Periklis Zisis**

1st Academic Supervisor: **Prof. Dr. Shun-Ping Chen**

2nd Academic Supervisor: **Prof. Dr. Johannes Gerdes**

Industrial Supervisor: **Eng. Dr. Ioannis Korovesis**

Completion Date: **25.03.2015**

Student:

PERIKLIS
First (Given) Name

ZISIS
Last (Family) Name

Date of Birth: 13.07.1986

Matr.-No.: 732218

1st Academic Supervisor: Prof. Dr. Shun-Ping Chen

2nd Academic Supervisor: Prof. Dr. Johannes Gerdes

Title: A study of RINA (*) a novel Internet architecture

(*) Recursive InterNetwork Architecture

Abstract: (max 10 Lines)

The Internet is changing and evolving rapidly. Nowadays, more and more applications are created continuously and due to the requirements of the networks, the Internet has become an architectural patchwork with increased complexity. Thus, new architectures are being proposed to solve the weaknesses of the current Internet.

RINA is a new network architecture, which was developed from scratch as an alternative to TCP/IP. It is based on inter-process communication and capitalizes on repeating patterns and structures as well as on the separation of mechanisms from policies. RINA attempts to solve the problems of the TCP/IP architecture like mobility, multihoming and addressing by providing a secure and configurable environment.

In partial fulfilment of the requirements of the **University of Applied Sciences Hochschule Darmstadt (h_da)** for the degree **Master of Science in Electrical Engineering** carried out in collaboration with **Industrial Enterprise**

Company: Internet Systematics Lab of Demokritos

Address: Patriarxou Grigoriou E32 Agia Paraskeui 15341 Athens, Greece

This Master Thesis is subject to a non-disclosure agreement between the University of Applied Sciences Hochschule Darmstadt (h_da) and the industrial partner.

(Signature)

1st Academic Supervisor:

Student:

PERIKLIS ZISIS
First (Given) Name Last (Family) Name

1st Academic Supervisor: Prof. Dr. Shun-Ping Chen

2nd Academic Supervisor: Prof. Dr. Johannes Gerdes

Declaration

I hereby declare that this thesis is a presentation of my original research work and that no other sources were used other than what is cited.
I furthermore declare that wherever contributions of others are involved, this contribution is indicated, clearly acknowledged and due reference is given to the author and source.
I also certify that all content without reference or citation contained in this thesis is original work.
I acknowledge that any misappropriation of the previous declarations can be considered a case of academic fraud.

Darmstadt, ...
(Date) (Signature)

A Study of RINA (*) a novel Internet Architecture
(*) Recursive InterNetwork Architecture

Hochschule Darmstadt
University of Applied Sciences, Germany

Zisis Periklis
Matriculation Number: 732218

Master's Thesis Report

Supervisors:

Prof. Dr. Shun-Ping Chen

Prof. Dr. Johannes Gerdes

Eng. Dr. Ioannis Korovesis

This page intentionally left blank

Acknowledgements

This Master's thesis project would not have been possible without the support of many people and I would like to thank them for helping me make my Thesis successful.

Firstly, I would like to express my sincere gratitude to my advisor Dr. Ioannis Korovesis of the institute Demokritos for giving me the opportunity to cooperate with him in Greece. He offered me valuable support and help throughout the thesis.

I would like also to thank my academic supervisors of Hochschule Darmstadt Prof. Dr. Shun-Ping Chen and Prof. Dr. Johannes Gerdes for their support and guidance during this Master's thesis research.

Last but not least, I would like to thank my family in Greece, for their continuous support, love and trust during my studies in Germany.

Abstract

The Internet is changing and evolving rapidly. Nowadays, more and more applications are created continuously and due to the requirements of the networks, the Internet has become an architectural patchwork with increased complexity. Thus, new architectures are being proposed to solve the weaknesses of the current Internet.

RINA is a new network architecture, which was developed from scratch as an alternative to TCP/IP. John Day, the originator of this idea, claims that there are some networking principles which are independent of technology and when applied, lead to a much simpler implementation and a unified theory of networking. Thus, he focuses on the fundamentals of networking and argues that in order to get a deeper understanding of an architecture, abstractions have to be used. The outcome is a simplified and a more easily managed model.

RINA is based on inter-process communication and capitalizes on repeating patterns and structures as well as on the separation of mechanisms from policies. The hierarchical model in which layers can be stacked on top of each other recursively, is also embraced by the architecture of SDN.

RINA attempts to solve the problems of TCP/IP like mobility, multihoming and addressing by providing a secure and configurable environment which is similar to the SDN trend. The simplicity and the features of this architecture, make RINA a topic worthwhile to be examined.

The thesis examines the differences between RINA and the current TCP/IP architecture, as well as how this new model attempts to solve the current Internet shortcomings. In addition, the implementation projects of RINA are given, among which the IRATI and PRISTINE are the most crucial. IRATI focuses on this new architecture and how it can be deployed in production scenarios such as data centers, by using OFELIA (OpenFlow testbed). PRISTINE on the other hand, focuses on programmability and aims at the implementation of a software development Kit for RINA.

Table of Contents

Acknowledgements **1**

Abstract **2**

Table of Contents **3**

Abbreviations **6**

List of Figures **9**

1 Introduction **11**

1.1 Background **11**

1.2 Motivation and Thesis problem **13**

1.3 Scope of the Thesis **14**

1.4 Thesis overview **15**

2 The evolution and the shortcomings of the Internet **17**

2.1 Origins of the Internet and the increase of network requirements **17**

2.1.1 Evolutionary research vs clean slate **19**

2.1.2 Purists vs pluralists **20**

2.1.3 The IAB Workshop **20**

2.2 Current Internet shortcomings **21**

2.2.1 Naming and addressing **21**

2.2.2 Routing, mobility and multihoming **23**

2.2.3 No indirection **25**

2.2.4 Protocols **25**

2.2.5 Security **26**

2.2.6 Architecture **27**

2.2.7 Quality of Service **29**

2.2.8 Network management **30**

2.3 Communication between distributed applications in the Internet **30**

2.4 Attempts to fix some of the Internet problems **31**

2.4.1 Overlays **31**

2.4.2 LISP, MIP and IPv6 ..... **33**
2.5 Summary of the chapter **35**
3 RINA and its implementation projects **36**
3.1 Phases of communication **36**
3.1.1 Data transfer mechanisms **37**
3.2 Terminology in RINA **39**
3.3 Communication between distributed applications in RINA **40**
3.4 Separation of mechanism from policy **41**
3.5 Inter-process communication and recursion **42**
3.6 IPC naming **44**
3.7 The DIF **44**
3.8 Structure of a DIF **46**
3.9 Phases of communication in RINA **54**
3.10 The DAF and the role of IPC manager **57**
3.11 Example of Inter-process communication by using DIFs **59**
3.12 Example of Inter-process communication by using DAFs and DIFs **61**
3.13 Features of RINA and how it differs from the current TCP/IP architecture **63**
3.14 How security is handled in RINA **72**
3.15 The ongoing projects of RINA **75**
3.16 Summary of the chapter **76**
4 RINA and Software Defined Networking **77**
4.1 Introduction to SDN **77**
4.2 Overview of the SDN architecture **79**
4.3 Current SDN architectures **80**
4.3.1 OpenFlow **81**
4.4 Forwarding devices, controllers, management applications and the role of abstractions in SDN **82**

4.5 How SDN is related to network virtualization **85**
4.6 SDN application cases **85**
4.7 Future development of SDN **86**
4.8 SDN and the relevance of RINA as a policy management architecture **87**
4.9 Summary of the chapter **88**
References **89**

Abbreviations

Term	Description
TCP/IP	Transmission Control Protocol/Internet Protocol
RINA	Recursive InterNetwork Architecture
IPC	Interprocess Communication
DIF	Distributed IPC Facility
SDN	Software Defined Networking
NREN	National Research and Education Network
NCP	Network Control Program
IETF	Internet Engineering Task Force
IMP	Interface Message Processor
ARPANET	Advanced Research Projects Agency Network
PoA	Point of Attachment
LISP	Locator Identifier Separation Protocol
IP	Internet Protocol
ID	Identifier
SN	Source Node
DN	Destination Node
MS	Mapping Server
MIP	Mobile IP
API	Application Programming Interface
QoS	Quality of Service
CLNP	Connectionless-mode Network Protocol
UDP	User Datagram Protocol
NAT	Network Address Translation
FTP	File Transfer Protocol
RJE	Remote Job Entry
DoS	Denial Of Service
VoIP	Voice Over IP
DNS	Domain Name System
URL	Uniform Resource Locator

ACK	Acknowledgement
MTU	Maximum Transmission Unit
MPL	Maximum Packet Lifetime
ISP	Internet Service Provider
AP	Application process
PPP	Point To Point Protocol
HDLC	High Level Data Link Control
DHCP	Dynamic Host Configuration Protocol
HTTP	Hypertext Transfer Protocol
PDU	Protocol Data Unit
CSNET	Computer Science Network
IT	Information Technology
IPv6	Internet Protocol Version 6
IPv4	Internet Protocol Version 4
BGP	Border Gateway Protocol
BER	Bit Error Rate
TCP SACK	TCP Selective Acknowledgement
TCP FACK	TCP Forward Acknowledgement
SNMP	Simple Network Management Protocol
RMT	Relaying Multiplexing Task
SDU	Service Data Unit
CRC	Cyclic Redundancy Check
EFCP	Error and Flow Control Protocol
DTP	Data Transfer Protocol
DTCP	Data Transfer Control Protocol
PCI	Protocol Control Information
CDAP	Common Distributed Application Protocol
RIB	Resource Information Base
CMIP	Common Management Information Protocol
IAP	IPC Access Protocol
CACE	Common Application Connection Establishment
DAF	Distributed Application Facility

IDD	Inter-DIF Directory
DAP	Distributed Application Process
CEP	Connection End Point
MPLS	Multiprotocol Label Switching
LTE	Long Term Evolution
GPRS	General Packet Radio Service
WLAN	Wireless LAN
NETCONF	Network Configuration Protocol
DCAN	Devolved Control of ATM Networks
I2RS	Interface To The Routing System
ForCES	Forwarding and Control Element Separation
GENI	Global Environment for Network Innovations
IAB	Internet Architecture Board
IRTF	Internet Research Task Force
FA	Foreign Agent
HA	Home Agent
VPN	Virtual Private Network

List of Figures

Figure 2.1 Saltzer's model **22**
Figure 2.2 the Internet **23**
Figure 2.3 Current Internet architecture **28**
Figure 2.4 Communication in the Internet **30**
Figure 2.5 Overlay structure **32**
Figure 2.6 LISP model **33**
Figure 2.7 MIP **34**
Figure 3.1 Communication in RINA **40**
Figure 3.2 Mechanism/policy **41**
Figure 3.3 Distributed Application Facility **43**
Figure 3.4 DIFs can be stacked according to the scope of the network **43**
Figure 3.5 Names of an IPC process **44**
Figure 3.6 Structure of a DIF **46**
Figure 3.7 Port allocation decoupled from synchronization **48**
Figure 3.8 DTP processing **49**
Figure 3.9 DTCP processing **49**
Figure 3.10 RMT **50**
Figure 3.11 SDU protection **51**
Figure 3.12 Common Distributed Application Protocol **52**
Figure 3.13 Enrollment phase **55**
Figure 3.14 Establishment phase **55**
Figure 3.15 Port-id to CEP-id mapping **56**
Figure 3.16 IPC Manager **58**
Figure 3.17 IPC between two processes **59**
Figure 3.18 – IDD DAF **61**
Figure 3.19 Creation of a common DIF **62**
Figure 3.20 Hosts connected via a router **63**

Figure 3.21 Routing in RINA **64**
Figure 3.22 Basic structure that a provider network might have in RINA **65**
Figure 3.23 Example of mobility in RINA **66**
Figure 3.24 Routing example with interfaces **67**
Figure 3.25 Shim DIF -over IP **68**
Figure 3.26 Example with mobile terminal **69**
Figure 3.27 User is connected also to the second interface (LTE) **70**
Figure 3.28 User is connected only to the LTE **71**
Figure 3.29 Port allocation decoupled from synchronization **72**
Figure 3.30 Current Internet Security stack **73**
Figure 3.31 Security in RINA **74**
Figure 4.1 Software defined network **79**
Figure 4.2 Westbound/Eastbound APIs **80**
Figure 4.3 Openflow Device **81**
Figure 4.4 Network abstraction **83**

Chapter 1
Introduction

1.1 Background

The Internet was created in simpler times. The creators and early users had a **common goal**; to build a network infrastructure in order to hook all the computers together [16].

The Internet as we know it today, interconnects computer networks and uses the standard Internet protocol suite (TPC/IP), in order to link numerous devices worldwide and to serve millions of users. It was originally developed by the government of the United States as a robust communication system, to support the computer networks in universities and research laboratories of the country.

For its first 20 years starting in the early 1980s, the Internet design has been guided by the end to end principles [17]. These end to end arguments suggest that specific application functions, **should not be built into the core** of the network but at the ends of it, in order to achieve a simplicity in the core. Many applications in the meantime have been designed in different ways, but according to the rules of the end to end design.

TCP/IP played a significant role in the evolution of the networks. It was initially designed for the ARPANET, the progenitor of what was to become the global Internet [1]. Because the combination of ARPANET and TCP/IP was successful, the whole product went into production immediately. Although there was lack of prior experience, the success was instant, the Internet grew exponentially and more and more computers were connecting to it. The network worked much better than anyone had any reason to expect.

However, due to the rapid expansion of the Internet, problems in routing, addressing and security started to appear and some of them exist until today. As presented in the IAB workshop in 2007, the today's Internet routing and addressing system is facing serious scaling problems [40].

The increasing user population, the overloading of IP address semantics and multihoming are some of the factors which enhanced the problem [40].

Furthermore, the last few years, new requirements have emerged for the Internet. These new requirements in turn, motivate new design strategies to accommodate the growing 'tussle' between the different stakeholders which are 'part' of the Internet, such as the commercial ISPs, the users, and the private network providers. Thus, to certain stakeholders, new mechanisms might be needed in order to serve their particular needs and the new requirements [16], [17].

Some examples of these new requirements are the following:

- No trust: there is no trust between the communicating end- points. In the beginning of networking, the networks were among people who knew each other. On the contrary, nowadays there are different kinds of attacks on the networks.

- Third- party involvement: as the Internet becomes mainstream, it inevitably moves from being an engineering curiosity to being a mirror of the societies in which it operates. There is an increase in third parties, which want to interfere between the communicating endpoints and satisfy their needs. These parties include: 1) users, who want to run applications and interact over the Internet, 2) ISPs, which sell Internet service with the goal of profit, 3) governments, which enforce laws, protect consumers etc.
  The governments for example, could spy on communications of parties, or control the access of certain parties to certain material.
  Another example which shows the third-party involvement is the design of DNS, in which the DNS names are used both to name machines and to **express trademark**.

- More demanding applications: new sets of applications with streaming audio and video requirements have emerged. These applications though, require high throughput and no delays. Thus, there are a lot of intermediate servers which position the streaming content close to the recipient, **moving from a simple end to end structure to a two-stage delivery via intermediate servers.**

  The above examples imply, that the world is becoming more complex than it used to be when the Internet began and the end to end principles were designed. The changing nature of the user base is pushing the Internet into new directions!

1.2 Motivation and Thesis problem

During my research regarding the state of the Internet, I discovered a view according to which the Internet expanded so fast without much research and some of the current problems could be avoided if tackled at the beginning. Through this, I found out the idea of RINA, which is an alternative Internet architecture and attempts to solve the current Internet shortcomings. Thus, I realized that this topic is very interesting and worth to be examined.

The current thesis will examine the logic of RINA and its relation to the current Internet architecture trends. The purpose of this thesis **is not to prove that RINA will solve the current Internet problems** but instead to present another interesting approach of how the Internet could be designed.

John Day, a computer scientist from the United States who has been involved in the development of the communication protocols of the Internet since 1970, is the originator of the recursive InterNetwork architecture idea [7].

RINA uses many of the lessons learned by previous network architectures, and brings them one step further by identifying that networking can be seen as a set of recursive layers which provide distributed IPC services over different scopes. In other words, there is a single layer called DIF which provides the required functionality and this DIF can be repeated according to the needs of the network [4]. Furthermore, in RINA, by separating policy from mechanism, the same mechanism can work across wide ranges of scope, QoS and bandwidth. As a consequence, RINA uses two generic protocols:

- **EFCP** which is a data transfer protocol and a combination of the DTP and DTCP
- **CDAP** which is RINA's application protocol

Thus, the resulting architecture is more simplified compared to the current one, with less protocols and mechanisms. With this simplified model the network designers will try to solve the problems of the current Internet architecture.

During my project and while searching for similar architectural topics, I discovered SDN, which is currently under a lot of discussion and it's also related to RINA [18]. Similar to RINA, the architecture of SDN attempts to solve the problems which exist in networks. I was proved right regarding the relation between these two architectures, when IRTF published an RFC regarding SDN [46].

SDN is an emerging paradigm which focuses on programmable networks and attempts to simplify network management [42]. To be specific, the focus of SDN is to make the networks programmable, by separating the control plane from the data plane. Through software applications and with the aid of a controller, the network devices (switches, routers) can be controlled. Thus, the whole behavior of the network can be managed and the complexity can be reduced.

1.3 Scope of the Thesis

The main scope of this Master's thesis, is to describe the model of RINA, as well as how it addresses and tries to solve some the current Internet problems. In particular, the main focus points of the thesis are the following:

- The shortcomings of the current Internet architecture
- The structure of RINA, including its protocols and mechanisms
- How RINA differs from TCP/IP as well as how it attempts to solve some of the Internet shortcomings
- How RINA can coexist with the current architecture, at least initially
- The ongoing projects of RINA
- How the recursive model of RINA is related to SDN

1.4 Thesis overview

The document contains various chapters, which cover details about RINA and provide a full description of how this architectural model works.

Furthermore, this document provides information related to SDN, which is an approach to manage networks through programmability and attempts to embrace technologies like RINA [18]. The details of each chapter are given below:

- Chapter 1, Introduction.

  This chapter provides a general background regarding the early state of the Internet, which was guided by the end to end principles. Since then, the Internet had a rapid expansion and as a result new requirements like more demanding applications and reliability issues have emerged.

  However, the new requirements, made networking and the Internet more complex and motivated new design strategies to accommodate the growing "tussle" between the different stakeholders (providers, users who want to run applications and the governments). The aim of these stakeholders is to interfere in communications, in order to satisfy their needs.

- Chapter 2, The evolution and the shortcomings of the Internet.

  This chapter provides a brief history regarding the origins of the Internet and how it evolved. The state of the Internet is "puzzling" the network community and there are many reasons behind this confusion. Some of them are the numerous personal devices, the increased requirements of some applications as well as the "big data". This "puzzling" is proved with different debates which exist, regarding the future of the Internet. The main focus of this chapter is the shortcomings of the current Internet architecture.

- Chapter 3, RINA and its implementation projects.

  This chapter focuses on RINA and its architecture, which is based on two principles:

  - Separation of mechanism from policy
  - Recursion.

  John Day, the starter of this idea, attempted to apply the concept of **separated mechanism from policy** taken from the operating systems, in networks!

He concluded that there are two basic mechanisms in the protocols and these mechanisms can be used under different policies. Thus, in RINA less protocols are required!

As far as **recursion** is concerned, John Day, noted that the protocol functions are repeated in different layers and he came up with a theory that networking is based on interprocess communication. This was first observed in 1972 by R. Metcalf of MIT project MAC [57].

The chapter also focuses on the differences between TCP/IP and RINA, as well as on the current projects of this new architecture.

- Chapter 4, RINA and Software Defined Networking.

  This chapter focuses on the architecture of SDN, which attempts to make programmable networks through software applications, in order to simplify network management and enable innovation and evolution. The main principle of SDN, is the **separation of control plane from the data plane**. To be specific, the control plane decides how to handle the traffic while the data plane forwards the traffic according to decisions that the control plane makes. Finally, this chapter presents the application areas of SDN and how this architecture is related to RINA.

Chapter 2

The evolution and the shortcomings of the Internet

This chapter firstly provides a brief history regarding the origins and the evolution of the Internet, from the early ARPANET to the rapid and global growth of it. Furthermore, the different views regarding the future of the Internet are presented, as well as how the idea of RINA which is a clean slate architecture came up. Finally, the current Internet problems are described, in relation to the architecture, addressing, routing, security and protocols as well as what is the 'root of cause' of these problems according to the proponents of RINA.

2.1 Origins of the Internet and the increase of network requirements

The development of the first electronic computers in the '50s, was a sign of the first steps in the history of the Internet. Packet switched networks were firstly introduced in laboratories in the US, England and France. Specifically, in the **1960s**, the Department of Defense in the US developed ARPANET [2]. ARPANET was the first packet switched network which led to the development of protocols for internetworking, in which multiple separate networks could be joined into a network of networks.

At the beginning, NCP was used as the communication protocol in ARPANET. It provided a reliable communication among applications running in different hosts. The hosts were connected to the network through IMPs, the first generation of today's routers [57]. Furthermore, there were only 3 applications: Telnet, FTP and RJE [7]. The first communication over ARPANET took place in **1969** between the University of California and the Stanford University.

In **1972**, Louis Pouzin, a French researcher, created the packet switched network CYCLADES, which was developed to explore alternatives to the design of the ARPANET and to further experiment on packet switching and routing [4]. Louis Pouzin claimed that the switches in the middle of the network didn't have to keep track of connections but to only pass the packets as they arrived.

Meanwhile, as ARPANET expanded, it became clear that NCP would not be capable of keeping up with the growth of the network.

In **1974**, a more robust suite of communication was proposed. TCP was introduced as a draft specification, which described how to build a reliable, host-to-host data transfer service over a network.

In **1981,** the access to ARPANET was expanded through the CSNET, which extended the networking benefits for the academic departments. Furthermore, the Internet Protocol was introduced in draft form and described how to implement an addressing standard and route packets between interconnected networks.

On January 1, **1983**, the TCP/IP was introduced as the networking protocol for ARPANET and made communication between nodes easier. At the same time, the NCP was shut down. Thus, this day is also known as 'flag day' [4]. The core idea behind TCP/IP, was to develop a protocol which allows interconnection between networks of different technologies.

Currently, Internet architecture is a controversial subject in the networking community. The ideas are changing fast and maybe in the future the Internet will be totally different from what we know today. The rapid growth of the Internet continues and more and more devices are connected to it, resulting to an increase of the network requirements. Server virtualization and the cloud services are driving the networking industry to reexamine the traditional architectures. Some of the reasons which are also puzzling the networking community are the following [19]:

- Traffic patterns: the applications need to access numerous servers and databases before returning the data to the end user device. Furthermore, there are so many devices from which the users want to access applications, anywhere and anytime.

- Increase of personal devices: the users, use personal devices such as tablets and smartphones to access the network. The privacy and the protection of data must be ensured. Thus, the work of IT is increased.

- 'Big data': it becomes difficult to process the large and complex data sets. This processing requires network capacity in the data centers and massive parallel processing on thousands of servers. As a result, the network operators face the problem of network scaling.

2.1.1 Evolutionary research vs clean slate

Due to the increased network requirements, there are currently different debates regarding the improvement of the Internet architecture. For example, there is a debate whether to focus on understanding and improving the current Internet (evolution) or to design new architectures from scratch (clean slate) [5]. In other words, there are different views whether to resolve the existing problems with backward compatibility and incremental deployment (e.g. create a new protocol) or to create a more secure and resilient 'new Future' without being constrained by the current architecture. Both approaches have advantages and disadvantages.

The evolutionary research proponents, claim that in the past there were also clean slate approaches like CLNP, but they weren't adopted from the Internet. Instead evolutionary approaches like NAT, and caching were eventually adopted because they worked well with the legacy architecture.

On the other hand, the clean state proponents claim that many issues of the today's Internet like reliability, support for mobile hosts or the management of a large network, can't be solved without fundamentals and 'rules of thumb'. Thus, they claim that by reconsidering the fundamentals of networking, a clean slate approach will allow to explore new designs and solve the current Internet problems.

The term clean slate means a fresh start and the idea began at Stanford University. The main topic was how the Internet could be redesigned from the beginning, without being based on the existing complex systems and networks but using the fundamental principles and experience gained from the past [3]. The fundamental principles remain stable across different scopes!

The program of the Stanford University sets two questions: a) **How the Internet will be in 20 years from now with the current scalability issues?** b) **With what we know today, if we were to start again with a clean slate, how would we design a global communications infrastructure?**

The researchers of the clean slate program, recognize that there are scaling and operational problems in the current architecture which need to be reconsidered.

Furthermore, there is a tendency of many NRENs to transform into Telecom operators mainly due to economic reasons [21]. This means that the commercialization of the Internet is growing and the academic influence is fading out. As a consequence, there are researchers who identify these problems and they move towards a clean slate approach.

One of these researchers is John Day, who claims that the design of the Internet wasn't based on scientific principles and notes that it is time for research and to reconsider the basic architecture. John Day is the pioneer of **RINA which is an example of this clean slate approach**!

2.1.2 Purists vs pluralists

Another debate regarding the state of the current Internet architecture, is between the Internet purists and pluralists [20]. The former group has a monolithic view of the architecture, where the center of it is a single universal protocol, currently the IP and around which all else revolves. Purists aim for flexibility due to the fact that the architecture will remain in place for a long time. According to this view, only **virtualization** provides a way to install new architectures. Virtualization refers to the technology for supporting, on a real network, a set of independent virtual networks. Furthermore, virtualization can provide a large scale test-bed for parallel experiments.

The latter group, the pluralists, has a different approach to the architecture, according to which, IP is just one component of the overall system i.e. the Internet. Pluralists claim that the evolving architecture can be defined as a set of the various existing protocols. They also put more emphasis on short term performance improvements.

This debate together with the evolutionary research/clean slate one, proves that the current Internet is a controversial issue and there are thoughts to make radical changes.

2.1.3 The IAB Workshop

The fact that the current Internet is facing problems was also recognized from the IAB, which held a workshop in 2007 in order to tackle some long standing concerns regarding routing and addressing [40]. The main objective of that workshop, was to identify the factors which have major impact on the scalability of routing and addressing systems. The growth of the routing tables due to multihoming as well as the overloading of IP address semantics were some of the problems which were acknowledged by the IAB workshop.

From the above, it is clear that the Internet architecture faces some problems which may increase as the networks expand.

2.2 Current Internet shortcomings

2.2.1 Naming and addressing

Naming and addressing wasn't a major concern in early data communications. In the beginning, the networks were simple and small. Hence, naming and addressing was not an issue.

Similar to the telephone system, in the early networks simple enumeration was used for addressing. Because ARPANET started as a research project, no one expected that it will grow so fast. As a result of the rapid expansion, the scientists of that time focused more on solving crucial problems such as routing of the packets rather than on naming and addressing issues [7].

The early work on networking was not careful to distinguish the concepts of an object and the name for that object. The name indicates 'what' we seek and the address 'where' it is [22].

However, the term 'address' causes problems when it is used for several different functions. To be specific, a number of problems arise when one name (IP address), is used for many functions i.e. **overloading the IP address semantics** [22]. In other words, the **IP address is the only name** in the TCP/IP architecture and has multiple uses: a) used directly by the routers, b) used to name the destination network, c) used to name the host of the end –to-end communication.

At the beginning, the address which was used on the IMP was 8 bits long [7]. The IMP was used to interconnect participant networks to the ARPANET. However, in the late '70s, the address size was expanded to 16bits due to the growth of the network. This means that with **more and more hosts added to the network the need for addresses would become bigger** [7].

The main problem of the ARPANET which exists until today, was the following: If there are two lines running to the **same** host from two different IMPs, they have two different addresses and they appear as two **different hosts [7].** This means that the addressing model needed to be reconsidered.

In 1982, a computer scientist Jerry Saltzer attempted to solve the naming and addressing issue, by applying the concepts of operating systems to networks. That time, there was enough knowledge regarding operating systems and as a result Saltzer tried to apply the same concepts to networks. He noted that there are four elements that need to be identified: applications, nodes, points of attachment to the network (PoAs) and paths [7].

According to Saltzer, an application can run in one or more nodes and should be able to move from one node to another without losing its identity.

Furthermore, a node can be connected to a pair of PoAs and should be able to move between them without losing its identity. The same happens in Operating Systems where the file names must not change if stored on a different disk.

Finally, the pair of PoAs should be connected by one or more paths.

Furthermore, Saltzer noted that there was a mapping from application names to node addresses and a mapping from node addresses to PoAs. As a result, routing was a sequence of nodes and PoAs (blue line in the Figure 2.1).

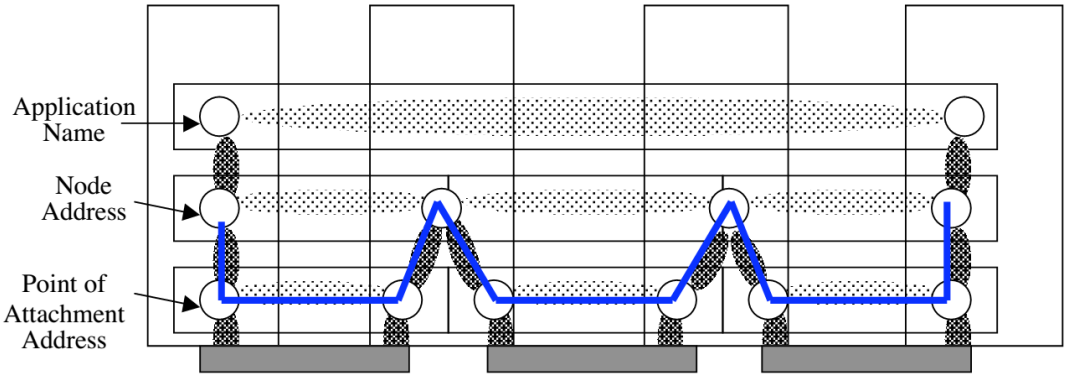

Figure 2.1 Saltzer's model

The Saltzer's model was based on operating systems but according to John Day, it was not completely correct for networks, since there may be more than one path between the same pair of nodes [7].

Thus, the researchers of RINA propose an extension of Saltzer's model in order to solve the naming and addressing issues. The next chapter describes in depth the RINA and its principles, as well as this extended model.

2.2.2 Routing, mobility and multihoming

As mentioned before, currently there is no notion of application names. The network has to use a combination of the interface address and transport layer port number, in order to identify different applications. On top of that, the interface (PoA) is named twice by the IP and MAC addresses (see Figure 2.2). **Due to the fact that the IP address refers to a PoA, the route is a series of PoAs** [23]**,** [10]. Hence, according to the researchers of RINA, there are problems with mobility and multihoming, when the application moves.

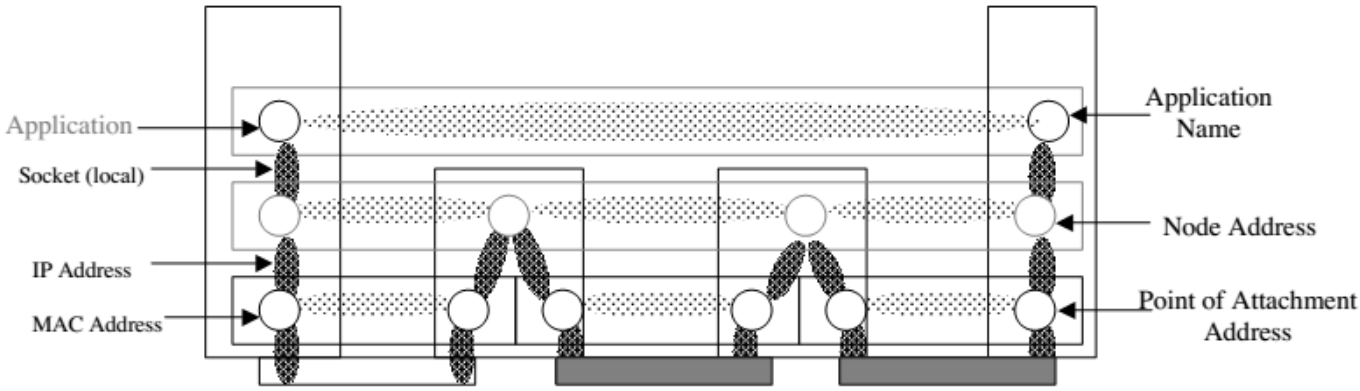

Figure 2.2 the Internet

Companies nowadays turn to multihoming, in order to increase the reliability of their services and to avoid network failure. Support for multihoming and mobility though was not a primary goal in the beginning of the Internet.

John Day claims that due to the naming and addressing issues of the current architecture, mobility and multihoming are inefficient.

Multihoming is a mechanism used to configure one computer with more than one network interfaces or multiple IP addresses [9]. Many nodes may need to be connected with more than one connection to the network at the same time and every backbone router needs to keep track of links to every network. There are hundreds of thousands of them. Thus, this leads to **scaling problems [4]**.

The problem with multihoming appeared very soon, in 1972, when an Air Force Base named Tinker joined the Internet and wanted to have two connections. However, this wasn't possible [23].

As mentioned before, in the current TCP/IP architecture the name is represented by the IP address. Hence, **the IP addresses are usually confused to be node names and the network can't understand that the two or more IP addresses of a multihomed node belong to the same node [40]**.

Mobility can be viewed as a special dynamic case of multihoming, which means that any architecture that provides multihoming will be capable to provide mobility too [31]. Mobility means that a host may wish to move to a different network while keeping the TCP connection open [22]. When this happens, the host unsubscribes from the old location and subscribes to a new one.

In a TCP connection, there is a combination of the IP source and destination address as well as of the IP source and destination port [22]. To get the packets to a different domain (usually the case in vertical hand-offs), a different IP destination address has to be used.

However, with a different address, **the packets will not be recognized as belonging to that connection** and the connection will break. This means that the node will lose its identity [13].

Furthermore, while the node moves from one network to another, this means also that one interface becomes inactive and another active. According to the researchers of RINA though, switching to another interface is costly [31].

Furthermore, during a switch from a fast network to a slow one and vice versa, the TCP sender does not have a fast way to adapt to the new characteristics of the channel and update the parameters (e.g. congestion window, transmission rate) in order to achieve the best utilization of the resources [13].

The above problems are caused, due to the fact that TCP wasn't designed with hand-offs in mind nor it is clear that it should have been, because when TCP was created no one expected the rapid growth of mobile devices in the future [13].

Routing is another concern regarding the current Internet architecture. The increasing user population, multihoming, traffic engineering are some of the factors which cause the growth of the routing tables [40].

For example, with traffic engineering, certain traffic of the Internet can be arranged to use or avoid specific network paths. As a result, there are multiple links and paths which lead to bigger routing tables [40].

Furthermore, a large portion of the growth of routing tables, comes from the de-aggregation of the address prefixes [40]. This happens, due to the fact that routing is performed on the interface [56] rather than on the node as well as to the fact that IP addresses are used both as locators and identifiers (overloading of IP address semantics).

Let's consider the following case:

How does a connectionless router know where to route the packets?

As already mentioned, the IP address doesn't actually tell you where the packet is going, it just names the destination network and the interface.

So the connectionless routers require more context than the connection-oriented ones, because they need to maintain connections to every other node on the Internet. This results to bigger routing tables. **The bigger the routing tables the more unscalable the networks become [4]**.
According to Cisco, currently the routing tables have officially passed the 512.000 route mark [6]!

2.2.3 No indirection

Besides the DNS, which is the only directory service in the current Internet and provides synonyms for IP addresses (using the well- known ports), there is no other form of directory that maps application names to nodes [24]. But what happens when the users and developers want to know where the applications are located?
What is used today for naming applications, are the URLs which use the IP address and a socket number as part of the application name [24]. However, if a parameter of the name changes, then the whole application name changes and as a result mobility and multihoming are affected.
In addition, the current architecture does not provide names for layers either. As a result, there is no way to describe on which layer a requested application is on. All discoverable applications are assumed to be accessible via the same layer instantiation [29].

2.2.4 Protocols

Currently, there are a lot of protocols and each one has a 'job' to do. Every protocol has a set of functions to achieve the basic requirements of that protocol, like error control, flow control etc. The operating region of the protocol determines which function has to be used. Furthermore, each of these functions is divided into a mechanism and a policy. In simple words, policy means what we want to do and mechanism how it can be done. Mechanisms are static and don't change in a protocol while the policies change and depend on the QoS required by the user [7].
In the current Internet architecture, TCP is used as an identifier of the connection and provides error and flow control. However, the **mechanism and policy of TCP are not separated** [4].
To be specific, TCP provides the feedback from receiver to the sender which allows the sender to retransmit lost packets. However, **TCP's header sends these feedback messages in the same packets which carry the payload.**

According to John Day, by separating mechanism from policy, the operating range of a protocol can be increased and as a result, an architecture would need **fewer protocols** [7]. Thus, the complexity of the networks can be decreased.

2.2.5 Security

As the Internet has grown and evolved, the TCP/IP architecture has shown signs of weakness regarding security. There are a lot of threats like DoS attacks, Trojans, viruses etc, despite the numerous security mechanisms which were developed. Any application can connect to any other application without permission [11].

At the beginning, there was no security because the networks were simple, small and everybody knew each other. Over the years though, the networks became more complex and many security issues have been discovered. As a result, many new protocols were designed in order to tackle the security problems. However, the numerous protocols also **increased the complexity** of the Internet architecture [11].

In the TCP/IP architecture, addressing is global, which means that the addresses are **exposed to applications**. Thus, any system can freely connect to any other system. Furthermore, TCP overloads the port-id, to be both an identifier for the application process and a connection-endpoint- id which identifies the data transfer connection. This results to a **well-known destination port [11], [56]**.

Anyone can use the well- known ports and as a result there are security risks with port-scanning attacks, connection-opening attacks, data-transfer attacks etc.

2.2.6 Architecture

Without doubt, the layering system played a significant role in the expansion of the Internet!
Layers were brought into operating systems by Dijkstra and later were adopted by the networking architecture. A layer can be described as a set of processes over a scope.
The Internet has five layers (see Figure 2.3): physical, data-link, network, transport and application. A brief description of each layer is given below.

Physical layer: Responsible for the physical transmission. It represents the medium over which data transmission can occur. The scope of this layer is usually point to point.

Data link layer: It is directly above the physical layer. It's the protocol layer that transfers data between network nodes over the physical medium. It also provides the means to detect and possibly correct errors which may occur in the physical layer.
The data link layer and the physical layer are closely tied together because the data link layer controls the physical layer in terms of when the physical layer can transmit. As a result, decisions of the data link layer drive the decisions about the physical layer.

Network layer: This layer is responsible for routing and provides the functional means for transferring data from a source to the destination. The routing is done with packet forwarding from one system to another. Network layer include protocols such as IP (for relaying datagrams across networks) and DHCP (for dynamic distribution of network configuration parameters such as IP address).

Transport layer: Provides end to end or host to host communication services for applications within the architecture. Some of the protocols in the transport layer are the TCP and UDP. TCP handles flow- control and provides a reliable communication while UDP provides unreliable packet delivery. The scope of transport layer is larger than the scope of data link layer.

Application layer: This layer is at the top of the stack and includes the protocols and interface methods with are used by the lower layers for communication. Some of the protocols are the HTTP, FTP, BGP, etc.

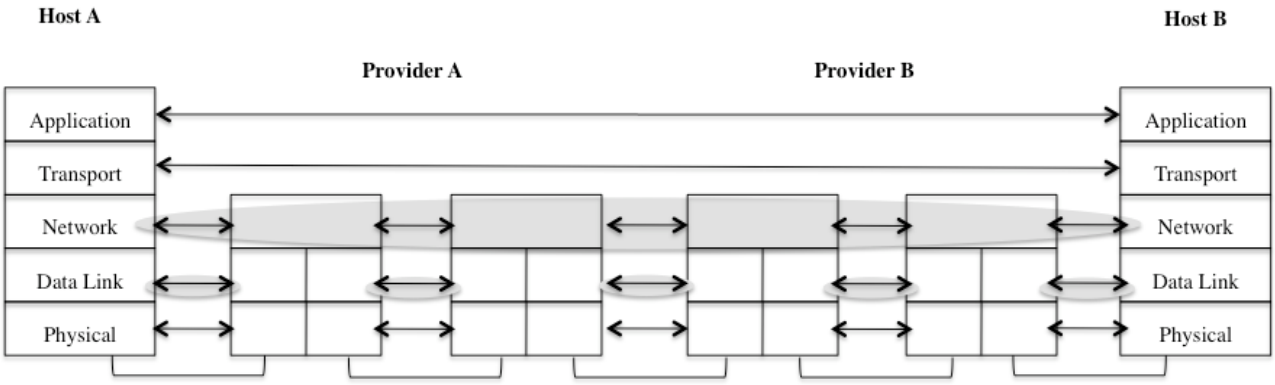

Figure 2.3 Current Internet architecture

In general, all the layers in the Internet are closely coupled and the decisions in one layer impose certain requirements on other layers. Each layer communicates only with the **layer directly above or below it.** Furthermore, **t**he current Internet architecture uses a fixed number of layers, with every layer responsible for a different function (functional layers).

According to the researchers of RINA, there are some issues with the current layered model.

The **first issue** is that the current architecture provides **only two scopes,** which is the data link (scope of layers 1 and 2) and the global (scope of layers 3 and 4). The researchers of RINA claim, that the current architecture lacks a mechanism in order to perform the same functions over different scopes [25]. For example a layer should be able to manage a wired network, another layer should manage a wireless network and a layer on top of them managing the end to end network.

As a result, the current architecture is not able to handle heterogeneous networks. In other words, the architecture lacks a mechanism to provide different configurations over different physical media [13].

In general, the Internet is originated from a wired network. Nevertheless, with the expansion of wireless technologies there is an effort to migrate the current Internet architecture to a hybrid environment. However, according to the proponents of RINA, due to the lack of scoping, the current Internet is not designed to handle heterogeneous networks in which mobile users can enjoy services while roaming between different networks [13].

Furthermore, another reason why TCP fails to adapt to a heterogeneous environment is, its congestion control mechanisms [13]. Congestion is the network state, in which a node or link carries so much data that may result in delay frame or packet loss.

In the wireless links, which are characterized by interrupted connectivity and frequent hand –offs, the BER is high.

The losses which happen in wireless links, **are falsely interpreted by TCP for losses due to congestion in the network** and therefore TCP triggers the congestion control mechanisms. This leads to reduction in window size and as a result to a **severe drop in throughput [13]**.

The **second issue** is the nature of layering. The layered structure which started in operating systems, proposed that layers don't have to repeat and the functions have to be independent [25]. However, in networks there were conditions that seemed to require functions to repeat. For example, the need to have relaying and error control in more than one layers. Nonetheless, this is opposed to the non-repetitive and the independent nature of layering.

The **third issue** is, that TCP/IP breaks the rule that the functions have to be independent [25]. In fact, the functions of IP are related to the TCP mechanisms! IP stands for a logical address which works as packet address. The TCP works with this logical address, helps the packets to reach their destinations and provides acknowledgement when the packet reached its destination. **If these two protocols are related, why to split them?**

2.2.7 Quality of Service

QoS is defined as the collective effect of service performances, which determine the degree of satisfaction of a user of the service. It characterizes bandwidth, delay, error rate, etc [48]. The modern applications like VoIP or real- time streaming applications have specific requirements. QoS requirements were not foreseen when ARPANET was created. However, due to the rapid growth of the Internet, the QoS requirements have increased.
According to the researchers of RINA, the problem of the current architecture, is that there is no built –in mechanism that allows the network to provide specific QoS [24]. Applications have no way of expressing their desired service characteristics to the network, other than choosing a reliable (TCP) or unreliable (UDP) type of transport.

2.2.8 Network management

Management is a vital component for delivering the requested network services. Naturally, the Internet wasn't designed with management in mind.

However, nowadays the network administrators face problems with configuration, traffic engineering and routing [28]. The management costs as well as the complexity of the networks are increased and the existing approaches often restrict the range of policies that can be employed to adapt to diverse network conditions. Due to the numerous mechanisms, protocols and configuration interfaces, there is no flexibility in the network management and the network administrators have to configure each network device separately [42]. Thus, this mode of operation has slowed innovation.

2.3 Communication between distributed applications in the Internet

This section makes a general conclusion, regarding communication between applications in the Internet. It is important to see how communication is performed firstly in the current Internet (see Figure 2.4), so we can make a comparison later on with RINA.

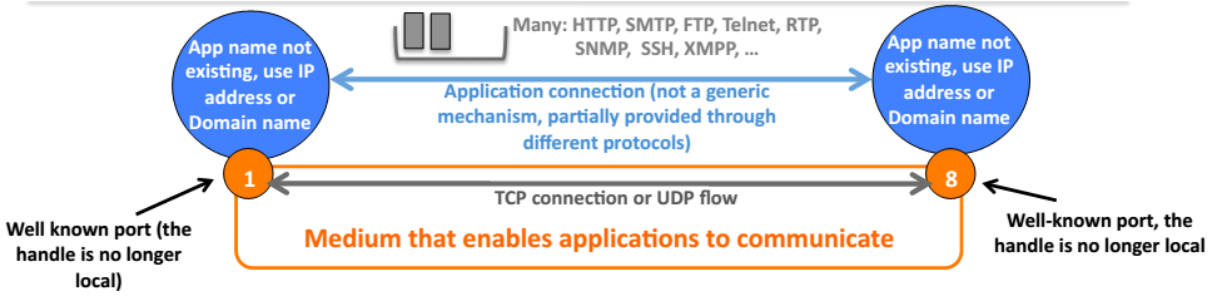

Figure 2.4 Communication in the Internet

For communication it is required [47]:

- **Application process naming**-> There are **no names** for applications but only IP addresses and ports
- Two types of **flows**-> TCP or UDP with **fixed characteristics**

- **Communication medium API**-> Need to know in which port an application is attached to (well-known port), in order to allocate a flow. Also, there is **no way to express desired properties of the flow**
- **Application connection**-> The applications have to **know in advance** which protocol is going to be used
- **Application protocol**-> There are **numerous protocols which increase the complexity of the networks**

2.4 Attempts to fix some of the Internet problems

2.4.1 Overlays

Because the ARPANET and TCP/IP worked well, the whole project was funded by the government and there was also no competition, it went into the production immediately [4]. Due to this hustle, temporary solutions were created in order to fix any problems that appeared. Nevertheless, due to the explosiveness of the Internet and the increase of applications and networks, these temporary solutions became permanent.

Network engineers have found a variety of ways to work around some of the shortcomings of the Internet [15]. For example the mobile telephone networks and other delay tolerant networks which are deployed in mobile and extreme environments (e.g. military networks) don't use IP. In fact they use their own specialized protocols.

In addition, many important enterprises such as Akamai and Skype use **overlays [15].** Akamai uses this solution, in order to provide high availability and low latency. It is a custom –built distributed system that runs on top of the core layers in order to use a different architecture. Thus, it is used both as an experimental platform and a deployment path.

An overlay has members, each of which is a process and offers a communication service to the above overlays and applications (see Figure 2.5).

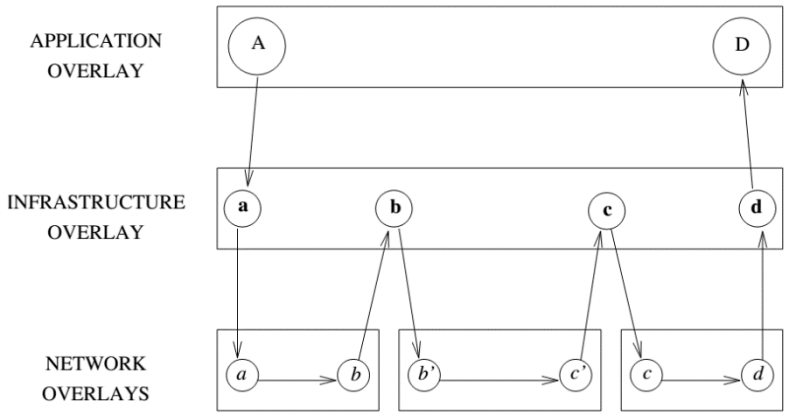

Figure 2.5- Overlay structure

Example [15]

The application overlay (see Figure 2.5) consists of the application processes A and D which want to communicate with each other. The infrastructure overlay consists of the processes **a, b, c** and **d** which represent the route from A to D. Finally at the lowest rank of the figure, there are three network overlays. The processes inside them represent the points of attachment to these networks.

This structure provides opportunities for mobility. For example if a process moves from one network to another, the ability to maintain session from A to D will remain because what is changed is the point of attachment and the route to reach the destination.

The above example proves, that there is a tendency to solve the current Internet shortcomings but with the use of other architectures.

However, the use of overlays is a controversial issue and there are researchers who claim that overlays suffer from limitations of their own [20].

According to these researchers, the overlays provide a fix to **specific problems** in the Internet architecture, whether for performance, availability, denial of service etc. In other words, the overlays provide isolated solutions and there is not much progress to determine how any of these solutions might work together. Furthermore, most current overlays, typically assume IP or a close cousin of it, as the basic architecture. As a result, they have not been the source of an architectural advancement.

2.4.2 LISP, MIP and IPv6

Many solutions have been proposed to solve the addressing problems and scalability of the routing system. The **LISP** which is proposed by IETF is one of them. Because the **IP address is overloaded and used for many functions** as noted before, LISP proposes to make a separation of the IP address into a **locator part** and an identifier [31]. In other words, LISP separates the address space, into end-systems identifiers for source and destination hosts, and routing locators where border routers act as routing locators for the end-systems (see Figure 2.6). The MS is used to store the mappings of endpoint identifiers to routing locators.

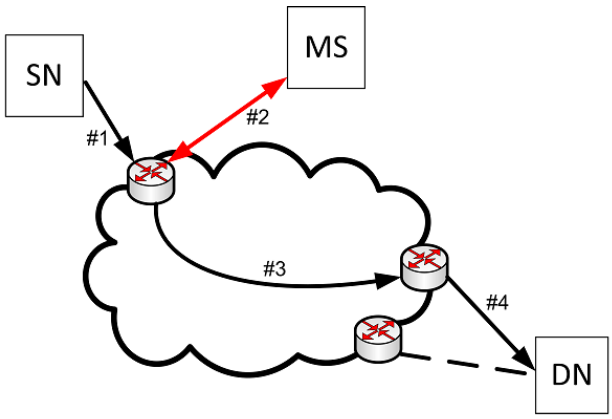

Figure 2.6 LISP mode

The basic procedure is the following:
The SN forwards the packet to its border router.
The source border router afterwards performs a lookup query for a destination endpoint identifier to routing locator, using the MS. Then the source router tunnels the data packet to the destination's border router and from there the packet is forwarded to the destination.
In the figure, the DN is attached to two border routers (multihoming).
If an active interface fails, the multihomed destination node, sends an update to its border router and through the MS, it switches to the operational interface.

Another proposed solution for the routing problem, is the **MIP** [31], which allows a mobile host to seamlessly move from its home domain to a foreign location without losing connectivity (see Figure 2.7). In MIP, there are two basic mechanisms: 1) a discovery mechanism which allows a node to detect its new PoA and 2) a registration mechanism, which is achieved by having a foreign agent update the location of the mobile node at its home agent. Since mobility can be seen as a dynamic form of multihoming, MIP can also be used to handle a change in the active interface (due to failure or re-routing) leading to a multihomed node. This is achieved by the home agent, which directs traffic to the currently active (operational or 'better') interface.

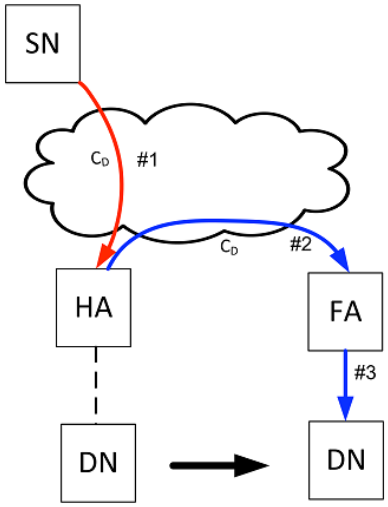

Figure 2.7 MIP

The basic procedure is the following:
Supposing that the DN moves, the SN sends a packet to the DN.
The DN moves to a new autonomous system and acquires a care-of-address at the FA. Then the FA updates the HA with DN's new location.

Both LISP and MIP solutions, help to reduce the size of the routing tables since several IDs can map to one location and be represented as one routing entry. However, according to the researchers of RINA, the location update in MIP as well as the switching between interfaces in LISP, can be costly [31].

IPv6 is another attempt to solve the addressing problem, the explosion of routing tables, as well as the IPv4 address exhaustion. IPv6 are 128-bit identifiers for interfaces.

The major advantage of IPv6, is the bigger address space. Furthermore, through hierarchical addressing, IPv6 limits the expansion of routing tables [49].

However, according to RINA proponents, IPv6 still fails to tackle the fundamental problem, which is **routing on the interface and not on the node itself,** leading to naming and addressing issues [7].

Finally, IPv6 wasn't designed to be backwards compatible. This means that the transition from IPv4 to IPv6 will be costly.

These can be some of the reasons which contribute to the slow migration process to IPv6.

2.5 Summary of the chapter

This chapter presented the evolution of the Internet as well as the "puzzling" which is caused to the network community regarding the future of the Internet. There are different views, whether the architecture of the Internet should evolve or create a new one from scratch.

The focus of this chapter is the current Internet shortcomings in naming, addressing, routing and security. Furthermore, there are attempts like LISP and MIP, which are proposed in order to solve the addressing and routing issues. The overlays, which are distributed systems and run on top of the stacks of the legacy architecture, is another approach in order to test new architectures and avoid some of the problems of the TCP/IP model.

Chapter 3

RINA and its implementation projects

This chapter presents the architectural model of RINA and its basic principles as well as some fundamental mechanisms and basic operations of communication which take place in RINA. Moreover, the differences between RINA and the current Internet architecture are presented and at the end the European ongoing projects are given.

3.1 Phases of communication

Before we move into depth regarding RINA, we must consider some fundamentals regarding communication between different systems. **This and the following section, describe the phases of communication as well as some basic mechanisms of protocols.**

When two systems exchange data, the phases and what has to be done in order for these two systems to communicate with each other are often neglected. Thus, the focus is mainly on the part of **data transfer**.

However, in order for communication to occur, two phases must take place first. Firstly**, enrollment** must be done and afterwards the phase of **establishment which is also known as synchronization or allocation**. After these two phases, **data transfer** can begin and the two systems can start communicating with each other [7]**.** These phases are described briefly below.

Enrollment: According to John Day, the enrollment phase creates, maintains, distributes and eventually deletes all the information which is necessary to create instances of communication. This phase makes an **object and its capabilities known to the network** and it is used to create all the necessary information for communication.

A typical example from the everyday life, is to place someone's name in a catalogue; which actually means to enroll this person and make him known. The enrollment phase has always been there but often it was neglected due to the initial configuration which was required.

Establishment: The establishment phase creates, maintains and deletes the **shared state** necessary to support the data transfer phase.
The shared state is used to share information about each system's state participating in the communication. During this phase, specific QoS requirements for data transfer are made if not fixed during the enrollment phase.
An example of this phase is when two people meet. Before they start discussing (data transfer phase), they greet each other.

Data transfer: The data transfer begins, when the actual transfer of data is affected by the requested QoS. This is the phase that we typically think of as communication. Typical example is when two people are discussing which means that they are exchanging data.

3.1.1 Data transfer mechanisms

Numerous mechanisms have been found to occur in many protocols. Some of them are used also in RINA and are described below [7].

Delimiting: this mechanism is used to indicate the beginning and the end of a PDU. In general, a PDU includes two parts: PCI and user data. The PCI is the part understood by the protocol and the user data the part which is not. There are two methods for delimiting a PDU: external and internal. **Internal** means that delimiting is included in the protocol. In other words, the PDU contains a length field from which its end can be calculated.
In **external** delimiting, a special bit pattern (flag) is used to denote the start and the end of a PDU. Moreover, in external delimiting the lower layer is used to delimit the PDU.

Relaying: this mechanism is used to forward PDUs from one network to another. Relaying is usually useful when there are networks which are not fully connected meshes. For example if there are two nodes which are connected to a third one but not to each other, by using relaying, the messages can be forwarded across that third node.

Multiplexing: this mechanism is used to combine multiple flows into a single one. In other words, PDUs from different flows and from different sources will be sent on the same outgoing flow.

Addressing: to identify the destination, as well as in some cases the source of the PDUs, the addressing mechanism is required.

Ordering: ordering means that the PDUs will arrive to the destination in the same order they were sent by the source. There are different kinds of ordering according to different applications. For example some applications may require ordering but may not require all PDUs to be received.

Authentication: this mechanism is used from the destination in order to authenticate the source.

Flow control: in general, flows are used for communication service between applications and they also transport well defined units of application data. A flow is locally identified by an application through the use of a port-id. Flow control is used to control the sending rate of the PDUs. There are cases in which the destination can't process the PDUs as fast as the source are sending them. As a result, flow control is necessary.

Policy selection: as already mentioned in the previous chapter, each function of a protocol has a mechanism and a policy. Policy represents what the user wants to do and the mechanism how it can be done. To choose a sequence number, or to set a maximum size of a PDU are some examples of policies. Policy selection allows to select policies during initialization.

Flow/connection identifier: this mechanism is used to distinguish multiple flows between source/destination pairs. Thus, protocols that support multiple instances of communications use for example port-ids to disambiguate one flow from another.

Retransmission control (acknowledgement): in order for the destination to inform the source that the PDUs have been successfully received, an acknowledgement mechanism is required. If the source doesn't receive an ACK then it automatically retransmits all PDUs.

Access control: this mechanism is required in order to check if the source has access to the destination.

Integrity/confidentiality: in order to protect a communication against unauthorized insertion or deletion of PDUs, the integrity mechanism is required. With confidentiality the user data is 'protected' from unauthorized users.

Data corruption: it includes mechanisms in order to protect PDUs from errors. For example CRC is a mechanism used to detect the corruption.

Sequencing: this mechanism allows large amounts of data to travel across networks quickly and in small separated packets.

3.2 Terminology in RINA

This section provides a list with some important terms which are used in the IPC model of RINA [7].

Application process: The instantiation of a program executing in a processing system intended to accomplish some purpose.

Distributed application: A collection of cooperating application processes that exchange information using IPC and maintain shared state.

(N)-DIF: A distributed application consisting of at least one IPC application in each participating processing system. The (N)-DIF provides IPC services to applications via a (N)-API.

(N)-IPC process: An application process that is a member of a (N)-DIF.

3.3 Communication between distributed applications in RINA

This section describes a general model of how communication between applications is performed in RINA (see Figure 3.1). More information regarding each characteristic of this model will be given in the next sections.

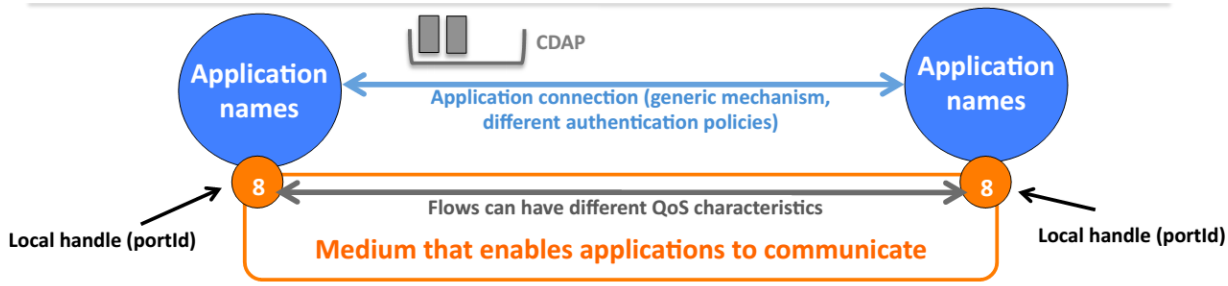

Figure 3.1 Communication in RINA

For communication between the two applications the following are required [47]:

- **Application process naming**-> RINA has a complete application naming scheme
- **Flows**-> The flows can have many characteristics depending on different application requirements
- **Communication medium API**-> Used to request allocation of flows to other applications, by name
- **Objects**-> Each application decides on their contents
- A generic **application connection** establishment procedure, with authentication policies
- A single **application protocol** -> CDAP

3.4 Separation of mechanism from policy

One of the basic principles of RINA, is to have as less protocols as possible in order to have a simpler architecture. This can be achieved by **separating mechanisms from policies**. This concept has a long tradition in operating systems and John Day attempted to apply that in networks [7]. As noted before, a protocol is composed of a set of functions and each function is divided into a mechanism and a policy (see Figure 3.2).

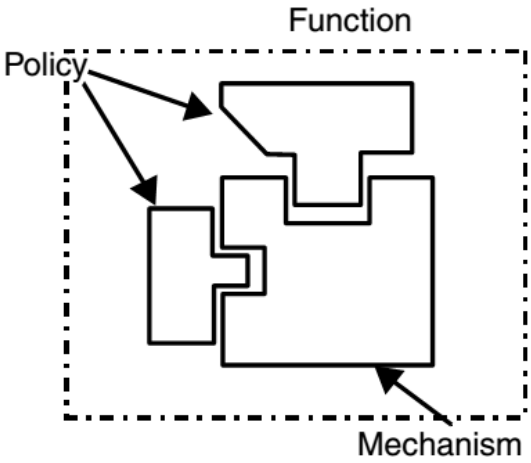

Figure 3.2 Mechanism/policy

Example: Let's consider the function for detecting data corruption. A specific CRC polynomial is required, which is **the policy** as well as to insert this polynomial into the PDU, which is done by **the mechanism**.

The choice of policy depends on the QoS required by the user and according to John Day an architecture doesn't require numerous protocols to accomplish different functions. What is required is to just use different policies.

After applying the concept of separated mechanism/policy in networks, John Day notes that there **only two kinds of mechanisms** in protocols [7]:

- Tightly bound: those associated with the transfer PDU and which are part of the PCI field, like CRC for detecting data corruption or generally the headers of a data transfer packet.

- Loosely bound: those associated with synchronization, flow control and which are not associated with the transfer PDU (not part of the PCI field).

John Day concluded that there are only two types of mechanisms in the protocols which can operate according to different policies [7]. **This enables the same basic mechanisms to be used in different ways and under different policies**. **As a result, an architecture will need less protocols!**
From the above, it was recognized that the only differences are in syntax (which are minimal and dictate neither policy nor mechanism) and as result in RINA only 2 protocols are needed: 1) **EFCP** which is used for data transfer and 2) **CDAP** which is used for the communicating applications. For more details about these protocols refer to section 3.8.

3.5 Inter-process communication and recursion

Besides the principle of the **separation of mechanisms from policies**, RINA is also based on another principle which is the **divide and conquer (recursion)**.
John Day as mentioned before, was one of the scientists who looked at the original requirements of the Internet. What John Day noticed is that the protocol functions are **repeated** in different layers. Thus, he claims that all the layers have the same functionality and they only differ in the scope, in the range of bandwidth, in the QoS and in the policies which are used. By embracing Robert Metcalfe, John Day supports the theory that **networking is IPC** [57], [7], [35].
IPC is a set of mechanisms for the exchange of data among multiple processes. These mechanisms allow the processes in **distributed systems** to communicate with each other and they also provide a mechanism that shields one process from failures of another [58]. The distributed systems (or computing) can be described as a set of autonomous computers which act as one large system and provide a spectrum of activities varying in the degree of decentralization [12]. The (IPC) processes may be running on one or more computers connected by a network and provide IPC services for the applications which are in the systems.
Unlike the current layered set of different functions, in RINA there is a single layer (DIF) of IPC that repeats over different scopes [26]. A **DIF is a distributed application** that provides IPC for a given range of QoS. Application processes which are in different systems are able to communicate using the services provided by a DIF.

Each instance of this repeating IPC layer **implements the same functions and mechanisms** but according to different policies and network needs. The greater the operating range in a network, the more IPC layers it may have.

A **set of distributed application processes is called DAF** [28]. The DAF performs different functions like weather forecast, genomics, communication service etc.

The DAF operates over a DIF (see Figure 3.3) and it is used to manage the DIFs which make up the whole network.

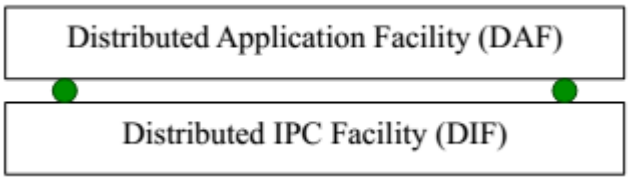

Figure 3.3 Distributed Application Facility

The relation between these two facilities is that the DIF is a specialization of a DAF and **provides only communication service, while the DAF has a more generic purpose**.

The **repeating structure of DIFs** scales 'indefinitely' [4] and provides better management of each IPC layer (see Figure 3.4) [56].Thus according to RINA, with this structure, the current problems of growing routing tables can be avoided, due to the more manageable scope of each IPC layer. Moreover, there is no need for purpose- built protocols for each layer. The same protocol can be used repeatedly in a protocol stack and can work across ranges of QoS and bandwidth. This makes the implementation simpler than the TCP/IP.

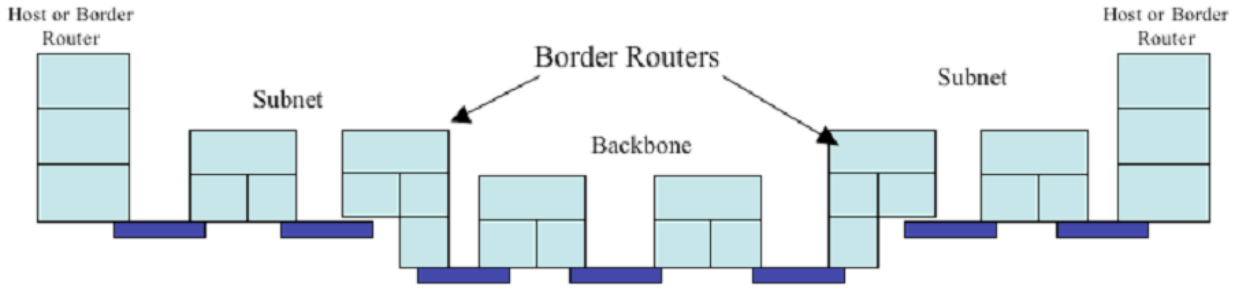

Figure 3.4 DIFs can be stacked according to the scope of the network

3.6 IPC Naming

The IPC process is an entity that provides IPC services for applications running on the same system. An IPC process has an external name in order to be used by other IPC processes. In order to facilitate its operation within a DIF, each IPC process gets also an address which is just a synonym (simply a short identifier used only within the DIF) with a scope restricted to that DIF [36]. The addresses are hidden from the applications and are internal to a DIF. Furthermore, in RINA the addresses are relative, i.e. the address of a process at one DIF is viewed as a name by a lower level DIF. An IPC process consists of the following identifiers (see Figure 3.5):

- The port-id refers to this instance of communication. It is also called Flow allocator Instance identifier.
- The CEP-id identifies the shared state of one end of the connection. It is also called Data transfer Instance identifier.
- The connection id identifies flows between two IPC processes, by concatenating the CEP-ids.

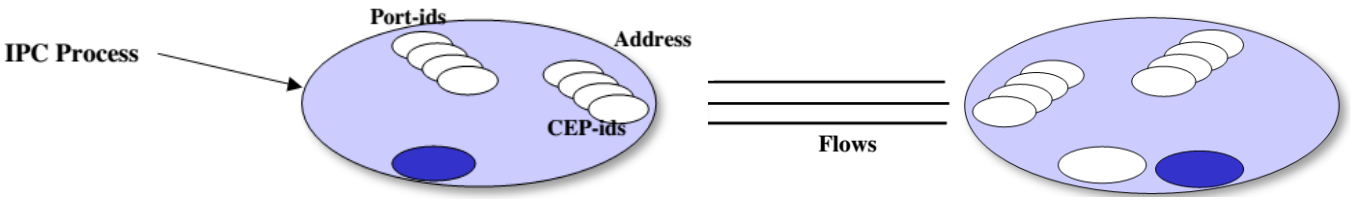

Figure 3.5 Names of an IPC process [36]

3.7 The DIF

A DIF is an organizing structure, grouping together application processes. It is what we generally refer to as a layer. Furthermore, it is a ‘black box’, which operates among multiple systems and it enforces strict layer boundaries, which means that what happens inside the DIF is not visible outside of it [4].

In the current Internet architecture, the layers don’t have names and it is difficult to describe on which layer the requested application is on. Thus, all discoverable applications are assumed to be accessible via the same layer instantiation [29].

In RINA however, the members of a DIF, are application processes and hence they have **application names [56]**. Any two application processes in different systems, can communicate using the services provided by a DIF.

The DIF is a distributed application consisting of at least one IPC process in each processing system participating in the DIF. A DIF provides the following service (interface) [30]:

- Allocate a flow to the destination application with a certain Qos
- Write data to the flow
- Read data from the flow
- Deallocate the flow resources

By separating mechanism from policy, it is possible to customize the behavior of a DIF, in order to operate on heterogeneous environments. The DIFs **can be dynamically instantiated**, and the policies can be tuned according to the network conditions (migration of an application, DoS attack, etc) [28]. The DIFs also have ranks ((N)-DIF, (N+1)-DIF, (N-1)-DIF, etc). An (N)-DIF may only know the names of IPC processes in a (N+1)-DIF. A (N+1)-DIF can use a (N)-DIF. The (N-1)-DIF is the DIF used by a (N)-DIF [45].

Furthermore, the DIFs have a scope. Each DIF provides IPC services over a limited scope. First level DIFs operate on top of a physical medium and their policies are optimized to deal with the characteristics of the physical medium [45]. The first level DIFs provide IPC services to second level DIFs, and so on. The protocols at each DIF are the same; they just use different policies to fulfill particular requirements. As the rank decreases the scope also decreases. However, all DIFs provide the same interface regardless of their rank.

RINA simply considers QoS to be a set of parameters inside the DIF. Thus, every DIF can provide every requested QoS, by using mechanisms which are hidden inside it [4].

To conclude, the DIFs perform a coordinated set of policy managed mechanisms to achieve the desired IPC service, rather than a single mechanism which happens in the current layered model of the TCP/IP architecture [28].

3.8 Structure of a DIF

This section provides a more detailed description regarding the structure of a DIF**.**

The DIF is a collection of IPC processes. The IPC process which is a member of a DIF, is an application process and relies on DIFs 'below' it for communication. **Each IPC process executes routing, transport, security/authentication, management functions** and provides an API to the user [24]. Thus, a DIF is a complete structure and provides all the functionalities!

The IPC API presents the service (a communication flow between applications) provided by a DIF and has four functions:

- **Allocation of the resources** to support a flow between the source and the destination application processes.
- **Deallocation of the resources** after the communication is completed.
- **Sending an amount of data (SDU)** to the port of the destination application process.
- **Receiving an amount of data (SDU)** from the destination application process.

Each IPC process consists of three major parts: a set of **fast mechanisms** for the **IPC data transfer,** a set of **slower mechanisms for IPC data control** and a set of **IPC management mechanisms** which operate at an **even slower rate** (see Figure 3.6) [56]. **The different rates, make the DIF capable of operating over different timescales and as a result it's more flexible compared to the layer of TCP/IP architecture** [28]. Each component of an IPC process can be configured differently **by using the appropriate policies** [24].

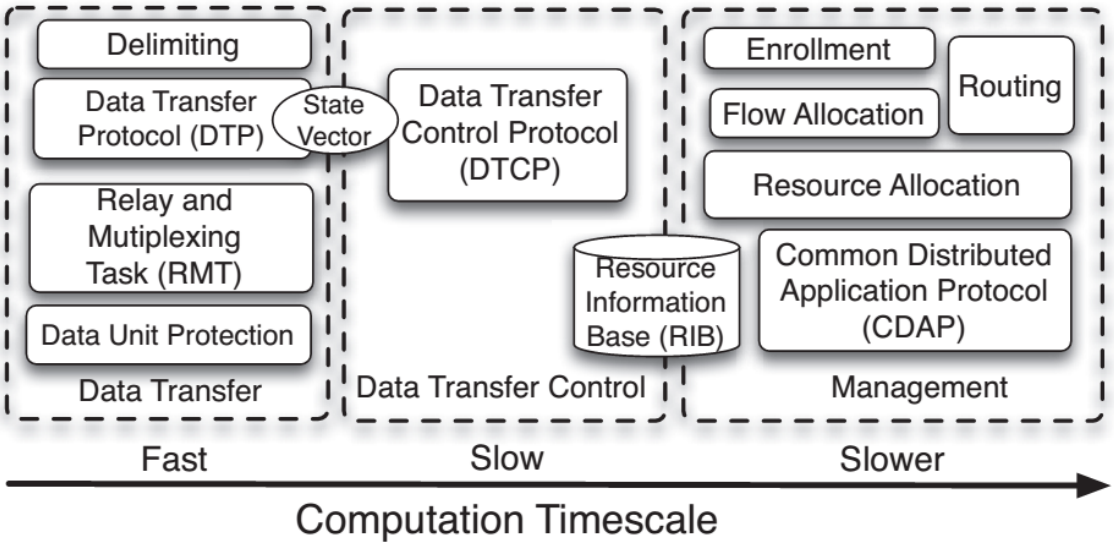

Figure 3.6 Structure of a DIF [8]

The **IPC transfer** functionality forwards data across, similar to the IP and UDP in the OSI layer architecture. It supports relaying (forwarding), multiplexing and per-flow data transfer. The computation is simple and fast!
It consists of the following functions (see Figure 3.6) [37]:

- Delimiting
- Relaying /Multiplexing Task
- SDU protection
- Data transfer protocol

The **IPC data transfer control,** implements the error and flow and controls the per-flow data transfer parameters.

The **IPC management** [37] task is responsible for routing, resource allocation, access control etc. Also it is used to manage the supporting DIFs. It consists of the following functions (see Figure 3.6):

- Flow allocation
- Authentication mechanisms
- The CDAP
- Resource Allocation
- RIB and RIB daemon
- IAP

The **delimiting** mechanism is used to denote the start and the end of a SDU [24]. In general, delimiting is done to ensure that the service maintains the identity on delivery.
The SDU is a piece of data that is delivered to a DIF but it has not yet been encapsulated into a PDU. The DIF will deliver the same SDU to the other side and in order to do this it must fragment it or concatenates it with other SDUs. Due to the fact that the receiver understands only the PDU, delimiting is required so that the SDUs are converted to user-data fields for PDUs [7].

For **data transfer,** the data transfer protocol (DTP) is used, which together with the data transfer control protocol (DTCP) comprise the EFCP, which is handled by the **IPC transfer control mechanism [28]**.

EFCP is one of the protocols which are used in RINA and in which the mechanisms are separated from the policies. It is in charge of supporting the data transfer related functionalities of an IPC instance within a DIF and it is also used to maintain synchronization and provide error and flow control [24]. This protocol is based on Delta-t protocol which is a soft state protocol and was developed by Richard Watson [38].

Briefly, Watson proved that the state of a connection at the sender and receiver can be safely removed **once 3 timers expire** and without explicit handshaking messages which currently happens in TCP/IP [56]. In other words, Watson followed the idea that in order to achieve reliable connections, flow control can be done by binding on 3 timers: maximum packet lifetime, maximum retransmission time and maximum time before acknowledgement [50].

The EFCP is composed of two cooperating sub-protocols, the **DTP** (IPC data transfer) and the **DTCP** (IPC data transfer control), loosely coupled through the use of a state vector. The DTP is **responsible for transferring the data** and includes mechanisms that are tightly coupled to the transfer PDU, such as sequencing, fragmentation, addressing [24].

The DTCP contains mechanisms which are loosely bound to the transfer PDU such as flow control and retransmission control mechanisms and provides feedback from the receiver (ack and flow control).

Unlike TCP, where a port number(TCP connection identifier) is mapped to one and only one TCP connection, in RINA **port allocation and data transfer are separate functions**, meaning that a single flow can be supported by one or more data transport connections (see Figure 3.7) [45]. This structure implies security advantages as it is described later on.

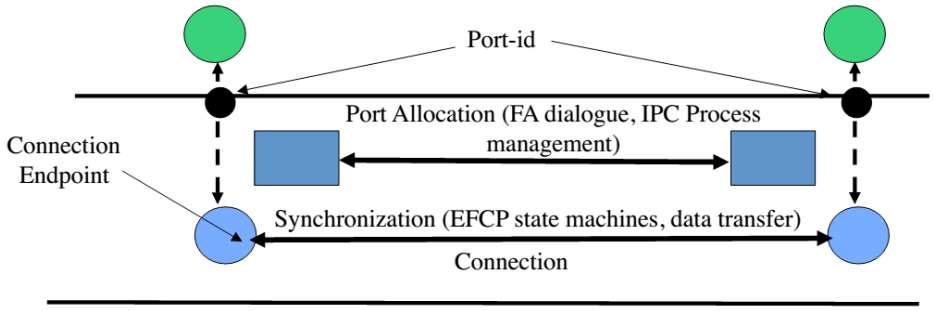

Figure 3.7 Port allocation decoupled from synchronization [32]

The port allocation state is created/deleted based on explicit requests, while the synchronization state is refreshed every time a dtp/dtcp packet is sent/received [47].

The EFCP operates between the IPC processes in the source/destination systems (see Figure 3.7). There is an EFCP instance for each different connection in each IPC process [45]!

PDUs go through an EFCP instance in one of two directions (see Figure 3.8) [45]:

- The PDUs generated by EFCP based on application processes SDUs, are passed to the RMT. The creation of a PDU involves delimiting, fragmentation and sequencing.
- Data PDUs received by the RMT from an (N-1) port, are passed to the corresponding EFCP instance for possible reassembly and delivery to the corresponding application process.

When a flow between two processes is allocated, an instance of DTP is always created (see Figure 3.8). The first step in this path is to delimit the SDUs posted by the application (right part of Figure 3.8) [45].

Depending on the policies, the DTP will provide tightly bound mechanisms [45]:

- Fragmentation/reassembly functions(breaking SDUs into more than one PDU)
- Concatenation/separation functions (combining multiple SDUs into one PDU)
- Sequencing

Afterwards, the RTM will multiplex the PDUs onto port- ids of the lower ranking DIFs, according to policies.

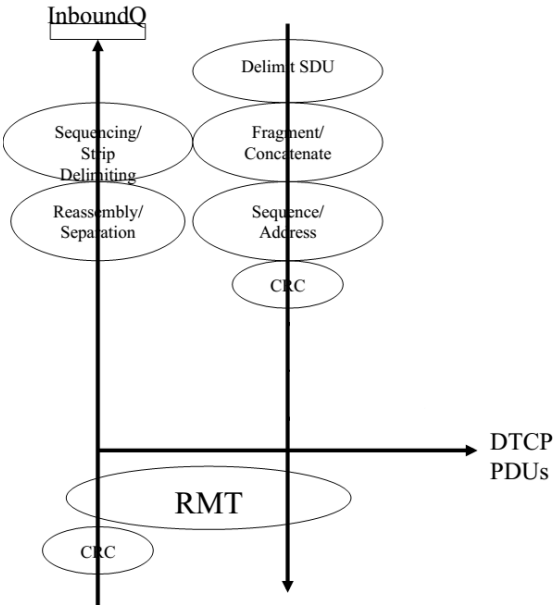

Figure 3.8 DTP processing

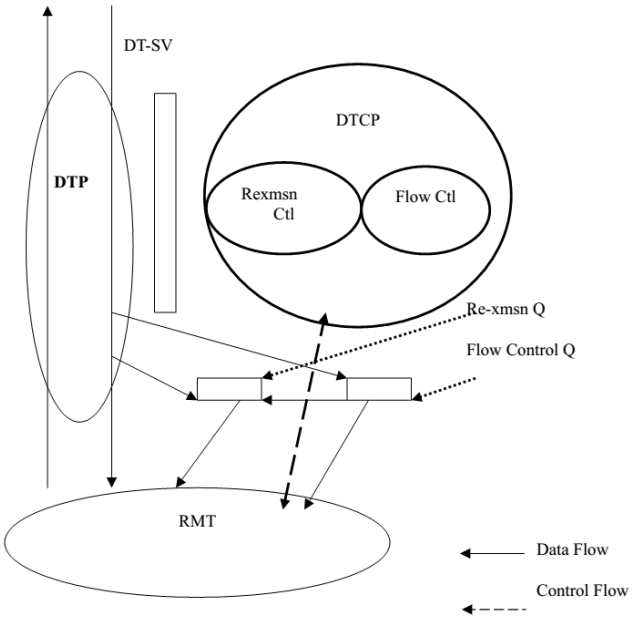

Figure 3.9 DTCP processing

On the other hand (left part of Figure 3.8), the RMT will relay incoming PDUs from (N-1)-port-ids and place them in an InboundQ in order to be served.

The flows which require flow control, transmission control or retransmission control will also have a companion DTCP instance allocated (see Figure 3.9) [45]. The DTCP controls whether the DTP PDUs can be posted to the RMT. It also uses the 3 timers of delta –t protocol, in order to ensure synchronization. Retransmission Control bounds two of them: i) maximum retransmission time and ii) maximum time before acknowledgement. The maximum packet lifetime is bounded by the propagation time on a (N-1)-DIF that doesn't relay, e.g. a wire [47].

The **RMT** is used to forward the messages (PDUs) from one IPC process to another and multiplex the flows (see Figure 3.10) [24]. It is the main place where QoS policies operate. The role of the relaying task, is to forward the PDUs based on the routing information and the QoS agreed. Thus, the relaying task consults the PDU forwarding table in order to choose the (N-1) flow to send a PDU over. The PDU forwarding table contains entries with the following information [45]:

- The QoS to be applied
- A list of port-ids to forward the PDUs to
- The destination address

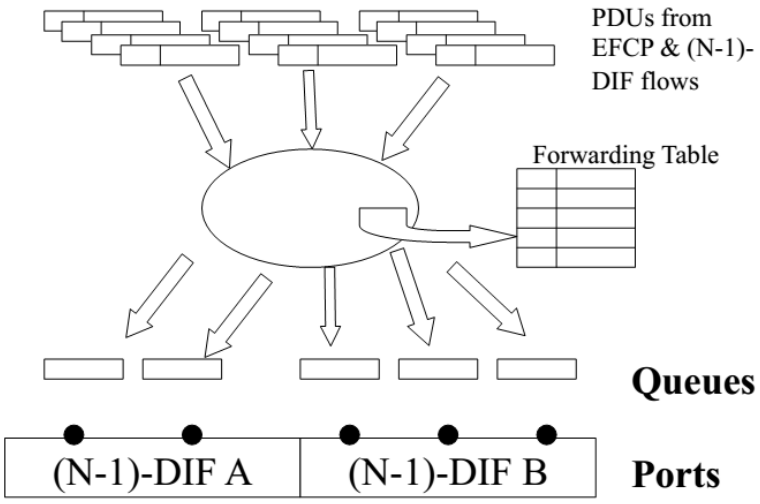

Figure 3.10 RMT [47]

The role of multiplexing task is to multiplex PDUs from different EFCP instances onto the points of attachment of lower ranking (N-1) DIFs. The RMT interacts with the SDU protection module, in order to provide any required protection for the SDUs passed to (N-1) –DIFs [45].

The **SDU protection** is a mechanism used to protect the SDUs from byte errors [24]. Applications may have different levels of trust in the communication mediums they use. Thus, a way to protect the SDUs sent through the flows is required (see Figure 3.11). It is an operation performed by the RMT to ensure that the SDU is not corrupted while in transit. Additionally, the SDU protection provides **confidentiality** and **integrity** and it also includes functions like data corruption protection. What is worth to note is, that this mechanism is **a component of the DIF and available at the bottom of every DIF**, and not a protocol or part of it.

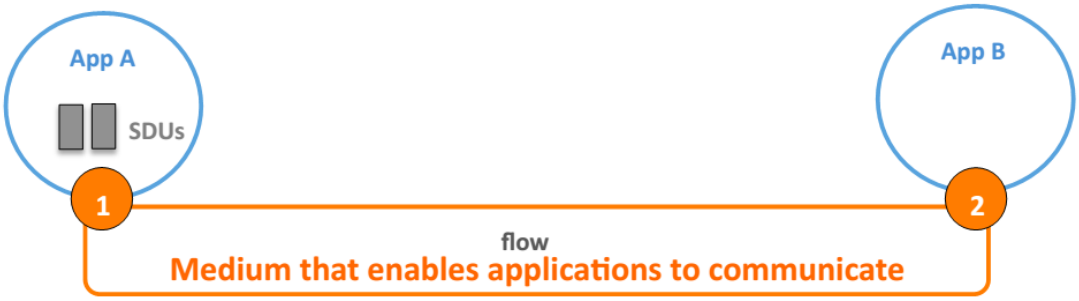

Figure 3.11 SDU protection [47]

The flow allocator is used to allocate and deallocate flow requests and for managing new flows among the IPC processes [24].
When an application process generates an allocate request then the flow allocator creates a flow allocator instance to manage each new flow. The instance has multiple tasks: 1) find the IPC process through which the destination application is accessible, 2) map the requested QoS to policies that will be associated with the flow, 3) create DTP/DTCP instances, 4) handle the flows 5) deallocate resources [45].

The **resource allocato**r is used to manage the resource allocation and to monitor the resources in the DIF [24]. It is the core of management in an IPC process and depends on policies.
Some of the functions of resource allocator are the following: creation and deletion of QoS classes, modifying data transfer policy parameters.

Once application processes have a **communication flow between them**, they have to set up an **application connection** before being able to exchange any further information.
CACE is the common procedure for establishing application connections (see Figure 3.12). The application connection allows the two communicating applications to [32]:

- Exchange application naming information
- Be authenticated by mechanisms such as authentication and access control in order to provide security
- Agree in the application protocol to be used for data exchange i.e. CDAP

CDAP is the second protocol in RINA and it is used by the application processes to exchange shared state (allocation phase) [32].
CDAP is an **object oriented application protocol** and performs operations similar to the CMIP which is also based on objects. According to John Day, after an application connection is established with the CACE, the applications can perform six fundamental operations: create/delete, read/write, and start/stop [7].
From application to application what changes is the objects being manipulated while the protocol remains stable. As a result, one protocol is enough for the applications.

Figure 3.12 Common Distributed Application Protocol

The uses of CDAP in a DIF [37]:

- Establishing a management connection, between the new member which through the enrollment phase wants to join a DIF and the members of that DIF.
- Maintaining the RIB
- Flow allocation

RIB is an object –oriented database which holds the shared state of the communication instances between applications. Each IPC process has a RIB, which stores all the information available to the IPC process such as resource allocation information, application names, mapping of addresses and established flows, represented as **objects [28]**.
It is mainly used to provide transparency i.e. the ability of hiding the complexity of the implementation of mechanisms of a system from both users and application programmers. The IPC processes communicate by exchanging operations (create/delete, read/write, and start/stop) on RIB objects [47]. These operations cause objects to perform appropriate actions. The RIB is populated by the CDAP.

The **RIB Daemon** is used to manage the information of the RIB database. It updates the RIB and ensures that all the necessary information for the task is available whenever is needed [28]. The RIB can be compared with a comprehensive routing table. It decodes the CDAP messages, and extracts the information of what action on what objects of the RIB and with what data is to be invoked.

The **IAP [7]** is a protocol which carries the source and destination application process names. Due to the fact that in RINA communication between processes is done by using names, a protocol to carry these names is required. It is also used to allocate the connection-identifiers which are created by the CEP-id concatenation, so that data transfer between two processes can begin [56].

3.9 Phases of communication in RINA

In RINA, the IPC processes in a DIF only communicate with other IPC processes in the same DIF usually on other nodes. This is similar to the current TCP/IP architecture where TCP protocol machines 'talk' to other TCP protocol machines on different nodes, but not to IP protocol machines. Moreover, as mentioned above, since the communicating elements in a DIF are application processes, they have application names. In RINA a source application process requests communication with another application process from its underlying DIF, **using the name of the destination application [26]**.

As mentioned before, there are 3 fundamental phases of communication. These are the following:
1) enrollment, 2) establishment and 3) data transfer.
However, the enrollment and establishment (allocation) phases are often neglected in the current TCP/IP architecture. In RINA though, these two phases which are part of the IPC management task (see Figure 3.6) play an important role.

Enrollment [32], [56]

In RINA, IPC processes need to be enrolled into the same DIF in order to exchange data [28]. **Enrollment** is the phase by which an IPC process communicates with another IPC process in order to join a DIF [47]. In order for an IPC process to join a DIF and make itself known inside that , the new member must know the name of the (N) - DIF that would like to join or the name of a member of that DIF.
In the Figure 3.13, there is a large (N)-DIF. The DIF A represents a single IPC process and wants to join the (N)-DIF. There is a discovery application mechanism which is called IDD (further described in the next section), through which the IPC process A learns that the first IPC process of the (N)-DIF is the only reachable member and that they **both share an underlying DIF which is the (N-1)-DIF (physical medium) [45]**. The IPC process A, can use the (N-1)-DIF in order to allocate an (N-1) flow to the reachable member. After a flow is allocated, the **CACE component** of CDAP, is used to establish an application connection between the two processes [47].
Once the address of the reachable IPC process is obtained and the application connection is established, **authentication is also performed according to the policies of this particular (N)-DIF**. If successfully authenticated, the new member gets an address i.e. a synonym for the **internal name** in order to get known to the other members of the (N)-DIF [28].

Moreover, the new member acquires information related to the supported QoS, to the policies of the (N)-DIF as well as to the supporting (N-1) DIFs through which can be reached [45].

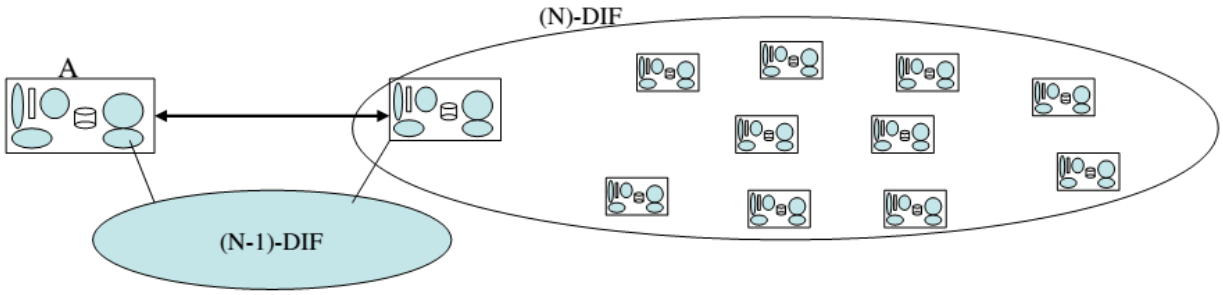

Figure 3.13 Enrollment phase [32]

Establishment [47], [37], [45]

After the enrollment phase, in which the new member A is enrolled to the (N) - DIF, the **establishment phase (or allocation) for allocation of resources** takes place. The resource allocator decides how many N-1 flows dedicated to data transfer will be allocated.

At this point, we suppose that the IPC process A wants to **allocate a flow** with the IPC process B. Both processes are inside the same DIF (see Figure 3.14). At the establishment phase, the two communicating processes share information about their states (shared state).

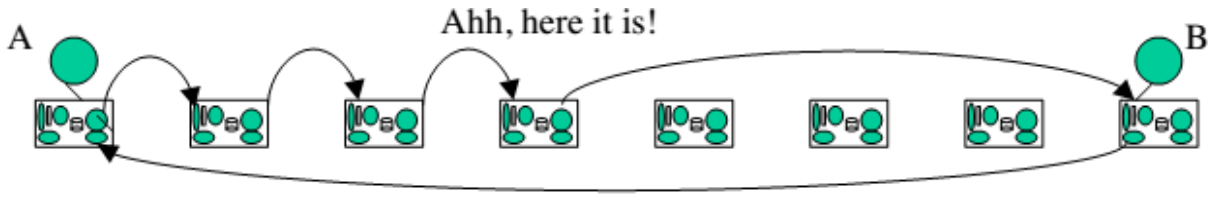

Figure 3.14 Establishment phase

The source application process (A) generates an **allocate request (IAP)** and the Flow allocator creates a **flow allocator instance**, in order to manage each new flow. The source application process (A) has already a port-id which was acquired when it joined the (N)-DIF (enrollment).

- When the **flow allocator instance** gets the **allocate request** of A, it instantiates an EFCP instance at the first point of the flow (A). The port-id is mapped to a CEP-id, which is used as an identifier for the EFCP instance. In general, the CEP-id is used to identify the **shared state of one end of the flow** (see Figure 3.15).
- Afterwards, the **flow allocator instance** checks its local directory cache. This directory maps the requested destination application names to a "Next Place" to look for it. In other words, it is used to find the IPC process through which the destination application (B) is accessible.
- If not found in its directory, it asks the neighbors. As a result, it points to the nearest neighbors until it finds B.
- When B is found, the **flow allocator instance** sends a **create flow request** to the address of the destination application process (B).**The request for the creation of the flow is a CDAP protocol exchange**. Afterwards, the destination sends back a **create flow response**. The source receives the **create flow response** **(CDAP) from the destination**, and if is positive, it instantiates an EFCP instance at the other end (B). The process B gets a port –id which is again mapped to a CEP-id which is used as identifier (see Figure 3.15).

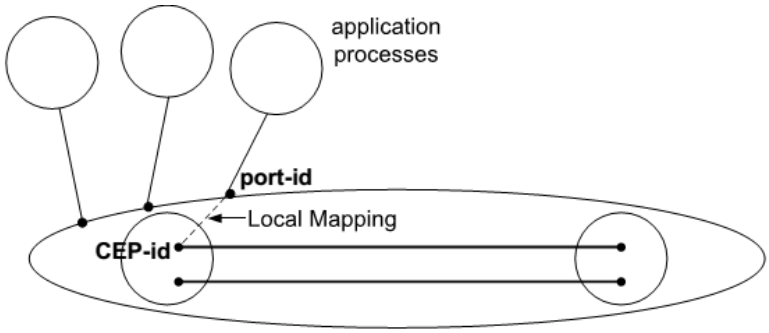

Figure 3.15 Port-id to CEP-id mapping

- The source and destination CEP-ids are concatenated through the IAP for use as a **connection id**, and the **flow allocation between A and B is now complete**. The described procedure has the following advantages: 1) Access control can be done, 2) B can be found even if it moves.
- At this point, either end may start the application protocol exchange. The data transfer phase between A and B can begin by using an underlying DIF!
- A can now start sending application SDUs to its peer. The SDU is delimited and transformed into user-data for a PDU [7].

- The SDU is delivered to the destination EFCP instance which is identified by the CEP-id and there it is processed.
- The resulting PDUs are delivered to the RMT for transmission.
- The RMT has already created a number of (N-1) flows of various QoS characteristics to various destinations.
- RMT accesses the forwarding table to choose the (N-1) flow to send a PDU over.
- The PDU is queued on an outgoing (N-1) flow in order to be sent. This is achieved by consulting the forwarding table.
- After the communication between A and B is complete, deallocation of the resources takes place.

3.10 The DAF and the role of IPC manager [28], [33]

As mentioned before, in RINA the applications communicate by using names.

How the source application knows where the destination application is located?

When an application wants to communicate with another application, there is a need for a component that can find the requested application and determines a DIF which will support the communication. The component responsible for this task is the IDD. The IDD is a distributed application or as called in RINA, a DAF, which is a collection of two or more cooperating application processes in one or more processing systems. The DAF as already mentioned, operates over a DIF in order to manage it and its members are called DAPs.

Every processing system maintains an instance of **IDD** which is an application process and acts as a directory which enables the system to make available its applications as well as to discover other applications. The DAPs **of the same DAF** exchange messages for the discovery of applications. As we mentioned before, the IPC processes must be on the same DIF in order to communicate [29]. Thus, when the destination application is found, a new DIF has to be created (in case there is no common DIF between the two systems) in order for the two applications to communicate.

From the above, we conclude that the IDD is responsible for three functions [29]:

1) Maps the source and destination application names to IPC processes through which they are available.

2) Discovery of applications- The DAPs exchange messages until the destination application process is found. To be specific, an IDD DAP can ask a peer DAP for a particular application by sending an IDD-request. This request contains the destination's IDD DAP name, the source's IDD DAP name and the requested application's name.
   - The peer DAP which will receive the request, it will check the destination IDD DAP name and if it is not itself, then it will forward the request to the appropriate DAP. Otherwise, it will look up its cache for the requested application process name and get the next IDD DAP that the request should be forwarded for the particular application name. When the destination application is found, the IDD will verify that the requested application is still there and the requesting application is allowed to access the requested application.

3) Creation of a supporting DIF- When the destination application is found, a common DIF between the two systems which carry the communicating applications, is required. For the supporting DIF there are two possible options: a) use an existing DIF and expand it to span from the source to the destination or b) create a new DIF from scratch. If the source system is not allowed to access any of the already existing DIFs or if the existing DIFs cannot provide the requested QoS requirements, then a new DIF will be created.

Another question which naturally arises is '**How do the applications know which DIF to use?**'
That is the role of the IPC manager (see Figure 3.16) [34]

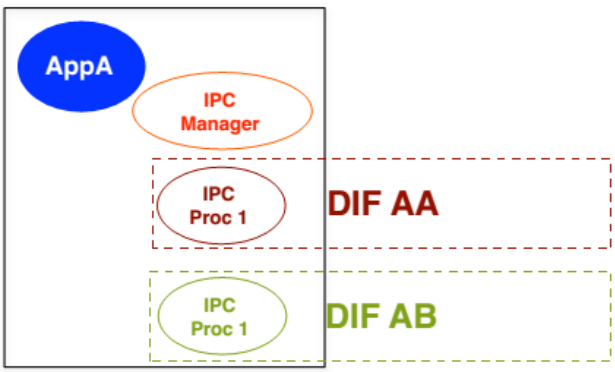

Figure 3.16 IPC Manager

The applications just want to connect to the destination application with a certain QoS, they don't have to care which DIF to join.

The **IPC manager or IPC resource manager** is a mechanism of a DAF which redirects the allocate requests from the applications to the proper DIF. It knows what applications are available on what DIFs. **The IPC manager is the mechanism which implements the IDD**!

3.11 Example of Inter-process communication by using DIFs

This example gives an insight of how communication between two processes can happen in RINA by using DIFs. We consider that the two communicating systems **share a common DIF** (DIFA) and we describe the process of communication of the two application processes. Let's consider two application processes (S and D) in two different systems (see Figure 3.17). The S application process resides in system 1 while the D resides in system 2. These two processes want to communicate.

There is also an intermediate node between these two systems as well as the DIF A, DIF B and the DIF C. The application processes A1,A2 and A3 comprise the DIF A, the processes B1,B2 the DIF B and the processes C1,C2 the DIF C accordingly [24].

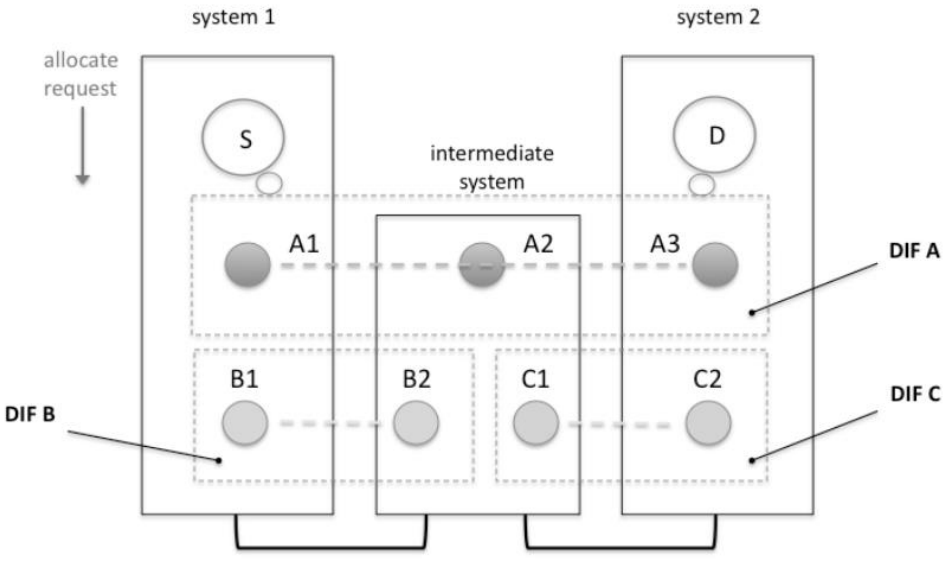

Figure 3.17 IPC between two processes

The procedure in order for these two processes to communicate is the following:

- Firstly, the DIF A maps the names of S and D to the processes A1 and A3 accordingly through the IDD.
- The source application process S makes an **allocate request** using the DIF A and specifies the destination application (D) name.
- The flow allocator of A1 process, receives the request and if it is well formed, valid and there are enough resources to honor it, it accepts it. Afterwards, the port-id of A1 is mapped to a CEP-id in order to create an EFCP instance at A1.
- The flow allocator of A1 looks up the local directory cache for the requested application process (D) and finds out that there is an entry that maps D to IPC process A3. Thus, it sends **a create flow request** to A3. This request is a CDAP protocol exchange.
- The flow allocator of A3 receives the **create flow request** and delivers it to the destination application process (D).
- The D makes an allocate response to A3.
- The flow allocator of A3 creates the necessary EFCP instance and sends a **create flow response** (CDAP) to A1.
- If the response is positive, the two EFCP instances (identified by the CEP-ids) are concatenated and the application processes S and D can start sending/receiving data to/from each other.
- After communication is complete, S and D make a deallocation of the resources.

3.12 Example of Inter-process communication by using DAFs and DIFs

The figure 3.18 represents an example of communication by using DIFs and DAFS. There is an IDD DAF which consists of five DAPs [29]. The DAF is also supported by underlying DIFs (DIF1- DIF6) which provide the IPC services.

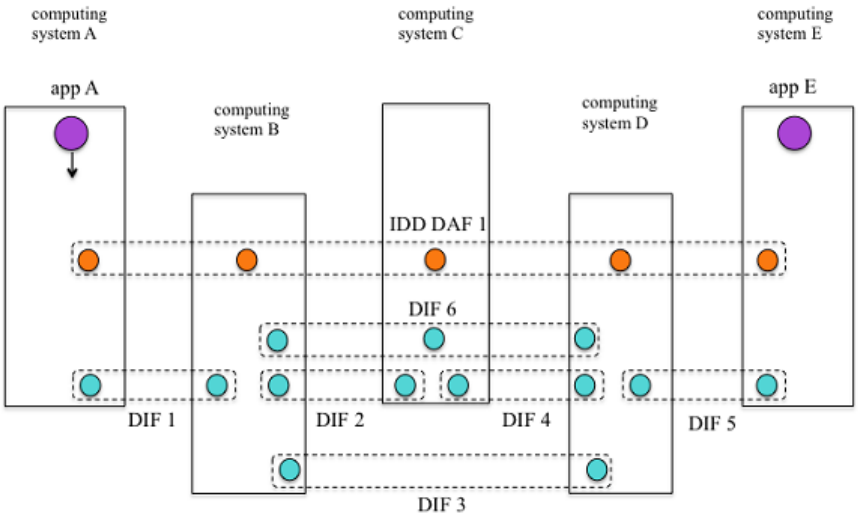

Figure 3.18 – IDD DAF

In order for the two processes (A and E) to communicate the two systems must share a common DIF as in the previous example.

The following steps are executed:

- Firstly, all the DIFs available to the source system (DIF1) are examined to see if the requested application can be reached through them.
- If there is an available DIF then it can be used for the communication. If there is no available DIF, the IDD is responsible to find the requested application and then attempt to create a new DIF between the two systems in order to communicate.
- The members of the DAF1 (DAPs) will start to exchange messages from one to another asking for the requested application by using its **name**. The messaging continues until the IDD DAP of the system (E) in which the requested application resides is found.
- When the destination system is reached, the IDD first verifies that the requested application is still there and that the requesting application is allowed to communicate with it.
- Afterwards, the IDD has to find a DIF that will support the communication between the two application processes.

In the figure 3.18 **there is no common DIF** between the two systems and as a result a new one has to be created by either expanding an existing DIF or creating a new one from scratch.
This is achieved by matching the request's QoS requirements with the QoS classes supported by the already existing DIFs. If none of the supporting DIFs can provide the requested service in the communication, a new DIF has to be created from scratch.

The figure 3.19 shows an example in which a new DIF (DIF5) is created from scratch to support the communication between two processes. To create a new (N) - rank DIF, a higher level application could create an IPC process and connect this process to one or more (N-1) DIFs [26]. This process is given the means to recognize allowable members of the DIF.

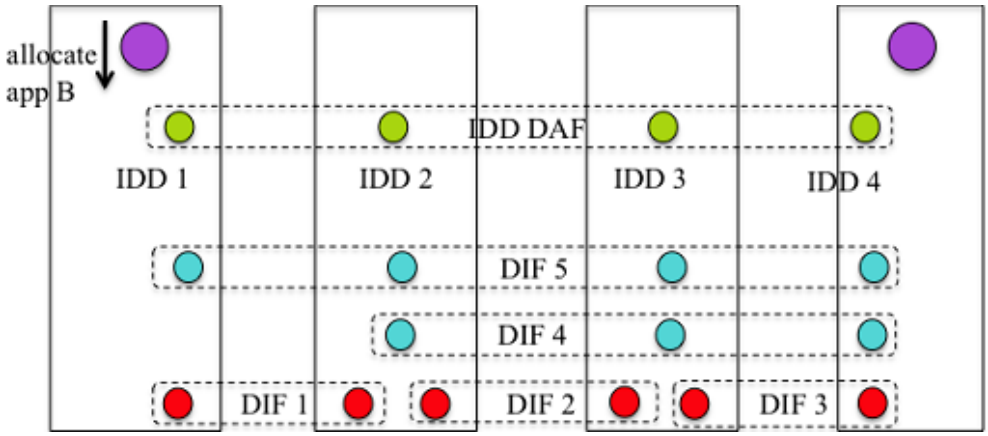

Figure 3.19 Creation of a common DIF [29]

3.13 Features of RINA and how it differs from the current TCP/IP architecture

In the figure 3.20 a general structure of RINA is represented, in which there are two hosts and in between them there is a dedicated system (router). There are also two bottom IPC layers (DIFs) attached to a physical link, as well as a higher level DIF for communication between the IPC processes which uses the services provided by the bottom DIFs [26].

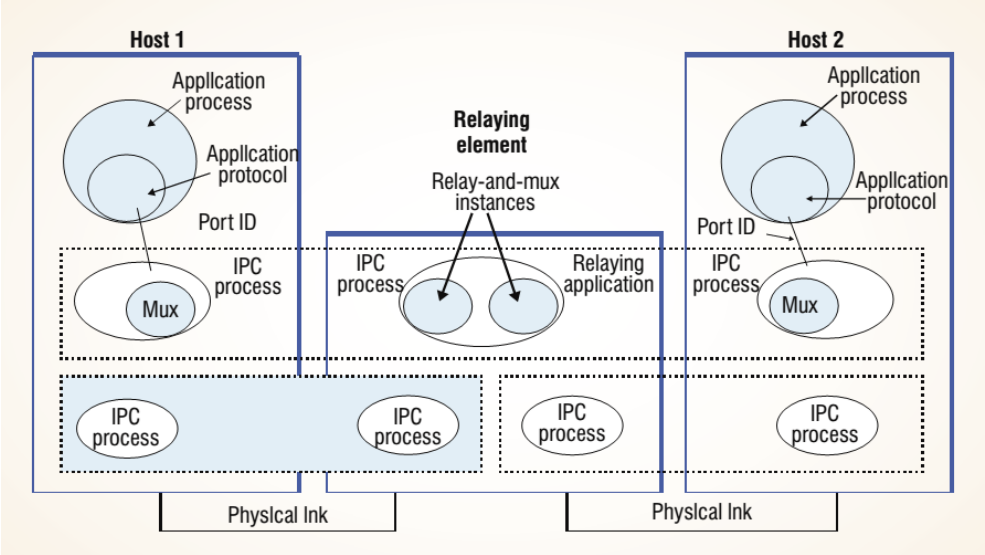

Figure 3.20 Hosts connected via a router

How this model differs from the traditional TCP/IP model?

Routing- The relaying processes in the routers and the multiplexing processes in the end hosts, are members of the same DIF (see Figure 3.20) which integrates both transport and routing tasks [26]. In the TCP/IP architecture there is no complete separation of transport and multiplexing/routing tasks.

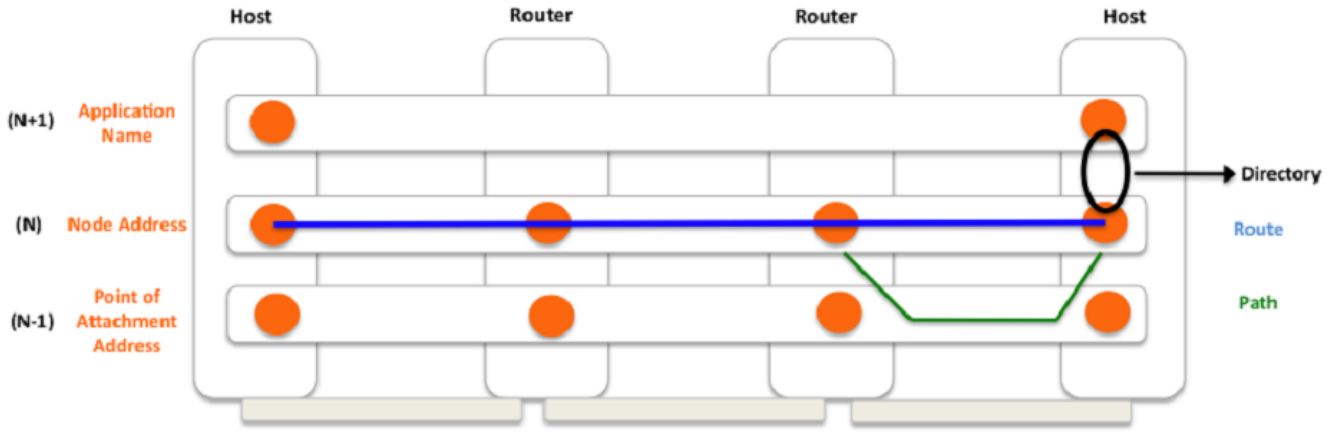

Figure 3.21 Routing in RINA

Furthermore, routing (relaying) is just an IPC process which exchanges connectivity information with other IPC processes of the DIF and it is also used to generate the forwarding table used by the RMT [45]. Unlike TCP/IP, in RINA routing follows the Saltzer's model with some extensions (introduction of a directory).

To be more specific, routing **is done on the node**, and not on the interface which happens in the current architecture. A DIF maps the source and the destination application names to node addresses (IPC processes) through the use of the **directory (IDD) [13]**.

Then, the route from the source to the destination application, is computed as a sequence of (N) – addresses, where the next hop is a node address [56]. Each IPC process knows how to map (N) – addresses to (N-1) addresses to determine the path to the next hop (see Figure 3.21).

In the figure 3.22, a basic structure with RINA configuration is depicted [27]. There is a top level DIF (T-DIF) which covers the network and provides end-to-end service to the hosts at the edges. The T-DIF is supported on the left by a lower DIF (with a scope of the host and the first router). The border router determines the next hop and multiplexes the traffic to the N-1 DIF (L-DIF) in which traffic is routed in the normal way. After crossing the backbone, the traffic is popped up a layer, goes to the T-DIF and through this to the destination. This structure can be repeated according to the network needs.

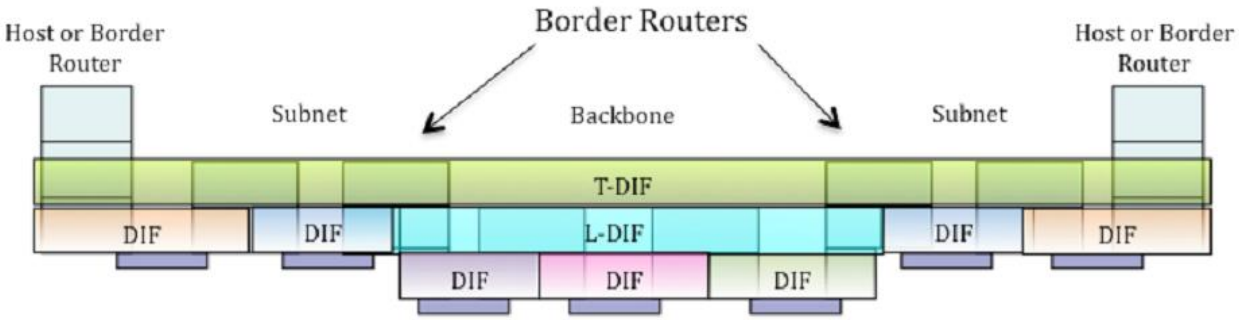

Figure 3.22 Basic structure that a provider network might have in RINA

The advantage of this structure is the following:

- By adopting a hierarchical addressing strategy (which aims to reduce the routing information in large networks) in the T-DIF so that all the nodes in the same subnet have the same prefix, the **number of routes can be reduced** because routing in T-DIF would only have to store routes to the border routers [27]. A border router can determine where to forward the PDUs, so no route calculation is necessary. Routes only need to be computed within the subnet. As a result, the size of the routing tables can be reduced!

Mobility and multihoming- By naming application processes and not interfaces, RINA supports mobility and multihoming without requiring any extra protocols [26]. Routing in a DIF as mentioned above, is done in terms of N-addresses. There are may be more than one paths to the next hop, but by routing on the address, the destination address is the destination **regardless of which interface the PDU arrived on**.

Multihoming is solved by recognizing that routing is a two- step process of picking the next hop and then selecting the path to the next hop. It can be seen as an IPC process having more than one (N-1) mappings.

Mobility can be seen as a dynamic version of multihoming. When the mobile host moves, it requires new PoAs in order to connect to the network. In RINA, each new PoA means that the mobile host joins a new DIF. Thus, while the host is moving, it joins new DIFs and drops its participation from the old ones [56].

Example: While the host M moves within the scope of the right (N-1) –DIF, a local routing update of **A** process is required since the (N-2) address changes (see Figure 3.23). By moving further to the left the host moves to a different (N-1) –DIF and this results in a routing update at the higher (N) – process B since the M's (N-1) - address changes.

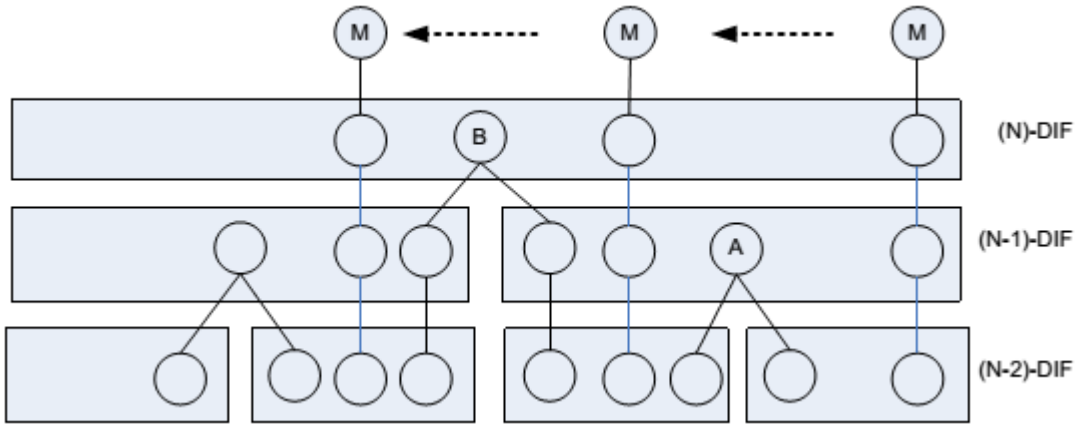

Figure 3.23 Example of mobility in RINA

The routing model of RINA provides another advantage regarding mobility. To be specific, if an active interface (path) to a node fails, RINA maps the node address to another operational interface (see Figure 3.24).

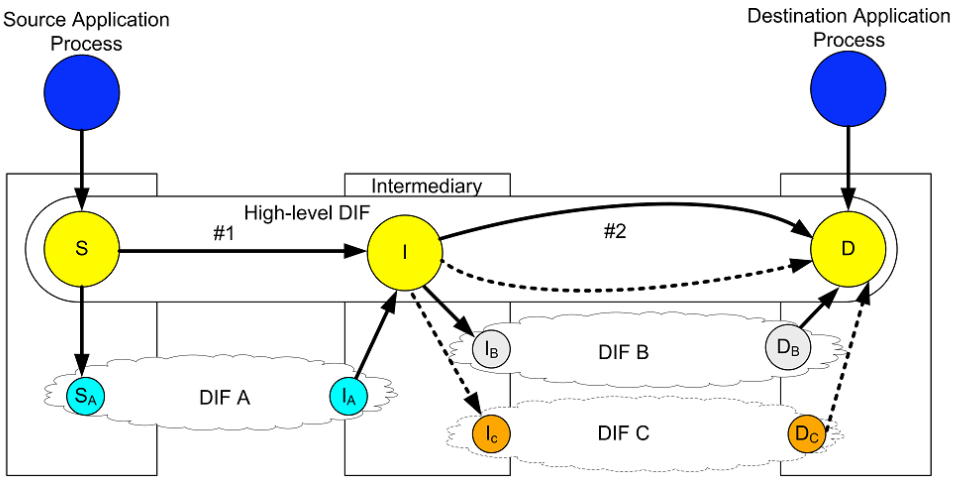

Figure 3.24 Routing example with interfaces

In the figure, a source S wants to send a message to D through an intermediate router which is connected to the destination through two different interfaces. If the link which uses the underlying DIF B goes down, the node **I** would locally adapt and start using the dotted link which uses the DIF C [31].

Naming and addressing scheme- Unlike the current Internet which doesn't really name nodes/applications and everything is accessed using "absolute addresses", in RINA all entities in the network (nodes, applications, PoAs) are named.
The communicating elements which are application processes, have **application names [26]**. These names are external, meaning that for a source application to communicate with a destination application the name will be used. The addresses on the other hand which are just synonyms, are **internal** identifiers used by the members of the DIF. They can be seen only by the members of the DIF and they are not visible outside it. Finally unlike the current Internet architecture, in RINA no global address space is needed but instead a global name space to name the applications [56].

A single error/flow control protocol and a single application protocol – The result of separating mechanisms from policies, is to have only one protocol for data transfer which is the EFCP and one for communication between the IPC processes which is the CDAP. This makes the implementation simple and there is also no need to develop protocols for different functions.

Support of QoS requirements- RINA provides multiple classes of QoS independent of the applications [27], [24]. Each DIF can support a set of QoS characteristics and deliver the requested service reliably without the need of extra mechanisms. The specific mechanisms such as error control and flow control which are used to provide the QoS, are an implementation detail, hidden inside the "black box". Furthermore, if the DIF determines that it can't deliver the requested QoS, it rejects the request.

Scalability- With the repeating IPC structure and the private addressing of DIFs, RINA attempts to have much better address scalability [26]. Requirements for router computation and storage capacities are structurally reduced due to hierarchical addressing. This strategy also leads to **greater robustness and more effective response to change,** due to the smaller routing table sizes.

Adoption- Another benefit of RINA is that no migration to the new architecture is required. RINA can be used over and under the layers of the current stack, as well as along with the current architecture. In other words, RINA can be used over IP, under IP (much as MPLS is deployed today) or alongside IP [47]. A Shim DIF can be used in order to make the "connection" between RINA and the current TCP/IP stack.

Shim DIF [47]: It can be placed above a non-RINA transport (wire, Internet etc) and provides the RINA API to allow an application to treat the transport as a RINA DIF. The researchers of RINA have already created Shim DIFs over IP (see Figure 3.25).

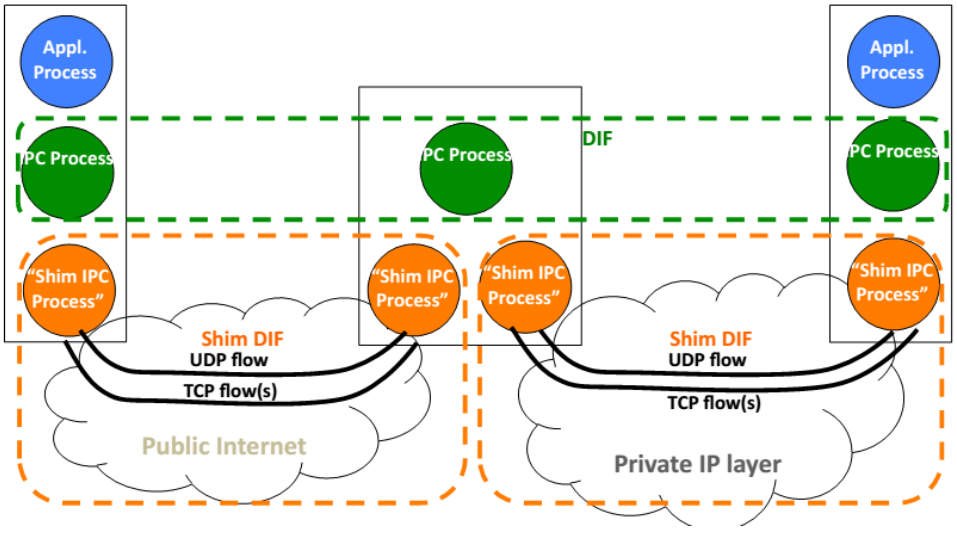

Figure 3.25 Shim DIF-over IP

The Shim DIF over IP layers presents an IP layer as if it's a regular DIF. The functions of the Shim DIF are the following:

- It wraps the IP layer with the DIF interface
- Maps the names of IPC processes of the layer above, to IP addresses in the IP layer
- Creates TCP and/or UDP flows bases on the QoS requested

Heterogeneous environments- Unlike the current Internet architecture, in which congestion control is in TCP and this according to the proponents of RINA results to problems in heterogeneous environments, in the model of RINA each DIF can perform congestion control confining the effects of congestion to that layer [27]. Furthermore, the structure of RINA seems to be suitable for requirements of transport over heterogeneous paths, because there is no need for mechanisms that work well for all types of physical media [13]. Each physical medium has its own dedicated DIF with specific characteristics. An example of RINA in heterogeneous networks is depicted in Figure 3.26.

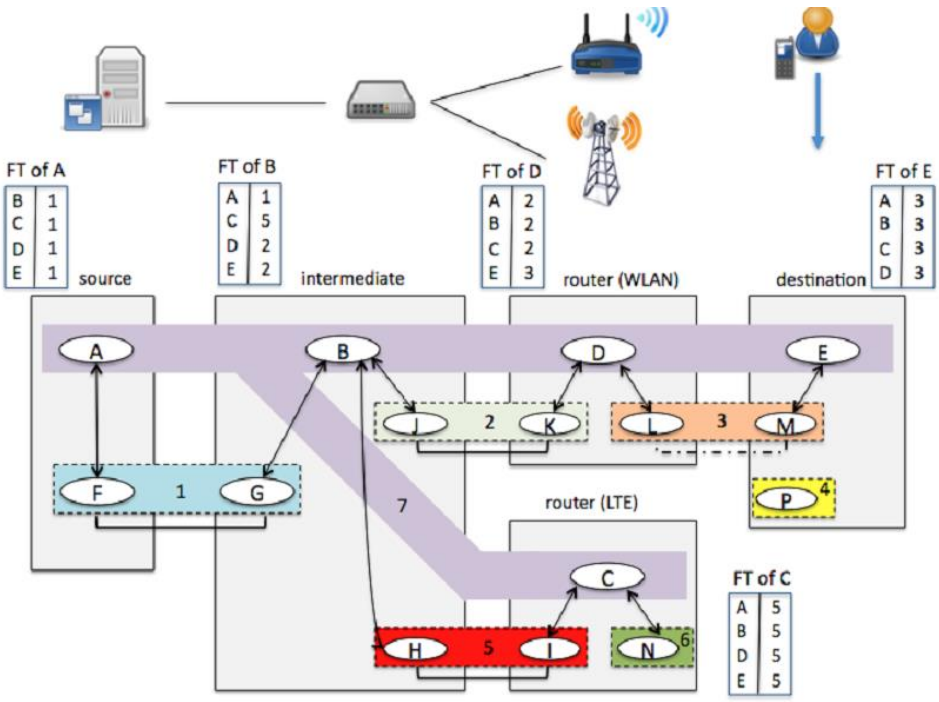

Figure 3.26 Example with mobile terminal

A user is moving around with a mobile phone causing hand-offs (Figure 3.26). The mobile phone has two interfaces (Wi-Fi and GPRS). The two access points are the WLAN router (Wi-Fi) and the LTE router (for GPRS). Both access points are connected to the same intermediate router that the server (source) is connected to.

An application process running on the phone (destination) has opened a connection with an application process running on a server and as a result data transfer begins. While the connection is open, the user moves, forcing a hand-off from one access point to the other.

Let's consider the case in which the user is in the range only of the Wi-Fi (see Figure 3.26). The server is represented with the source system and the mobile phone with the destination system. The intermediate system is a router which connects the server with the two access points. There are 7 DIFs which include IPC processes (A-P). The DIFs 1-6 are low level DIFs and the DIF 7 is the top level DIF (higher rank). Every IPC process of the top level DIF, has a forwarding table which maps the IPC processes to the low level DIFs in order to know where to forward the PDUs.

All the low level DIFs are above physical links and these **DIFs have different characteristics according to each physical medium**, providing the top level DIF with properties such as delay, loss etc. The dotted line represents the wireless link and the continuous line the wired links. DIF 4 represents a second interface in the mobile phone but without a connection to an access point. For data transfer, the route from A to E is computed (through the WLAN interface).

For a PDU to go from A to E the route is the following (according to the tables): A will forward it to DIF-1, B will forward it to DIF-2, and finally the PDU will go to DIF-3.

Now let's consider the case in which the user enters in the range of the second access point and uses the LTE interface (Figure 3.27). This is the case of **multihoming** in which the node has two interfaces connected to two different access points (Wi-Fi and LTE). A new DIF -4 is formed between the mobile phone and the LTE. The forwarding tables also change because now a different route can be used to reach the destination.

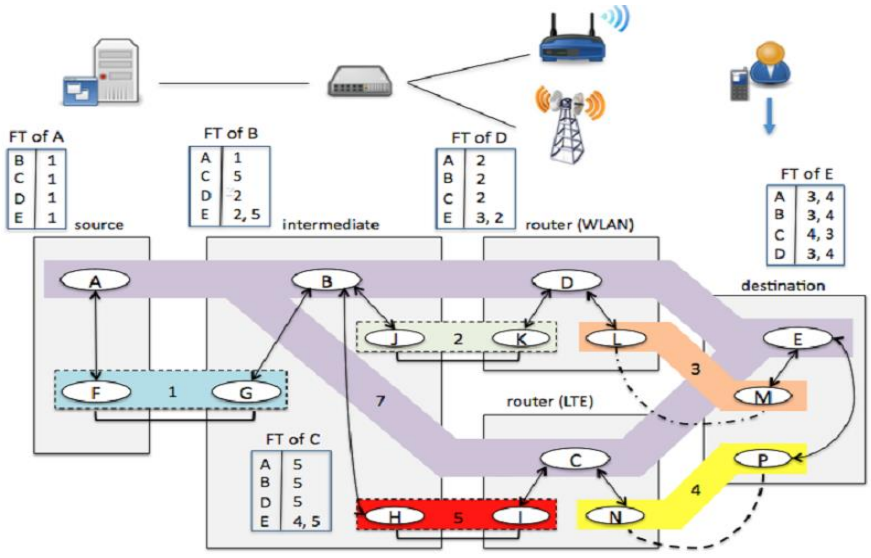

Figure 3.27 User is connected also to the second interface (LTE)

In the final case, the user moves out of the range of the first access point while is connected to the second one (see Figure 3.28). This means that the bind between the processes E and M drops. This change will cause the routing tables to be updated.

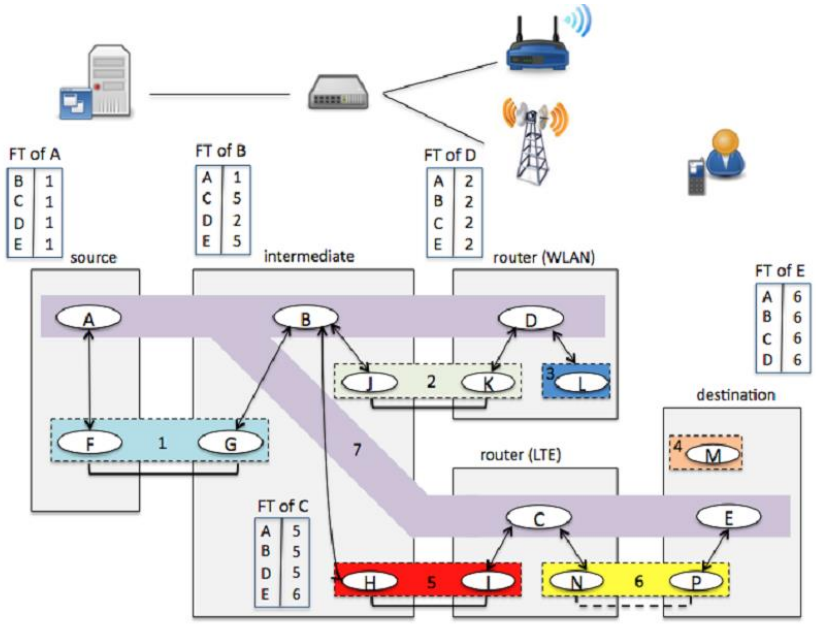

Figure 3.28 User is connected only to the LTE

The above example shows the flexibility of RINA with dynamic DIF instantiation and adaptation, in mobile situations. In heterogeneous environments, when the mobile host moves, it joins new DIFs and drops its participation from the old ones. Furthermore, the flexible routing model of RINA, allows a fast switching between interfaces.

3.14 How security is handled in RINA

According to the proponents of RINA, the recursive architecture is more secure and more resistant to different kind of attacks (data transfer attacks, port-scanning attacks etc) which happen in the current TCP/IP architecture [11]. The current Internet has a global address space, which allows any system to connect to any other system.

However, in RINA the addresses are internal to a DIF and hidden from the applications. The DIF is a securable container which provides encryption.

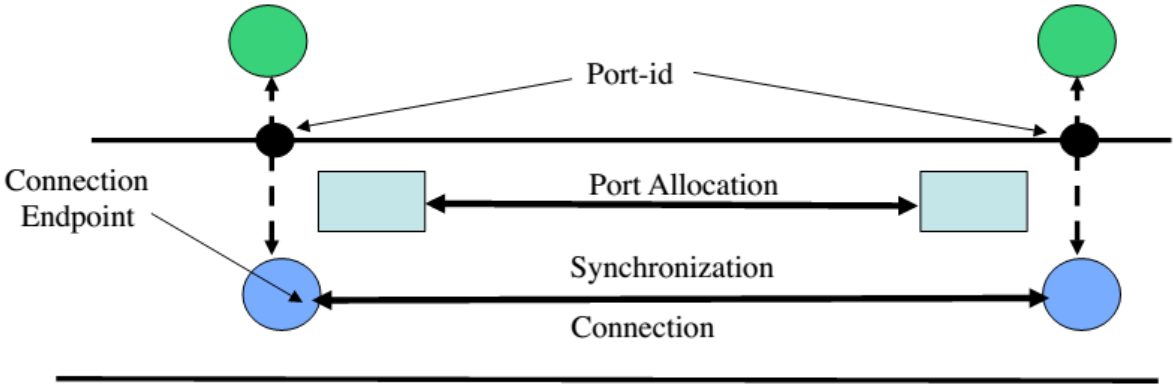

Figure 3.29 Port allocation decoupled from synchronization

In RINA **port allocation is not coupled with synchronization** (see Figure 3.29) **and as a result, applications don't listen to a well-known port** [11]. Instead, an application process requests service using the destination application-name and in order to join a DIF (enrollment) it must first be authenticated. In addition, due to the fact that RINA uses a soft state data transfer protocol (EFCP) which is based on delta –t, a flow state is deleted after 2xMPL at the receiver and 3xMPL at the sender [56]. On the contrary, TCP relies on handshaking messages which are vulnerable to attacks.

Thus, unlike to the current architecture, RINA is less impervious to attacks from outside the DIF. Moreover, different authentication policies within each DIF, can provide a range of security levels.

Port –scanning attacks and how they can be tackled in RINA [11]

In a port scanning attack, the attacker explores the 'open' ports to which processes are listening. In RINA, a service is accessed by the application name and the requesting applications can't see the addresses. Thus, there are no well-known ports. As mentioned before, there are **local** port-ids for each communicating process, which are mapped to CEP-ids which are used for data transfer. In RINA, however an attacker might try to scan application names, but this is difficult because the names are taken from a namespace and they have variable length. Also, the attacker has to be a member of the DIF and this requires authentication during enrollment.

Difference in the number of protocols between TCP/IP and RINA [39]

This section describes how security in RINA can be simpler in respect to the number of protocols which are used.

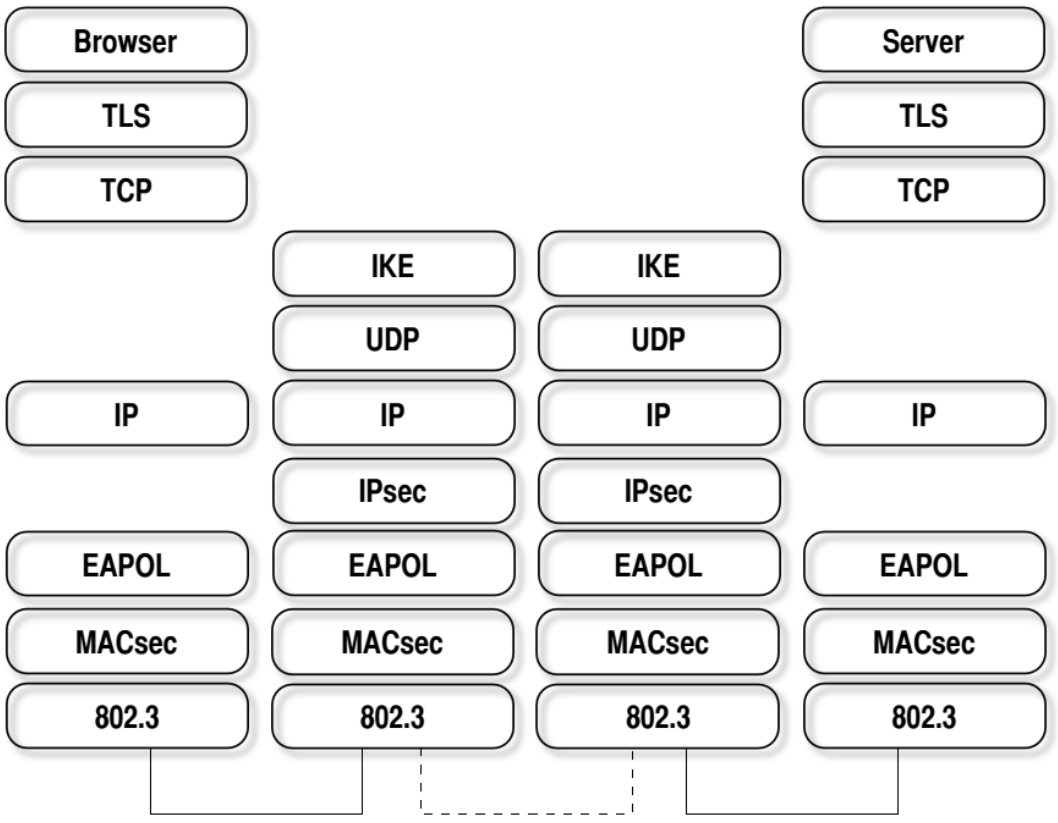

Figure 3.30 Current Internet Security stack

The Figure 3.30, shows that there are currently numerous mechanisms and protocols for security. In the Figure, there is a server and a browser which want to communicate with each other through an application protocol (HTTP). To add security to an "Internet" network, entire protocols are added to the stack. These protocols are separated, operate independently and they are stacked as needed in order to provide the requested network service.

In RINA however, there are less protocols and mechanisms due to the separation of policies from mechanisms. Thus, there are no additional redundant security mechanisms.

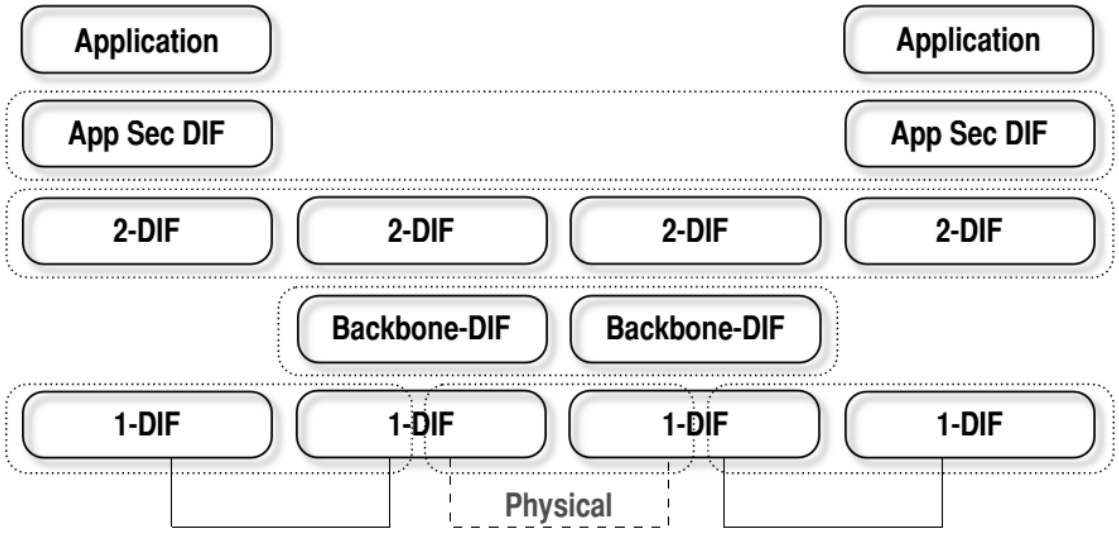

Figure 3.31 Security in RINA

Let's assume that there is an HTTP running over a network (for comparison reasons) and the two communicating applications are a server and a browser (see Figure 3.31).

There are 6 physical links. The (1)-DIFs are used to manage each of these links. Above them, there is another DIF which includes two Backbone-DIFs which actually represent the border routers.

Above, there is a single "network' denoted as (2)-DIF. The APP Sec DIF is used for transport security between the two applications and includes two members (IPC processes) for the source and destination applications respectively.

Above the App Sec DIF the two communicating applications are located. In RINA, the communicating applications use the same security mechanisms as the processes running on the underlying DIF. In this example we assume that the network will provide security in the data transfer, between the server and the browser.

Since every DIF uses only two protocols (CDAP, EFCP), this means that together with the HTTP, in total **only 3 protocols** are required. Thus, unlike TCP/IP, in RINA far less protocols are required!

3.15 The ongoing projects of RINA

There are currently four projects which focus on RINA.

IRATI [51]: IRATI's goal is to explore RINA and to develop a model which can be deployed in production scenarios such as data centers, campus networks and VPNs. This model uses the OFELIA testbed (OpenFlow) for experimental activities such as routing [61]. The OFELIA facility provides a programmable Ethernet network and offers a set of islands around Europe composed of OpenFlow switches and virtualized servers [55]. TUB is one of these OFELIA islands which is located in Berlin. All the resources are controlled by the OFELIA Control Framework which is a set of software tools for testbed management.

OFELIA offers a geographical diversity and multi-domain environment. As a result, experimentation with different network topologies can be performed, as well as dynamic modification of the network topology in order to test different network conditions (e.g. link failures). The behavior of the RINA stack can be tested among these complex scenarios.

IRATI is currently working on open source prototypes over Ethernet for UNIX-like OS, as well as over UDP/IP. These prototypes will enable experimentation and evaluation of RINA compared to TCP/IP.

PRISTINE [53]: This project aims at the implementation of a software development Kit for RINA.PRISTINE intends to develop and implement the internals of RINA that include the programmable functions for:

- Security of application processes
- Providing protection and resilience
- Supporting QoS

PRISTINE attempts to demonstrate the applicability by using 3 use cases: 1) datacenter, 2) distributed cloud and 3) carrier network.

To be more specific, PRISTINE will investigate and trial the use of RINA- based solutions in a datacenter environment, by addressing issues related to mobility of virtual machines, in order to have an efficient utilization of resources as well as multihoming support and high reliability [62]. For example, with specific routing and load balancing policies the amount of the datacenter components can be reduced, leading to an improvement in the deployment cost.

Furthermore, with the routing model of RINA which supports mobility, it becomes easier to relocate running VMs to different physical machines, in order to have high availability in a datacenter.

IRINA [52]: IRINA proposes to study RINA as the foundation of the next generation NREN and GEANT architectures. The strategy of IRINA is to take the leadership in research networking back from the vendors [63]. The objectives are the following:

- Perform a use-case study of how RINA could be better used in the NREN scenario with the benefits that RINA brings in terms of mobility, QoS, multihoming etc.
- Involve the NREN and GEANT community in the project, in order to discuss and get valuable feedback.

ProtoRINA [54]: ProtoRINA is a prototype which has been developed from the Boston University and tested on its campus and on the GENI testbed. GENI provides a virtual laboratory for networking and research. In other words, it provides an experiment infrastructure with resources and allows the researchers to program experimental networks.
ProtoRINA is not restricted to the IP and enables experimentation with new control and management applications. ProtoRINA has been used to demonstrate the RINA architectures and its advantages as well as to experiment with different policies. Different experiments like enrollment into a DIF or how RINA supports provision of a virtual private cloud service were performed.

3.16 Summary of the chapter

This chapter focuses on the architectural model of RINA. It is also presented how RINA differs from the current TCP/IP architecture and how it attempts to solve some of current Internet problems. Finally, the ongoing projects of RINA are presented. These projects prepare the tools and the platform which the network and application developers can use, in order to advance the goal of RINA for a better Internet.

Chapter 4

RINA and Software Defined Networking

This chapter focuses on SDN which is currently under a lot of discussion and it is closely related to RINA. I included this chapter on my thesis as I was doing my research on RINA and was searching for similar architectures. I was proved right regarding the relation of these two architectures, when IRTF published an RFC regarding SDN. The architecture of SDN and its popular mechanism which is the Openflow, are the focused points of this chapter. Finally, this chapter presents the application fields of SDN, as well as how SDN can embrace the recursive model of RINA.

4.1 Introduction to SDN

In the traditional networks there are many individual network components and the management becomes difficult and complicated. In order to express different network policies, the network administrators need to check and configure each network component separately. They also use configuration interfaces which vary across vendors and as a result this procedure is complex and error prone. Furthermore, there are many sets of protocols for solving each problem. In addition, the current networks are vertically integrated which means that the control and the data plane are bundled inside the networking devices reducing flexibility and innovation [18].

The idea for programmable networks and separation of the control, has been around for many years. For example Active Networking [41],[59] which started in the 90s, was an approach to network control, by using a programming interface which exposed resources (storage, packet queues) on network nodes. NETCONF [41], [60] was another approach to network programmability, through which the configuration of network devices could be performed simultaneously.

These approaches contributed in many ways (e.g. programmable functions, network virtualization) to SDN [41].

SDN is an emerging paradigm which focuses on programmable networks and attempts to simplify network management as well as to enable innovation and evolution. The term has its origins at Stanford University. It refers to the ability of software applications to program network devices and therefore to control the behavior of the whole network. The main principle of SDN is the **separation of control plane from the data plane**. In SDN, the control plane decides how to handle the traffic while the data plane forwards the traffic according to decisions that the control plane makes.

The network is considered programmable because there is a network management layer which focuses on programming the control plane [18].
An interface is provided by the network management layer, which allows the network managers to manage the network easily without being concerned about low- level network details.
The result of the data/control plane separation is, that the switches become simple forwarding devices and the control logic is implemented in a **logically centralized controller**. As a consequence, there is flexibility and programmability in networks. Moreover, it is considered simpler and less error-prone to modify a network through high level languages [42].

Characteristics of the SDN architecture [42]

1) The control plane is decoupled from the data plane. The control functionality is removed from the devices of the network in order to have simplicity and flexibility.

2) The forwarding decisions are not destination-based but instead they are **flow-based**. A flow is represented as a set of packets which include a match criterion and a set of actions.

3) Control logic is moved to the SDN controller which is used to control the whole network.

4) The network is programmable through software applications running on top of the SDN controller.

4.2 Overview of the SDN architecture

The Figure 4.1 shows the general architecture of SDN [18]. At the bottom of the Figure, there are network devices which include switches and/or routers. Each device includes a process which hides its internal details. The **Network Device Interface** (Southbound API) provides a way to access these processes of the devices and in general to control the forwarding devices. This link must remain available and secure. An example of this interface is the OpenFlow protocol.

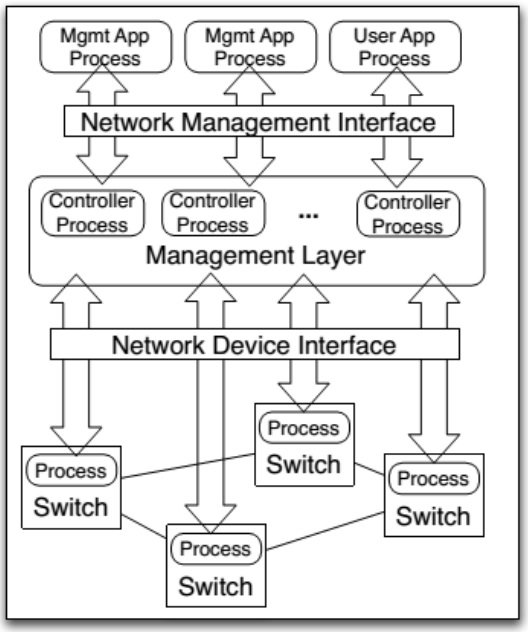

Figure 4.1 Software defined network

The **Management Layer** consists of controller processes which may run on one or more physical servers. These processes provide the required control functionalities for the network.

The **Network Management Interface** (Northbound API) provides a way to manage the network through the management application processes. To introduce a new functionality in the network, is achieved by just adding a new software application to run on top of the management layer. There is currently no common northbound interface, thus this remains an open issue.

Eastbound and Westbound APIs [42]

In SDN there can be more than one controllers. Each controller implements its own east/westbound API. The role of these interfaces is to import and export data between the controllers as well as to perform other functions such as monitoring. All in all, they are important in order to provide a common compatibility **between different controllers** (see Figure 4.2).

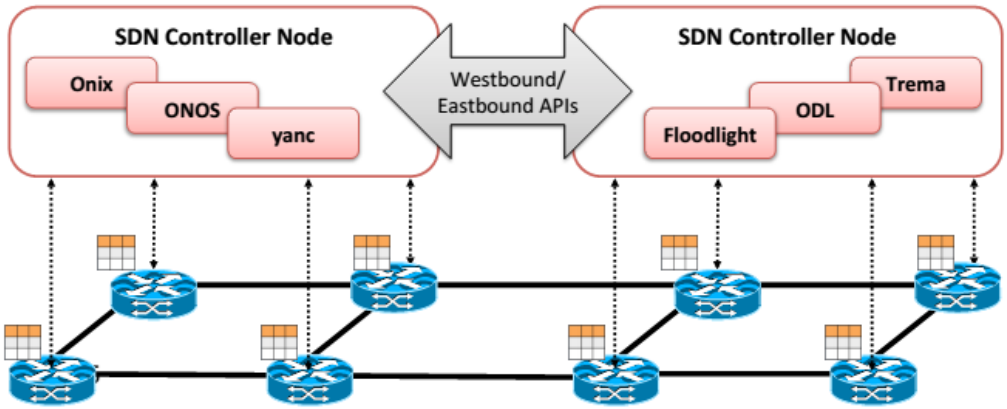

Figure 4.2 Westbound/Eastbound APIs

4.3 Current SDN architectures

In SDN, the network device interface (Southbound API) can be supported by any mechanism which provides communication between the control and the data plane. **Openflow** is such a mechanism which is currently very popular in SDN [43]. It follows the basic SDN principle, of separation between the control and data plane and it is used for information exchange between these two planes. The advantage of Openflow is, that it allows the network administrators to configure devices of a network which are **from different vendors,** in a uniform way. Furthermore, the vendors don't need to expose the internal details of their devices and as a consequence Openflow is supported by major vendors which are already developing Openflow switches.

There are also other SDN architectures, like **I2RS** which provides an interface to the routing system for real-time interaction through a collection of management interfaces. **ForCES** is another architecture which is proposed by the IETF group and has the control element separated from the forwarding element but within a close proximity (e.g. same box) following the current TCP/IP architecture for the network [14]. Among these SDN architectures though, Openflow is currently the most popular.

4.3.1 Openflow

Most of the modern Ethernet switches and routers contain flow –tables to implement firewalls, NAT, QoS etc. Each vendor has its own flow table. However, the researchers of OpenFlow identified that there is a common set of functions inside the switches/routers [43]. Thus, OpenFlow exploits this set of common functions.

It follows the principle of SDN by decoupling control from the data plane. Openflow provides an open protocol in order to program the flow table in different switches and routers. Furthermore, researchers can control their flows by choosing the routes their packets will follow! Thus, there is flexibility.

The Figure 4.3 depicts the Openflow protocol. There is an SDN device which communicates with a SDN controller with the help of the OpenFlow protocol. The switch contains flow tables which consist of flow entries (rules) in order to determine how the packets will be processed and forwarded. Depending on the entries, the OpenFlow switch can behave like a router, switch, firewall or something in between.

The entries contain: [43], [42]

- Match rules, in order to match the incoming packets of the flow by looking at the header
- Statistics, to determine for example the number of received packets of a flow
- Set of actions, to determine how the matched packets are going to be handled

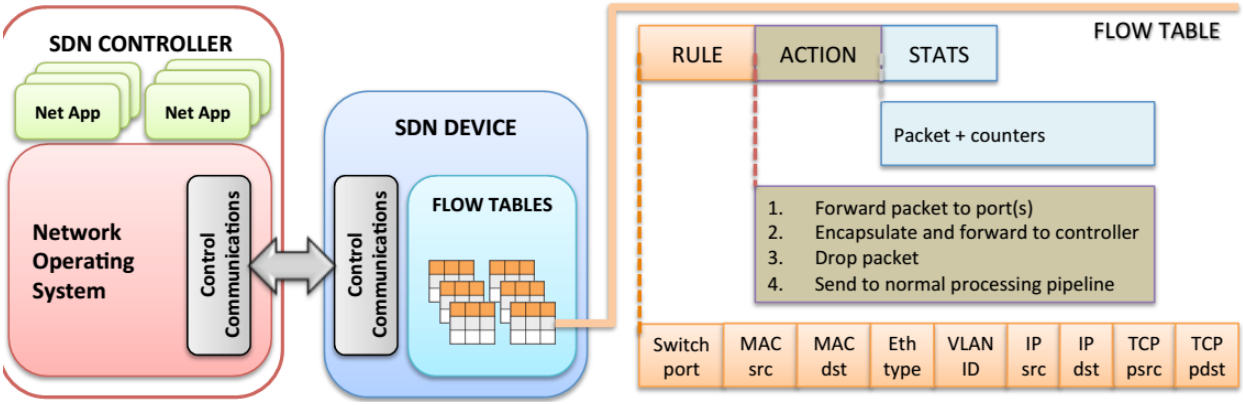

Figure 4.3 Openflow Device

Procedure [43]: When a packet arrives at the Openflow switch, its header is extracted and matched with the match rules. If a matching entry is found, the switch applies the appropriate set of actions which is related to the matched flow entry.

These actions can be the following:

- Forward the flow's packets to a given port(s).
- Encapsulate and forward the flow's packets to a controller.
- Drop the flow's packets.

If there is no matching entry, the action will depend on the instructions defined by the table-miss flow entry (default entry), which every flow table must contain.

4.4 Forwarding devices, controllers, management applications and the role of abstractions in SDN

This section provides a more detailed description regarding the components of a software defined network.

Forwarding devices

The forwarding devices or switches in SDN terminology [46], comprise the physical network equipment and they can be routers, switches, access points etc. These devices don't have embedded control to take autonomous decisions.

Apart from the physical, there are also softwarized devices, when for example there are virtual machines communicating.

The control plane needs **a flexible forwarding model**. To achieve this, **a forwarding abstraction** is needed in order to hide the details of the underlying hardware [42].

In an OpenFlow network, apart from the pure Openflow switches which forward packets between ports and completely rely on a controller, there are also hybrid switches (used today) which have both Openflow and non-Openflow ports in order to support the traditional operations and protocols.

Controller

A controller is a software stack running on a hardware platform and it's a critical element in an SDN architecture because it is the key in order to generate the network configuration based on the policies defined by the network operator. The control is logically centralized and provides a **specification abstraction** so that the whole network can be seen as a single system [42].

This abstraction can be seen as 'network virtualization' which provides a "virtual image" of the network. Underneath this "virtual image" there is the physical network. This gives us a simplified model (see Figure 4.4) and enables the SDN architecture to be applied over a wide range of applications and heterogeneous environments. In addition, the network operators can programmatically configure this simplified network abstraction, rather than having to configure each device separately.

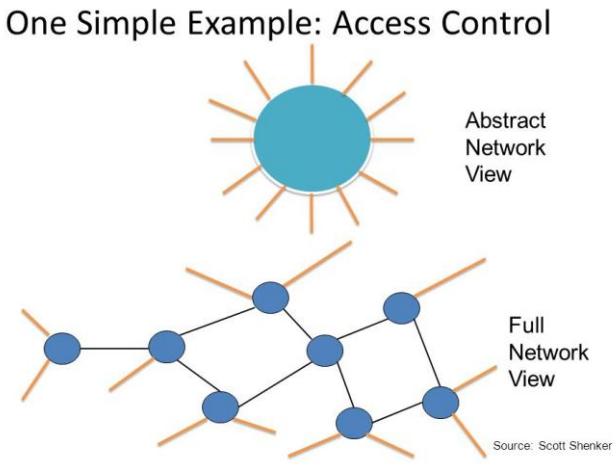

Figure 4.4 Network abstraction [44]

The Figure 4.4 depicts a case of a network in which we want to do access control. Because the behavior we want to specify is that **A** can or cannot talk to **B**, we don't care about all these switches. Thus, we can specify an abstract view of the network.

From the above we conclude that a software defined network can be defined by **three fundamental abstractions [42]**:

1) Forwarding- to allow any forwarding behavior while hiding the details of the hardware.
2) Specification- to allow a network application to express the desired behavior of the network without being responsible to implement that behavior itself.

This can be achieved by having an abstract view of the whole network as one virtual 'big switch'. This also simplifies the task of application developers as they don't need to think about a sequence of switches but rather see the network as one switch.

3) Distribution- to have a logically centralized controller.

Centralized vs Distributed controller [42]: In SDN architecture, the networks may have either a centralized or a distributed control plane or a combination of them (hybrid).

A **centralized** controller represents a single point that manages all the forwarding devices. However, it also represents a single point of failure for the entire network. Thus, OpenFlow allows the connection of multiple controllers to a switch.

A **distributed** controller can be a centralized cluster of nodes or a physically distributed set of elements. The first case can offer high throughput for dense data centers while the second one can be more resilient to logical or physical failures.

A **logically centralized but physically distributed** (hybrid approach) control plane, decreases the look-up overhead by having a simplified central view of the network (specification abstraction).

Reactive vs Proactive Policies [42]: In a reactive model, the forwarding elements must consult the controller each time a decision must be made when a packet reaches a switch. In a proactive model however, the controller pushes policy rules to the switches and rarely needs to be consulted.

Management applications

These applications implement the control-logic that will be translated into commands to be installed in the data plane for controlling the behavior of the forwarding devices [42]. For example in order to perform routing from A to B, a routing application has to instruct the controller to install the appropriate forwarding rules in the devices which are in the path from A to B.

There are different management applications which can perform routing, load balancing, security policy enforcement etc.

4.5 How SDN is related to network virtualization

SDN and network virtualization relate in three main ways [14]:

- Virtualization for testing SDNs – By decoupling the control from the underlying data plane, it is possible to test SDN control applications in virtual environments before the application is deployed on an operational network.
- 'Slicing' an SDN - It is easy to virtualize (slice) an SDN switch by dividing the traffic flow space into 'slices'. Each slice is managed by a different SDN controller and as a result, third –party service providers can deploy services on the network without having to install their own infrastructure.
- Abstraction- As mentioned before, with virtualization, an abstraction of the network is given. This increases flexibility and the network administrators can configure the network easier.

4.6 SDN application cases

SDN has application cases in multiple environments like in **enterprise networks** [41] where there are often large networks which have strict security and performance requirements. SDN can be used to adjust the network policies in order to increase performance and monitor the network as well as to provide a simplified network by using a network controller.

The **Data centers** in which the traffic management is crucial, is another field in which SDN can be useful [41]. An example of this, was presented by Google in early 2012, which has deployed a software-defined network to interconnect its data centers across the globe.

In data centers, the cost of hiring engineers to write control programs to run over multiple switches proved to be more cost-effective than purchasing switches which don't support new features and they are not flexible. SDN can benefit the data center networking by solving different problems such as live network migration, network management, failure avoidance etc.

There are also different projects which examined how SDN could be used **in small networks** at home as well as in wireless access networks.

Finally, SDN can be applied to **campuses**. Starting from Stanford University in which Openflow-enabled switches were deployed, the researchers of SDN attempt to deploy OpenFlow networks in other campuses too [43]. To win the confidence of network administrators, OpenFlow enabled switches attempt to **isolate the experimental traffic (processed by flow table) from production traffic**!

4.7 Future Development of SDN

There are currently research efforts of particular importance in order to unleash the full potential of SDN. Some of the ongoing efforts are the following [42]:

- Switch designs- Design switches which are able to perform well in heterogeneous implementations.
- Performance- The current OpenFlow switches support only around 200 control events per second. As a result, there are different proposals such as adding more powerful CPUs in order to overcome this limitation.
- High availability of the controllers- The controllers need to be able to handle the pressure of different objectives from the applications they host.
- Scalability- Several efforts have been done to tackle the scaling concerns of a software defined network. DevoFlow, Onix and Kandoo are some of these efforts.
- Security- There are numerous threats for an SDN architecture. Security is a top priority in SDN and the experts believe that this issue needs to be addressed and to be further investigated. There are some efforts like STRIDE with which it is possible to identify different attacks to an OpenFlow- enabled network.
- Migration to SDN- There are projects like RouteFlow which implements an IP level control plane on top of an Openflow network, allowing the underlying devices to act as IP routers under different arrangements. Panopticon is another approach which implements SDN inside the legacy networks.

There is a lot of activity around SDN. Emerging topics like the migration path to SDN or how to extend SDN towards carrier transport networks, require further research.

4.8 SDN and the relevance of RINA as a policy management architecture

As I was doing my research on RINA, after my initial assimilations I zoomed out to get a wider perspective and find comparisons between it and other architectures. As a result, I found out that there are some similarities with SDN. RINA supports SDN, by allowing users to set up a private network (by using DIFs) on top of physical networks and instantiate a variety of mechanisms such as routing and authentication rather than just forwarding as in OpenFlow [8].

The basic similarity between these two architectures is that they make a **clear distinction between the management, data transfer and data transfer control**. SDN separates the control from data plane and includes also a management plane for the programmability of the network and RINA separates the IPC data transfer from IPC control and IPC management tasks.

Furthermore, **scoping** is an aspect of both architectures. In RINA, each DIF of the network provides services over a certain scope. Similarly, in SDN the management layer is responsible for managing the network over a certain scope and for providing policy management which means that the management of the network can be done with high level policies (set of rules) instead of low level network device configurations [18]. By having a policy based management, leads to a simplified device and network management.

However, most existing SDN management layers are **limited to a specific management scope and it is not easy to define new scopes [18]**. Moreover, there are still open issues regarding the weaknesses of existing SDN management layers, such as the weak QoS support and security. Thus, the existing SDN management layers are tied to the current Internet architecture.

As a result, the researchers of RINA believe that a better approach is to build a management architecture on top of a new network architecture that avoids the shortcomings of TCP/IP architecture. The solution according to the researchers, is to adopt RINA and its recursive model [18], [46].

Due to the better manageability support and dynamic DIF instantiation of RINA, a DIF with a management scope can be dynamically instantiated and recursively formed over existing management scopes. RINA supports nested scopes which means that a large scope can span multiple existing scopes. Thus, collaboration across different administrative domains can be performed. Furthermore, with the explicit support of QoS, RINA can help the end –users to improve the application performance.

With the management architecture of RINA as well as with the recursive API which provides, the distinction between the southbound and northbound APIs in SDN, can be eliminated! Through the RINA API, the applications can be programmed recursively over different scopes.
Thus, RINA is an approach which is taken into consideration from the SDN researchers and offers a promising direction for SDN [18]!

4.9 Summary of the chapter

This chapter examines SDN and provides a brief description of this architecture. The main principle is the programmability of networks, through the separation of control from the data plane. Openflow is the most popular mechanism in a software defined network. It's an open standard which enables researchers to run experimental protocols in the campus networks. Finally, this chapter presents the applications fields of SDN and how RINA can be embraced as a policy management architecture.

References

[1] Evolution of the Internet, from http://en.wikipedia.org/wiki/Internet

[2] ARPANET, from http://en.wikipedia.org/wiki/ARPANET

[3] Clean-slate, from http://en.wikipedia.org/wiki/Clean_Slate_Program

[4] The last Waltz and Moving beyond TCP/IP, from http://rina.tssg.org/docs/PSOC-MovingBeyondTCP.pdf

[5] Clean slate versus evolutionary research, from http://cacm.acm.org/magazines/2010/9/98030-future-internet-architecture-clean-slate-versus-evolutionary-research/fulltext

[6] Routing table entries according to Cisco, from http://blogs.cisco.com/sp/global-internet-routing-table-reaches-512k-milestone

[7] Book: John Day, Patterns in Network Architecture- A return to Fundamentals.pdf

[8] RINA-Boston University prototype, from http://csr.bu.edu/rina/papers/BUCS-TR-2013-013.pdf

[9] Multihoming, from http://en.wikipedia.org/wiki/Multihoming

[10] We got trouble, from http://irati.eu/wp-content/uploads/2013/01/3-AddressingtheProblem130123.pdf

[11] Accessing the Security of a Clean-Slate Internet Architecture, from http://www.cds.caltech.edu/~doyle/wiki/images/3/3d/RINA_Security_11209.pdf

[12] Ethernet: Distributed Packet Switching for local computer networks, from http://research.microsoft.com/en-us/um/people/pcosta/cn_slides/metcalfe76ethernet.pdf

[13] Transport over Heterogeneous Networks Using the RINA architecture, from http://rina.tssg.org/docs/Transport_over_Heterogeneous_Networks_using_the_RINA_Architecture.pdf

[14] The Road to SDN: An Intellectual History of Programmable Networks, from https://www.cs.princeton.edu/courses/archive/fall13/cos597E/papers/sdnhistory.pdf

[15] Internet evolution and the role of Software engineering, from http://www2.research.att.com/~pamela/fose.pdf

[16] Tussle in Cyberspace: Defining tomorrow's Internet, from http://david.choffnes.com/classes/cs4700fa14/papers/tussle.pdf

[17] Rethinking the design of the Internet: The end to end arguments vs. the brave new world, from http://nms.lcs.mit.edu/6829-papers/bravenewworld.pdf

[18] SDN Management Layer: Design Requirements and Future Direction, from http://csr.bu.edu/rina/papers/BUCS-TR-2014-006.pdf

[19] Software-Defined Networking: The new Norm for Networks, from https://www.opennetworking.org/images/stories/downloads/sdn-resources/white-papers/wp-sdn-newnorm.pdf

[20] Overcoming the Internet Impasse through Virtualization, from https://homes.cs.washington.edu/~tom/support/impasse.pdf

[21] State of the Internet & Challenges ahead O.Martin, March 2008 doc, from http://www.ictconsulting.ch/papers.html

[22] Endpoints and Endpoint Names: A proposed enhancement to the Internet architecture, from http://mercury.lcs.mit.edu/~jnc/tech/endpoints.txt

[23] Why Loc/Id Split Isn't the Answer, from http://rina.tssg.org/docs/LocIDSplit090309.pdf

[24] Is the Internet an unfinished demo? Meet RINA! , from http://rina.tssg.org/docs/Is_the_Internet_an_unfinished_demo_-_Meet_RINA.pdf

[25] How in the heck do you lose a layer? , from http://rina.tssg.org/docs/How_in_the_Heck_do_you_lose_a_layer.pdf

[26] "Networking is IPC": A Guiding Principle to a Better Internet, from http://www.cs.bu.edu/fac/matta/Papers/IPC-arch-rearch08.pdf

[27] Bounding the router table size in an ISP network using RINA, from http://csr.bu.edu/rina/papers/Bounding-the-router-table-size-in-ISPs-using-RINA.pdf

[28] RINA: An architecture for Policy-Based Dynamic Service Management, from http://csr.bu.edu/rina/papers/BUCS-TR-2013-014.pdf

[29] Layer Discovery in RINA networks, from http://rina.tssg.org/docs/CAMAD-final.pdf

[30] Prototyping the Recursive Internet Architecture: The IRATI Project Approach, from http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=6786609

[31] On supporting Mobility and Multihoming in Recursive Internet Architectures, from http://www.cs.bu.edu/techreports/pdf/2010-035-rina-mobility.pdf

[32] Welcome to the RINAissance, from http://rina.tssg.org/docs/DublinIntrotoRINAPt2-140120.pdf

[33] DAFs and Management in RINA, from http://rina.tssg.org/docs/DublinMngmt140120.pdf

[34] Experiences with Implementing RINA, from http://rina.tssg.org/docs/20120307_PhDCourse-FutureNet-RINA.pdf

[35] An Introduction to RINA, from http://rina.tssg.org/docs/FutureNetTutorialPart1-100428.pdf

[36] Things they never taught you about naming and addressing, from http://rina.tssg.org/docs/FutureNetTutorialPart2-100415.pdf

[37] The Nuts and bolts of RINA, from http://rina.tssg.org/docs/FutureNetTutorialPart3-100504.pdf

[38] On the Performance and Robustness of managing reliable transport connections, from http://www.cs.bu.edu/fac/matta/Papers/deltaT-gc10-submit.pdf

[39] Security in RINA, from http://irati.eu/wp-content/uploads/2013/01/6-Security130123.pdf

[40] Report from the IAB Workshop on Routing and addressing, from http://tools.ietf.org/html/draft-iab-raws-report-01

[41] A survey of SDN: Past, present and future of programmable networks, from https://hal.archives-ouvertes.fr/hal-00825087/document

[42] SDN: A comprehensive survey, from http://arxiv.org/pdf/1406.0440.pdf

[43] Openflow: Enabling Innovation in Campus Networks, from http://archive.openflow.org/documents/openflow-wp-latest.pdf

[44] Stanford Seminar- SDN at the Crossroads, from https://www.youtube.com/watch?v=WabdXYzCAOU

[45] IRATI, Investigating RINA as an Alternative to TCP/IP, from http://irati.eu/wp-content/uploads/2012/07/IRATI-D2.1.pdf

[46] Software-Defined Networking (SDN): Layers and Architecture Terminology, from http://www.rfc-editor.org/rfc/rfc7426.txt

[47] RINA detailed Overview and Implementation discussions, from http://irati.eu/wp-content/uploads/2012/07/ImplementationOverview.pdf

[48] Qos – Wikipedia, from http://en.wikipedia.org/wiki/Quality_of_service

[49] IPv6- Wikipedia, from http://en.wikipedia.org/wiki/IPv6

[50] The Delta-t Transport Protocol: Features and Experience, from http://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=65288&url=http%3A%2F%2Fieeexplore.ieee.org%2Fxpls%2Fabs_all.jsp%3Farnumber%3D65288

[51] IRATI project, from http://irati.eu/

[52] IRINA project, from http://www.geant.net/opencall/Optical/Pages/IRINA.aspx

[53] PRISTINE project, from http://ict-pristine.eu/

[54] ProtoRINA project, from https://github.com/ProtoRINA/users/wiki

[55] OFELIA testbed, from http://www.fp7-ofelia.eu/

[56] “Networking is IPC”: A guiding principle to a better Internet-presentation, from http://www.cs.bu.edu/fac/matta/Presentations/IPC-arch.pdf

[57] Strategies for Operating Systems in Computer networks, from http://dl.acm.org/citation.cfm?id=569929

[58] Interprocess communication, from http://doc.utwente.nl/18379/1/mullender92interprocess.pdf

[59] Active networking, from http://en.wikipedia.org/wiki/Active_networking

[60] NETCONF, from http://en.wikipedia.org/wiki/NETCONF

[61] IRATI, investigating RINA as an alternative to TCP/IP, from http://irati.eu/wp-content/uploads/2012/07/IRATI-D4.2.pdf

[62] PRISTINE, Use cases description and requirements analysis report, from http://ict-pristine.eu/wp-content/uploads/2013/12/pristine_d21-usecases-and-requirements_draft.pdf

[63] RINA: An opportunity for NRENs to lead Internet Research, from https://tnc2014.terena.org/core/presentation/61