UNIVERSITAT POLITÈCNICA DE CATALUNYA (UPC)

Contributions to Routing Scalability and QoS Assurance in Cloud Data Transport Networks based on the Recursive InterNetwork Architecture

by

Sergio Leon Gaixas

A thesis submitted in fulfillment for the
degree of Doctor of Philosophy

in the

School of Informatics of Barcelona (FIB)
Computer Architecture Department (DAC)

April 2018

Contents

List of Figures

List of Tables

Abbreviations and Symbols

AS	Autonomous System
ASN	AS Number
ASO	Address Supporting Organization
BGP	Border Gateway Protocol
ccTLD	Country Code Top-Level Domains
CDAP	Common Distributed Application Protocol
CNNSO	Country Code Names Supporting Organisation
C/U	Cherish/Urgency
C/U Matrix	Cherish/Urgency Matrix
C/U Mux	Cherish/Urgency Multiplexor
C/U/R Cube	Cherish/Urgency/Rate Cube
DAF	Distributed Application Facility
DC	Data Centre
DCCP	Datagram Congestion Control Protocol
DDT	Delegated Database Tree
DFZ	Default Free Zone
DIF	Distributed IPC Facility
DNS	Domain Name System
DTCP	Data centre TCP
DTP	Data Transfer Protocol
ΔQ	Degradation of Quality
ΔT	Time Difference
ECN	Explicit Congestion Notification

EFCP	Error and Flow Control Protocol
EID	Endpoint Identifiers
FA	Flow Allocator
FTP	File Transfer Protocol
gTLD	Generic Top-Level Domains
HTML	HyperText Markup Language
IANA	Internet Assigned Numbers Authority
ICANN	Internet Corporation for Assigned Names and Numbers
IP	Internet Protocol
IPC	Inter Process Communication
IPCP	IPC Process
IPSec	Internet Protocol security
ISP	Internet Service Provider
LAN	Local Area Network
LISP	Locator/ID Separation protocol
MA	Management Agent
MAC	Media Access Control
MILP	Mixed Integer Linear Programming
MPLS	Multi-Protocol Label Switching
MTU	Maximum Transfer Unit
NAT	Network Address Translation
NMS	Name-space Management System
PCI	Protocol Control Information
PDU	Protocol Data Unit
PN	Programmable Networks
PM	Protocol Machine
QoS	Quality of Service
QTA	Quantitative Timeliness Agreement
QTAMux	QTA Multiplexor
RA	Resources Allocator
RINA	Recursive InterNetwork Architecture

RLoc	Routing Locators
RMT	Relay and Multiplexing Task
RIB	Resource Information Base
RTT	Round-trip Time
SCTP	Stream Control Transmission Protocol
SDN	Software-Defined Networking
SDU	Service Data Unit
SMPT	Simple Mail Transfer Protocol
SRV	Service Record
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
URI	Uniform Resource Identifier
VLAN	Virtual LAN
VoIP	Voice over IP
VPN	Virtual Private Network
VPN-DIF	VPN-like DIF

Summary

With an increasing number of devices and heterogeneous distributed applications, it is becoming evident that the best-effort service delivered by the current TCP/IP Internet fall short to supply the actual Quality of Service (QoS) requirements of applications. Not only that, but the global scope of the IP layer become the cause of great scalability problems with increasingly higher costs to solve. During TCP/IP lifetime, multiple solutions that aim to overcome the limitations of the model have appeared (BGP, NAT, LISP, etc.). Even so, all these solutions end being constrained by the same networking model that they try to improve. Given that, most solutions end simply breaking and patching the stack itself of TCP/IP.

Aiming to solve those problems, the Recursive InterNetwork Architecture (RINA) proposes a new clean-slate Internet architecture that returns to the roots of networking. Instead of patching the current TCP/IP stack, RINA proposes a recursive networking stack with focus on inter process communication, where each layer, called Distributed IPC Facility (DIF), performs the same set of tasks. While performing the same tasks, RINA DIFs are fully configurable by mean of programmable policies, definitions of how to perform such tasks. In addition, RINA provides complete support for QoS by the mean of QoS Cubes, or QoS classes defining the capabilities of each DIF. With the use of QoS Cubes, RINA is capable to provide a standardized way to express the capabilities of each layer. In addition, given that information, RINA also allows for applications and upper processes to express their QoS requirements in terms of accepted latency, losses, average rate, etc. The contributions in this thesis take profit from the recursive stack of RINA and the use of policies to propose and analyse old and new solutions for both QoS and scalability, which would not be compatible with the current TCP/IP Internet.

In terms of improvements of QoS services, the work in this thesis takes profit from the information on flow requirements provided by the applications itself, to improve the QoS assurances provided by the network. It proposes the use of ΔQ-based scheduling policies, providing improved QoS assurances, better matching the requirements of flows. In contrast to simpler QoS differentiation solutions, where QoS services are provided in a priority order, ΔQ aims to provide "good enough" service for all flows in the network, resulting in a more appropriate sharing of resources. In this work, these policies have been tested in backbone-like networks, showing interesting improvements with respect to common QoS differentiation solutions like MPLS-based VPNs. In addition to the use of

ΔQ policies in the core of the network, this thesis also considers the provisioning of QoS services to end-users, being that the ultimate goal of networking. In order to allow that, it is required to impose some limits on what end-users can send to the network, in order to limit the amount of priority traffic that potentially greedy users can send into it. In that regard, while enforcing strict rate-limits per QoS would be trivial in RINA, a new ΔQ-based rate-limiting policy that aims to limit the amount of priority traffic in a more user-friendly way is also explored.

In terms of improved scalability, this thesis also considers different measures to improve forwarding and routing within large-scale networks. First, as for the use of policies that could profit from specific network topologies, a new forwarding policy that mix both topological rules, namely forwarding decisions based on the location of nodes, and exceptions, namely forwarding entries overwriting rules upon failure, is proposed. With this policy, costly forwarding table lookups in large tables are replaced with fast and simple forwarding rules based on the location of nodes and their neighbourhood. Given the specific topologies used most commonly in large data centres nowadays, it is found the use of the proposed policy to be a perfect match for those scenarios. Test for different data centre topology showed clear improvements, requiring only a small fraction of all forwarding information despite the large size of such networks, depending that in the number of concurrent failures in the network rather than on the size of it. In addition, this thesis also considers the use of topological routing policies to populate such exceptions upon failures. The use of topological routing solutions resulted in reduced complexity for computing paths and less routing messages. In addition to topological solutions, the use other routing solution, not well suited for the IP environment are also investigated. Specifically, it is shown how a Landmark routing solution, a routing solution of the family of compact routing, could be implemented within RINA. Finally, efforts are also devoted to analyse the importance of path selection for ensuring QoS requirements and how it is not required to reach extremes solutions, like the use of connections, to provide the required services.

Resumen

Con un número cada vez mayor de dispositivos y aplicaciones distribuidas, se está volviendo evidente que el servicio best-effort ofrecido por la actual Internet TCP/IP no satisface los requisitos de calidad de servicio (QoS) de las aplicaciones. No solo eso, sino que el alcance global de la capa de IP se convierte en la causa de grandes problemas de escalabilidad, requiriendo costes cada vez más altos para ser resueltos. Desde la implantación de TCP/IP, han aparecido múltiples soluciones que tienen como objetivo superar las limitaciones del modelo (BGP, NAT, LISP, etc.). Aun así, todas estas soluciones terminan restringidas por el mismo modelo de red que intentan mejorar. Dado esto, la mayoría de las soluciones terminan simplemente rompiendo y parcheando la pila misma de TCP/IP.

Con el objetivo de resolver esos problemas, la Recursive InterNetwork Architecture (RINA) propone una nueva arquitectura de Internet que vuelve a las raíces de la comunicación en red. En lugar de parchear la pila actual de TCP/IP, RINA propone una pila de red recursiva con enfoque en la comunicación entre procesos, donde cada capa, llamada Distributed IPC Facility (DIF), realiza el mismo conjunto de tareas. Mientras realizan las mismas tareas, las DIF de RINA son completamente configurables por medio de políticas programables, definiciones de cómo realizar tales tareas. Además, RINA brinda soporte completo para servicios de QoS por medio de los Cubos QoS, o clases de QoS que definen las capacidades de cada DIF. Con el uso de los Cubos QoS, RINA es capaz de proporcionar una forma estandarizada de expresar las capacidades de cada capa. Además, dada esa información, RINA también permite que las aplicaciones y los procesos de capas superiores expresen sus requisitos de QoS en términos de latencia aceptada, pérdidas, uso promedio, etc. Las contribuciones en esta tesis sacan provecho de la pila recursiva de RINA y el uso de políticas para proponer y analizar soluciones, antiguas y nuevas, para QoS y escalabilidad, que no serían compatibles con la Internet TCP/IP actual.

En términos de mejoras de los servicios de QoS, el trabajo en esta tesis aprovecha la información sobre los requisitos de flujo, proporcionados por las propias aplicaciones, para mejorar las garantías de QoS proporcionadas por la red. Propone el uso de políticas basadas en ΔQ, proporcionando garantías de QoS mejoradas, que coinciden mejor con los requisitos de los flujos. A diferencia de las soluciones de diferenciación de QoS más simples, donde los servicios de QoS se proporcionan en orden de prioridad, ΔQ pretende proporcionar un servicio "suficientemente bueno" para todos los flujos en la red, lo que resulta en una repartición de recursos más apropiada. En este trabajo, estas políticas

se han probado en redes tipo backbone, que muestran mejoras interesantes con respecto a las soluciones comunes de diferenciación de QoS, como las VPN basadas en MPLS. Además del uso de las políticas de ΔQ en el núcleo de la red, esta tesis también considera el suministro de servicios de QoS a los usuarios finales, siendo ese el objetivo final de las redes. Para permitir eso, se requiere imponer algunos límites a lo que los usuarios finales pueden enviar a la red, con el fin de limitar la cantidad de tráfico prioritario que usuarios codiciosos puedan enviar. En ese sentido, aunque imponer límites de velocidad estrictos por QoS sería trivial en RINA, también se explora una nueva política de limitación de tasas basada en ΔQ que pretende limitar la cantidad de tráfico prioritario de una manera más beneficiosa para los usuarios.

En términos de escalabilidad, esta tesis también considera diferentes medidas para mejorar el reenvío y el enrutamiento dentro de redes de gran escala. Primero, en cuanto al uso de políticas que podrían beneficiarse de topologías de red específicas, se propone una nueva política de forwarding que combina reglas topológicas, es decir decisiones basadas en la ubicación de nodos, y excepciones, es decir entradas que sobrescriben reglas en caso de error. Con esta política, las costosas búsquedas en tablas grandes se reemplazan con reglas de rápidas y simples basadas en la ubicación de los nodos y su vecindad. Dadas las topologías específicas más comúnmente utilizadas en los grandes centros de datos hoy en día, se encuentra que el uso de la política propuesta es la combinación perfecta para esos escenarios. Pruebas en varias topologías comunes para centros de datos mostraron mejoras claras, que requieren solo una pequeña fracción de toda la información sobre la red, a pesar del gran tamaño de dichas redes, dependiendo esta de la cantidad de fallas concurrentes en la red y no del tamaño de la misma. Además, esta tesis también considera el uso de políticas de enrutamiento topológico para poblar tales excepciones en caso de fallas. El uso de soluciones de enrutamiento topológico dio como resultado la reducción de la complejidad en el cálculo de rutas, junto con un menor número de mensajes de enrutamiento. Además de las soluciones topológicas, también se investiga el uso de otra solución de enrutamiento, no adecuada para el entorno de IP. Específicamente, se muestra cómo una solución de enrutamiento Landmark, una solución de enrutamiento de la familia de enrutamiento compacto, podría implementarse dentro de RINA. Finalmente, también se dedican esfuerzos a analizar la importancia de la selección de rutas para garantizar los requisitos de QoS y cómo no se requiere llegar a soluciones extremas, como el uso de conexiones, para proporcionar los servicios requeridos.

Dedicated to my brother and parents,
with special thanks to my always supportive
advisors Jordi Perelló and Davide Careglio

"When it's all said and done, the Wired is just a medium of communication and the transfer of information. You mustn't confuse it with the real world. Do you understand what I'm warning you about?"
- Serial Experiments Lain

Chapter 1

Introduction

The current Internet architecture, based on the TCP/IP stack, faces many challenges, not considered or apparent in its early days. Among those, there are two big groups, commonly related. In one hand, challenges related to the scalability of the network, mobility or multi-homing of nodes (a.k.a. device), with great impact on the cost of the network and its capabilities. On the other hand, challenges related to the quality of the offered service and traffic engineering, with great effect to how the end-users view the network.

With a penetration rate already close to half the total human population[1], and even higher number of connected devices[2], the current Internet has grown multiple magnitudes larger than its original size. Not being though with the current scale in mind, scalability problems are more serious and notable as the network grows (e.g. IPv4 pool depletion [3] or different cases of BGP table overflow [4][5]). While the number of connected devices by itself entails some challenges, those have been commonly solved by the means of multiple hacks in the architecture. On the other hand, more than the sheer number of devices, their connectivity has a greater impact to the scalability on the network, with multi-homing, mobility or the renumbering of nodes having a large repercussion in the growth of routing tables in the backbone of the Internet.

As the network grow, new and heterogeneous services are also supported, creating a diverse and rich environment for network users that greatly expanding that of its origins (e.g. file-transfer, telnet, e-mail, etc.). The apparition of new applications also resulted in the need of new and varied service requirements for those. In that regard, the current TCP/IP Internet find one of its greatest weakness, providing a default best-effort service in which applications are incapable of stating its own communication needs, only being possible to differ between services in an end-to-end basis with the use of different data transport protocols.

As new problems emerged, different solutions for those appeared, hacking the model in order to circumvent its limitations. As opposed to these solutions, the Recursive InterNetwork Architecture (RINA) proposes a redefinition of the network model, instead of trying to repair the existing architecture. RINA is not an extension of the current architecture, but a new advance towards the original concept of InterProcess Communication (IPC) defining any data transfer process between applications and devices, centred in only a couple of primitive operations. While not as mature as TCP/IP solutions, RINA offers an enhanced and more open environment for improvements, centred in the inherent recursivity of the network, the ability to configure each layer and a "by-default" support of for Quality of Service (QoS).

The main contribution of this work is in the study of the capabilities that the use of RINA opens in the fields of quality of service and network scalability. With respect to QoS, the work in this thesis bases on the ideas behind the Degradation of Quality (ΔQ) in order to provide improved QoS services within RINA networks. ΔQ states that, given the relation between throughput, delay and losses, it is possible to analytically predict the degradation that for flows traversing a congested node would suffer. Considering that, ΔQ-based scheduling solutions can target specific different maximum degradations for flows traversing the network depending on their quality requirements, as opposed with common priority-based solutions that enforces most of the degradation of quality to low requiring flows. With respect to network scalability, this thesis focus in the research of routing and forwarding solutions that either take profit from the topology of already existing networks (e.g. data centre networks) or that take some topological properties of the network to reduce routing tables at the cost of slightly longer paths. In addition, it is also considered the effect of routing and forwarding decisions with respect to the degradation of the service suffered by the flows traversing the network.

The first chapters of this thesis serve as an introduction to the general idea of networking and the current TCP/IP Internet. Chapter 2 provides an overview of networking theory and general topics. The idea of networking is not bounded to a specific architecture, but only provide general rules on how communication should be done. Those include the idea of naming and addressing of nodes, the requirement of using shared communication protocols or the distinction between the different levels of coupling of communication shared state. Parting from the general idea of networking, in Chapter 3 the current TCP/IP Internet architecture is analysed. TCP/IP has its roots in a networking environment that largely differs from the current one. A large number of problems have been emerging from that, from the lack of a real QoS support to major scalability problems that have resulted in large-scale Internet outages. This thesis analyses the root of some of those problems, as well as the different solutions that have been emerging to solve them.

Chapter 4, introduces the InterNetwork Architecture (RINA) and its recursive approach towards a new Internet model. RINA proposes a different approach to networking, departing from the functionality-based TCP/IP, where the networking stack is defined by recursively stacking the same kind of layer, the DIFs. Instead of defining those layers to

perform a specific function (e.g. transport or security), DIFs are defined by the networking domain that they encompass, and network managers are free to configure them given the specific needs of that networking domain. In addition, RINA provides a complete support for QoS within the DIFs, and a shared API that allows to recursively propagate network requirements.

Chapter 5, explores the opportunities that RINA opens with respect to QoS and its joint work with ΔQ theory on quality degradation. Within this thesis the different requirements (including both the traffic requirements, physical properties, etc.) required to provide reliable QoS assurances to the network are analysed, and a ΔQ-based scheduling policy for RINA networks is proposed. QoS assurances provided by ΔQ are compared to simpler QoS differentiation provided by common scheduling solutions (e.g. weighted fair queue). This thesis also analyses the benefits of ΔQ and QoS assurance in backbone networks with respect to the sharing of quality degradation. Finally, given that providing QoS guarantees to the end-users, where applications run, is the final goal of ΔQ, this thesis also analyses the use of rate limiting policies in order to avoid an incorrect usage by greedy users, as well as propose a new rate limiting policy with enhanced freedom of decision.

Chapter 6 explores the possibilities that RINA opens for topological routing solutions towards enhancing the scalability of the network, as well as the importance of routing for QoS and resource allocation. This thesis introduces the Rules&Exceptions forwarding policy, a forwarding policy that takes profit from the properties of well-known network topologies to leverage the scalability constrains on large networks. Then, making use of the previously introduced policy, the benefits of topological solutions within the specific scenario of data centre networks is analysed. In addition to topological solutions, this thesis also explores the possibility of compact routing solutions within RINA, something not doable in the current TCP/IP Internet, as well as provide a brief study of the requirements of routing in a QoS enabled environment.

Finally, Chapter 7 draws some conclusions from the research outcomes, and briefly examine possible future works in relation with the content of this thesis.

Chapter 2

Networking

In this thesis, different quality of service and routing solutions for RINA networks are introduced. These solutions share all the particularity of not being implementable for the current TCP/IP based Internet, or, at least, having a large implementation cost. Being the main environment for the proposed solutions one that breaks from the current Internet model, the first chapters of this thesis are focused on introducing some fundamentals principles presents in any network and how those are implemented within the current TCP/IP networking model, as well a present fundamental problems presented in the use of that networking model.

2.1 Networking overview

The purpose of this chapter is not to lay down the design for an ideal networking architecture, but to present the fundamental principles of networking and convey various guidelines on how a networking model should be designed. First, let us define "what" is networking. A good definition can be found in the Merriam-Webster dictionary, ≪Networking is the exchange of information or services among individuals, groups, or institutions.≫ [6]. While this definition does not specifically reference the scenario presented in computer networks, it clearly defines the goal of any kind of networking, namely the translocation of information between different locations. The specific case of networking in computer networks, or simply networking from now on, shares most of the principles of traditional networking, while imposing higher requirements on the process itself (i.e. the translocation of information has to be done faster, securely, etc.). But, at the end, networking is networking.

2.2 Networking fundamentals

Citing John Day in its book "Patterns in Network Architecture" [7] ≪Networking is distributed InterProcess Communication and only Inter Process Communication≫. As it names suggest, InterProcess Communication (IPC) is the act of exchanging information between different processes in a computer system, and networking is just that, only expanded to distributed systems. Networking is not bounded to a specific architecture, and even less to an implementation.

Borrowing again the definition from John Day, ≪An architecture is A set of rules and constraints that characterize a particular style of construction≫. In fact, a networking architecture is no more than the set of rules that defines how to reach the goal of communication itself. Given that direct any-to-any communication is something not viable in most networks, requiring connecting all pairs of nodes either by the means of one-to-one or shared mediums, networking architectures tend to be divided in various layers. In such layers, sets of IPC Processes (IPCPs) are connected as a sub-network from the whole network, defining a networking domain. IPCPs in that networking domain, then can provide paths to upper layers that would be seen as a direct links between nodes.

At the bottom of the stack there is the layer 0 or the bearer of communication. Commonly, this bottom layer is the one representing the physical links connecting the different devices. But, that is not always necessarily true. In fact, the bottom layer is simply the last layer known by the stack, providing seamlessly direct links the nearer neighbours. And, while that is what the physical layers do, other technologies, unknown by the stack, can also act as layer 0.

Something important when defining layers is to give names to the different IPCPs in there as so to identify all communicating parties. In this regard, Jerry Saltzer noted how names, addresses and paths are required to define a complete addressing architecture [8]. As Saltzer noted, in order to communicate within any networking domain, three elements are required:

- Name, defining *who* is the node in the networking domain.
- Addresses, defining *where* is the node in the networking domain.
- Routes, defining *how* to reach the node in the networking domain.

This separation between name, addresses and routes may not be always requires (e.g. in a point-to-point network), but provides a general rule as to how to define general network architectures. It has to be noted that this rule is not only valid for computer networks, but also in many other forms of indirect communication. The simpler example of that could be sending a letter, an example where we know *who* has to receive it, *where* that someone can be located and the post office knows *how* to reach that location.

In any communication network, IPCPs communicate in order to share either information for IPCPs on their upper layers or to maintain a shared state between the different IPCPs forming part of that same communication. In order to perform that communication, IPCPs make use of shared schemes known as protocols, namely a set of rules and methods shared by all communicating processes. Those protocols exchange information in the form of self-contained pieces of data known as protocol data units (PDUs). Those PDUs are formed by the protocol control information (PCI) and the service data unit (SDU), also known as header and user-data respectively. PCIs contain information about the state of the communication process and are interpreted by the protocol. That commonly contains information like the source and destination of the communication process, the size of complete PDU or how the information should be treated during its transmission. In contrast, SDUs contain data providing from upper IPCPs (either application data or upper PDUs) and cannot be (or at least should not be) interpreted by the protocol of the current layer.

As stated, PCIs contain information aimed, in part, to maintain a shared state in the communication. Depending on how tight that shared state is, the communication can be defined from a simple association, with minimal shared state between nodes, to a complete binding, with a fully shared state between nodes.

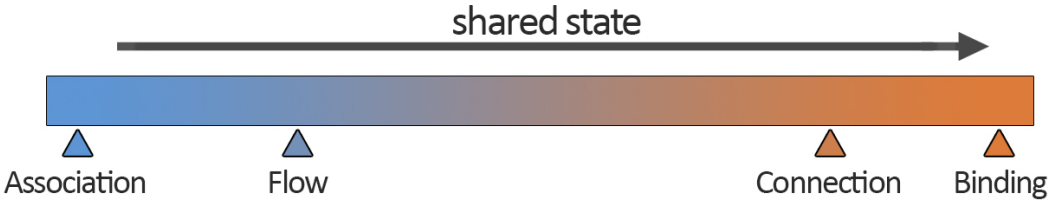

FIGURE 2.1: Levels of coupling of communication shared states.

Fig. 2.1 shows a common simplification of how shared states define the communication flows. A part from the extreme cases of associations and bindings, communication tend to be divided between flows, with loosely coupled shared states, and connections, with tightly coupled shared states. The space between flows and connections is not a discrete jump, but allows a wide range of combinations in terms of how thigh are the distinct elements coupled. Given that, this work refers as flow indistinctly for flows, connections and in-betweens. In particular cases, it may still distinguish between connectionless and connection-oriented treatment of the flows, denoting flows with a loosely or tightly coupled shared state with respect to specific elements (e.g. information routes, resource allocation, etc.).

Within a computer network, any two nodes with communication capabilities goes through three different phases in order to perform any communication:

1. Node Enrolment

   In this first phase, the different objects required for the communication are created and configured accordingly to the layer configuration. This includes the creation of

the IPCPs themselves, if those have not been created yet. In order for neighbouring nodes to communicate, it is required for both to be correctly enrolled in the same layer.

2. Flow Allocation

   After an IPCP is configured for a specific layer, it can then start new communications. In this phase, new flows can be allocated, for which the required shared state need to be configured in the appropriated nodes.

3. Data Transfer

   After a flow is allocated, information can be exchanged. In this phase, the communication protocols will exchange PDUs, containing both PCIs (to maintain the shared state) as well as SDUs (the exchanged information).

As a note, with the exception of data transfer, both node enrolment and flow allocation can be completely done without any exchange of information, as long as the layer configuration allows it. An example of that can be seen in layers with static configuration, for which enrolment is sometimes self-contained in the same node, or in physical links, where data is sent as independent packets (e.g. Ethernet frames), requiring only a minimum shared state.

2.3 Protocols and task

As stated previously, in order to communicate, IPCPs make use of shared schemes known as protocols. In order to be usable, those protocols need to be known by both parties and an agreement on its configuration need has to be reached before making communication possible. In that regard, protocols are divided into mechanisms and policies. Mechanisms define the functions of the protocol itself and cannot be changed. In contrast, policies defines the configuration of the protocol and can be negotiated before (and even during) the communication. In resume, mechanisms define *what* to do, while policies define *how* it has to be done. As an example, mechanisms and policies can be compared with a simple communication in person, when talking would be the mechanism and the language the policy.

With regard to networking, communication mechanisms can be categorized between mechanisms related to the enrolment of nodes, mechanisms related to the transfer of data and mechanisms related to the control of that transfer. Tables 2.1 and 2.2 sum up some basic communication mechanisms related to this work.

In addition to the protocols, IPCPs also have to perform different tasks in order to ensure the communication. Like protocols, tasks are also collections of mechanisms and policies. A common example of communication task presented in most IPCPs is that of relaying

TABLE 2.1: Basic communication mechanisms for data transfer

Name	**Mechanism /Policy**	**Description**
Addressing	Mechanism	The protocol has to provide unambiguous addresses for all IPCPs in the networking domain.
	Policy	How address are assigned are assigned is left as policy. E.g. static addressing (e.g. MAC addresses [9]), dynamic per-request (e.g. DHCP [10]), etc.
Flow Identification	Mechanism	Any protocol that supports multiple communication flows between the same pair of IPCPs requires an unambiguous way to differentiate them.
	Policy	Identification can be by the means of unique flow-id within all the networking domain, a pair of source and destination ports (e.g. UDP [11] or TCP [12]), etc.
Relying (Forwarding)	Mechanism	Whenever a PDU has to be transferred from one IPCP to another, one or more N-1 flows have to be selected in order to relay that PDU to the destination IPCP or a closer one.
	Policy	How to select the N-1 flows is left to policy. Common approaches are those that use forwarding tables mapping destination to N-1 flow [13], but other aproaches are possible (e.g. rely to next IPCP in ring network [14]
Multiplexing	Mechanism	Manages the movement of PDUs from the layer N into N-1 flows.
	Policy	How PDUs have to be scheduled. Scheduling is required whenever congestion can occur in the network, but also can be used to, for example, enforce traffic patterns. Different scheduling policies are commonly used depending on the requirements of the network (e.g. best-effort [15], Weighted Fair Queue [16], etc.).

and multiplexing PDUs. In any system, this task is composed of multiple mechanism, including the decision of PDU forwarding and the scheduling of colliding data, even so how that same forwarding or scheduling are done is decided via policies (e.g. a forwarding table or a FIFO scheduling). Although tasks are similar to protocols in their form, unlike those, their configuration is not required to be shared between nodes. Given this, different policies for the same task can be configured in neighbouring IPCPs, allowing configuring each task given the specific requirements of the node.

TABLE 2.2: Basic communication mechanisms for data transfer control

Name	**Mechanism /Policy**	**Description**
Flow Control	Mechanism Policy	Mechanism that prevents for a sender to send more data than the receiver can take. Flow control can be done with a credit scheme or a pacing scheme (sometimes both in parallel). In a credit scheme the receiver tells the receiver how much octets or PDUs it can send at any moment (e.g. TCP's sliding windows [17]). In a pacing scheme, the receiver tells the sender how fast it can send data, either as PDU/s or octets/s (e.g. Token Bucket [18]).
Congestion Control	Mechanism Policy	This mechanism tries to protect the network from periods experimenting collapses due to congestion. Different policies can be considered depending on the network. For example, Ethernet uses collision detection [19] to stop sending on collisions, TCP uses congestion avoidance to reduce the throughput when encountering losses [20], other policies uses congestion notification techniques to inform of congestion [21], etc.

2.4 Chapter summary

The goal of networking is to translocate information between distributed processes in the network, and in order to do that, the different networking architectures follow basic principles, independent to the architecture. With that in mind, this chapter introduces some of the principles behind networking, or distributed IPC. Knowing these principles helps to understand some of the main problems with the current TCP/IP Internet architecture, given its interpretation of those (e.g. incorrect separation of name, address and route; fixed stack; strict separation between flow and connection (UDP vs. TCP); etc.).

Chapter 3

TCP/IP Internet, problems and solutions

A networking architecture is a set of rules that define how to reach the goal of communicating nodes in a distributed network. While Internet has been constantly evolving, its fundamental principles have been maintained in as an Internet architecture based on the TCP/IP Internet protocol suite [22]. The TCP/IP started its development in the decade of the 70's, marking its steps to its complete adoption in the flag day of the 1th of January of 1983 [23], when it replaced the previous networking protocol in the ARPANET (the predecessor of the current Internet). From those days, the TCP/IP model has not changed too much in its core. In this chapter overviews the Transmission Control Protocol/Internet Protocol (TCP/IP) Internet protocol suite and describe how it implements some of the fundamental principles of networking. Then, de remark some of the major problems of this model with respect to today's and future environments, as well as introduce some of the more widespread solutions for them.

3.1 Today's Internet

From the very beginning, the current Internet architecture has been based in the fundamental principle of providing simple best-effort communication over a large but resilient network with global connectivity. With this Internet model, the core of the network is maintained as simple as possible, removing most of the communication intelligence from it. In contrast, the extremes of the network manages most of the communication intelligence, being all flow control tasks done in an end-to-end basis, instead of behind hidden within the network.

Similar to the OSI model [24, 25], the TCP/IP Internet protocol suite is based on a hierarchy of layers (Fig. 3.1), each with a specific functionality from which they are named (Link, Network, Transport and Application). This layering bases in the design

of TCP/IP architecture as a standard protocol suite for large area networks, connecting multiple network segments. As it is, the current Internet is a collection of thousands of networks globally distributed. This makes of the Internet a global network that provides connectivity between all nodes in it before that is even required.

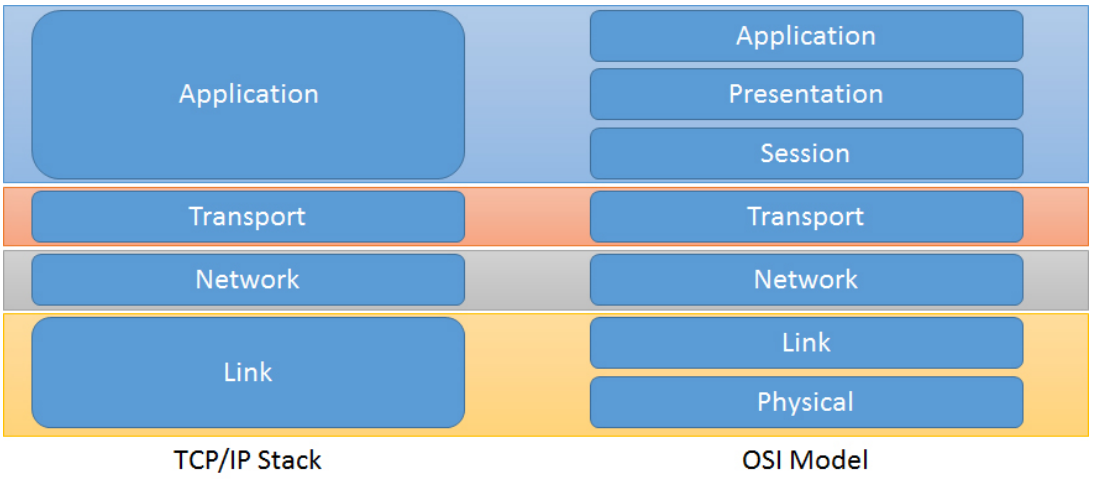

FIGURE 3.1: Comparison of TCP/IP and OSI stacks

In the TCP/IP stack, the layer 0 is named as Link layer and contains both the physical layer itself and the basic protocols running over it. This provides a source for direct connectivity between physically connected nodes in the network. Multiple technologies are supported as a bearer for TCP/IP, both electrical and optical, only requiring to use protocols capable of packet-based communication (e.g. Ethernet[26], Token Ring[14], Frame Relay[27], ATM[28], etc.).

Over the different link layers, there is the Network layer. This layer is mainly composed by the Internet Protocol (IP) with its two versions IPv4[29] and IPv6[30]. The IP protocol is in charge of providing a simple best-effort communication between the members of the network, but without performing any flow control on the flows. Depending on the scope, this layer can define either a private or public networks. Fig. 3.2 shows the default headers of both IPv4 and IPv6 protocols. While there exists other differences between IPv4 and IPv6, the most notable is that of the increased address size in the latter one, going from 32 to 128 bit addresses. This is something required to endure the constant grow of the network, as IP assigns addresses directly per interface, addresses being also commonly used as name for the nodes.

In the case of public networks, the network commonly known as "Internet" is a specific case of public global area network connecting most public and private networks around the globe. In order to manage such a large area network, the Internet Corporation for Assigned Names and Numbers (ICANN) [31], and its subsidiaries (IANA[32], ASO[33], CNNSO[34], etc.), are in charge of decisions like the assignation of addresses, names, etc. within the network.

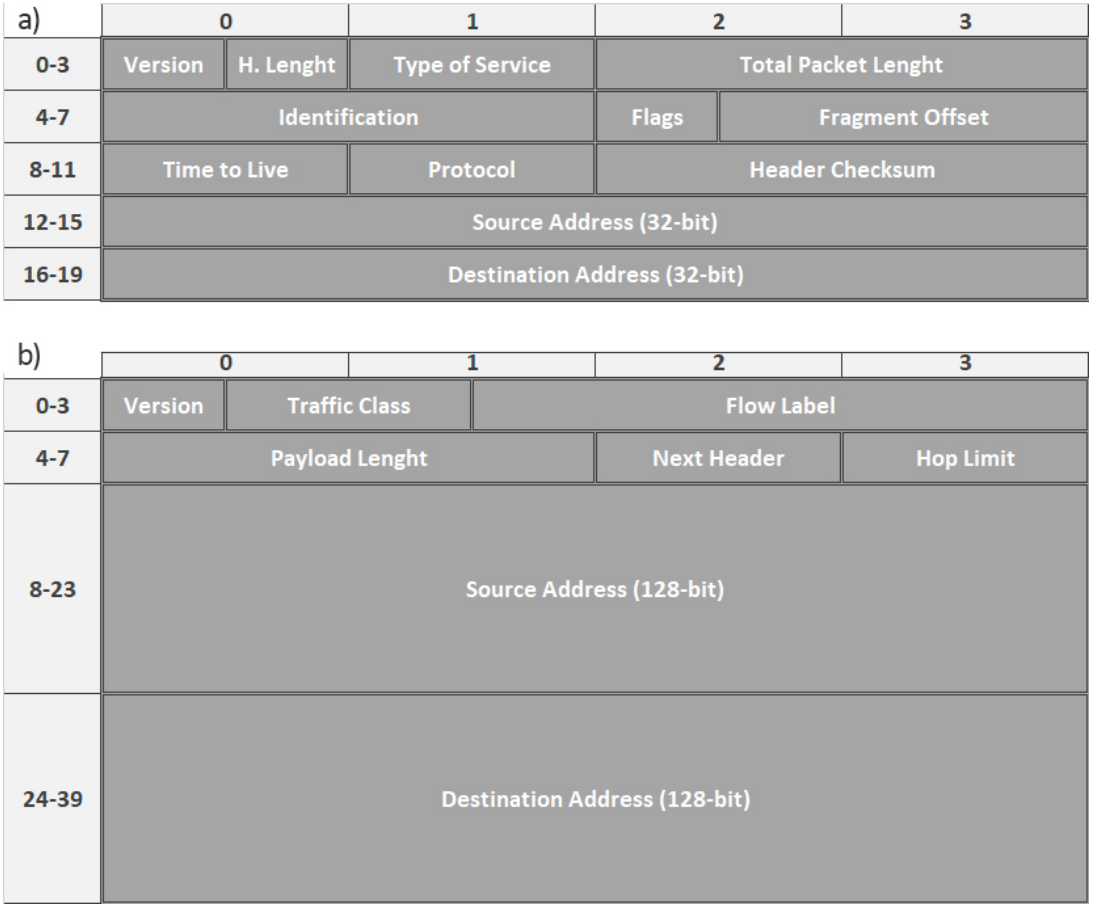

FIGURE 3.2: Headers of the IP protocol. a) IPv4; b) IPv6.

While the link layer is tightly mapped to the physical link technology connecting the nodes and the network layer bases on two different versions of the IP protocol, that is not the same for the transport layer. In conjunction with the IP layer, providing connectivity between all the nodes in the network, the transport layer is placed over it, providing transport capabilities (data transfer and data transfer control). Unlike the IP layer, the transport layer do not connect nodes themselves, but the applications running on them. In that regard, transport protocols do not encounter the same requirements for large addressing spaces as IP networks. In contrast, transport protocols tend to use what is known as ports (commonly with a length of two bytes) to address specific flows between specific pairs of nodes. In addition, as for IP, transport protocols tend to use addresses as synonyms of application names, creating what is commonly known as "well-known ports"[35].

The transport layer is formed by multitude of protocols, being the most known the User Datagram Protocol (UDP)[11] and the Transmission Control Protocol (TCP)[12]. Each of these protocols provide a specific implementation of data transfer and data transfer control mechanism and policies. This approach of having fixed implementations provides a handful of easy to use transport protocols, each designed for a specific use case, but

also has some drawbacks as will be seen later. Table 3.1 gives a small description of some of the most used transport protocols and their benefits.

Table 3.1: Common Transport Protocols

Protocol	**Acronym**	**Description**
Transmission Control Protocol	TCP	Provides reliable, ordered and error-checked streams of data. Connection oriented and provides flow control mechanisms based primarily on losses, although it can use ECN.
User Datagram Protocol	UDP	Provides unreliable but error-checked transmission of units of data named "Datagrams", never bigger than the MTU of the network layer. Connectionless without any flow control mechanism.
Stream Control Transmission Protocol	SCTP [36]	Provides reliable, ordered and error-checked transmission of units of data. Connection oriented with flow control mechanisms. Sometimes, used as a transport layer +1/2 over UDP or TCP.
Datagram Congestion Control Protocol	DCCP [37]	Provides reliable and error-checked transmission of units of data. Connection oriented with flow control mechanisms based on ECN. Sometimes, used as a transport layer +1/2 over UDP.
DataCenter TCP	DCTCP [38]	Provides reliable, ordered and error-checked streams of data. Connection oriented and provides flow control mechanisms. Modification of TCP for data centre environments.

3.2 Major problems with today's Internet

As stated before, the TCP/IP protocol suite, used in the current Internet, is defined by an architecture that fixed the layering on the network based on specific networking functionalities. In addition, protocols used in there do not have a noticeable separation between mechanism and policy, but instead different "protocols" are used for the same goals (e.g. TCP and UDP for data transfer). All this results in some problems unavoidable within the current networking model that limits the growth of the network and its ability to adapt to an ever-changing environment. With that in mind, this section lists some of

the main problems of the TCP/IP Internet protocol suite, focusing principally in those related to the scalability of the network, and its adaptability to new requirements.

3.2.1 Fixed Stack

First of all, the root of most problems with the current Internet is the fixed stack that supports it. Having a fixed stack in the network has interesting benefits. For instance, as layers are fixed and network protocols are limited (IPv4, IPv6, ARP[39], etc.), it becomes easier to implement hardware solutions going only up to some specific layer (e.g. switches with only link-layer functionality or routers with link and network functionalities), something that allows for cheap commodity hardware. It also becomes easier to analyse the network with simple tools (e.g. traceroute[40]). But, as the network grows, these benefits gets more and more occluded by what the network cannot provide.

For instance, while not a problem at the beginning given the small number of nodes, having a unique global network layer entails multiple scalability problems, as that requires to, not only name, but provide a route to all nodes in the network. This is a recurring problem that has already cost millions in hardware and has caused important Internet outages, like when different cases of BGP table overflow occurred in 2008[4] and 2014[5, 41, 42]. This also has an important implication in how the network is designed, as it defines the stack before the network. Given this, multiple networking solutions are removed from the very beginning and it is also difficult to merge public and private networking domains, or, what is the same, provide communication between private and public networking domains without using solutions that modify the stack itself.

However, not only the fact that the network is defined after the stack and not otherwise is what causes problems, but it is also a problem how the layering of the stack is defined. As stated before, the TCP/IP stack defines its layers based on their functionality. While this serves to remove more complex mechanisms from lower layers, it also limits the functionalities of the different networking domains (e.g. it is not possible to provide secure flows directly in the networking layer). Given the problems that carries the fixed stack, most solutions used to solve problems in the current Internet bases on the modification of the stack itself. Those effectively solve some of the problems with the current TCP/IP Internet, but do that at the cost of breaking the model itself, losing at the same time part of the benefits that a fixed stack provides.

3.2.2 Incomplete Naming/Addressing

At the earlies of networking, the first important network, the Advanced Research Projects Agency Network (ARPANET)[43], was only but a small network connecting few universities and research institutions. In that network, the Network Control Program (NCP)[44,

45] was the protocol in charge of managing the communication between hosts in the network. Thought to work in a small network, NCP was designed in a way that provided addressing based on the physical interfaces of nodes, rather than addressing the nodes themselves. This encounter some problems already in the early years, when, in 1972, the Tinker Air Force Base wanted a double connection through two distinct interfaces for redundancy[7, p.xvii]. It was not until the first of January of 1983 that the ARPANET changed from NCP to the TCP/IP protocol suite in the widely known Flag Day[23]. At that moment, Saltzer had already stated the requirements for a complete naming and addressing scheme in distributed computer networks [8], namely the separation between name, address and route. Even so, ARPANET was still a small network, and the old naming by interface of NCP was migrated to the new networking model, without considering future problems that this could carry.

It was not after the year after ARPANET closed that the network begin gaining popularity. The popularization of the World Wide Web (WWW)[46, 47] after 1991 propelled an exponential grow of the new Internet, an amalgam of different networks (PSINet[48], Alternet[49], CERFNet[50], etc.). While this new growing Internet was not the same old small ARPANET, it took TCP/IP as his default architecture, and, with that, all its scalability limitations. For starters, as Internet maintained an addressing scheme based on naming the connected interfaces, rather than in nodes themselves, the aggregation of addressees become something not always possible, as more and more sub-networks relied on multi-homing to improve their interconnectivity and increment reliability.

In addition, a more evident problem with the addressing in the current Internet is the depletion of the IPv4 address-pool [3, 51]. At this moment, more than half of the world population has access to Internet [1] (3.8e9 users as of June 2017) with a total of more than 2e10 connected devices in the network[2]. It has to be noted that only a fraction of those devices owns a public IPv4 address and most user devices ends sharing the same address. Even so, the large number of connected devices, the global scope of the IP layer and the interface-based addressing has resulted in the depletion of the public address-pool of IPv4, being most if not all the addressing space already assigned.

3.2.3 General Routing and Forwarding

Given the constant expansion of Internet in the last years and the tendency to multi-home services, routing tables have grown at a fast pace. This has a great impact especially in the Default Free Zone (DFZ) routers [52], located at the backbone itself of the Internet. The number of BGP entries in those nodes has become of a great concern in the latter years, in part as a cause of the aforementioned Internet outages of 2008 and 2014, caused when the amount of prefixes stored in BGP tables surpassed the 256K and 512K limit respectively. And, while the problem has been clearly noted, the burden on those routers keep increasing as those require to store an incrementing number of routing prefixes each year[53][54][55].

Scalability is one large problem of the IP network, requiring larger and larger routing and forwarding tables when approaching to the core of the network. This problem has its root in the TCP/IP network architecture itself: the fixed stack, having a unique global network, and the interface-based addressing. As these shortcomings of the model cannot be overcome with solutions that faithful follow the fixed stack, most solutions used in the current Internet focuses on patching the stack adding new intermediate layers or repeating existing ones. While these solutions move away from the traditional stack, their usefulness makes them widely accepted. However, even these solutions encounter some problems, and sometimes that is caused by one of the same things that has helped to the current TCP/IP Internet to grow at that pace, the use of cheap commodity hardware, that, faithful to the Moore law, became cheaper and more powerful as the network grew.

With access to cheap commodity hardware, these new solutions end up relying on existing hardware rather than develop something new, designed specifically to the solution's environment. This means that, independently on the network topology or requirements, generic solutions based on traditional forwarding tables are being used (e.g. use of BGP in data-centres[56]). As those are designed to work in any topology, it becomes almost impossible to take profit from the specific topology of the more regular networks. This results in a movement of the focus of network managers from searching the most optimal solution for their current scenario to search the most optimal configuration to use with some existing protocols.

3.2.4 Lack of QoS support

One of the key point of the current TCP/IP Internet is that it is a global network that provides resilient best-effort services. This marks the base-point for the common lack of QoS differentiation in Internet. On the beginning of Internet, this was one of those "not seen" problems, occluded behind a non-heterogeneous environment and the relatively small capacities of the network. At that point only a handful number of applications populated the network (e.g. Telnet[57], FTP[58], e-mail[59], RJE[60], etc.). In part, that was a result of the low popularization of the network at that point and the fact that the access to the network was done with low speed dial-up links. With maximum speeds of only 64kbps, dial-up links imposed a high added latency to all outgoing packets, something that at the same time limited the apparition of new distributed applications, as those that would require high bandwidths or low latencies could not receive the appropriate service.

It was not until the collision of multiple factors (i.e. the implementation of broadband connectivity[61], the popularization of the web 2.0[62], smartphones[63] and social media[64], online-gaming[65], etc.) that people mentality started to change and more and more varied applications started to populate the network. With the emergence of new applications, networks requirements have become more diverse, however, being designed with the provision of best-effort services in mind, the current TCP/IP Internet is not

capable of fully understand and provide them. And, while different solutions have been used to provide QoS differentiated services in the TCP/IP Internet, those solutions end encountering an almost impassable barrier in the lack of means for applications to reliable inform of their communication requirements, having to rely in most cases in the use of well-known ports to differentiate applications.

3.2.5 End-to-end flow control

Another problem, related to the improper layering and minimum capabilities in the network layer, is the lack of short-term flow control. While congestion in a network can occur at any layer (from a busy link to a full socket buffer), it is commonly the congestion on lower layers what really affects the behaviour of a network. Within the TCP/IP Internet protocol suite, the enforcement of flow control is done mainly in the transport layer (by protocols like TCP, DCCP, etc.). While this moves most of the computational cost related to congestion control to the network end-points, it also moves the action away from the points of congestion, resulting in a less optimal solution. That is not the only problem, but only part of it, as not only flow control is only done end-to-end in most cases, but the use or not of flow control mechanisms is something entirely left to the end-users. This has some dangerous implications, as protocols like TCP will try to adapt to the state of the network, while others like UDP will simply overflow the network without considering what other flows do.

In addition, a great problem in nowadays networks, filled with wireless devices, is that congestion control mechanisms tend to work based on packet losses rather than on information from lower layers. Coming from a wired environment, congestion control mechanisms are designed with the idea that "losses = congestion". That is mainly true in wired environments, where links are reliable enough to be considered "lossless". There, if a packet is lost, it can be assumed that the cause has been congestion in some node along its path. That is not the case for the wilder environment of wireless networks. With unreliable links, the probability of losing independent packets grows larger and that makes of losses an unreliable indicator of congestion. In this regard, the TCP/IP Internet protocol suite already has ways for nodes to inform of congestion, like with the use of the ECN bit, even so, the usage of this has been widely ignored, both at congested nodes and end-points.

3.3 Diverse solutions to Internet problems

The previous section outlined some of the main problems of the TCP/IP Internet protocol suite and the history behind them. As stated multiple times along this chapter, in order to maintain a correct service yet those problems, new solutions are constantly

been developed. Even so, while these solutions mainly bases on the TCP/IP model, in order to surpass the shortcomings of the model, it is common for those to break one or more of the default assumptions in it (e.g. add new layers, separate name from address, etc.). With that into consideration, this section outlines and analyses few of the most important solutions that maintain working the current TCP/IP Internet.

3.3.1 BGP

The backbone of Internet is formed by multitude of Autonomous System (AS), namely a group of networks with their own independent addresses. Each of those AS has their own AS number (ASN), assigned by the same authority in charge of assigning IP addresses, the IANA. Each of those AS manage its own group of sub-networks independently of the rest of ASs, commonly in a transparent way. This creates two different kind of networking domain within the backbone of Internet, the private or internal domains (within ASs), and the public or external domain (between ASs).

For these environments, the Border Gateway Protocol (BGP) [66] was designed. As ASs manage their networking domain independently of the rest of the network, having a unique routing domain in all the backbone would be counter-productive (having to filter and forcefully aggregate all information on the AS borders. BGP uses a path-vector[67] approach, where the paths to the different ASN are propagated ensuring that there are no loops in the network. Alongside the ASNs, the IP prefixes owned by that AS are also shared. This has some benefits with respect to directly computing the path for each prefix, as some AS may own a large number of IP prefixes. With that in mind, BGP defines two different usages, External BGP (eBGP)[68] and Internal BGP (iBGP)[69], for routing between border routers in different ASs or within the same AS. In the case of eBGP, while an AS can have multiple border routers, those are considered as if they were all the same, simply an access point to the AS.

While BGP manages the routing between ASs, as those manage their own networking domains, routing within an AS's private domain can be done in multiple ways. For example, protocols like the Enhanced Interior Gateway Protocol (EIGRP)[70] provide a distance vector approach that not only consider the distance in hops between nodes, but also the bandwidth of links, supported load, etc. Even more, it is also possible to use BGP within an internal BGP domain, something useful to manage large ASs structured as a collection of smaller networks. Not only that, but BGP has also been used as a substitute for IP based routing solutions in large networks like data centres (e.g. Facebook's data centres[56]).

While BGP has large benefits versus any kind of flat routing solution working directly on IP prefixes, it also has some clear problems and special peculiarities that affects its potential efficiency. First, while using ASNs to compute routes between ASs, removing the address by interface limitation within the BGP domain, at the end, eBGP requires

to store the mapping between ASs and IP prefixes. This is not a problem of BGP itself, but of the unique scope of the IP network layer and how IP addresses are assigned in the internal domains, meaning that BGP tables requires to store a constantly growing number of AS prefixes. This, as seen before, has been the cause of major Internet outages in the recent years, as some backbone routers were not prepared to accommodate that large number of AS prefixes.

In addition to the scalability problem, BGP also has a peculiarity related in how it works that limits its reaction speed upon failures and slightly increments its communication costs. In fact, while BGP is used to configure the network layer, it is a routing protocol that works on top of everything, like an application. While it may not seem a big problem, given its common running environment, the fact that BGP runs over TCP (specifically on the port 179) has, as stated, some inconvenience that would be solved if it was a protocol running directly over IP (or even on the link layer). For starters, BGP is the only routing protocol of its kind that runs over TCP. This provides it reliable connections between neighbours, ensuring that updates would be correctly delivered, but at the same time increases the size of such updates and limits its monitoring capabilities. In that regard, in order to be sure that the connection with a neighbour is alive, BGP use keep-alive messages sent every 60 seconds. While failures in a specific backbone link are not something common and keep-alive messages can be spaced in order to reduce the weight on the routing protocol, that also means that a new failure may take a whole minute to be notified, disconnecting all flows in the network using that specific link. And, while some approaches to reduce these reaction times have been taken into consideration (e.g. hardware capable of inform BGP of link failures), those are not widely implemented, nor always useful (e.g. its not useful to only know about outgoing link failures if there is more than one hop until the neighbour node).

3.3.2 MPLS

In contrast with the large networking domains, managed by multiple different entities (e.g. Internet backbone), in smaller networks it is possible to enforce a higher control in the network. Is in those networks that solutions like the Multiprotocol Label Switching (MPLS) [71] can be used in order to better optimize the usage of the networking resources and provide better services to its users. With the use of MPLS, network managers are capable of creating virtual circuits, introducing improvements in terms of:

- Traffic engineering
- Failure protection
- QoS support (e.g. voice vs. data)
- Service separation/private networks

- Etc.

As most solutions for the TCP/IP stack, MPLS does not take use of one of the pre-existing layers, but instead adds a new layer between the link and network layers. Within that layer, MPLS works adding an extra MPLS header, containing a stack of MPLS labels, as shown in Fig. 3.3. Those labels, of 4 bytes each, give information about the next router in the path, as well as allows to define a QoS class for the packets (using the QoS or Experimental bits). With the use of those labels, MPLS routers are able to forward the packets in the network in more complex ways, and being able to easily adapt to changes in requirements or network failures. While the full stack of tags could be set at the source of an MPLS path, that is not even required, as during the transport, MPLS routers can, not only pop labels from the stack, but also swap or push new labels into it.

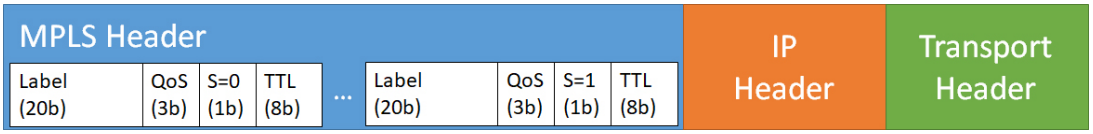

FIGURE 3.3: MPLS header.

While MPLS improves the management capabilities of the network and adds QoS support, this solution is far from perfect. Instead, it has some clear inconveniences (as always, some caused by the use of the TCP/IP stack itself). First, headers in MPLS have no fixed size, but depend on the depth of the stack, something that is not a problem on itself, but adds incertitude with respect to the MTU of paths in the MPLS networking domain (something far from expected near lower layers). Second, labelling decisions are done based on information stored on IP and transport layers headers. That is something that should be avoided in any solution as, not only breaches the separation between layers, but is unreliable as a solution, as it takes as granted knowledge about the upper layers (e.g. may work with TCP/UPD but fail with an unknown protocol, secure transmissions, etc.). This has a large impact in the provided QoS support, as it is based not in service requirements but mostly in the use of well-known ports in the transport layer to differentiate the kind of application using each flow (e.g. differentiate between voice-over-IP and "Internet" traffic.

3.3.3 Domain Name, URI and DNS

As the number of devices in the network started to grow, it became clear that remembering the address of services was not a viable solution. Given that, in the early days of ARPANET, host names started to be assigned to the different devices in the network. In order to have an updated mapping between host name and address, each computer in the network relied in a hosts file downloaded from a specific computer at the Stanford Research Institute (SRI). As the rapid growth of the network make it impossible to maintain

a unique centrally organized registry, ARPANET introduced in 1983 the Domain Name System (DNS)[72][73], a distributed register for mapping host names and node addresses.

As of today, host names have evolved to what is known as domain names. Domain names consist in one or more labels concatenated and delimited by dots (e.g. google.com, images.google.com, etc.). Those define a hierarchical, as seen in Fig. 3.4, being the right-most label what is known as top-level domain (e.g. google.com belong to the top-level domain com). Those top-level domains forms the DNS root zone [74] and are separated between country code top-level domains (ccTLD) and generic top-level domains (gTLD). While the number of top-level domains is limited by the ICAAN, that have been gradually growing, with more than 1000 gTLD as of 2016. Even so, special top-level domains, such as localhost or example, are left reserved for testing purposes.

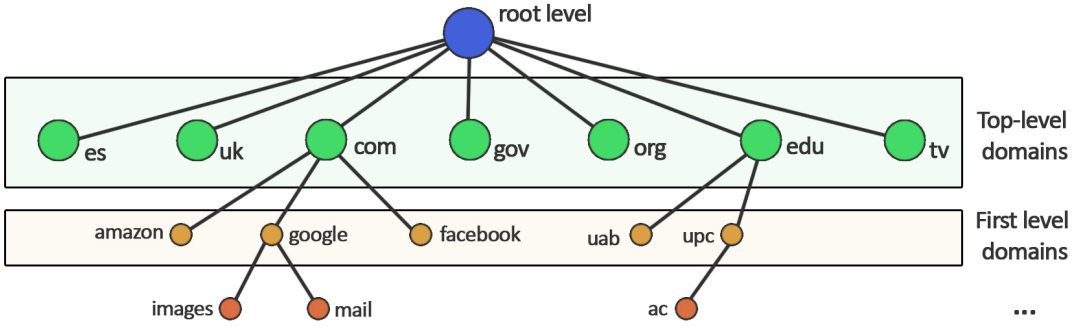

FIGURE 3.4: DNS Namespace Hierarchy

Complementing domain names, the Uniform Resource Identifier (URI) is used to access specific services and resources on the network. URIs are defined as follows:

scheme:[//[user[:password]]host[:port]][/path][?query][#fragment]

In there, the scheme indicates the application protocol in use (e.g. ftp or http), the host gives either the address or the domain name of the host hosting the service, the optional parameter port overwrites the well-known port used by the application (e.g. 8080 instead of 80 for http), and the rest of the parameters are used to set user credentials, define the accessed file, etc.

While DNS is used commonly only to locate hosts by name (either of general purpose or for mail servers, with Mail eXchange [75] entries), that is not its unique purpose. Instead, DNS can also be used to locate applications (e.g. html, ftp, smtp, etc.) Given that, DNS may act as an all purpose application-location dictionary, something that would reduce the use of well-known ports. While removing well-known ports this way would not reduce the vulnerabilities of public servers [76], as those would be accessible with a simple query, it would remove some configuration constrains on the server-side (e.g. being able to have multiple http servers accessible in the same machine, without specifying the port in the URI). Even so, this usage is something far from common and only used by specific

applications (e.g. the online game Minecraft allows the use of SRV DNS entries to specify the port where private game servers are located).

3.3.4 VPN and NAT

Between the fixed stack and the global IP layer, the TCP/IP stack encounter some problems both in terms of scalability and security. For example, being faithful to the fixed stack of TCP/IP, it would be impossible to interconnect nodes in distant private networks without either giving them public addresses (something that could create dangerous security breaches) or having a large link layer interconnecting the border routers of each network (something that could have a great impact in the cost of the network). The scalability problems of IPv4 would have had also a higher impact much earlier, as any device with access to the Internet would had required its own public address in the IPv4 addressing scheme, something that would require to IPS to provide wide ranges of addresses to its clients.

In order to interconnect distributed private networks securely through the public Internet, the use of a Virtual Private Network (VPN)[77] is one of the more common solutions. In a VPN network over the Internet, two networks are bridged by the means of either network or transport layer tunnels (e.g. IPsec or virtual routers). The use of those tunnels, not only allows interconnecting distant networks, but also serves to secure the communication, as most of those protocols provide encrypted communication by default. While the use of VPNs break the fixed stack of TCP/IP, in truth, their use shows a more recursive Internet, closer to the true form of Internet.

In contrast with VPNs, where the objective is to connect two networks, Network Address Translation (NAT)[77] is a security mechanism designed to limit the connectivity between two networks. Designed to allow access to public networks from private networks in a secure way, NAT hides a whole private network within one unique public address. At the border between both networks, NAT translate between private and public addresses based, commonly, on the used ports in the transport layer. Given that, it not only hides nodes in the private network (as those can only be reached from the exterior if a NAT entry is already in place), but also helps to the conservation of IP addresses, as only one public address is required for the whole network. In truth, NAT is probably one of the main causes why the IPv4 addressing space has not been depleted much earlier, given the fast increment in connected devices in households [78].

Currently, NAT is a necessary evil, even so, while NAT has helped to overcome one of the major problems of IPv4, the depletion of the addressing pool, with the future migration to IPv6, and its larger addressing scheme, its usage should be reduced (e.g. providing ranges of addresses to households instead of a unique address). NAT not only works by the means of taking information from multiple layers, but it also modifies the headers of both of them, “translating” IP addresses and ports at the will of intermediate nodes,

without informing the end-nodes of flows. This affects to a large number of applications, especially those based on peer-to-peer communication or voice-over-IP, as it is a common case for both nodes to be behind a NAT network, therefore being unable to directly open a communication. In those cases, different techniques of NAT traversal [79, 80, 81] need to be used, trying to automatically create NAT entries or predict the port translation in the NAT router (something not always possible).

3.3.5 LISP

By itself, the unique addressing space used in the current TCP/IP Internet results in large scalability problems. In addition, given the way how addresses are assigned and the design of the TCP/IP Internet protocol suite, this addressing space does not provide any distinction between the identity (name) and location (address) of nodes in the network. One of the more visible and notable results of this is the constant growth of routing tables in the DFZ, consequence of the increment on number of multi-homed services, lack of aggregability of addresses, etc. Solving part of these problems, the Locator/ID separation protocol (LISP)[82, 83] implements separation of the IP addressing space into two different addressing spaces: Endpoint Identifiers (EIDs) and Routing Locators (RLOCs).

LISP works in a similar way as VPNs does, bridging different distributed networks into a bigger one, but adding a dynamic component on that bridging, incrementing its scalability and manageability. Within a typical LISP environment, two different kinds of networks can be found: a bearer network (typically the Internet), providing global connectivity, and a distributed set of LISP networks. Within a LISP environment, each LISP networks should have assigned a unique prefix within the addressing space of EIDs, avoiding the common problem of current IP networks with unaggregable prefixes to the same node. That prefix would be used to identify all end-point nodes within that LISP network, having each a unique address within that network. Then, in the bearer network, RLOCs are used to locate the border routers to specific LISP networks (public IP addresses in the case of Internet). While this sounds quite similar to a large VPN comprising multiple distributed private networks, the trick of this solution is the fact that the interconnection between LISP networks is not done typically in a manual or static way, but in a more dynamic way.

LISP border routers use the Delegated Database Tree (DDT) [84], a DNS-like indexing system, that maps LISP prefixes to their RLOCs. With the use of the DDT, it is not required for each pair of networks in the same LISP environment to be constantly bridged between them, but, instead, this bridging can be done on demand, whenever a new flow between two of those LISP networks is required. The use of the DDT not only gives LISP a way to provide dynamically its VPN like connectivity, but it has other usages. For example, while the initial goal of LISP was to reduce the constrains that the unique addressing space and strict relation between name and address in the Internet inflict to the backbone network, the use of LISP provides a great solution for mobility, hiding the

movement of sub-networks or nodes from to the bearer network (e.g. host mobility in a data centre network with the data centre fabric as a bearer). It also allows for simple traffic-engineering in the network, being possible to, not only provide multiple RLOCs for the same LISP prefix, but to give different priorities to those, allowing for traffic to be distributed between the different RLOCs where a specific EID prefix can be located. These behaviours can be seen in Fig. 3.5, showing how the use of LISP can provide multi-homing solutions without increasing the complexity of the backbone network. In this example, communication between two different nodes under a LISP environment is performed, without any knowledge of it from the nodes itself, as the border routers of the LISP networks are the ones in charge of providing the encapsulation over the bearer backbone network. At the same time, as multi-homing is managed in an upper level, the backbone does not requires to manage the aggregation of extra prefixes, as those are outside of its reach.

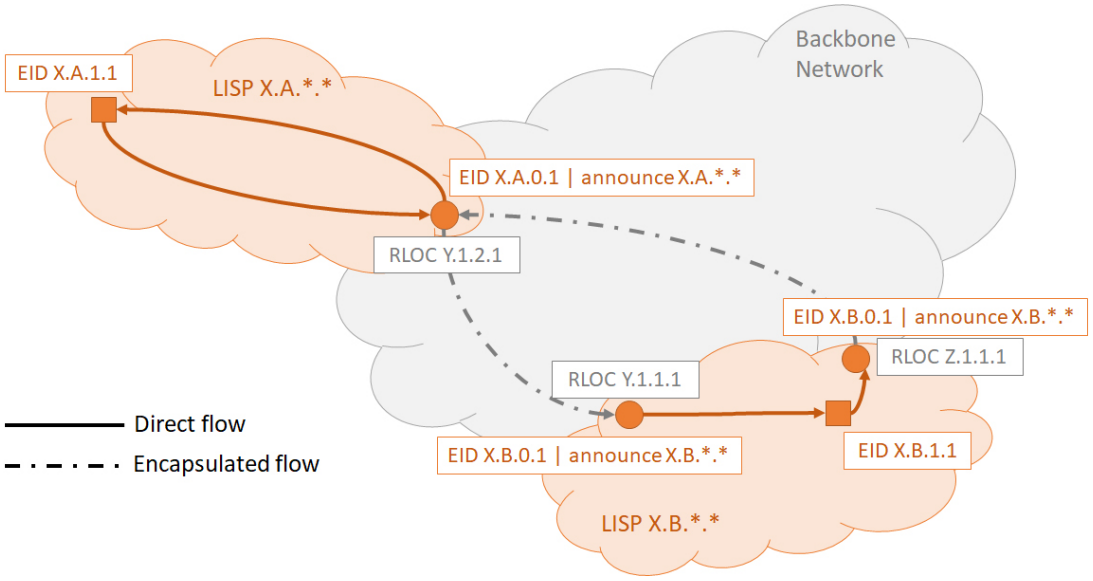

FIGURE 3.5: Example of simple LISP communication

In addition, while LISP by default creates a whole new layer on top of the bearer network, it is possible for hosts in a LISP environment to communicate with non-LISP hosts. In order to do that, it is required for LISP network to share a global prefix (or set of prefixes) between the EID and RLOC space, in order to avoid conflicts between addresses. For example, in a LISP over Internet environment, this would require for the LISP network to use only prefixes assigned to it by the ICAANA and private addresses (those not being accessible from the non-LISP hosts). First, for communication between a LISP host and a non-LISP host, upon reaching the border router, communication is done directly without encapsulating the packet, as the end-point is already at the designed address. On the other hand, for communication from a non-LISP host and a LISP host, communication is done by the ways of a LISP proxy, a gateway between non-LISP and LISP networks that would encapsulate the packets and send them to the appropriate LISP border router. This behaviour can be seen in Fig. 3.6, as opposed as that seen for pure LISP communication

seen in Fig. 3.5. In this example, a node within a LISP network communicates with one in the bearer network. As in the common case, both nodes do not need to consider the specific circumstances of the network, and are only the border routers and LISP proxy the ones in charge of accounting for the movement between networking domains. For packets coming from the LISP network, the border router knowing that the destination is outside the LISP environment, thus not requiring the encapsulation of the packets towards another LISP border router. And for packets coming from the bearer network, the LISP proxy knowing that the destination is in a LISP network, thus requiring for them to be encapsulated towards the nearer border router of that network.

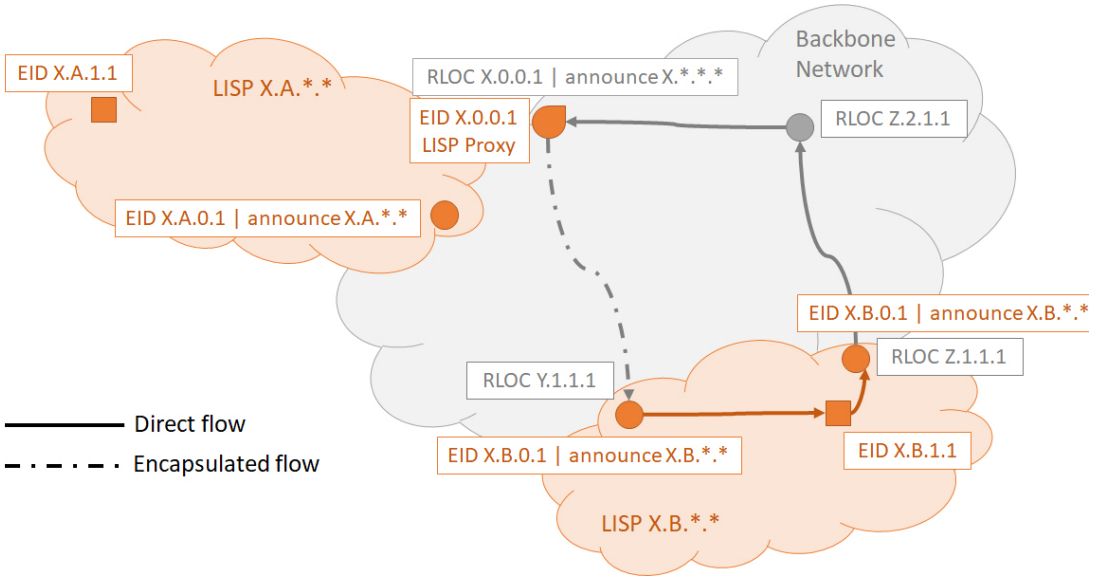

FIGURE 3.6: Example of LISP-bearer communication

Being border routers the main exceptions, LISP has the interesting property that it does not requires specific hardware designed for it. Instead, LISP takes profit from the already existing TCP/IP commodity hardware. As a bearer, LISP can use both IPv4 and IPv6 networks to bridge the different border routers (e.g. using the current IPv4 Internet). On top of that, LISP puts an IPv4 or IPv6 layer (or both), for which only border nodes know that they form part of a LISP environment and not a simple TCP/IP network. This makes of LISP a transparent solution to its users, being those not aware if in some part of the network it has been used.

While LISP, in the same way as simple VPNs does, breaks the fixed TCP/IP stack, adding new layers as required, it is a great improvement towards network scalability and mobility, something of great importance with the fast growth of cloud computing and the number of mobile devices. While still limited by the constrains imposed by the TCP/IP Internet protocol suite, LISP provides a clear view of how an scalable Internet should be, and the inherent recursively of networking domains (e.g. using LISP to connect distributed networks over Internet or using it to separate the connectivity within a data centre network from the public addresses of its hosts).

3.3.6 SDN and virtualization

Within networks, there are two distinct planes that work jointly in order to provide the desired communication services, namely, the data plane (or forwarding plane) and the control plane. In one hand, the data plane (strongly related to the hardware) focuses on making fast local decisions to provide the connectivity, forwarding, error detection, etc., between neighbouring nodes. On the other hand, the control plane, with a more global network view, is in charge of configuring the data plane to ensure that the requirements of services are eventually met for the supported flows.

In conventional networks, devices tend to be vertically integrated black boxes, where the control and data planes become tightly bounded into the same physical device. Networks typically are highly heterogeneous, consisting of multiple devices of different vendors, with distinct capabilities, performance, interfaces, etc. This heterogeneity, and the lack of common interfaces between devices, force networks to be configured, usually, in a manual way or using proprietary management systems. As a result, the overall management of a network composed by several devices becomes something not trivial and prone to errors. Aiming to solve the aforementioned issues, the paradigm of Programmable Networks (PN)[85] introduces a whole new family of network models that yield improved network control based on a clear separation between control and physical resources. This separation allows placing the control plane outside of devices, while maintaining the data plane as a black box. This delivers an abstraction of the forwarding capabilities, allowing for fast and automatic responses to changes on the communication requirements and network failures.

Among the family of PNs, Software-Defined Networking (SDN)[86, 87] has been gaining support in the latter years and is becoming the most accepted proposal. In SDN, the control plane is divided into the control layer (managing the data plane) and the application layer, containing the business logic (see Fig. 3.7). This separation of the control plane not only allows to more easily convey communication requests between SDN applications and the SDN controller(s) managing the network, but also permits a modular structure where changes on modules becomes greatly simplified (e.g., allowing the introduction of new upper-layer protocols). SDN provides an abstraction of both resources and functionalities of forwarding devices to service providers, allowing the centralization of the network control in the so-called SDN controller entity. This centralization not only allows an automated configuration of networks, removing the need of over-provisioning and making configuration less error prone to a great extent, but also paves the way to using cheaper forwarding devices by removing most of the control logic from them.

While the northbound API that connects the SDN controller with the application layer consists primarily in adhoc solutions (although efforts are being made to make it more standardized), for communication via the southbound interface there are only few protocols commonly used, such as OpenFlow[88]. OpenFlow, developed by the Open Networking Foundation (ONF)[89], is an industry standard for the southbound interface

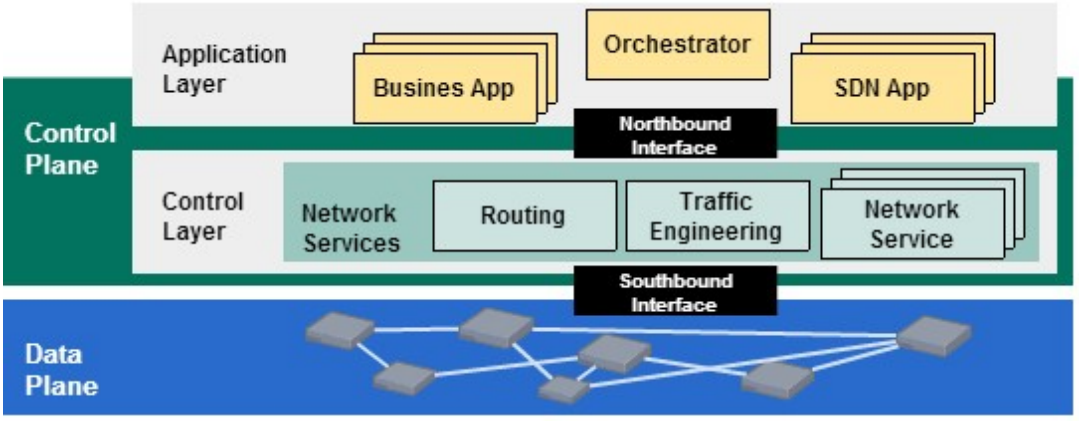

FIGURE 3.7: SDN architecture diagram

communication, considered to be the first and most well-known southbound interface for packet switching networks, providing a common interface used by most of big vendors (HP, IBM or Infinera are some of the vendors producing OpenFlow enabled devices).

In addition to the paradigm change that SDN proposes, towards a more abstracted network, in the latter years, the virtualization of networking resources has also been gaining the focus of attention. The virtualization of hardware (e.g. servers or network devices) consists in its emulation in software. Given the growth of network requirements and the increasingly good relation power/price of general purpose hardware, the use of virtual devices, with the ability to work in a similar way to traditional hardware solutions would, as increased considerably. Network virtualization (NV)[90, 91] abstracts the traditional hardware based networking into logical virtual networks, running on top of an independent physical network. In addition, NV does not provides only simple switching and routing solutions. In the form of Network functions virtualization (NFV)[92, 93], NV allows to add virtualized services to the network, like firewalls or load-balancing, helping to the centralization of the network management and allowing to provide network solutions on demand, without the need to alter the physical bearer network. This gives an alternative way to the design, deployment and management of networks, that works perfectly with the capabilities of network management provided by SDN.

Given the possibilities of centralized management and the virtualization of networking resources, the combination of SDN and NFV provides a new and dynamic environment where networking solutions can be deployed on demand of the requirements. In addition, as those solutions become more and more disjoint from the specific hardware, it greatly reduces the cost of introducing new solutions (e.g. related to traffic-engineering, routing scalability, etc.). Even so, SDN and NFV are not a panacea that would solve all the problems related to networking, but only an improvement that opens the path to a more scalable and cost-effective network. In fact, while SDN and NFV allows for new solutions in the network, at the end, the currently inevitable usage of the TCP/IP Internet protocol

suite at the top of the stack will still limit any possible solution, as it is not possible to know the requirements of flows traversing the network.

3.4 Chapter summary

This chapter does a small introduction to the TCP/IP Internet protocol suite and some of the major problems of this networking model as Internet architecture for nowadays requirements. The fixed stack of the TCP/IP Internet, alongside an incomplete naming scheme, is the cause of one of the major problems nowadays, the lack of scalability of the network. In addition, the current Internet lack of reliable means to differentiate the requirements of flows traversing it, something inherited from the lack of needs for such a reliable differentiation in the origins of the model.

After a quick analysis of the problems with the TCP/IP Internet protocol suite, some of the more common solutions for those problems are shown, most being patches that work breaking the fixed stack of TCP/IP, but always working within the TCP/IP Internet model (with the exception of SDN and NFV, only limited by the implementation). Network-centric solutions, like BGP or MPLS, that work towards providing the best service in specific environments. The implementation of a more complete naming scheme, with the implementation of domain names and the name mapping provided by DNS. Solutions like VPN, NAT or LISP, that generate new networking domains, or expand the existing ones, providing security at the same time that improve the global scalability of the network. And even solutions like SDN and NFV that propose a new paradigm for networking, moving away from the traditional device-centric network.

This recollection of problems and common solutions remarks that the core itself of the current Internet, the TCP/IP Internet protocol suite, does not represents the real environment of networking, and that it may be the moment for a big change.

Chapter 4

The Recursive InterNetwork Architecture

Citing the RFC 1958 [94], ≪The principle of constant change is perhaps the only principle of the Internet that should survive indefinitely≫. It is true that the current TCP/IP Internet model does great efforts to maintain itself afloat and adapt to the continuous increment in network sizes and changes in its usage. Even so, this model is based in strict principles that obstruct that same evolution necessary to adapt to evolving needs. This has resulted in a networking model that has been patched again and again, but still does not succeeds to accommodate the current needs. Given that, it may have arrived the time for a larger change, not in the model, but of the model itself.

The Recursive InterNetwork Architecture (RINA), presents itself as a potential replacement for the current TCP/IP Internet model. Based on John Day's definition of distributed networking as "IPC and only IPC"[7], RINA is a clean-slate architecture for computer networking that aims to correct all the design flaws of the TCP/IP Internet protocol suite, and create a new network capable of evolving and adapt to any changes. Proposed by John Day and Pouzin Society[95], RINA is an alternative Internet model that changes the idea of networking layers. While the current TCP/IP Internet or the OSI model assign layers based on specific IPC functionalities on specific networking domains, RINA swaps this to a more natural way, defining layers based on the logical networking domains (e.g. similar to how LISP put extra layers above the backbone, MPLS manages physical sub-networks, etc.). In RINA, layers are called Distributed IPC Facilities (DIFs). Those DIFs share all the same base design, centred in the idea of providing distributed IPC communication between IPC processes (IPCPs) within a networking domain. Unlike layers in functionality-based models (e.g. TCP/IP and OSI), DIF are designed as programmable layers capable of performing any of the functions needed to provide IPC services to applications or higher level DIFs, implementing all the same set of functions and mechanisms. This provides a unique and common Application Programming Interface (API) at all layers, providing a great amount of freedom to network designers, as

those are now able to stack the different layers as required, something that marks the recursive nature of RINA.

While all DIFs share the same set of mechanisms, that does not means that all are the same (as would happen when extending the TCP/IP stack with extra IP layers). Instead, DIFs are fully configurable by the means of policies. Protocols and tasks in a DIF can be adapted with the use of different policies in order to get the best performance for each networking scope. RINA also provides full QoS support, by the mean of QoS Cubes, namely sets of QoS requirements such as maximum latency and jitter, losses, data rate, burst rate, etc. that a DIF can support. As for policies, those QoS cubes can be configured specifically for each DIF, and their specifications shared with other DIFs through the means of the common API. The recursive nature of RINA, configurable policies and QoS support, all merges to provide multiple benefits with respect to the current functionality-based TCP/IP model. For starters, given that recursivity is an inherent part of the model, it becomes easier to divide the network based on smaller networking domains (e.g. like LISP proposes in IP networks), reducing the size and complexity near the core of the network. In addition, given the common API between DIFs, not only those are easily stacked, but it is also possible to request specific flow requirements and query the capabilities of bearer DIFs. This allows for more informed and automated solutions for network management, less error-prone and easy adaptable to dynamic changes in the network than traditional management (similar to what SDN propose, but with a supportive networking model behind). Finally, while its recursively allows to freely create new layers, it also allows to reduce the number of networking domains whenever those are repeated only to add functionalities. For example, while a TCP/IP network has both network and transport layers, those are in truth the same networking domain, only separated to provide different transport capabilities to the IP flows (e.g. connectionless with UDP or connections with TCP). Instead, in RINA those two layers could be merged within a unique DIF, and simply provide the differentiated transport by their QoS Cube.

As a note, RINA uses the same nomenclature described in Chapter 2 for general networking architectures. In that regard, this thesis refers indistinctly between packet and protocol data unit (PDU), and between data and service data unit (SDU). In the same way, it follows the same definition of protocol, task, mechanism and policy as stated previously.

Fig. 4.1 shows a simple comparison between the TCP/IP and OSI stacks and the RINA stack. When comparing those stacks side by side, it is clear the difference between functionality-based stacks (e.g. TCP/IP and OSI), where each layer is defined by a specific set of functionalities, which gives it its name, and a networking domain based stack (e.g. RINA) where layers are defined by their networking domain. In addition to the common DIFs, two special layers can be seen on the top and bottom of the RINA stack. On top of the communication, the Distributed Application Facilities (DAFs). DAFs are a special case of DIF connecting not simple IPCPs, but applications with

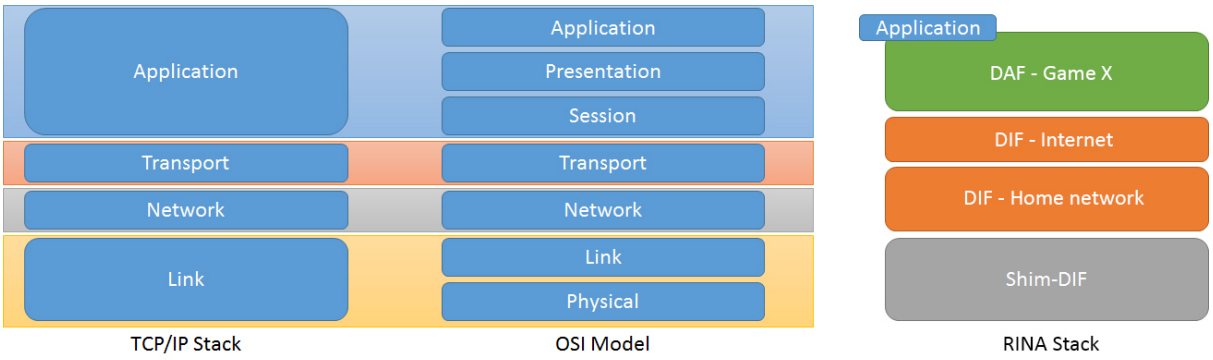

FIGURE 4.1: Comparison of TCP/IP, OSI and RINA stacks

IPC functionality. Unlike DIFs, DAFs does not provide IPC functionalities to upper processes, being only a collection of Distributed Application Processes (DAPs). On the opposite side, the shim-DIFs, namely "wrapper" DIFs used to provide a RINA API for a specific non-RINA technology[96]. As opposed to common DIFs, shim-DIFs have limited functionality, dependent on the bearer technology, and commonly are used to provide a unique point-to-point flow between neighbouring nodes (e.g. an Ethernet shim-DIF connecting two routers).

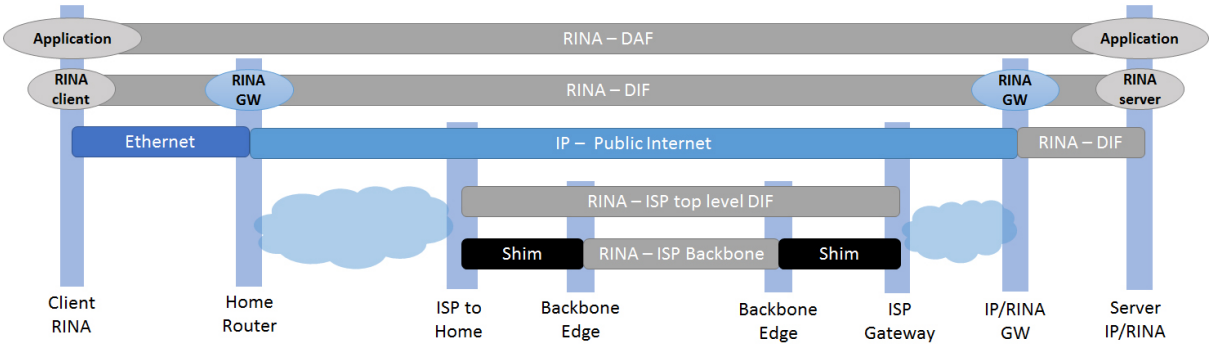

FIGURE 4.2: Mixed RINA/IP scenario

Despite the clear differences between RINA and the TCP/IP model, both are compatible enough to allow for a progressive migration between them at little expense. Indeed, the migration towards RINA does not require turning off the entire Internet overnight to be deployed, in a similar fashion as did the Flag Day for TCP[23]. Conversely, it can start either by replacing lower layers with RINA, while keeping IP services on top, or by using any existing network protocol (IP, Ethernet, UDP, WDN, etc.) as a bearer for RINA in the form of specific shim-DIFs. Fig. 4.2 shows a simplified example of how RINA could be introduced as a networking model within an ISP as a backbone network. This has multiple benefits, providing an improved management of the backbone resources, similar to what solutions like MPLS could enable. In addition, it could be used to easily differentiate other services that could be running over that same backbone (e.g. video streaming services, voice calls, etc.), and would prepare the ISP to provide true RINA services to its clients. On the other hand, that same IP Internet network could be used as

a bearer for RINA, in a similar way that VPNs or LISP networks would use the existing IP network to bridge distributed networking domains.

4.1 QoS Cubes

As stated before, RINA provide a complete support for QoS in its DIFs, by the mean of the QoS Cubes, namely QoS classes for flows providing statistical bounds on metrics like data rate, latency, losses, and so on. While all DIFs provide the same kind of service to upper processes, the characteristics of the offered service may vary between DIFs, thus resulting in different DIFs supporting different set of QoS Cubes. Given those QoS Cubes, applications are able to request specific bounds in the experienced end-to-end QoS of flows, for which the DIF will automatically assign the QoS cube that better suits them. In addition, given the recursivity of DIFs, upper IPC processes (IPCP) are also capable of requesting specific QoS levels for flows, making it easier for them to ensure their own QoS Cubes.

QoS Cubes provide statistical bounds on different metrics related to the quality of flows, during normal utilization (e.g. 99.99% of time), being the most important between those the following:

- Maximum delay. Maximum delay that a PDU can suffer.
- Maximum jitter. Maximum variation of delay between close PDUs.
- Maximum PDU loss. Maximum % of lost PDUs.

In addition to the quality assurance, QoS Cubes may also define part of the "contract" that the flow source has to enforce. This include different parameters like:

- Maximum bandwidth usage. Limiting the data rate on mid-term basis.
- Maximum burst allowed and burst duration. Limiting the data rate on short-term basis.
- Maximum SDU size. Limiting the size of data packets.

QoS Cubes, as for the rest of RINA API, have the same format for all DIFs, allowing upper IPC processes and applications to know what to expect in the worst case from the flows provided by underlying DIFs without needing to know how the lower DIFs are managed. While QoS Cubes provide useful information of the expected service of flows, that is not the only use of QoS Cubes, but simply a useful by-product for network management and applications to improve their services. Instead, the importance of QoS

Cubes comes from the ability to request explicit QoS requirements for flows. Unless QoS Cubes, QoS requirements are not a fixed set within a DIF, but may vary depending the current needs. In order to ensure its own QoS guarantees for upper processes, a DIF will require specific requirements for each of its lower flows (e.g. low bandwidth requirements at the edge of the network but higher requirements on the central part). As the usage from upper flows changes, requirements to maintain the QoS guarantees may also changes and IPCPs can request the replacement of lower flows with those new requirements. Being more diverse than QoS Cubes, processes may choose their requirements between a large set of parameters, like:

- Accepted delay. Requirements on the maximum allowed delay for correct usage.
- Accepted jitter. Requirements on the maximum allowed jitter for correct usage.
- Accepted PDU loss. Requirements on the maximum accepted losses for correct usage.
- Average/Maximum bandwidth. Information about the expected bandwidth usage.
- Maximum burst size and duration. Information about the expected burstiness of the flows.
- Require in order. Requirement to get all PDUs in order or not.
- Accepted gap. Allowed gap between received PDUs (0 if all PDUs are required).

While some of those QoS Requirements are directly translated into the best QoS Cube to ensure them, what those provide is information for the IPCPs to ensure the flow needs within the DIF. For example, if a flow requires low delay and no losses (e.g. real-time stock market), the IPCP will try to put the flow into a QoS Cube with the higher requirements in delay and losses, in order to avoid lost PDUs that will need to be retransmitted (while enabling retransmission just in case). On the other hand, if the only requirement is having no losses (e.g. file-transfer), the IPCP will put the flow, in most cases, in a QoS with lower requirements for losses, as the added delay of retransmission is not an issue. On the opposite side, if a flow only requires low delay but accepts losses (e.g. VoIP-like service), the IPCP will put it instead in a QoS Cube with low delay assurance, but it will probably not enable the retransmission of lost PDUs. In resume, QoS Requirements says what the flow owner needs, but how that is ensured is up only to the DIF. As for applications, this also changes how users see the network, as those do not need to know what kind of transport protocol is required for its service, but only what kind of service it expect to get from the network. This effectively reverse the roles of application and network, as it is not the user who decides "how" (e.g. using TCP vs UDP) and the network provide some service that way, but the user decides "what" and is then the network who decides "how" to reach that service.

4.2 Protocols and Tasks

A RINA stack, as said before, is formed by repeating once and again the same type of layer (DIF) as required by network managers. While these DIFs are all capable of performing all networking functions required of IPC and the freedom to stack them provides interesting benefits with respect to functionality-based models like TCP/IP, it is clear that, if all DIFs behaved the same, it would still impose a large constrain in order to adapt to the specific requirements of each networking domain. In order to avoid that, RINA DIFs can be configured via policies, namely programmable behaviours that defines how their protocols and tasks will work.

As can be seen in Fig. 4.3, the different mechanism in a RINA IPCP can be divided given their requirements of fast response and how many times are those called. From those, three main groups can be extracted, with tasks related to data-transfer being called with maximum priority in a per-PDU-basis, data-transfer control tasks being also priority, but secondary to data-transfer, and DIF management tasks as the slower/long-term tasks.

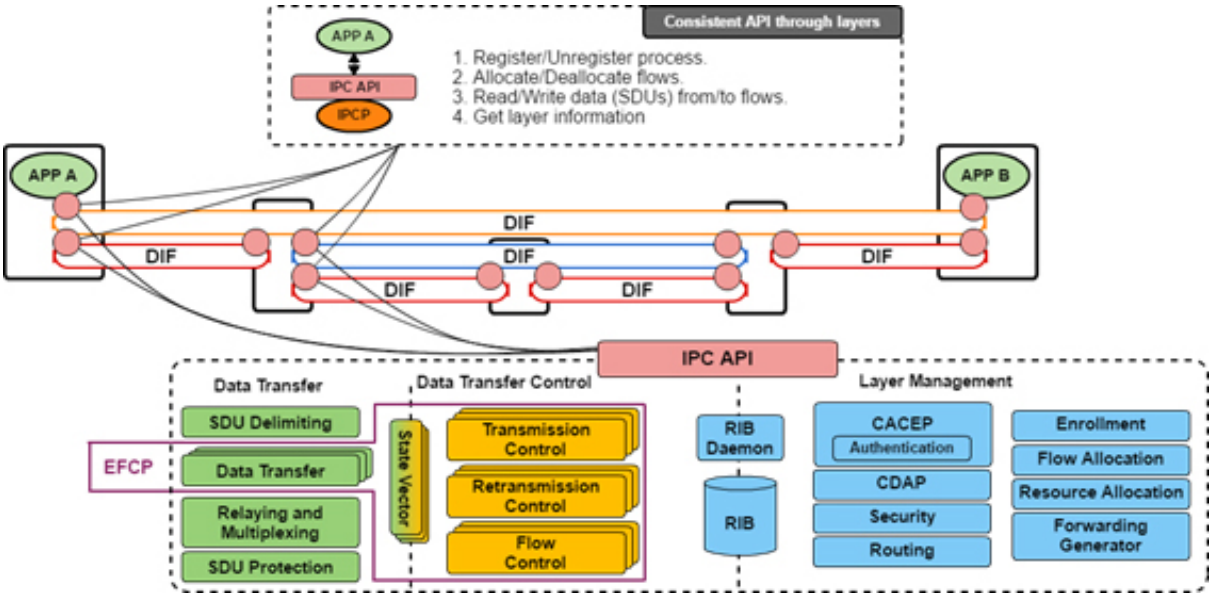

FIGURE 4.3: RINA IPCP's Architecture

Comparing the RINA IPCP's architecture to that of the TCP/IP stack, there are some similarities that remark the difference between both models. First, like in the Network layer, RINA's data-transfer tasks focus on the forwarding of data through the network (the DIF in this case), this being done as fast as possible in a per-packet basis. Then, like in the Transport layer, RINA's data-transfer control tasks manage the different flows, avoiding congestion in the network and, if required, ensuring that all data is received. Finally, the management tasks span what currently is performed by multiple protocols controlling the network, like routing, security measures or resource allocation.

While performing similar task as TCP/IP, in contrast to it, RINA does not bases its configuration in the use of specific protocols for each scenario, but focuses only in a small

set of configurable protocols. In fact, all DIFs are composed of only two protocols: a data transfer protocol called Error and Flow Control Protocol (EFCP) and an object-oriented application protocol called Common Distributed Application Protocol (CDAP) that carries all the information exchanged by DIF management tasks (usually known as control plane in TCP/IP terms). Both protocols can be adapted to the different requirements of each DIF via policies [97]. And, not only protocols, but multiple mechanisms of tasks can also be configured with policies, being an example of that the scheduling and forwarding functions.

4.2.1 Network Management System and CDAP

Within a RINA network, the creation and management of DIFs is performed by the Network Management System (NMS), a distributed application composed of one or more Manager(s), together with the Management Agents (MA) located at each RINA device. The NMS is responsible of the creation of IPCPs in RINA devices, as well as their configuration and enrolment in their DIF. The NMS manages many key functions related to the configuration of IPCPs and their policies, like neighbour discovery, flow allocation, resource monitoring or centralized management.

Given its distributed approach, NMS are capable to provide different configuration solutions, ranging from a fully distributed one, where each devices' MA acts as a network manager, to a fully centralized one, where an extern manager manages the entire network with MAs acting as no more than actors to its decisions. In addition, unlike IPCP tasks and policies, NMSs are not bound to a unique DIF, but can also be used to manage multiple levels of DIFs if required. Furthermore, different NMSs can interact in the management of DIFs. This opens an environment for dynamic and scalable network management (similar to that provided with solutions like SDN[86, 87] in the TCP/IP model), in conjunction of a common and exhaustive API that limits the risk of human failures and misconfigurations.

Related to the management of the DIF, the Resource Information Base (RIB) is a database of relevant information present in each IPCP. The RIB is used to share information between the different policies on the IPCP, the Manager Agent and other IPCPs in the DIF. In addition to simpler information lookups, the RIB also allows for policies to subscribe to certain information, allowing for policies to work with both a request/respond behaviour or subscribe/publish behaviour. In order to exchange information between the different IPCPs and managers in the network, RINA uses a common communication protocol, the Common Distributed Application Protocol (CDAP). As for the RIB, this protocol allows for both request/response and subscribe/publish communication modes in order to allow interaction between IPCPs and share relevant information between peers. CDAP defines 6 distributed operations that can be performed to a set of object in the RIB; create/delete, start/stop and read/write. Using those simple operations, manager agents and the different policies in the IPCPs can do both, share relevant

information and perform request operations, like request the allocation of new flows, share policy updates between peers, manage DIF enrolment, etc. CDAP works specifying the data model for each task as objects, providing a naming scheme and a set of callbacks that are executed on layer management tasks when a particular action on an object is invoked remotely. This programmability allows network administrators to properly configure each DIF with the policies that better adapt to its scope, operating environment and offered levels of service.

As it is a protocol already within the IPCP, CDAP takes profit from the capabilities of the DIF and serves to abstract how information between processes is really shared. This allows simplifying distributed policies (e.g. routing policies), as now, those do not need to consider how to share information. This contrast with how distributed protocols work with the current TCP/IP Internet protocol suite, as those need to either manage themselves all its communication or rely it to another protocol. The BGP routing protocol[66] is a clear example of this, as, this routing protocol used to configure IP forwarding tables on backbone routers, use a TCP connection to both share information between connected peers and monitor that connectivity, meaning that it is a protocol over the Transport layer who manages the Network layer. As opposed, a similar routing policy in RINA would run within the same DIF, with CDAP managing the communication between peers as well as connectivity being monitored by the manager agent.

4.2.1.1 Flow allocation

Between the different management tasks of the MA, perhaps the most important would be the allocation and supervision of flows in de DIF. Within the DIF management tasks, the Flow Allocator (FA) is responsible of the allocation and management of flows between IPCPs. When allocating a new flow, the FA is responsible of selecting the appropriate QoS Cube for its requirements (potentially considering also the DIF status or the flow's destination) and allocating the required EFCP instances in both extremes of the flow. In a similar way, it is responsible of reallocating flows (e.g. replace its QoS Cube) or remove them when those are not needed anymore. Decisions taken by the FA will also be informed to any potentially affected policies by the means of the RIB and CDAP. This allows for example to seamlessly add pseudo-connection-oriented behaviours by the mean of routing policies subscribed to FA updates (e.g. a distributed routing policy that use knowledge of flows to pre-compute forwarding paths for priority flows), or scheduling policies auto-configurable with knowledge of the flows traversing the node.

In contrast with the TCP/IP architecture, where applications listen to specific ports for flows, RINA takes a different approach where processes are access by name. Fig. 4.4 shows the general workflow for flow allocation in RINA. First, in a similar way as how IP addresses are commonly queried in the Internet, the MA in the source node queries the Namespace Manager (NSM) for the address of a reachable node containing an instance of the required process. This call can be completed within the same node if that entry is

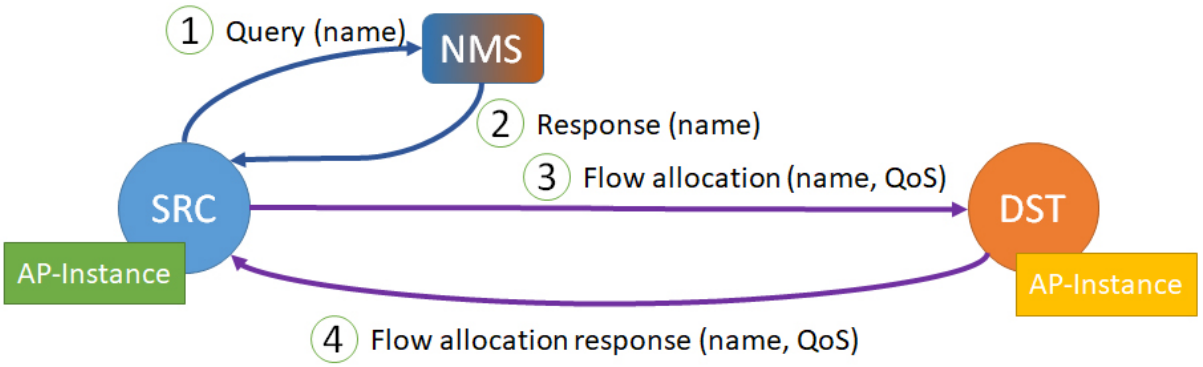

FIGURE 4.4: RINA IPCP's Architecture

already within the RIB (from a previous call, pre-configured, etc.) or in a distributed way (with a DNS like system), but, if successful, a list of DIFs and node addresses containing instances of the desired process would be obtained. At this point, the MA decides for the best destination containing the desired process and try to allocate a new flow towards it.

In order to allocate a flow between two nodes, the FA requires from both to be in the same DIF (a natural requirement) and for the destination to be reachable. In order to decide if a destination is reachable, the FA can make multiple considerations, all depending on the current DIF. While a simple policy could directly try to reach that destination, more complex ones may consider the remaining amount of resources on lower flows to that destination or even request multiple flows to lower DIFs in order to reach that destination. RINA EFCP is based on R. Watson's Δt [98], a communication protocol that bases on timers, rather than synchronization messages, in order to manage connections between IPCPs. With Δt, RINA assumes that all connections between IPCPs always exists, meaning that the FA can easily manage the EFCP instances associated to them, for example, removing EFCP instances of idle flows, replacing old instances with new ones whenever a rollout on sequence number is close, etc.

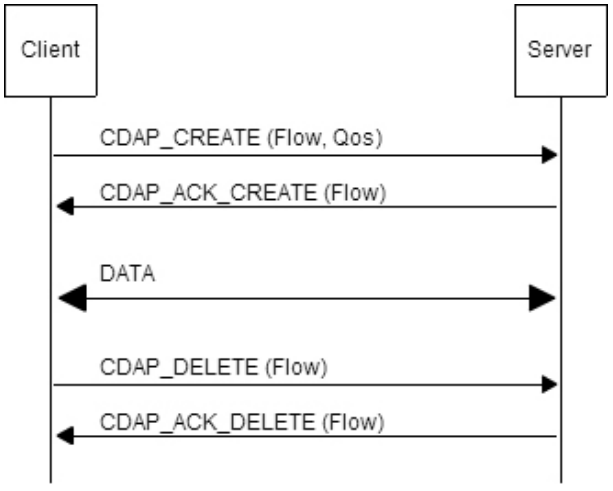

FIGURE 4.5: Simplified fow lifespan's diagram

As new flows need to be requested and allocated before being usable (EFCP instances need to be created), it becomes impossible to randomly search for open ports in a destination nodes. This provides a great improve in security with respect to the TCP/IP Internet, being that the source of one of the most common attacks in those networks (either by using well-known ports or by trying to guess them). In addition, the extra control that this kind of flow enrolment provides, as well as the recursive nature of RINA, provide an enhanced wall toward other types of attacks, like denial of service attacks trying to overflow the network.

Additionally, it is important to remark that, when a FA receives a new flow request, it can decide to which instance of the destination process send it (in case of having multiple instances available) or even request the creation of a new instance of the destination process. This has great importance and shows the benefits of the separation of process name and address in RINA, easily allowing for dynamic environments with replicated services, something of great importance nowadays with the extended use of cloud based services, but that, with the TCP/IP Internet, can only be reached by means of patches like the use of modified DNS servers for load balancing [99].

4.2.2 Error and Flow Control Protocol

Within RINA, the Error and Flow Control Protocol (EFCP) encompasses most of the task related to data-transfer and data-transfer control. The EFCP is composed by two sub-protocols, the Data Transfer Protocol (DTP) and the Data Transfer Control Protocol (DTCP), connected through the use of a common state vector for each flow. Modelled after Richard Watson's Δt transmission protocol[98], EFCP manages all tasks related to ensure reliability and flow control within the DIF. With the use of Δt, the EFCP is capable of managing reliable flows without requiring of the use synchronization messages (e.g. SYN). Instead Δt bases on the use of different timers (RTT, retransmission and ACK time) in order to manage the lifespan of a flow status, ensuring that the state vector of a flow will be maintained while data is in transit, deleted if there is no data being transmitted and re-created again when the flow re-starts transmitting data.

When a flow (src $\rightarrow$ dst) is allocated, an associated EFCP instance is created to manage the flow during its lifespan. With this EFCP instance, a DTP instance performs the functions directly related to the transport of SDUs (including sequencing, fragmentation and reassembly or SDU protection) and flow control. In addition, in case that the flow requires it (e.g. flow requires retransmission on lost PDUs), a DTCP instance would be in charge of managing all transmission and retransmission control for the flow. This DTCP instance will then use the shared state vector (written by the DTP instance) in order decide when PDUs should be retransmitted, manage buffers and windows, decide when feedback (e.g. ACK) must be sent, etc.

While being the unique transport protocol in RINA, EFCP is highly configurable. This include EFCP policies for multiple different sub-tasks as manage timers (in both sending and reception), congestion control and notification, sending rates, RTT estimation, etc. This configurability, and the recursivity of RINA, allows for configurations specifically tailored to each networking domain, no matter its scale or usage (e.g. a data-centre backbone would differ from a home network). In addition, having a unique data-transfer protocol in the whole DIF has some benefits with respect to the approach taken in the TCP/IP Internet protocol suite. The most important point in this regard would be that, instead of having a heterogeneous environment (with multiple transport protocols), all flows are managed following the same rules. This avoid incompatibility problems like the ones caused between TCP (with congestion control) and UDP (without congestion control). Related to this, having a unique data-transfer protocol also allows for more reliable congestion notification mechanisms, as all IPCPs in the DIF interpret those notifications in the same way (e.g. early notification vs. notification after losses).

4.2.2.1 Relay and Multiplexing Task

As its name suggest, the Relay and Multiplexing Task (RMT) takes the role of relaying and multiplexing SDUs and PDUs to upper EFCP instances and lower flows respectively. The RMT is a fast task that works in a per-PDU basis and is the main responsible of the transmigration of data between source and destination. There are multiple policies related to this task, the more important being those related to forwarding and scheduling.

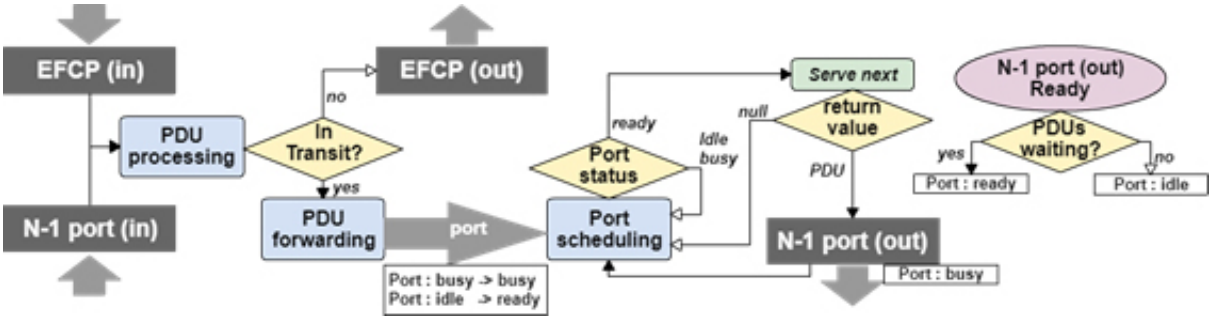

FIGURE 4.6: RMT workflow

In order to understand how the different policies related to the RMT interact, it is important to know how the RMT work. Fig. 4.6 shows a simplified view of the RMT workflow, from the arrival of a PDU from a RMT port, linked to an N-1 flow, or from an upper EFCP instance to its departure towards an EFCP instance or RMT port. On arrival, a PDU first is processed in order to check it has already reached its destination, in that case being directly forwarded to the EFCP instance towards its directed, or, if it is still in transit, being forwarded to the forwarding policy.

The forwarding policy then decides the next N-1 port(s) towards the PDU has to be relayed. At this point, a copy of the PDU is processed for each output port. Depending

on the IPCP configuration, at this point, if the port is idle, the PDU could be directly relayed to it. Otherwise, the PDU is forwarded to the scheduling policy. Then, whenever the port is ready, and if scheduling policy has PDUs to send, the scheduling policy is called and the next PDU to send is selected and relayed to the N-1 output port (or the call is re-scheduled for a latter iteration if the policy requires it). In addition to deciding the order in which PDUs have to be served, the scheduling policy also decides when is required to drop a PDU (e.g. because congestion) or when is required to inform of congestion.

4.2.2.1.1 Scheduling policy

In any network susceptible to congestion, how data is treated during those stages of congestion is key to provide QoS guarantees to flows. As for other policies, RINA do not enforce a specific way on how scheduling policies should work, but only a few hooks for the different events related to the scheduling of PDUs:

- PDU arrived: return void

  A function called whenever a new PDU waits to be forwarded to a neighbouring IPCP through an RMT port. It takes as a parameter a reference to the specific PDU and the destination RMT port.

- Port ready: return PDU/nil

  A function called whenever a RMT port is idle but still with PDUs waiting to be served. It takes as a parameter the idle RMT port and requesting the next PDU to serve. Its call has two possible outcomes, return the next PDU, in which case the RMT port will start processing the returned PDU, or return nothing, in which case the function will be called again in the next execution of the RMT.

This freedom allows for slightly more complex solutions that simple queue based ones (FIFO, Weighted Fair Queue, etc.) where extra information can be considered at the time of performing scheduling decisions. In addition, given that scheduling policies are not even forced to process PDUs whenever the port is ready (work conservation not enforced), scheduling policies can enforce extra control over the flows traversing each IPCP. All this allows for the usage of scheduling solutions tightly adapted to each scenario, providing this way improved QoS assurances for flows.

Chapter 5 considers the implications of scheduling policies in the assurance of QoS requirements, at the same time that introduces a ΔQ-based [100, 101, 102] approach for scheduling within RINA networks.

4.2.2.2 DTCP and Flow Control

While the RMT and the scheduling task manage short-term congestion as part of the DTP functionality, long term congestion (and how to avoid it) is managed primarily by the DTCP. DTCP has multiple functionalities, including:

- Maintaining the order of PDUs.
- Retransmission of lost PDUs.
- Traffic shaping (either enforcing a maximum rate or a specific pattern).
- Congestion control/avoidance.

Depending on the QoS requirements, assigned QoS Cube of the flow and the capabilities of the DIF itself (or the flow's path), DTCP can be configured performing any subset of its functions (e.g. order without retransmission for VoIP-like flows, ordered with retransmission and maximum rate, etc.). While it is not even required for all flows to use DTCP within its EFCP instance (only DTP is required), it is preferred to have a minimum configuration with congestion control active for all flows, as this provide an improved control of the resources of the network. This remarks one of the primary benefits of RINA, its recursively, as this allows to perform flow control at any layer, approaching the solutions to the focus of the problems.

Having a congestion control mechanism is a requirement whenever it is possible to oversaturate the network. In the current TCP/IP Internet model, transport protocols like TCP are the ones in charge of managing that. The use of those protocols, while achieve their objective, have multiple problems that avoids them to produce optimal results, some of those listed below:

- Shared network with non-controlled protocols (e.g. UDP).
- End-to-end control may locate the control mechanism far from the focus of congestion.
- Congestion decided by drops/out of order do not consider losses in the medium.

In contrast with TCP/IP, as RINA has a unique protocol in charge of data-transfer, the EFCP, instead of multiple transport protocols the first problematic point is already solved whenever the DIF is reliable (there are no malicious IPCPs), something controlled to a lower or higher degree at enrolment. With regards to the second point, the inherent recursivity of RINA already solves it to a high degree, allowing lower DIFs to manage congestion wherever it occur, and recursively inform of that to upper DIFs when needed. As opposed to the first two points, the third is one more complex to solve and highly

dependent on the policies in use at each DIF. While not commonly used, TCP/IP already has a method for informing of congestion before dropping packets, the Explicit Congestion Notification (ECN) [21]. In RINA, a similar approach is taken where a congested IPCP (commonly at the scheduling policy) can mark congestion in a PDU with ECN bits or even signalize directly congestion to the flow source.

In addition to flow control, the use of DTCP and the recursively of RINA also provide interesting benefits to the current Internet, and its increasing wireless structure. The reason for this is that, while TCP/IP was designed for a more reliable environment, where congestion in the network was the primary cause of losses, the current Internet has an ever increasing number of wireless devices (laptops, smart-phones, sensors, etc.), communicating through unreliable mediums. In those networks, losses due to the nature of the medium are common, but TCP/IP does not only force the retransmission of the lost packets, but consider them to be caused by congestion, although solutions are being considered (e.g. coded TCP [103]). In contrast, this is no problem in RINA. First DTCP is not designed to consider all losses as congestion (although that can be defined by policy). Secondly, and most important, the recursive nature of RINA allows to retransmit lost PDUs at the extremes themselves of unreliable links, simply having the required configuration for DTCP in those flows. This not only avoids end-to-end retransmission of the lost data, but effectively increments the average bandwidth of flows (both thanks to avoid the end-to-end retransmission and having no effect on flow control in upper layers), but most important, reduces the perceived latency for the lost PDUs.

4.2.3 Forwarding and Routing policies

The role of the forwarding policy within a DIF is simple, whenever the RMT has to process a PDU not directed to itself (or multi-casted also to other nodes) it has to decide to which(s) neighbouring IPCP forward it and how. This policy is called almost in a per-PDU basis, as part of the RMT execution when receiving a new PDU (only exception if the IPCP is the destination), meaning that it has to be processes as fast as possible to avoid bottlenecks, something that may be difficult to avoid in large DIFs (e.g. data centre networks).

Unless common forwarding table solutions for the TCP/IP Internet, forwarding in RINA not limited to the use of variations of forwarding tables, but, instead, any valid forwarding function can be used as a policy. Forwarding policies can take different information from the network status and PDU headers to perform its decisions in more complex ways. This opens the path to new and interesting forwarding solutions capable of taking profit from the different topological properties of networks. This, in conjunction with an addressing scheme adapted to the network, is one of the main points that help to improve the scalability of DIFs and avoid bottlenecks on forwarding functions when working with inevitably large DIFs.

Complementing the fast-paced Forwarding policy, the slower paced Routing and Forwarding Generator policies are in charge of provide the correct configuration for forwarding given the network state. As those policies are not limited by the fast pace of a per-PDU function, complex solutions can be considered, resulting then in the possibility of using more optimal and scalable Forwarding policies.

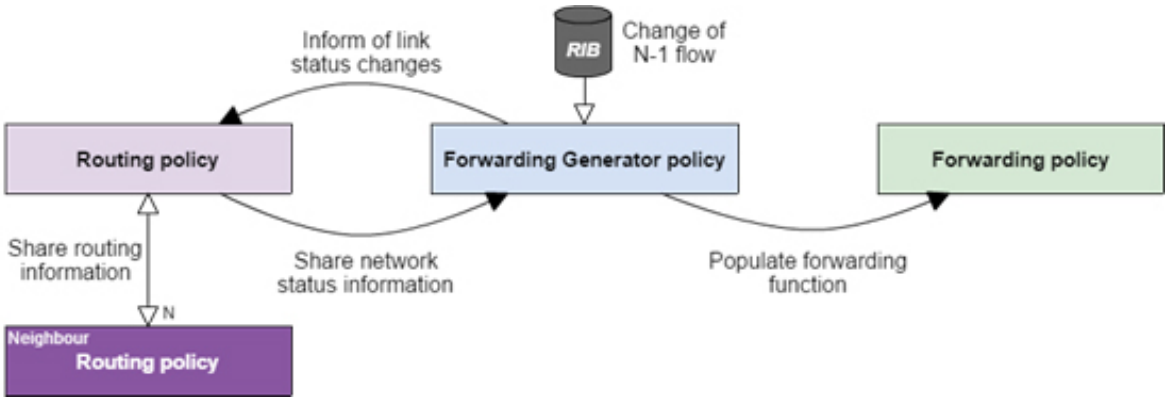

FIGURE 4.7: Workflow on network status changes

As can be seen in Fig. 4.7, the workflow upon changes in the network status is divided between changes in the own IPCP connectivity, managed by the Forwarding Generator policy, and changes in the general connectivity of the DIF (managed by the Routing policy). First, the Forwarding Generator is in charge of monitoring the IPCP connectivity and configure the Forwarding policy upon changes in the network connectivity. Upon changes in the own connectivity, the Forwarding Generator informs to the Routing policy, and, if a N-1 flow was lost, it re-configures the Forwarding policy to avoid the lost flow. The Routing policy, for its part, is responsible of receiving network updates from both the Forwarding Generator policy and neighbouring IPCPs' Routing policies, compute the new network state and inform the Forwarding Generator policy of the IPCP of any change. With those change, the Forwarding Generator then is responsible of re-configure the Forwarding policy to adapt the new network status.

Given this division between tasks, network managers can configure DIFs in a more flexible ways, adapting to the specifics of the network. For example, a small DIF could use a simple link-state Routing policy in conjunction with a Forwarding Generator policy that use an entry per destination. As the network grows, the Routing policy could be changed to a distance-vector one, without requiring changes in the Forwarding Generator. At the same time, it could be possible for some IPCPs to be connected through multiple N-1 flows, ensuring different QoS guarantees. In those cases the Forwarding Generator policy could also be replaced with one that take in consideration the QoS guarantees of lower flows as the DIF QoS Cubes, while using the same Routing policy as other nodes in the DIF. In resume, given this separation of tasks, network managers can use any combination of policies, as long as those are compatible (Routing with other IPCPs Routing policies, and Forwarding Generator with Forwarding and Routing in the same IPCP).

Chapter 6 considers different approaches to forwarding in different environments, from simple forwarding table solutions for small scenarios to complex smart topological solution for larger ones. Those are also complemented with appropriate Routing and Forwarding Generator policies, considering also scenarios with different policies depending on the IPCP location within the DIF.

4.3 RINA scenarios

RINA provides multiple benefits and improvements with respect to the current TCP/IP Internet protocol suite (e.g. recursive network domains, recursive QoS assurance, variable and adaptable addressing schemes, etc.). While its implementation is not yet at the point where it can be used in a production environment, multiple projects (i.e. FP7 PRISTINE [104], FP7 ARCFFIRE [105], etc.) have been studying how the use these benefits provided by the use of RINA could be used in different key scenarios to improve scalability, provide better services, reduce costs, etc.

In this section, we are going to examine briefly some of those scenarios and the most direct benefits that the use of RINA could bring in.

4.3.1 Service Provider Network

While service provider networks are the base of Internet, as seen before, the limited scalability of the current TCP/IP Internet reduces their capabilities to provide better services to their clients, as well as increasing their costs. In order to manage their networks in the most efficient way possible, provider networks tend to be organized into a hierarchy of interconnected sub-nets, reducing this way the scope of management functions. Given the recursive nature of RINA, it provides an environment well-suited to this kind of networks, creating, in the form of DIFs, small isolated administrative domains within the same operator network. Fig. 4.8 shows an example of this same hierarchical structure, where IPCPs on the different systems form a 3-layers structure consisting of a backbone (Tier 1) DIF, few Regional (Tier 2) DIFs and multiple Metro (Tier 3) DIFs. This same hierarchy can be seen in Fig. 4.9, showing how these DIFs are commonly interconnected.

Each of those DIFs can be configured with different properties in mind, corresponding to their specific function within the provider's network, and isolated between them. For example, as the backbone DIF deals with large flows of aggregated traffic with more deterministic behaviour, it could profit from a more connection-oriented approach, including optimised resource allocation and path recovery mechanisms. On the other hand, Metro DIFs dealing with more varied traffic coming directly from the users of the network would require more connectionless-oriented approaches, taking into consideration the different applications on the network. In this regard, not only the recursivity of RINA provides

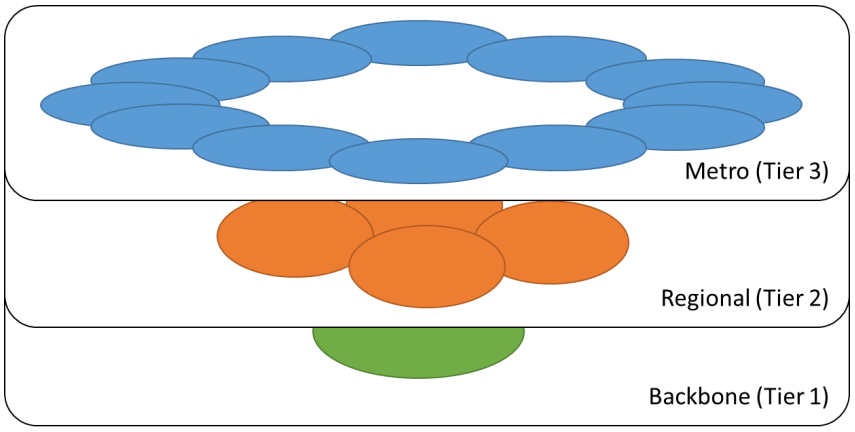

FIGURE 4.8: Example of ISP network DIFs hierarchy

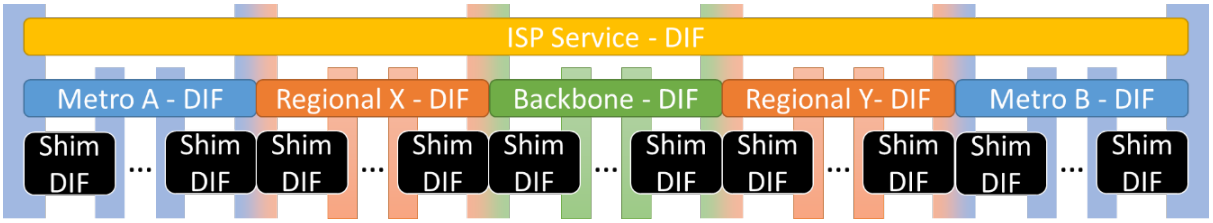

FIGURE 4.9: Example of DIF structure on ISP network hierarchy

a well-suited environments for these networks, but that is also enhanced by the high configurability of DIFs provided by the use of programmable policies.

In addition to the performance of the ISPs themselves, with respect to the services that those provide to their users, while TCP/IP allows to provide only few limited services, mainly related simple best-effort Internet services and Internet-TV services, with RINA those could be improved taking profit from its new capabilities. Those improvements include:

- QoS-enabled Internet services. A RINA-based Internet would profit from all the QoS capabilities provided by the use of QoS Cubes.
- Remove requirement for NAT. While IPv6 could increase the number public addresses given to users, temporarily removing the need of using NAT, the recursivity of RINA would permanently remove the problems related to having a number of addresses lower than that of devices.
- Enhanced Internet-based services. The use of RINA with its programmable policies and QoS Cubes could allow for improved Internet-based services over the same ISP network, including the TV on demand or Internet-based phone and video calls. In addition, given RINA's multihoming capabilities, those services could include home-and-go features, being accessible both at homes and on the go (e.g. same phone number accessible at different locations).

4.3.2 Cloud Network

Cloud computing focuses on the use of computational and storage resources up in a cloud of devices, commonly located in one or multiple data centres. The use of cloud computing has multiple benefits, like the use of leased resources on demand. This removes the need of buying multiple high-end equipment to perform resource-consuming tasks, for example when those have sporadic behaviours (e.g. 3D-video rendering or data analysis) or with variable load during the day (e.g. web-service with focalized user-base). Cloud computing not only reduces costs, but also allows to move web-services and resources close to their users (e.g. serving resources from the closest data centre), improving this way the service that those receive. In addition, given the incremental popularity of smart-device (smartphones, connected devices, sensors, etc.), the use of the cloud provides cheap means to externally increase the storage and computational capabilities of such devices.

While cloud computing provides large benefits, it also has important requirements, specially related to its scalability and dynamism. In order for a cloud network to work correctly and with low costs, it is required to have efficient means to dynamically allocate not only computation and storage resources, but also allocate and configure the required networking resources for that network to work. In that regard, the rigid stack of TCP/IP lacks both the configurability and dynamism required for such tasks. Given that, in order to provide the required capabilities, current clouds make use of solutions like private networks and SDN that forces a solution outside of the stack, as pure TCP/IP-based solutions are not capable of providing such services.

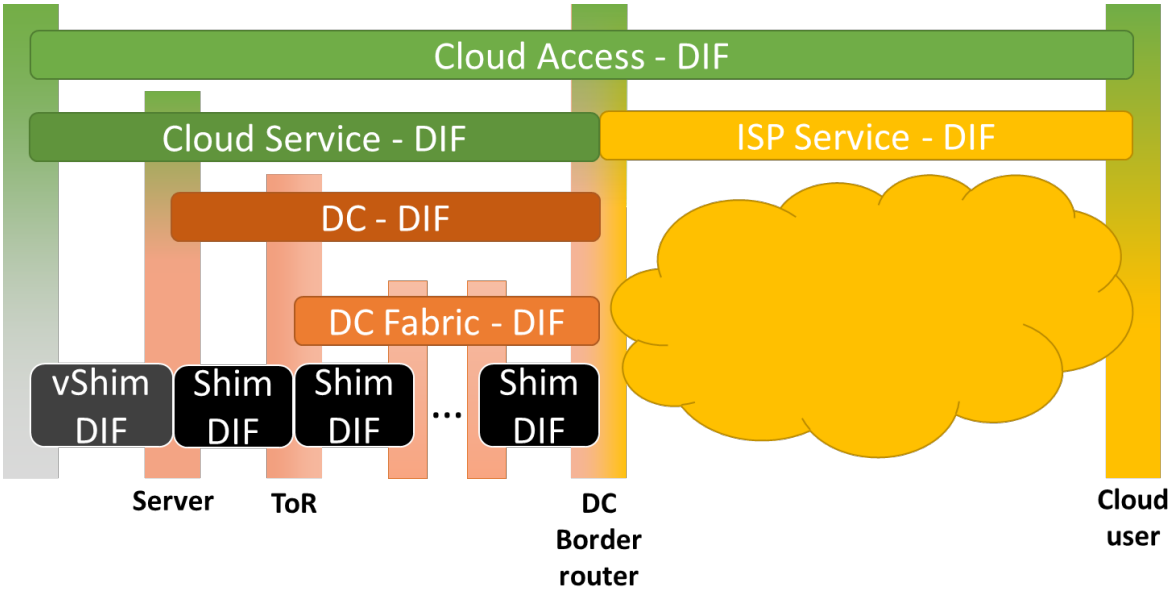

FIGURE 4.10: Example of DIF structure of a cloud network

Fig. 4.10 shows an example of a possible RINA-enabled cloud network. While cloud services would run in a data centre environment, the cloud and data centre networks would be isolated one from another, being impossible for nodes in the cloud to access

other DC nodes. In the same way, users of the cloud would be only capable of accessing its services once they are being enrolled in its DIF. Given the capabilities of the RINA stack, highly focused on scalability, configurability and dynamism, its usage in cloud networks could enable the implementation of new and more optimal solutions within the networking model itself, reducing that way the costs of such cloud networks at the same that improves quality of communication within the network. Some of these improvements include:

- Independent configuration of the data centre lower layers. Each data centre can have its physical network managed by a DIF with policies tightly configured for its specific network, independently of cloud or other usages of upper layers.
- Enhanced mobility of resources. Given the name-based location of resources and its recursivity, it becomes easier in a RINA network to either create or reallocate resources as needed.
- Enhanced QoS assurance. The recursive QoS support that RINA provides allows to any network, including clouds, to define their specific set of QoS requirements, which will be effectively translated to lower layers. In addition, as RINA allows the use of connection-oriented resource allocation for flows alongside connectionless flows, it is possible to enforce the allocation of resources only for the most requiring tasks.
- Security. Users can only access to cloud services after enrolling in the DIF, and any information on the data centre or other services running in it can be accessed.

4.3.3 Distributed Cloud Network

In addition to the traditional data centre-based cloud, where computational and storage resources are located in one or multiple data centres, another trend for cloud networks and applications is the sharing of resources between the same users. Unlike not being commonly referred as such, data sharing in P2P networks is a clear example of distributed cloud computing, in this case, centred on the sharing of storage resources and information. In a similar way, distributed cloud solutions have been used for years in the environment of online games where, in some games, the main servers only acted as managers, gathering the different users, and any client on the network could act as a server for the game itself, hosting the match. Even so, the usage of distributed clouds is not limited to only those use cases, but, as the technology improves and the resources on user-side increase, newer scenarios take profits from its capabilities. Two examples of this can be seen in the early phases of cryptomining [106], before the common use of GPU-based mining, or multiple scientific distributed computing projects, like SETI@home [107]. In those cases, cloud computing is used not for simply sharing storage resources or move part of the load from the main servers to some of the clients, but to parallelize large tasks, that would require

a large number of dedicated computing resources, using simply the unused resources of the clients of those networks.

While distributed clouds have been used for specific and punctual problems, their capabilities are approaching those of small data centre-based clouds, and proof of that is the service provide by SlapOS [108], a decentralized cloud service that uses storage and computing resources at common Internet users in exchange of subsidize part of the Internet bill. Unlike other distributed cloud systems, where the whole cloud focuses on a specific task, and the clients of the cloud are the same that provide the different resources, this kind of distributed cloud reassembles more closely to that of datacentre-based clouds. Fig. 4.11 shows how a distributed cloud is created over one or multiple networks (e.g. IP Internet or possible RINA networks). This decentralized structure of distributed clouds has some important drawbacks with respect to the data centre-based clouds, especially considering that all communication is done over the Internet. Still, distributed cloud networks like that provided by SlapOS offer an increased reliability against localized disasters (e.g. fire on a data centre or natural disasters, Fig. 4.12), as their resources are heavily dispersed, and the possibility of use already existing resources. Even so, given that those distributed clouds currently work with and over the TCP/IP stack, communication on those is highly limited by the strictness of the model itself. In this regard, the use of RINA in such a networks is being studied, as it would provide different benefits already as a RINA over IP solution. Some of these benefits include:

- Enhanced scalability and reliability. Possibility of using custom addressing and routing policies adapted to the specific network, not limited to common TCP/IP solutions.
- Possibility of centralized management within the same model. RINA networks can be managed in a distributed or centralized way without requiring external tools.
- Replication of services. RINA's name-based node location allows to easily locate services at different nodes at any point of the network, solving the unreliability problems related to running services on user's locations.

4.3.4 Network Function Virtualization

As seen before, the virtualization of resources is something more and more common every day. In the especial case of NFV, where the network function themselves are virtualized, this has interesting benefits, as it allows for a fast introduction of improvements on the network, not being limited to the lifespan and cost of the hardware itself. Concerning the requirements of NFV, the first thing to be noted is that the inherent structure of the network that it defines is that of a recursive network: services are composed of multiple connected Virtual Network Functions (VNF), namely virtualized network tasks run on

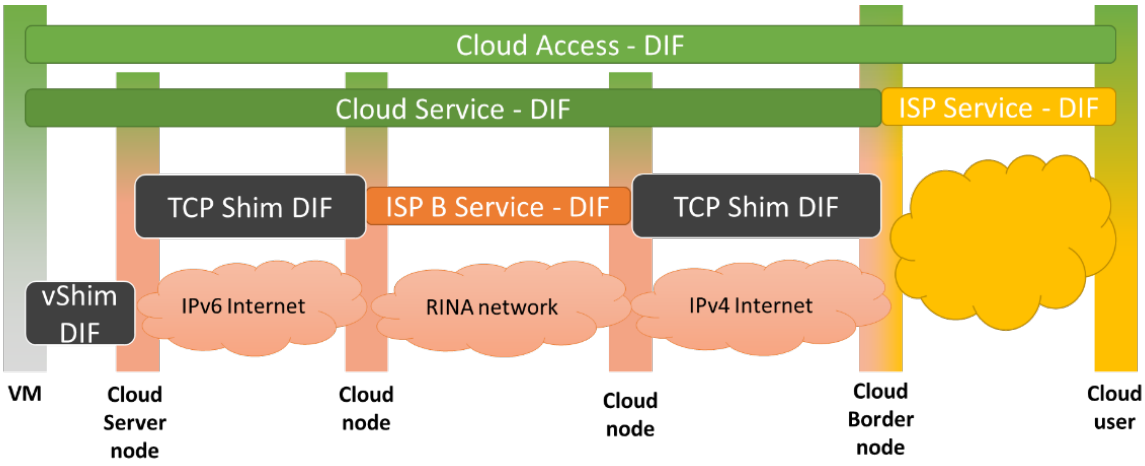

FIGURE 4.11: Example of DIF structure of a distributed cloud network

commodity hardware, and those are themselves built from multiple VNF Components (VNDCs), or sub-tasks. Those services could then to either build a NFV-enabled network or as a network service for upper layers, being then recursivity one of the inherent properties of VNF scenarios. In that sense, the use of RINA in the construction of VNF scenarios, as well as any kind of virtualized network, provides already some promising benefits. Not only with respect to its recursivity, as can be seen in Fig. 4.13, but NFV also imposes some network requirements that imposes extra cost to be achieved in current TCP/IP networks. NFV requires mechanisms for security, resiliency and elasticity in order to be able to perform their functions as required. In addition, in order to provide its services, NFV requires of a coherent abstraction layer to support the interaction between the different models to allow for simple development and execution environments, something that fits well in RINA with the use of programmable DIFs and its management with the use of CDAP.

Between the different benefits, that RINA could provide to a network with NFV services, few of the most important are:

- Authentication mechanisms. Both IPCPs and network managers need to be authenticated as part of the enrolment in the DIF.
- Access control to resources. In addition to IPCPs and manager enrolment in the DIF, the access of users and applications to specific resources within the DIF can also be controlled.
- Content-based security. Security mechanisms can include the encryption of messages and/or authenticity checks.
- Resiliency of communication. From the use of resilient QoS Cubes (with high priority and retransmission on EFCP) to having forwarding policies with fast rerouting upon failures.

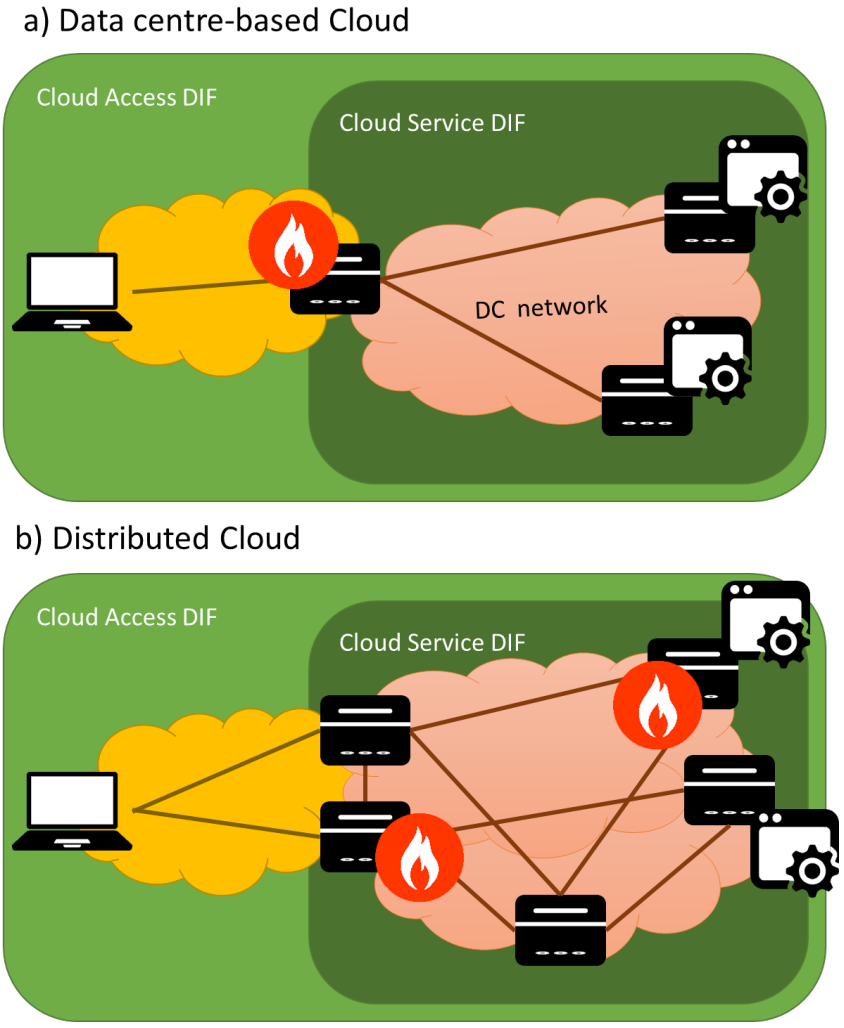

FIGURE 4.12: Comparison of Data centre-based (a) and Distributed clouds (b) reliability upon natural disasters

4.4 Chapter summary

This chapter provides an overview of the Recursive InterNetwork Architecture (RINA) and some of the peculiarities that makes it an interesting replacement for the current TCP/IP Internet. In contrast to TCP/IP, RINA offers a recursive and fully configurable Internet model with QoS support, capable of evolve with the usage of the network. In addition, with a common API, shared across all its layers, RINA provides an easy configuring networking environment for network managers, providing a more complete view of the network requirements and capabilities for automatic dynamic configuration. Not being RINA itself the goal of this work, this thesis only provides a limited view of the model. To extend on the view of RINA, interesting documentation can be found at [109] and in the thesis of Vladimír Veselý [110], who leaded the creation of the RINA simulator (App. A).

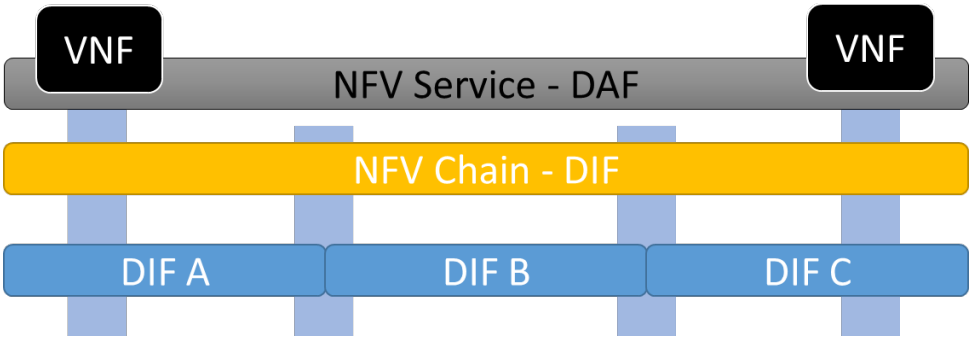

FIGURE 4.13: Example of DIFs structure for the creation of NFV services

While the RINA model by itself provide an interesting solution, one of its greatest points is the use of policies that allows to configure each networking domain in the most suited way. Next chapters expand on that, proposing different scheduling and routing solutions capable of taking profit from the RINA environment to provide a scalable, QoS-centric network.

Chapter 5

Scheduling and Quality of Service in RINA

As seen in previous chapters, the current TCP/IP stack lacks the ability to effectively respond to the increasing variety of communication requirements of heterogeneous distributed applications. Not only that, but this networking model does not even provide a standard mean for applications to express their requirements for communication quality in terms of maximum delay, data loss, data rate, etc., being almost the only ensured "quality" the reliability or not with the use of TCP, UDP or other transport protocol. Of course, as new requirements emerged, new solutions to allow some kind of QoS differentiation in the IP networks appeared. For example, solutions like MPLS VPNs guarantee a minimum bandwidth with some internal QoS differentiation at the IP layer, but usually at the expense of greatly degrading the remaining traffic to a best-effort treatment using the remaining resources or a costly underutilization of resources.

In RINA, all flows traversing a DIF are assigned to one of its QoS Cubes, depending on their QoS requirements. Those QoS Cubes provide average and worst-case measures for the service that the flow should expect, being then responsibility of the DIF to ensure that those expectations are meet, or in the worst-case scenario, inform that it is not possible to achieve the required outcomes. In this context, RINA provides an enhanced medium for QoS-based solutions. With its recursive, multi-layer architecture and its high configurability RINA allows for each DIF to be configured with specific policies that allows to best deliver the expected outcomes of flows, all while maintaining a common API between layers.

This chapter discuss some key points related to scheduling and the assurance of QoS requirements. In special, it focus on RINA's scheduling policies centred on the ideas of the Degradation of Quality, or ΔQ, to analytically ensure QoS guarantees for flows. The rest of the chapter goes as follows. In Section 5.1, the ideas behind ΔQ are expanded and the default ΔQ scheduling policy, the QTA Mux, is introduced. Section 5.2 lists some of the considerations needed to provide QoS assurances. Section 5.3 does an extended analysis

of some key scenarios with respect to the assurance of QoS requirements. Additionally, analytical ΔQ-based configurations and compare the resulted behaviours with those of common solutions are also provided. In Section 5.4, the first results of the QTA mux policy on a physical network are shown. In Section 5.5, a variation of the default ΔQ policy is proposed, aiming to provide limitations on the usage of the network by end-users, something required in order to avoid greedy behaviours. Finally, Section 5.6 resumes the ideas of this chapter.

5.1 Degradation of Quality

Applications depend on information to complete computations, and distributed computation necessarily involves "translocation" of information generated by one computational process to another, located elsewhere, this is IPC. Instantaneous and completely loss-less translocation is physically impossible, thus all translocation experiences some 'impairment' relative to this ideal. If information takes too long to arrive (and/or too much of it is missing) then the computations cannot proceed, and the application fails to deliver the requested service with acceptable performance, thus the impairment of the translocation must be suitably bounded.

ΔQ [100, 101, 102] is a measure of the impairment of the translocation of information between two points. This impairment has several sources, including the time for signals to travel between these points and the time taken to serialise and de-serialise the information. In packet-based networks, statistical multiplexing is an additional source of impairment. Multiplexing is a real-time "game of chance" played out between packets of flows sharing resources at some network element. The result of this game is that the onward transfer of each packet to the next element along the path may be delayed or may not occur at all (the packet may be lost). In these "games", when one packet is discarded, another is not. Similarly, when one is delayed more due to scheduling decisions, another is delayed less - i.e. this is a zero-sum game in which quality impairment (loss and delay, as measured by ΔQ) is conserved. Thus, in packet networks, ΔQ is an inherently statistical measure that can be thought of as either the probability distribution of what might happen to a packet transmitted at a particular moment from source A to destination B or as the statistical properties of a stream of such packets. ΔQ captures both the effects of the network's structure and extent and the impairment due to statistical multiplexing.

Whether an application delivers fit-for-purpose outcomes depends entirely on the magnitude of ΔQ and the application's sensitivity to it. The odds of the 'multiplexing game' are affected by several factors, of which load is one; since capacity is finite, no network can offer bounded impairment to an unbounded applied load. What an application requires is for the network to translocate the amount of information it needs to exchange with an impairment no greater than it can tolerate. A formal representation of such a requirement is called a Quantitative Timeliness Agreement (QTA), which provides a way

for an application and a network to negotiate performance. In effect, the application "agrees" to limit its applied load in return for a "promise" from the network to transport it with suitably bounded impairment. This idea is embodied in the design of the QTA Multiplexor (QTAMux), to be discussed later.

As some applications are more sensitive to losses than others, and the same can be said for latency, it can be said that some flows are more cherished (require lower losses) or more urgent (require less latency). Hence, their requirements can be mapped into a Cherish/Urgency (C/U) matrix, that is, an NxM matrix with relative latency and losses at each edge (a 4x3 C/U matrix is shown in Table 5.1). This has a straightforward implementation called a Cherish/Urgency multiplexor (C/U Mux) [111], included within the QTAMux design. A C/U mux provides differential loss probability using a shared buffer with higher thresholds for packets of more cherished flows, and differential urgency by giving higher precedence service for packets of more urgent flows. Just as total delay is conserved under scheduling [112], ΔQ is conserved in C/U multiplexing.

TABLE 5.1: 4x3 Cherish/Urgency matrix

Cherish/Urgency	**"Lossless"**	**Max Cherish**	**Mid Cherish**	**Min Cherish**
Max Urgency	A1	A2	A3	A4
Mid Urgency	B1	B2	B3	B4
Min Urgency	C1	C2	C3	C4

5.1.1 Degradation and congestion

The use of the QTAMux provides short-term treatment for the flows traversing congested nodes, sharing the degradation that those suffer depending on their QoS requirements and QTA. This, in connected-oriented networks, with a tight control of the flows traversing the network, is enough to ensure an effective share of degradation between flows, being congestion mainly generated by small bursts of colliding data. Conversely, in less restrictive/more competitive environments, like that of connectionless networks, resources may be overbooked, having peak times where users may want more resources than available (common use case for today's Internet).

In cases of long-term congestion, like those common in overbooked networks, the short-time treatment of the QTAMux is not enough to assure the best usage of the resource of the network, while it ensures that the degradation caused by that congestion is shared as expected. Instead, congestion control mechanisms are required to reduce the usage of resources in the network. Lucky, RINA allows any DIF to enforce its own data transfer control policies in its DTCP, meaning that measures can be taken within the focus of congestion, taking into consideration also both the QoS and QTA of flows. For that, it would be work of the QTAMux and the P/Ss to decide when inform of congestion.

Considering the notification of congestion, given the different QoS requirements and different QTA of flows, simple solutions like marking congestion in all flows traversing a congested node would result in an undesired share of resources (e.g. both priority traffic and non-priority traffic would have its data-rate similarly reduced). Instead, the different QTA in the DIF may require some special considerations depending on the usage of resources by the different flows (e.g. QTA may require for priority traffic to maintain its rate while non-priority traffic is over some % of the total occupation).

With flows distributed in a C/U Matrix depending on their requirements, a common assumption could be that the more cherished flows have also higher requirement for its data-rate to be maintained with more than other flows. Even so, that may not always be true, as it is entirely possible for the requirement of a flow in terms of avoiding losses to not be related to its requirements in terms of maintain its data-rate in a congestion scenario (e.g. a voice flow, allowing some losses, but requiring a constant rate of PDUs sent at 50Hz). With that in mind, when considering the QoS Cubes of RINA and QTA, a third dimension, related to the order how data-rate has to be reduced upon congestion, sometimes is required when distinguishing requirements. In those cases, instead of a C/U Matrix, a Cherish/Urgency/Rate Cube (C/U/R Cube) would be considered, adding the stated third dimension to the decision of the QTAMux (although that is left for future work).

5.1.2 QTAMux

The QTAMux is designed in order to fulfil the QoS requirements of flows, to the higher possible extent, and ensure that flows behave within the contracted QTA. Fig. 5.1 describes the sub-modules and operation of the QTAMux scheduling function. When a PDU arrives, it is classified based on its QoS Cube, flow identifier, etc., depending on the desired flow treatment. This classification results into a stream queue where the PDU will be stored and the Policer/Shaper (P/S) for that queue. The P/S is then informed of the new addition to the queue and the size of the added PDU. After this point, the specific PDU is abstracted to queue references. Then, the P/S may decide at some point to send a traffic class reference (a tuple <queue, cherish level, urgency level>) to the C/U Mux. The C/U Mux then does its function and whenever it has queues ready to be served and the output port is ready, it selects the queue reference for which the first PDU will be served.

5.1.2.1 Policer/Shaper

Within the QTAMux, each queue has an associated Policer/Shaper (P/S) instance in charge of control the internal behaviour of its flows (intra-flow contention). P/Ss are able to drop packets from the queue (either the first or the last one) or delay when the next

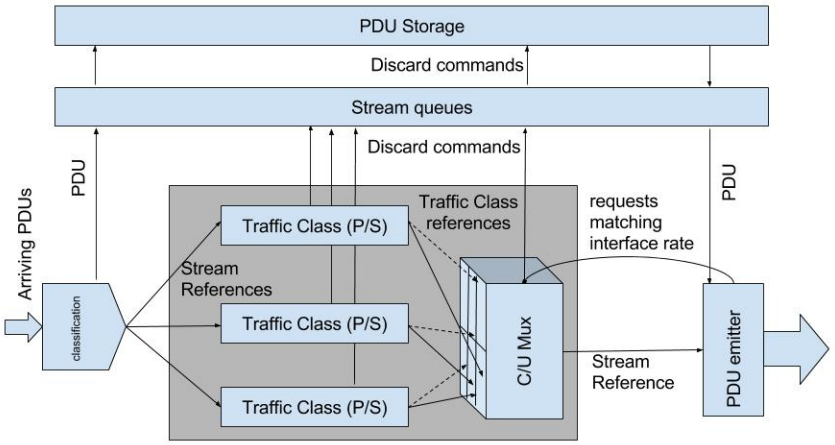

FIGURE 5.1: QTAMux modules and workflow

queue reference will be sent to the C/U Mux. The main objective of P/Ss is to ensure the compliance of the QTA of the arriving flows. In addition, the P/S also decides the cherish and urgency levels of the traffic class references passed to the C/U Mux, being that not required to be a fixed pair of values, but vary depending on the behaviour of flows. In general, the functionality of Policer/Shapers can be summarized as follows:

- Burst control. A P/S may control bursts, either dropping all arriving PDUs after some burst limit or statistically dropping PDUs as the burst size grows.
- Traffic class spacing. A P/S may delay the sending of traffic class references in order to smooth long bursts or to enforce an instantaneous rate limit.
- Selecting cherish and urgency levels. Cherish and urgency levels can be either fixed or determined (either deterministically or statistically) dependent on current burst length.
- Inform of breach of QTA as ECN. Instead of simply dropping PDUs upon breaches in QTA, the P/S may also inform of the breach of QTA, either after it has been breached or progressively near its breach.

Note that P/Ss perform most of their decisions based on bursts rather than the average rate at queues. The motivation of that is that both C/U Mux and P/Ss should perform only fast point-to-point decisions based on the instantaneous state of the network. Instead, mid and long terms decisions should be taken by EFCP instances at the DIF endpoints and the resource allocation modules, being in that case the reconfiguration of P/Ss or even the C/U Mux one of the possible decisions.

5.1.2.2 C/U Mux

The C/U Mux module is the module in charge of inter-flow contention. The main functionality of this module is to decide if a received queue reference has to be discarded because of too much congestion (and with that a PDU dropped from it) and when accepted queue references should be served. At the input, the C/U Mux receive traffic class references containing both the queue reference and the cherish and urgency levels for that. At this point, based on the cherish level and the amount of queue references currently stored in it, the C/U Mux may decide if discard or not the queue reference. In order to take that decision, for each cherish level, the C/U Mux has configured multiple ranges and a discard probability after that threshold. Then, the queue reference is discarded statistically with the discard probability given by the range that includes the current number of stored queue references. For simplification, the implemented C/U Mux works with only 3 ranges:

- $[0, th_p)$. Do not discard.
- $[th_p, th_a)$. Discard with probability p.
- $[th_a, \text{inf})$. Discard all.

If the queue reference is not discarded, then it is pushed into a priority queue of queue references, being its priority the urgency level (urgency 0 being the higher priority). Finally, whenever the output port is ready and there are queue references waiting in the C/U Mux, the next queue reference is extracted from the priority queue and that queue is served.

It has to be noted that, while a possible implementation for the C/U Mux based on priority queues has been presented, other implementations could be considered whenever they provide the required inter-flow contention. An example of this could be a variation of the C/U Mux taking decisions based on how much time a queue reference has been stored and how much in-queue delay is allowed for each urgency level. While not considered in this work, such variation of the C/U Mux could provide a finer grade of QoS assurance in scenarios with a wide range of similar latency requirements.

5.1.2.3 Congestion notification

In order to support the signalization of congestion on flows, the QTAMux has to decide when and how to mark congestion on the departing PDUs. In traditional solutions, where each queue is independent, this would be done at the same moment a PDU is inserted in a queue, marking congestion whenever a certain threshold in queue occupation had been surpassed. On the other hand, when considering ΔQ and the QTAMux, that is something not that simple. First, P/Ss may modify the behaviour of flows, and second,

PDUs may be dropped at different points, and not only in their arrival to the queue. Considering that, it becomes clear that it becomes more useful to inform of congestion at the departure of PDUs from the C/U Mux, rather than mark PDUs at their arrival. In addition, this way allows for early congestion notification, being that notified by the next served PDU rather than the last inserted.

Considering that there is a whole combination of heterogeneous flows, all sharing the same buffer space, straightforward decisions based on queue length are also not reliable in this case, as overall congestion should have different effects depending on the flow. Having this in mind, the QTAMux has to decide if a departing PDU has or not to notify congestion considering both its status at the arrival and departure of that PDU. Considering that, a simple approach to ECN marking within the QTAMux could be considered:

- For each queue, a counter of "remaining ECN" is maintained, storing the number of notifications waiting to be send with PDUs of that queue (always lower or equal to the current size of it).
- When accepting a queue reference into the C/U Mux, the counter for that queue is incremented if the number of queue references already in the C/U Mux surpasses a threshold for the queue's rate level.
- When serving a queue, if the counter for that queue is not 0, the outgoing PDU notifies congestion and the counter is also decremented.

In addition to the notification in C/U Mux, a P/S can also notify congestion in cases of breach of contract. Although, in this case, additional conditions should be noted, like how restrictive is the contract or the RTT between nodes, as it may cause undesirable reductions of rates for more restrictive QoS, while those are at the same time the ones with higher requirements.

5.2 Considerations when providing QoS Assurances

In order to be able to properly provide QoS assurances in any given network, it is required to have sufficient knowledge about that specific network and the flows traversing it. Almost anything can impact on the provided service: if the expected traffic is much lower than the capabilities of the network, requirements will be less constricting than in an overbooked scenario; if link latencies are high, that will have higher impact on the flows latencies than scheduling; retransmission of lost PDUs add delay and extra load into the network; etc. Given that, it is not enough to place the scheduling policy that best suits the QoS Cubes in the DIF, but, instead, how those are configured tend to be of higher relevance.

The previously QTAMux policy is a clear example of this. By itself, this scheduling policy provides a clear differentiation of the available services. Even so, without a correct configuration, that differentiation may come to waste, providing a service no better to that provided by a best-effort solution. For example, using the QTAMux in a small scenario with two QoS Cubes, one with low latency and other lossless, the QTAMux configuration would place those QoS Cubes in the down-top diagonal of a 2x2 C/U matrix (classes A2 and B1 respectively). While this gives a good base point to fulfil the QoS requirements, from this point, multiple things could go wrong with the configuration: the threshold for A2 traffic in the C/U Mux could be too low, over-dropping on bursts; rate limits in the P/S for either one of the two QoS Cubes could be configured too tight, having PDUs wait too long on the P/S queues and being dropped in uncongested scenarios; etc.

With that in mind, this section analyses some of the key properties of any network with impact on the assurance of QoS requirements. These include not only the flows and their requirements, but also the intrinsic properties on the traversed network and their capabilities, as well as the placement of the most suited policies in each place.

5.2.1 Knowing the expected usage

The first and most important thing to consider in order to configure, not only the scheduling policies, but also the set of available QoS Cubes in a DIF, is knowing what will be the usage of that specific DIF, or at least what to expect. Depending on the DIF, flow requirements may vary greatly. For instance, every application has its own service requirements. In one extreme, sometimes a applications can have as few requirements as to only require a potential translocation of resources (e.g. an application that monitors a low priority system). In the other, applications may have strict requirements, requiring things like a maximum allowed delay, loses, etc. (e.g. a 4K video transmission in direct).

Not only it is important to know the common set of QoS requirements in the DIF, but also the traffic patterns followed by those applications. That is important, as, even if two applications share the same requirements, how those are achieved may vary greatly. Let us consider for example three applications with similar QoS requirements but following different traffic patterns:

- An application that monitors an urgent system, with small updates every few seconds or on events.
- A high-quality voice application, sending data at constant intervals, but with their size depending if the source is in silence or not.
- A video transmission, sending large amounts of data at a constant rate.

While those three applications require a high priority treatment, requiring low to no losses and minimum delay, the way that may be achieved could be completely different within

the same DIF. For starters, knowing the low amount of traffic generated by the monitor application, for those flows, only a small rate-limiting would be considered (to ensure it remains within the requested rate), but no more flow control mechanisms would be in use (neither window-based, nor rate-based), as PDUs would be sent at intervals time larger than the RTT. In contrast, while practically no flow control would be in place, it most probably would have active the re-transmission mechanism, ensuring that even in the improbable case that a PDU is lost, it will arrive latter. On the contrary, when considering voice and video applications, the situation is the opposite, as lost PDUs are not re-sent (as either way would be discarded given their late arrival), but the higher rate would require of flow control mechanisms in use (most probably rate-based ones). That could be taken even beyond this, as, if the DIF configuration allows it, video transmission flows could be allocated as connections, having each their bandwidth completely allocated along their path, something that matches perfectly with their large rate and constant behaviour.

Apart from the applications requirements and their traffic patterns, it is also important to consider how much applications of each kind will be supported in the network, or, what is the same, how traffic aggregates in the DIF. Considering the QTAMux scheduling policy for example, while its behaviour is based on the differentiation of services by their requirements, it also imposes some limits in the processed flows. These limitations, imposed by the P/S, limit the rate at which the aggregate of one or more QoS Cubes can send through a link, allowing this way for less requiring flows to also send through that link. In addition, while those limits could be simply computed as the maximum rate if all applications are sending at its maximum at the same time, that would be counterproductive, as that would be way more bandwidth than the amount really in the network. For example in the case of voice flows, those tend to follow an ON/OFF pattern where the flow varies from talk-spurt to silence, which characteristics vary for example given the speakers language [129]. If only the rate of ON and OFF periods and their average duration are considered to compute the minimum, average and maximum rates of flows, all flows would be considered separately, always at their maximum. Instead, if the aggregate of all voice flows is considered, more meaningful values can be extracted from it. For example, Table 5.2 shows the probability of having, from N voice flows, M or less in the ON state, if talk-spurts take 1/3 of the flows time in average.

TABLE 5.2: Probability of at least M voice flows simultaneously in the ON state

N	$\mathbf{P}(M = 0)$	$\mathbf{P}(M \leq 1)$	$\mathbf{P}(M \leq 2)$	$\mathbf{P}(M \leq 3)$	$\mathbf{P}(M \leq 4)$	$\mathbf{P}(M \leq 5)$
1	0.66667	1.00000				
2	0.44444	0.88889	1.00000			
3	0.29630	0.74074	0.96296	1.00000		
4	0.19753	0.59259	0.88889	0.98765	1.00000	
5	0.13169	0.46091	0.79012	0.95473	0.99588	1.00000

With that into consideration, instead of considering the maximum rate, it is more correct to consider the maximum rate under some high probability. For example, if 99.5% is considered as "high probability", for 5 simultaneous voice flows in this network, instead of considering the aggregate rate of the 5 flows at their maximum, it could be considered that the network will have 4 flows simultaneously at its maximum at most, being this way almost sure that the aggregate would be under that rate (4 at max + 1 at min).

Taking this into consideration, in order to effectively assure the QoS requirements of multiple services in the network, it is required to know, not only the requirements of flows traversing the network, but also different properties on the aggregates, like the average bandwidth usage or the maximum the probability of having more than X flows of the same kind, etc.

5.2.2 Knowing the network

Flow requirements and their aggregates are not the only thing to consider in order to provide QoS assurances, but only the starting point. Specially, it is of great importance to consider the specific characteristics of the network, as those would affect directly to the offered service. For instance, is not the same to consider a network covering small distances that one covering large distances, as in the first case link latencies could be negligible, but in the second those could be the main source of delay in the network. Taking this into consideration, there are four main point to consider when configuring most networks:

- Link latencies.
- DIF layering.
- Hardware/software characteristics.
- Nodal degree.

In the first case, as mentioned before, link latencies are one of the main network properties to consider whenever configuring a DIF. Those impose strict minimums in the observed delay in any communication, impossible to be reduced by any configuration or policy. Firstly, when configuring the set of available QoS Cubes, in any DIF, there is no QoS Cube capable of ensuring a delay lower than the minimum link latency, so that already gives the minimum range to consider. Secondly, if the DIF configuration allows it, it also gives some limits in flow allocation, given direct feedback to applications to applications trying to allocate flows with requirements physically impossible to meet. Finally, when allocating flows, it can be used to assign different QoS Cubes to applications with similar requirements, based in the minimum distance to the destination. With that differentiation, it may be possible to provide completely distinct treatments to flows going to a

near location that to flows going to a distant one. For example, for the voice application considered previously, large distance flows could be treated with the higher priority in both delay and losses, as those cannot afford to retransmit lost PDUs. On the other hand, for flows covering smaller distances the retransmission of lost PDUs could be done within the required transmission time, so those can have slightly lower priority when considering losses in scheduling policies.

How the network is layered also has great importance in how well QoS requirements can be ensured. For instance, while on the lower layers the underlying shim-DIFs tend to behave in a regular way, in higher layers it is only possible to rely in what the QoS Cubes of N-1 layers guarantees. This has a significant impact in the behaviour of the flows in the current DIF, as those do not gives exact values (e.g. 5ms latency), but a range of what to expect (e.g. average 3ms and maximum 5ms of latency, with a standard deviation of 0.7). If paths formed by the concatenation of multiple of those N-1 flows are considered, that needs to be taken also into consideration, meaning that flows would be even more unpredictable. For example, that same N-1 flow latencies of 3ms on average and 5 of maximum, would translate to an average path latency of 9ms and maximum of 15ms for a 3 hops path, without considering the effects of congestion in the N DIF. Not only that, but it is also important to consider the effect of all headers added by the current and lower layers, as those will affect differently flows with large-size PDUs than those with small-size PDUs.

Then, there are also the intrinsic characteristics of the hardware and software that provides the service. In this regard, the most obvious thing to consider is the rate of the physical links and how that is reached. For example, let us consider how a 1Gbps Ethernet link would transmit data coming from upper layers. First, depending of if a vlan is being used or not in that link, the Ethernet frame will have different header sizes, affecting both to the resulting available link rate and the sending time of frames. Then, it need to be taken into account that consecutive Ethernet frames require a minimum separation of 12bytes that, while do not affect the time required to send frames, reduces the available link rate. Then, even when considering the same kind of link, there are different variations in their behaviour depending on how those are used. For example, that same rate of 1Gbps from a limited 10Gbps Ethernet link, while using the same protocol, will suffer only 1/10th of the delay when sending each frame, and the minimum gap between frames will have no effect, as that will be already included in the packet separation added to limit the traffic. On the contrary, to achieve that 1GBps rate by merging multiple 100Mbps links, while small burst could be sent at the same time, flow would not only suffer from 10 times more delay when sending each frame, but also from longer inter-frame gaps in each link, as well as potentially requiring the reordering of packets on arrival.

While being the most evident, links are not the only hardware that will affect the provided service. In this case, it is also important to consider the specific hardware and software that constitute the different nodes in the DIF. For instance, clear differences can be observed between the use of hardware or software based switches/routers [135], or even

more when considering the use of general purpose hardware (e.g. servers, phones, etc.) [149]. Any device take some time to process incoming data, as well as that created at the same device, and that is something unavoidable. With purpose built hardware, that time can be minimized, but never removed completely. Even more that delay is not something regular, but that tend to have small variations, not only given the size of the processed packet, but also of other factors like the rate at which those arrive or the collision with other packets in the node [128]. When considering general purpose hardware, the effects on the observed service is even larger. Unlike with purpose built hardware (even if it has some software based functionalities), in general purpose hardware, networking capabilities share the different resources of the machine (CPU time, memory, etc.) with other unrelated processes, all managed by a common kernel. This has important implications, as it means that, in this case, the networking related processes do not always have the focus whenever there is something new to process. A clear example of this can be seen in how some drivers manage the physical networking resources. For example, going back to the Ethernet link, as network processes cannot gain focus whenever the output port is ready, and sometimes that focus gain can occur with some delay, these links are designed to maintain a small outgoing buffer, capable of storing multiple frames ready to be sent [150]. While this avoid the need for the networking process to gain focus each time the outgoing port became free, takes away part of the control of scheduling policy, making it send packets on burst whenever the buffer has space, rather than one by one.

Finally, it is also important to consider how many source of collision there are in each node. For instance, a rely-only node with only two ports, both with the same characteristics, most probably, will never suffer from any kind of congestion, except if caused by a failure (e.g. a temporary desynchronization), and will act mostly as an extension of the links at both sides. In contrast, a node with a high nodal degree could possible suffer from instantaneous congestion in an outgoing port, even if the port usage is low, as multiple packets can be directed to it from different sources. Not only that, but that could also delay the processing of packets, if too much arrive at the same time. This collision from concurrent sources is something of great importance to scheduling and has to be taken into consideration in order to correctly configure those policies. In fact, to avoid moments of false congestion due to concurrent arrivals, it is required to at least have enough buffer to accommodate bursts of packets resulting from a random collision of concurrent flows from different sources and same destination.

5.2.3 Locating policies and layering

Finally, it is also important to take into consideration where to locate each policy and how the different layers interact. Before all, it has to be considered the fact that shim-DIFs do not tend to offer any service differentiation, but only a basic translocation service (e.g. transmission through an Ethernet link, a tcp or udp flow, etc.), and, when offered, it tends to be simpler that what RINA may offer (e.g. QoS differentiation of vlans). Given that,

in order to provide any service differentiation in the network, it is commonly required to have at least one level of normal DIFs over the shims. Fig. 5.2 shows an example of QoS differentiation in the shim-DIF vs. in a RINA DIF. In this simple example, it can be seen that, while sometimes it is possible to differentiate services directly within the shim-DIFs, that is limited to what the hardware supporting it is capable, and does not easily offers the possibility to dynamically adjust it. In the case of this example, QoS differentiation based in vlans commonly offers nothing more than simple priorities between vlans or shaping the flows to a maximum rate [151]. In contrast, QoS differentiation in a normal RINA DIF is free to implement any necessary policy, as well as dynamically reconfigure them as needed. Even so, the benefits not always surpass the cost of a more complex solution, and, sometimes, even a best-effort solution in the shim-DIF could provide the required service, without requiring to add extra layers.

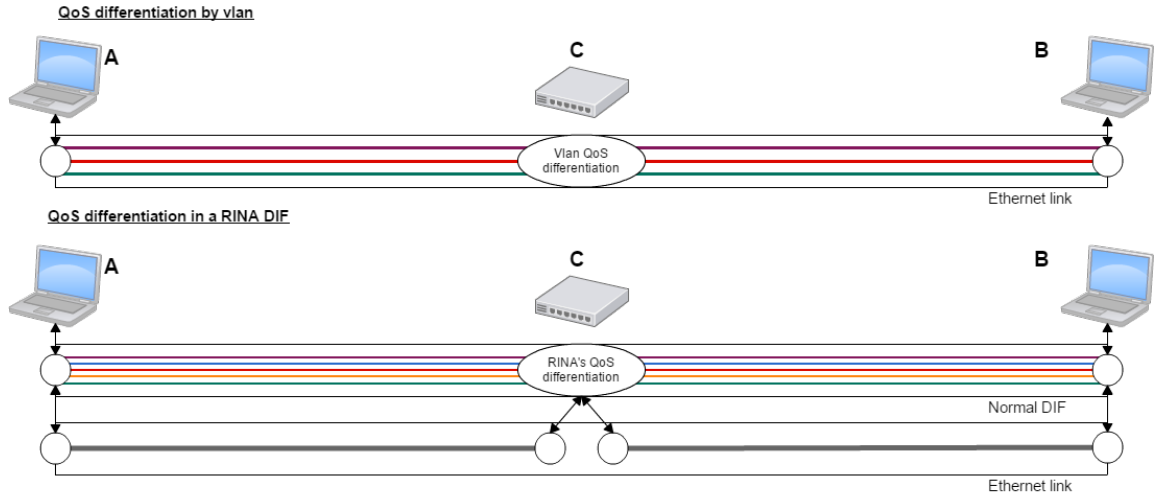

FIGURE 5.2: QoS differentiation in shim-DIF vs. normal-DIF.

Another think that needs to be considered is how QoS Cube in upper DIFs aggregate in lower DIFs. A common scenario with QoS when approaching the lower layers is that where the number of QoS Cubes either decrements or transforms into similar QoS Cubes for the aggregates. This is something important to consider as it defines the two most common scheduling scenarios for a DIF (outside of connection-oriented scenarios):

- Equivalent QoS Cubes. Each QoS Cube in the N-DIF has an equivalent QoS Cube in the (N-1)-DIF. As each RMT port in the N-DIF manages only flows with one QoS Cube, there is no mean in differentiating flows.
- Aggregation of QoS Cubes. Multiple QoS Cubes in the N-DIF aggregate into the same QoS Cube in the (N-1)-DIF. Depending on the QoS Cube requirements of the N-DIF, it may be useful to perform QoS differentiation in the RMT-port.

That is not all, but it needs to be considered alsto that the behaviour of RMT-ports may differ greatly depending on the N-1 flows supporting it. For instance, as seen before,

when considering scheduling over an N-1 flow over an Ethernet shim-DIF, scheduling is done practically at link's rate. This gives great control about the behaviour of flows, as the only unmanaged part of the translocation process would be that of the small buffer at the Ethernet port. In contrast, part of this control may be lost at the moment that the lower layer performs some kind of flow control. In that case, flows may have to deal with large EFCP buffers that can unexpectedly grow (e.g. after receiving an ACK for multiple PDUs). That, unless the network is already on a situation of congestion, means that most scheduling policies will send all arriving PDUs directly to be stored into the N-1 EFCP buffer, without any real control in their resulting order. This can only be avoided until some point if the size of the EFCP buffers of the N-1 DIF can be minimized, something not always possible depending on the scenario. Similar behaviours may happen if the N-1 flow performs retransmission on losses, as PDUs with lower priority in terms of the N-DIF may be re-sent before high priority ones still waiting on queue. Fig. 5.3 shows a small comparison between the effects of scheduling in a N-DIF depending on the EFCP configuration of the N-1 flow. This example shows a situation where a burst from three different sources arrives with the same destination port, while its queues (both scheduling and N-1 EFCP) are empty. The arriving PDU are either of high priority or low priority and the scheduling policy sends them down in a strict order based on that. Then, there are two different behaviours based on how much buffer the N-1 flow allows. In the first case, there is an EFCP with a small left window of only 3 PDUs that, while takes few PDUs to fill up, does not take long to give the control to the scheduling policy in the N-DIF (similar to what would happen with an Ethernet shim-DIF), so most PDUs end correctly ordered by their priority. In contrast, given a larger buffer in the EFCP (in this case a left window of size 20), the buffer needs more time to fill up, resulting in this case in all PDUs being served in order of arrival.

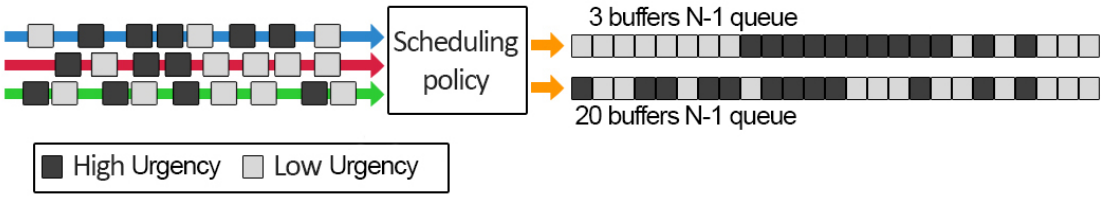

FIGURE 5.3: Comparison between scheduling over N-1 flows with small and large EFCP buffers.

With all this into consideration, it can be seen that there is no "fit-all" solution when it comes to scheduling functions, but each network location requires its own distinct configuration. For example, in the same network there can be have extreme cases when the aggregation of a full array of QoS Cubes may benefit from using the QTAMux scheduling policy in one DIF, but the above DIF may use a simple best-effort policy with a simple 1-to-1 relation between QoS Cubes.

5.3 QoS differentiation vs. assurance

The current IP networks, lack the ability to respond to the increasing variety of communication requirements of heterogeneous distributed applications, not even providing standard means for applications to even express their communication quality requirements in terms of maximum delay, data loss, etc. Even so, it is of course possible to perform some QoS differentiation at the different layer of the TCP/IP stack. For example, solutions like Multi-Protocol Label Switching (MPLS)-based Virtual Private Networks (VPNs) guarantee a minimum bandwidth with internal QoS differentiation, but usually at the expense of degrading the remaining traffic to a best-effort treatment using the available resources.

In contrast to IP, RINA provides an environment with a complete support for QoS where a large variety of information about flow requirements and their expected service is available for flow allocation policies, network administrators, etc. While it is possible to directly implement IP solutions in RINA networks, their behaviour is mainly centred on a strict QoS differentiation (e.g. strict priority, bandwidth division, etc.). While simple solutions based on QoS differentiation, or even best-effort ones, may be useful in multiple scenarios, their do not tend to provide a fine control of the provided service. For more complexes scenarios, with multiple QoS Cubes and tight requirements, those strict solutions may not be enough to provide reliably QoS assurances for the different flows. In contrast, RINA allows for the use of more specific solutions, capable of providing more reliable QoS assurances. Those policies (e.g. QTA Mux scheduling policy), may use the available information on QoS requirements and flows to provide fine-grained QoS service beyond strict QoS differentiation.

This section focus on the comparison between simple QoS differentiation and the QoS assurances provided by ΔQ-based scheduling policy. Given the importance of QoS-aware networks and cloud computing for future network scenarios like 5G, remote applications for smart devices, sensor networks, etc., providing and assuring differentiated and bounded levels of QoS on backbone networks is a must, making of these networks an interesting study case. With that into consideration, this section work is based on the results of multiple simulations in a small scenario emulating a small backbone network providing QoS services. Those services are provided to both common users and data centre networks, comparing there the service provided by a MPLS-like network and a RINA network with ΔQ-based scheduling (using a limited version of the QTA Mux).

5.3.1 Service provider scenario

The scenario under consideration, illustrated in Fig. 5.4, is that of a network provider offering different types of services. Some are targeted to providing connectivity between distributed data centres, while others are for general best-effort traffic to common users. The provider network consists of two main layers: the Top Level DIF, which is exposed to

customers and allows them to access the provider's IPC services, and the backbone (BB) DIF, which is an internal layer where most of the routing and resource allocation policies are enforced. With that in mind, the backbone DIF, in charge of providing adequate differential traffic treatment for the different traffic classes, will be analysed.

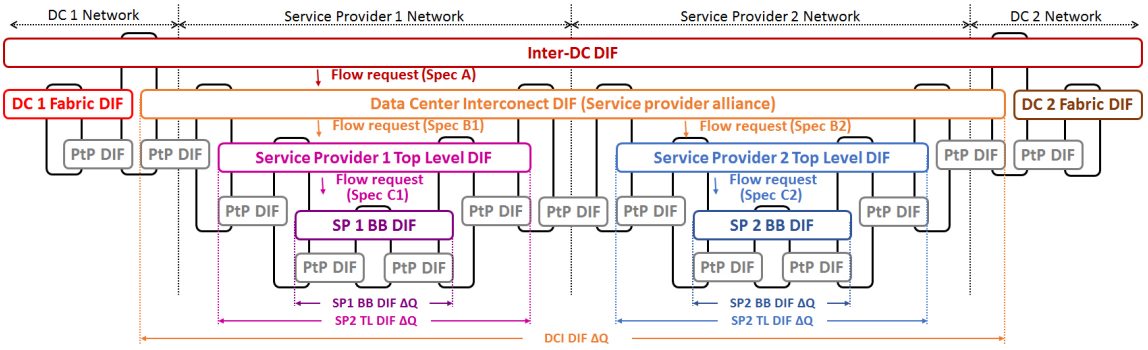

FIGURE 5.4: DIF configuration in the service provider scenario

To analyse the benefits of a RINA backbone DIF with ΔQ-based policies, a DIF with the same internal structure as the 10-node IP/MPLS layer of the Internet-2 backbone network [164] as shown in Fig. 5.5 has been considered, where link latencies are derived from the real physical distances between node locations. In that figure, circles represent nodes in the DIF and solid lines flows provided by the multiple N-1 level point-to-point DIFs shown in Fig. 5.4 (N-1 flows). Moreover, it is assumed that all N-1 flows have a capacity of 10 Gbps and impose a Maximum Transmission Unit (MTU) of 5KB. Note, however, that the achieved results would also be representative of scenarios with higher N-1 flow capacities (40 or 100 Gbps), provided that the offered loads are scaled accordingly.

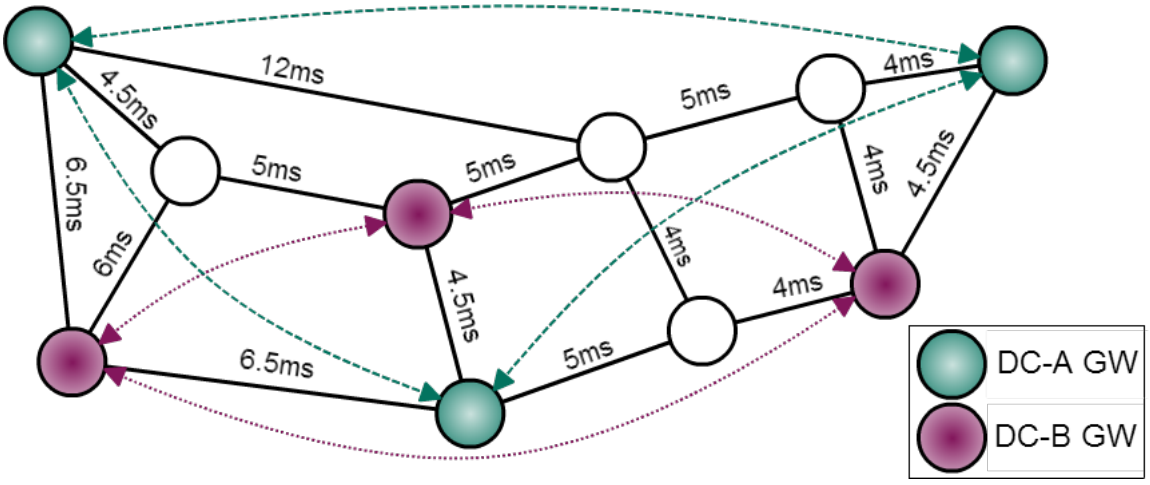

FIGURE 5.5: DIF configuration in the service provider scenario

In this scenario, there are located two distributed data centres (DCs), each one situated in three different geographical locations. These exchange two different classes of inter-DC traffic directly mapped down to the ISP Backbone DIF, whose characteristics are detailed below:

- Gold/Urgent traffic (GU): Assured bandwidth, minimum latency and jitter. Essentially lossless.
- Silver/Non-Urgent traffic (SN): Assured bandwidth, low latency and jitter. Nearly lossless (weaker requirement compared to GU traffic).

A 1:4 GU:SN traffic ratio has been assumed when generating the inter-DC flows. Moreover, each pair of data centre locations exchanges up to 2 Gbps and 1 Gbps in DC-A and DC-B, respectively. This inter-DC traffic mix has to share the available lower-level N-1 flows (hereafter referred as "links" for simplicity) capacities with background traffic flows of two different classes:

- Sensitive Best-effort (sBE): Minimum latency and jitter. Allows losses but preferable to limit consecutive packet losses on flows to less than 3 (it accepts losses for voice and video streaming, sensor updates, etc.).
- Best-effort (BE): Traffic class with the lowest requirements in terms of losses, delay and jitter.

Each pair of nodes exchange sBE and BE flows (representing a full-mesh background Internet traffic) with a 3:7 sBE:BE traffic ratio, so that all DIF links are filled with a similar load level. The motivation behind this is to assume a worst-case scenario (in terms of congestion) in which all links are heavily loaded. Furthermore, a simple Equal Cost Multi-Path (ECMP) forwarding with memory per flow [140, 132] is used in order to perform load balancing without introducing jitter or the need for reordering caused by disjoint path usage by the same flow.

5.3.1.1 Configuring RINA ΔQ-POLICIES

Traffic classes in the scenario give some approximation on QoS requirements derived from user expectations (assured bandwidth, low latency/jitter, etc.). However, more specific information is needed for the distinct types of flows in order to complete the DIF configuration.

It can observed that, while there are requirements for low latency, the stationary latency caused by the distance between nodes may be higher than any delay introduced by nodes due to medium or moderately high congestion. This has two important implications when configuring the DIF. Firstly, it is clear that following the shortest paths (in terms of hops) may not provide optimal latency for delay-sensitive flows. Instead, the total stationary latency can be used as a metric for routing those more urgent flows, resulting in an overall improvement of their experienced performance in terms of latency for urgent flows. Secondly, although the stationary factors may dominate the average latency, the

congestion in network nodes is the source of jitter. In order to control this jitter, strict requirements on the degradation of latency are considered in each hop, in terms of packet service times (PST, i.e. the number of packets preceding their departure from queues).

Apart from considering latency and jitter degradation, the focus is the sharing of loss degradation amongst competing flows. Primarily, given the importance of inter-DC traffic, it is required to make them almost lossless, even in overloaded links. After assuring that, the configuration will try to guarantee that sBE and BE flows do not suffer excessive losses upon high congestion. Particularly, the aim is ensuring that sBE flows do not suffer consecutive packet losses, as this can negatively affect real-time and streaming applications' performance.

To reify these goals, the following end-to-end requirements (w/ high probability) are set on flows:

1. GU traffic must experience zero losses up to at least 150% aggregated load (i.e., relative to the total link capacity and with all types of traffic equally scaled).
2. SN traffic should be supported without losses up to at least 120% aggregated load.
3. Losses of sBE and BE traffic should be below 0.05% up to at least 95% aggregated load.
4. GU and sBE flows should not exceed 50 PST of variable latency up to at least 120% aggregated load.
5. SN and BE flows should not exceed 1000 PST of variable latency up to at least 120% aggregated load.
6. sBE flows should not lose more than 3 consecutive packets up to at least 110% aggregated load.

5.3.1.2 QoS to Cherish/Urgency classes

Requirements of the GU, SN, sBE and BE traffic classes can be mapped into the 4x2 Cherish/Urgency matrix shown below in Table 5.3. In order to simplify the configuration, the same worst-case scenario is considered at each node, considering ΔQ in a way that requirements are met in the longest paths, sharing ΔQ equally between each hop. To achieve this, requirements are set per hop for the different levels of cherish and urgency depending on the current aggregated load on the link, subsequently illustrated in Tables 5.4 and 5.5.

Being in a RINA scenario where the load of the flows can be controlled at source and destination IPCPs in a trusted way, the requirement for flow policing is minimized, so it is omitted from the analysis. Considering that, instead of using the full QTAMux

TABLE 5.3: QoS to cherish/urgency classes

Cherish/Urgency	**"Lossless"**	**Max Cherish**	**Mid Cherish**	**Min Cherish**
Urgent	GU	-	sBE	-
Not urgent	-	SN	-	BE

TABLE 5.4: Maximum avg. loss (%) per load per hop

Load/QoS	0.90	0.95	1.00	1.20	1.50
GU	0	0	0	0	0.001
SN	0	0 0		0.001	0.01
sBE	0.001	0.04	0.2	-	-
BE	0.002	0.05	0.3	-	-

TABLE 5.5: Maximum PST req. per load per hop

Load/QoS	<10	<10	<10	<10	<15
GU, sBE	0	0	0	0	0.001
SN, BE	<75	<125	<200	<225	<250

scheduling policy, the scheduling policy is configured only with the C/U Mux, without shapers, having only then a C/U Mux based on: i) a first heuristic threshold; ii) a second absolute threshold of total buffer occupancy. When the total occupancy reaches the heuristic threshold, i.e., congestion starts to occur, a proportion of the incoming packets will be independently randomly dropped. With this simple heuristic threshold, the probability of consecutive losses can be largely reduced before reaching the absolute threshold while, at the same time, allowing the different cherish classes to better share losses.

5.3.1.3 ΔQ analysis

As stated before, the triad Bandwidth-Latency-Loss has two degrees of freedom. In this scenario, a worst-case is assumed where the input rate is fixed at the maximum load, and have to decide how to share ΔQ between delay and losses. In the common case, having largely elastic traffic would cause the average load to not exceed 100% by much, however transient traffic loads can be higher, which is why this example want to ensure QoS guarantees even under considerable overloads. With the fixed load assumption, it is possible now to know the ratios between the offered load of the different QoS cubes. These ratios under the considered link load levels (90, 95, 100, 120 and 150%) are used then to configure the resources in the different nodes in a way that accomplishes most requirements when possible.

This configuration should be computed per node. However, here, it is simplified to a single configuration that ensures the requirements in worst-case scenarios, thus obtaining a compliant configuration for any node (although most probably not the optimal one). To this end, it is required for configurations to ensure per hop requirements for all QoS classes in scenarios where the ratio of load between GU/SN and sBE/BE is 1:9, 2:8 and 3:7 (1:4 between GU and SN and 3:7 between sBE and BE). Given that the offered load is fixed, with the C/U-mux configuration above, the only remaining point to configure is the different buffer thresholds. In order to configure such thresholds, the analytical approach described in [5] is considered. To include the use of the first threshold in the analysis, the most pessimistic view of introducing PDU drop is considered, namely: for each queue, for lengths beyond the first threshold, all PDUs are discarded, while for the other queues PDUs are not discarded until their second threshold. To reduce the possible set of configurations, the probabilistic threshold for each QoS class is fixed at 10 buffers below the absolute one, except for GU for which only an absolute threshold is considered. In addition, the same absolute thresholds is considered for GU and SN (A) and for sBE and BE (B). As for the drop probability applied when the heuristic threshold is exceeded, that is set to 0.1 for SN and sBE and 0.2 for BE.

Table 5.6: Parametrised buffer thresholds per QoS

QoS	Heuristic Threshold	Drop Probability	Absolute Threshold
GU	120	1.0	120
SN	110	0.1	120
sBE	90	0.1	100
BE	90	0.2	100

After analysing the possible configuration ranges, the best one was the one presented in Table 5.6, which fulfils the imposed requirements while allowing for a small margin of error in case of having non-Poisson arrival patterns.

5.3.2 Simulation results

In order to validate the configuration of the scenario, a simulations with the RINASim is performed, a simulation software developed in the FP7 PRISTINE Project [104, 113]. As the DIF is configured in a way that ΔQ requirements are ensured per hop in a worst-case scenario, the first test aims to check whether the expected behaviour computed via analysis of loads and requirements is delivered in a simple single-hop scenario. Then, the service provided in the full scenario is considered, analysing how well the Δ-Q based policy compares to an MPLS scenario using Weighted Fair Queue (WFQ) scheduling and a simpler baseline best-effort scenario (BBE).

5.3.2.1 ΔQ configuration validation

In order to validate the modified C/U Mux policy and its configuration, the small scenario in Fig. 5.6 is considered. This scenario is composed of a sequence of three nodes interconnected by two 10Gbps links. For each offered load value and ratio between inter-data centre flows and best-effort ones (1:9, 2:8 and 3:7), a 10s simulation is performed. In each run, 1000 flows are established, served following an exponential distribution and with average data rate distributed between 7 and 13 Mbps (At 100% load) and spread between the distinct QoS given the stated rates between QoS. To add more randomness, packet sizes were uniformly distributed between 2KB and 4KB. For simplification, in this and later simulations, the use of abstract links for the shim-DIFs is considered, adding only delay per PDU based on the PDU size and link rate, but without considering other peculiarities, like the inter-frame gap of Ethernet flows or similar peculiarities of the medium.

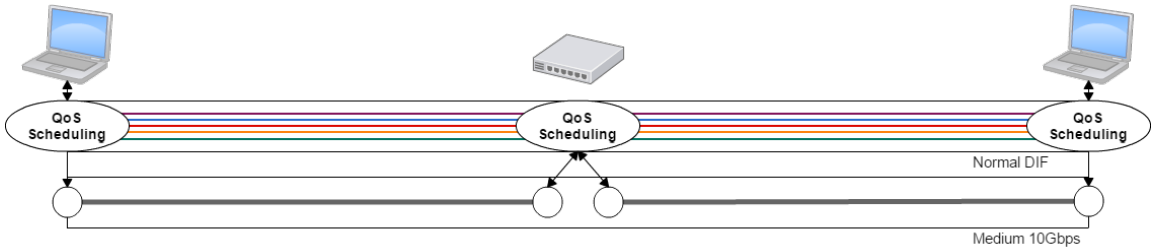

FIGURE 5.6: DIF configuration in the service provider scenario

Fig. 6. For single hop flows at distinct loads: a) Average drop (%); b) Average delay; c) Max delay.

The results from these first experiments confirm those from the analysis. As shown in Fig. 5.7, inter-DC flows (GU and SN) remain practically lossless in all the experiments, successfully fulfilling their loss requirements. In addition, the losses experienced by sBE and BE flows are within the allowed limits and their losses match pretty much the overload in the network. In terms of latency, as shown in Fig. 5.8, urgent flows encountered less than 10 preceding packets, having to wait on average for 0 or 1 packets to be served, as shown in Fig. 5.9. Less urgent flows also remained within acceptable delays, correctly shared between both classes. As a note, is interesting to see how the increment in losses for sBE and BE flows resulted in smaller latencies for non-urgent flows given the changes on accepted loads for each QoS.

5.3.2.2 Simulating the full backbone DIF

Having validated the configuration of isolated nodes, next are the backbone DIF-wide experiments. For these experiments, the traffic matrix is computed in a way that all links are equally loaded, as mentioned in the previous section. The previously stated

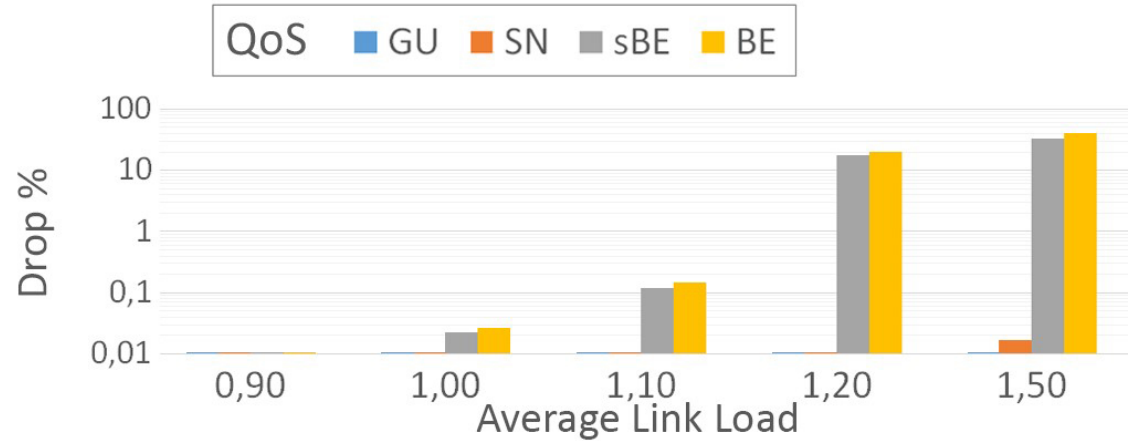

FIGURE 5.7: Average drop (%) for single hop flows at distinct loads

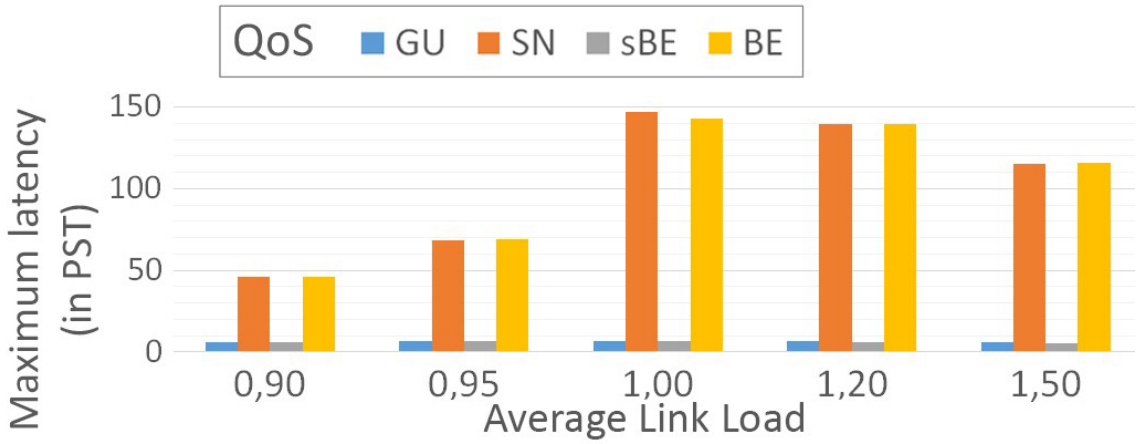

FIGURE 5.8: Maximum delay for single hop flows at distinct loads

distribution of bandwidth for a full load is considered, allocating the same type of flows (7 to 13Mbps at 100% load, 2-4KB PDUs, etc.) between the distinct pairs of nodes, and then scaling the flow data rate to the desired DIF load. For inter-DC traffic there are set up 40 GU and 160 SN flows (Avg. 2 Gbps at 100%) between DC-A nodes and 20 GU and 80 SN flows between DC-B nodes (Avg. 1 Gbps at 100%). In addition, as the main interest is the degradation of sBE and BE flows, 30 sBE and 70 BE extra flows are set between those same pairs of nodes, enough to get comparable data with respect to the other two classes. Finally, the network is filled with multiple point-to-point sBE and BE flows, in a 3:7 ratio, between all pairs of nodes in order to reach the targeted 1000 flows per link.

Regarding the considered scenario, there are two points to note. Firstly, a worst-case scenario is considered, with no congestion control in use, whereas RINA allows (and promotes) a multi-layer congestion control, with fast detection and reaction, which would reduce the rates of sBE and BE flows once network congestion starts to happen. Nonetheless, as congestion control is disabled in these tests, the resulting scenario is one where only scheduling actions are taken to respond to congestion problems. Secondly, statistics

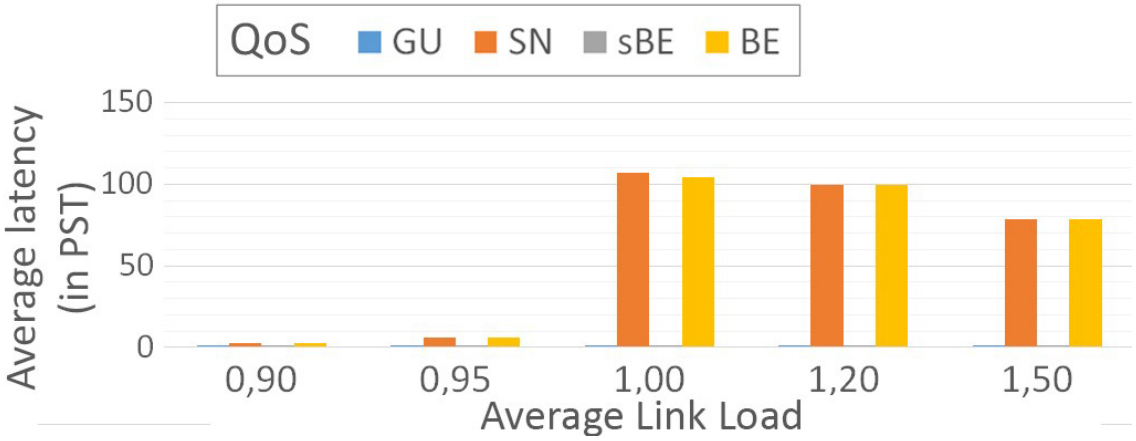

FIGURE 5.9: Average delay for single hop flows at distinct loads

are only computed for flows with multiple hops whose ΔQ increases along their paths, whereas "dumb" flows used to fill links are point-to-point. This means that their ΔQ does not affect the behaviour at other nodes, that is, the losses at one congested node do not reduce the incoming rate of PDUs at downstream nodes, which would happen if the network had been filled with multi-hop flows.

The goal of this section is not to simply ensure that end-to-end requirements are meet when using RINA + ΔQ, but also compare these results with other solutions currently in use, in particular a baseline entirely Best-Effort scenario (referred as BBE) and an MPLS-based VPN [13,14]. In both cases, the same routing is used, for fair comparison purposes, where urgent flows use latency as the metric to optimize and the rest use number of hops. For the Best-Effort scenario, a simple FIFO queue is used at each node with the same 120 packets absolute threshold as the one used for the GU class. For the MPLS-based case, a Weighted Fair Queuing (WFQ)-based scheduling is configured where 260 buffers are distributed as follows: 80 for GU and SN and 50 for each best-effort class. In this last case, in order to ensure the loss levels for GU and SN flows, 40% of the available bandwidth is reserved for GU flows and 30% for SN flows, while the remaining bandwidth is shared between sBE and BE flows following a 2:1 ratio.

From these experiments, some interesting results in favour of RINA + ΔQ-based policies can be found. In RINA, GU and SN flows are lossless, while losses in sBE and BE flows are well distributed and satisfy the requirements previously stated, as can be seen in Fig. 5.10. Those results can also be compared against the WFQ-based scheduling policy configured to satisfy the requirements of GU and SN services, while allowing some differentiation between sensitive and non-sensitive best-effort flows. As can be observed in the same figure, BE flows are the only ones experiencing dramatic losses. These losses also happen earlier, given the division of buffering space and low priority. Regarding the BBE baseline case, while being the last one experiencing losses, as all packets are accepted until reaching the 120 packets threshold, all classes share losses uniformly, failing to ensure GU and SN loss requirements.

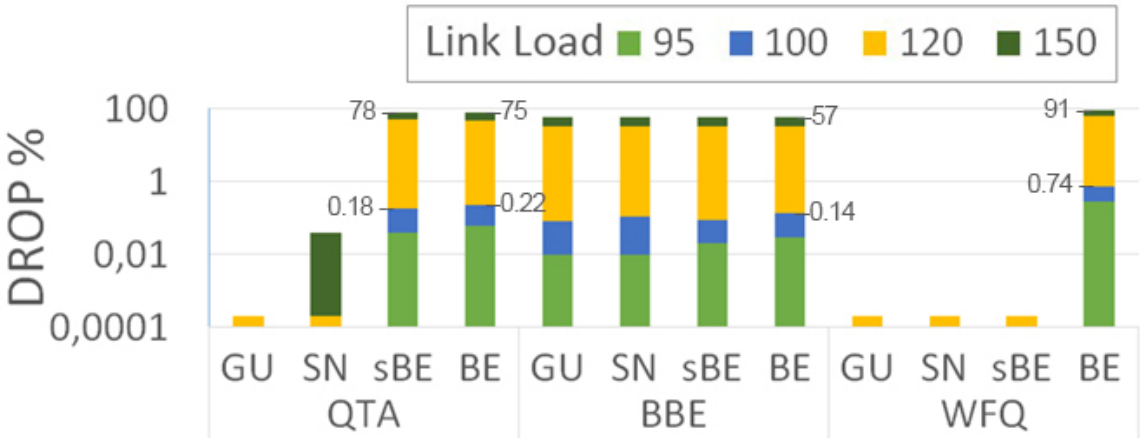

FIGURE 5.10: Average drop for GU, SN, sBE and BE flows depending on the scheduling policy used in the network.

In terms of latency, as variable latency is not of great importance in this scenario, the focused instead is set on the maximum jitter in PST experienced by each traffic class. In the RINA + ΔQ scenario, the requirements per hop are ensured, thus meeting the constraint of ensuring minimum jitter for urgent flows, while also limiting it for the less urgent ones, as can be seen in Fig. 5.11. In contrast, both WFQ and BBE encounter problems. As for BBE, equally sharing the available resources among all flows increases the jitter for urgent flows to unacceptable levels. On the other hand, WFQ provides good service to both GU and SN traffic (even better than required). However, this is achieved at expenses of increasing sBE and BE losses and jitter.

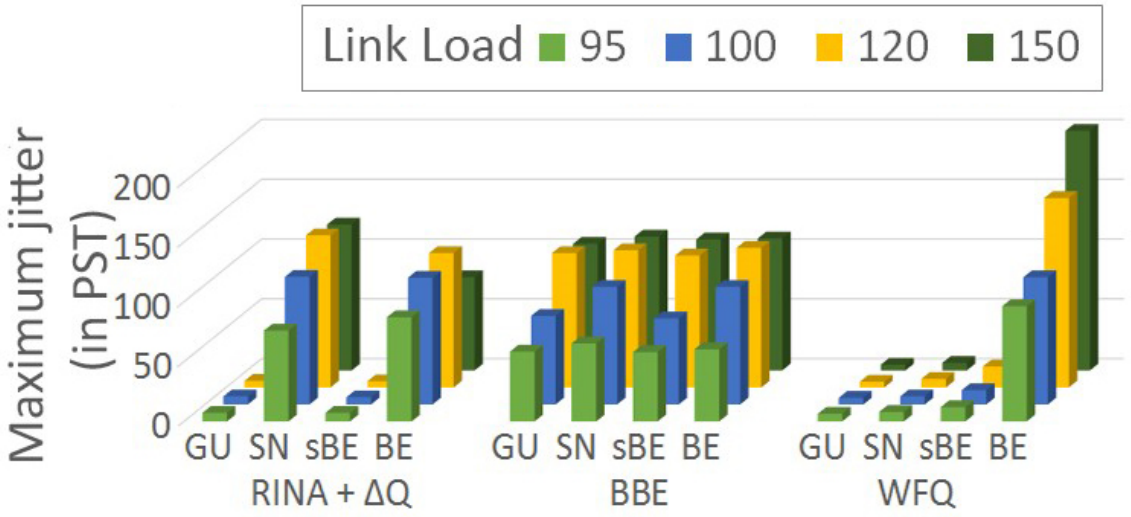

FIGURE 5.11: Maximum jitter in PST for GU, SN, sBE and BE flows depending on the scheduling policy used in the network.

Conversely, in addition to providing better assurance on delay and losses, the RINA + ΔQ-based policies also meet the requirements of avoiding multiple consecutive losses under light congestion. Particularly, for a link load of 110%, even thought there are

necessarily high losses in average, the soft requirement of having at most 3 consecutive losses in sBE flows was pretty much fulfilled. Under higher loads, it becomes nearly impossible to avoid consecutive losses given the multiple points of congestion. However, those are still limited to, for example, less than 1% situations of more than 3 consecutive packets for an offered load of 120%.

5.4 QoS assurance with the QTAMux

The focus of the previous section was set in asserting the benefits of QoS assurance versus simple QoS differentiation. For that, a simple C/U Mux scheduling policy with statistical analysis of flows was used to show how a correct configuration of the network can affect the provided service. After reaffirming the benefits of QoS assurance, this section analyses the use of the complete QTAMux scheduling policy in a more constrained and realistic scenario, assesing the behaviour of the QTA Mux scheduling policy by the means of simulations mimicking the more random behaviour real networks.

5.4.1 Scenario under study

For these experiments, a "distributed cloud" scenario was considered, where, instead of owning the physical network, connectivity between nodes is provided by a third party network. Specifically, this scenario emulates a cloud networks over a sub-set of Amazon AWS infrastructure [148], depicted in Fig. 5.12.

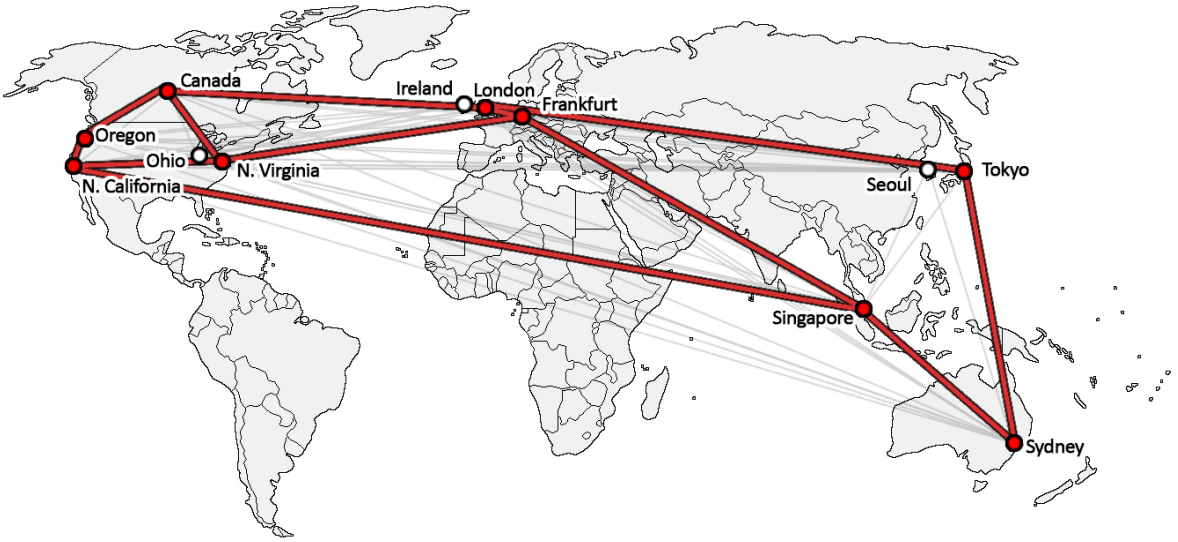

FIGURE 5.12: Overlay cloud backbone network over Amazon AWS infrastructure.

This sub-set of Amazon AWS locations gives us a 9 nodes fully connected backbone, with each node also having external connectivity. In order to reduce the reserved resources,

our cloud networks do not take use of all the available links in the backbone, but reserve capacity in only a small fraction of those (12 from the 72 available) and route traffic between nodes when required. A complete cloud scenario, as depicted in Fig. 5.13, would be divided at least in two different networking domains. First, in the backbone of the cloud, connecting the different border routers with the reserved links, there is the Cloud Backbone DIF. This DIF has two main functionalities, provide connectivity between the different locations, and aggregate the different flows traversing it (enhancing flow control). On top of that, there is the Cloud DIF. This DIF provides connectivity between the different applications running in the cloud servers, as well as connectivity to the cloud clients.

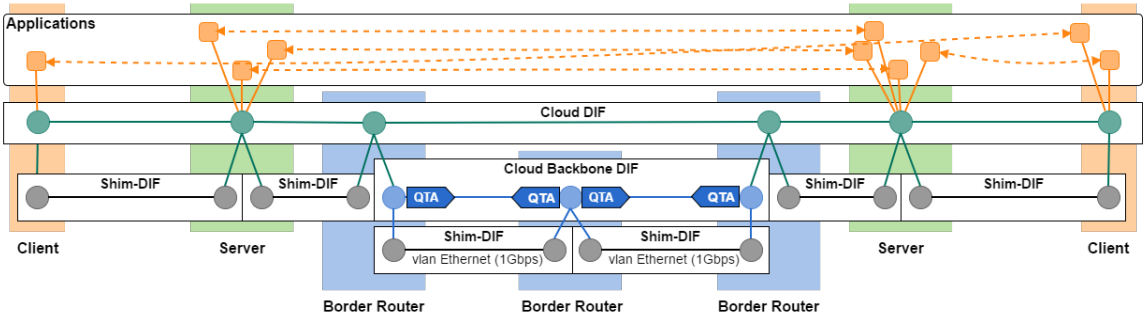

FIGURE 5.13: DIF layering of the full cloud scenarios.

Given the large amount of resources that emulating the whole scenario would require, tests are limited to only one the cloud backbone, generating traffic directly in the border nodes. In this scenario, for each pairing of nodes, a link emulating a 1Gbps vLan is considered, with added latency equivalent to that between the relative AWS locations (previously computed).

In this network, a 3x3 C/U matrix is considered, defining the possible QoS Cubes. Then, in the cloud backbone DIF, the QTAMux policy is configured having P/S with rate limited as shown in table 5.7. As a note, while table 5.7 defines a limit for the C3 QoS Cube, that is only the remaining bandwidth left for the purpose of scenario, being in truth the P/Ss for C3 QoS Cube configured to not limit traffic.

TABLE 5.7: Defined QoS Cubes and P/S bandwidth limits.

Cherish/Urgency	**Max Cherish**	**Mid Cherish**	**Min Cherish**
Max Urgency	A1 - 50Mbps	A2 - 50Mbps	A3 - 100Mbps
Mid Urgency	B1 - 50Mbps	B2 - 100Mbps	B3 - 150Mbps
Min Urgency	C1 - 100Mbps	C2 - 150Mbps	C3 - 250Mbps*

In this network, the following 5 kinds of traffic are considered, being some of the most common applications in the current Internet:

- Voice (QoS Cubes A1 and A2). Based on G.722 [162] at 128Kbps. Multiple small flows with PDUs sent at constant intervals at 50Hz, and their size depending on voice and silence periods. Do not requires retransmission on losses.
- Video (QoS Cubes B1 and B2). Based on YouTube HD and fullHD qualities [163]. Multiple big flows of 1.5 and 3 Mbps each. Requires retransmission on losses.
- Data (QoS Cubes B3, C2 and C3). P2P like flows. Multiple flows at maximum 2Mbps. Requires retransmission on losses.
- Signalling (QoS Cubes A3). Large flow sending constant updates at maximum possible rate. Do not requires retransmission on losses.
- Information Updates (QoS Cubes C1). Large flow sending constant updates at maximum possible rate. Requires retransmission on losses.

Given these different applications, the traffic matrix for each scenario as follows is created as follow. First, in order to avoid random behaviours, a static forwarding is configured, and the network filled to the point of congestion. The traffic matrix is computed in an iterative way, adding 10 Mbps of traffic between each pairs of locations at each iteration. After each iteration, for each overflowing link, the path traversing them are removed from the next iteration. This stops when there are all flows are overflowing. After this first step, it is known how much bandwidth the network will support between each pair of locations. Then, considering the bandwidth limit for each QoS, that is distributed between the different applications as follows:

- Voice. Set as much voice flows as the available rate divided by 80Kbps, rounded down.
- Video. Set as much FullHD flows (3Mbps at application level) as possible, then add HD flows if possible (1.5Mbps at application level).
- Data, Signalling and Information Updates. A unique flow with all the available bandwidth (overbooked it 1.3 times for C3 traffic).

It has to be noted that, although voice flows have a maximum rate of 128Kbps (in the upper level), given their large number, statistically those can be securely allocated closer to the average bandwidth rather than their maximum.

5.4.2 Experimental results

Given the large variation on path latency, in this scenario there is a large variation between flow services. For example, flows traversing short paths (e.g. London to Frankfurt or

Oregon to N. California) will not only suffer from small delays, but as those only traverse one link, those will also suffer a lot less potentially losses due to congestion. On the other hand, flows traversing slower paths (e.g. Oregon to Sydney or Tokyo to N. California) will suffer a higher overall ΔQ, not only with large latencies, but with increased probability of being dropped due to congestion, as those also traverse more nodes. With that in mind, instead of focusing in an average service, that would be biased given the smaller number of extremely bad cases, results focus in one of the possible "bad" cases, specifically, flows from Sydney to N. Virginia, following those the path Sydney → Singapore → N. California → N. Virginia.

For the experiments, an ad hoc simulator has been used. While having a discrete behaviour, the simulator has been configured to mimic the peculiarities of real scenarios that make them less predictable (e.g. competition between processes for CPU time or non-exact sleep times), resulting in behaviours similar to those observed when using the RINA SDK [118] in physical machines. First, in order to verify the scenario, we performed some test in a "perfect" scenario, where all actions where performed with precise in-simulation times. These simulations resulted in the expected behaviours with respect to the amount of traffic in the network, something with special importance for voice flows. In this case, while the aggregate of voice flows for both QoS Cubes had a peek bandwidth that highly surpassed the 50Mbps limits (over 75Mbps), an inspection of the different emulated links showed that most averaged really close to those 50Mbps (it has to be noted that shaping already avoided the aggregate to surpass that). On the other hand, not only the average was close to the statistical limit, but we can be sure that there were not big bursts of traffic surpassing that, as, as can be seen in Fig. 5.14, shaping in voice flows (QoS Cubes A1 and A2) did not suffer from losses due traffic shaping.

With respect to the service provided to the different QoS Cubes, in terms of losses, Fig. 5.14 shows that those are distributed correctly according to their position in the the C/U Matrix, with no losses in *1 traffic, only few losses in *2 flows and most losses on *3 traffic (it has to be noted that those are mostly caused to the overbooked C3 flows). Then, with respect to urgency, while results still resemble what we expected, there are some variations due to the use of shaping. In this case, in Fig. 5.15 we can see how latency is added to the different QoS Cubes (not considering the fixed latency added by the distance between nodes, in this case 213 ms). As can be seen, while latency is more or less correctly distributed, there are some cases where that is not so true. In the first case, in general, as the cherish priority lowers, also does the latency suffered by flows. While this is something commonly caused by the fact that PDUs on those flows are lost under high congestion, in this case there is also the fact that highly cherished traffic also has more strict bandwidth limits, resulting in higher latencies during shaping. In a similar way, we found that, for non-cherished traffic, urgent flows (A3) suffered higher latencies that less-urgent ones (B3). In this case, this variation is due multiple factors, being the most important the use of traffic shaping, as signalling traffic (A3) was designed to always reach the imposed limit, thus having most PDUs suffering the effects of shaping.

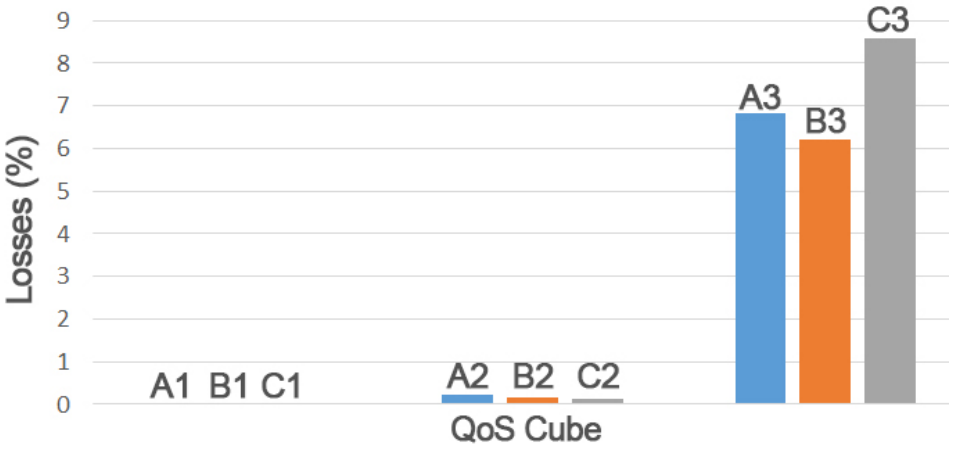

FIGURE 5.14: Distribution of losses in between flows from Sydney to n. Virginia ("perfect" scenario).

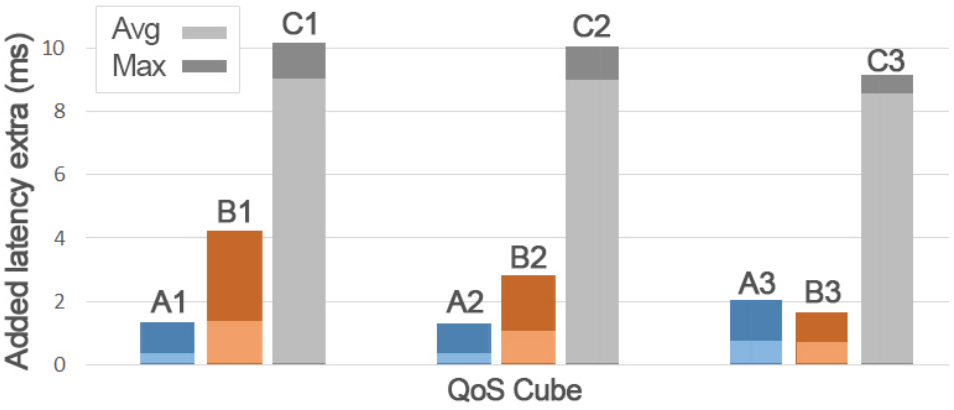

FIGURE 5.15: Latency of flows from Sydney to n. Virginia ("perfect" scenario).

Then, to assert the behaviour of the policy in a more production-like environment, we firstly compared the "perfect" results to those of a more wild environment. In order to do that, the new simulations not only delayed randomly the execution of simulated applications and IPC calls, but also added bigger buffers on the simulated links, limiting the control that the QTAmux policy has over the out-going flows. In this regard, Fig. 5.16 and Fig. 5.17 show a comparison between the results in Fig. 5.14 and Fig. 5.15 respectively. As can be seen, while there are small variations between the different scenarios (due to the randomness in the applications and difference between simulated environments), the behaviour of flows is maintained in general close to that seen before, suffering most flows slightly higher degradation in this second case.

After ensuring the correct behaviour of the QTA Mux policy in a less predictive scenario, the next step was to check the specific behaviours of the different applications and the variation between the ΔQ received by the different flows of the same kind. In this case, we focused on the different voice flows in that one path between Sydney and n. Virginia. Fig. 5.18 and Fig. 5.19 show respectively the average and maximum added latency that the different flows suffered. While there is a clear difference between the maximum and

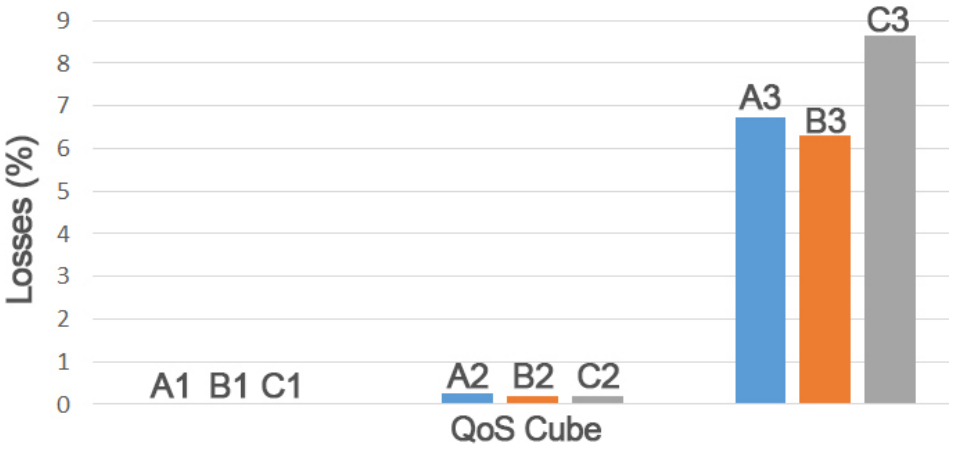

FIGURE 5.16: Distribution of losses in between flows from Sydney to n. Virginia ("production" scenario).

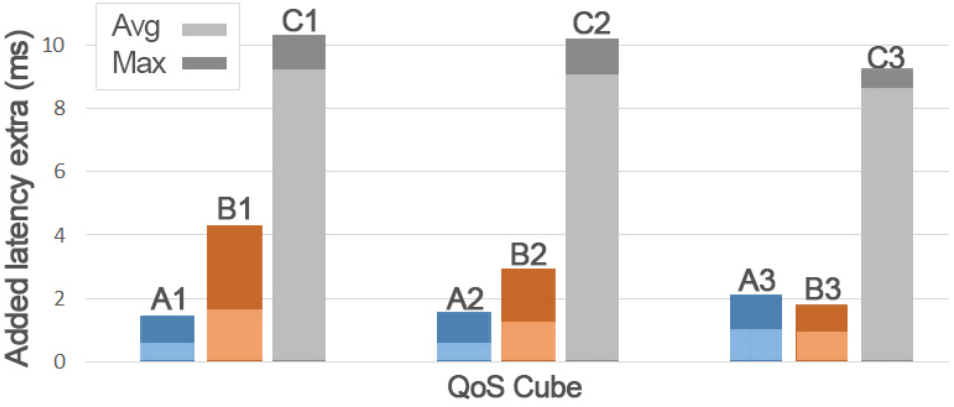

FIGURE 5.17: Latency of flows from Sydney to n. Virginia ("production" scenario).

average delay suffered, suffering up to 1 extra ms of latency en some cases, the behaviour between the different flows remains consistent, being all within acceptable margins (jitter of all voice flows within 5% of the inter-frame time of 20ms). Even so, it has to be noted that, in this scenario, although suffering small losses, voice flows using the A2 QoS Cube suffered slightly less jitter than those using the A1 QoS Cube, probably due the suffered losses.

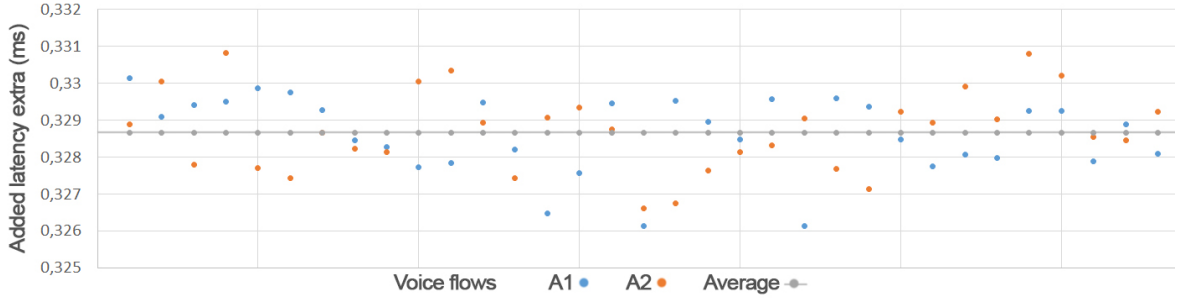

FIGURE 5.18: Average added delay of voice flows from Sydney to n. Virginia ("production" scenario).

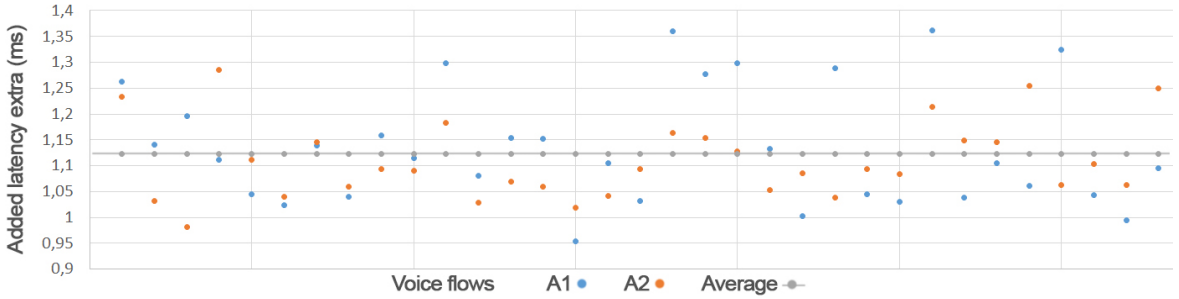

FIGURE 5.19: Maximum added delay of voice flows from Sydney to n. Virginia ("production" scenario).

It has to be noted that, while the simulator mimicked the observed behaviour of real hardware (unlike most discrete simulations with precise timings), these tests did not contemplated the use of congestion control mechanism. Given that, the obtained results cannot be directly extrapolated to real networks, as those would reduce the amount of non-cherished traffic before reaching those levels of losses. Even so, in terms of suffered latency and losses on more cherished flows, simulated results should provide a good "worst-case" approximation, as those are less affected by congestion control mechanisms in the long run, and even should exhibit an even better behaviour with less competition for resources (after reduce the non-cherished traffic).

5.4.3 Conclusions

Real environments suffer from more random behaviours (e.g. competition between processes for CPU time) than perfect ones, making them less predictive. Given this, it is not enough to ensure that the behaviour of scheduling policies works as expected on the more controlled simulated scenarios as done until now, but also to ensure that those behaviours are also correct in more realistic scenarios where not everything happens when expected. In that regard, this section shows how the QTA Mux policy is capable of correctly share ΔQ between the different flows traversing the network in base to their requirements in terms of both cherish and urgency. In order to do that, a cloud backbone scenario has been used, based on the current infrastructure of Amazon AWS services, supporting different common applications (voice, video and data). Results on that networks shows that, not only ΔQ is correctly shared in most cases, but also that the behaviour of the policy when all process become less predictable still close resembles that of a perfect scenarios.

5.5 Enabling QoS for End-Users

Between other limitations to effectively handle the nowadays' increasing variety of heterogeneous distributed applications, the current TCP/IP-based Internet lacks a standard

way for such applications to express their QoS requirements, forcing network managers to guess them based on the transport protocol used (i.e., TCP, UDP), ports, or past information. In this regard, RINA emerges as an enhanced QoS-aware environment capable of providing a complete QoS centred environment. With the use of the QoS Cubes, RINA allows a differentiation of flows based on QoS requirements, rather than in simple priorities. In this context, a DIF is responsible of ensuring that the service provided to supported flow meets the levels defined in the associated QoS Cubes. As networking resources are limited, RINA policies have to manage how they can be shared among flows to meet all their QoS demands.

When considering end-users, a common treat is their greediness with respect to computational resources. If nothing avoids it, most applications will request more resources, less delay, more security, etc. than they really needs. Allowing these behaviours may result in undesired scenarios. For instance, all flows may receive the best QoS Cube in the network, resulting in a best-effort treatment that would remove most of the benefits of having different QoS Cubes. Even worst, few applications may try to work by the rules and request their required QoS, ending receiving a degraded treatment with respect to the rest of flows filling the network with high priority traffic (that will still receive an almost best-effort treatment). Especially in large networks, like the current Internet, this is a huge problem, as a large overbooking of resources and congested nodes is the norm.

This section focus on rate limiting policies that would enforce strict limits on what end-users can request to the network, avoiding this way an unfair sharing of resources. The rest of this section is organized as follows. Subsection 5.5.1 introduces a new ΔQ-based rate limiting solution for home users. Subsection 5.5.2 provides an in-depth description of the rate limiting scheduling policy. Finally, Subsection 5.5.3 provides some initial results comparing the proposed solution to the RINA's default per-Flow rate-limiting.

5.5.1 ΔQ-based Rate limiting for end-users

Thanks to QoS Cubes and a dynamic management, RINA is able to provide dynamic and reliable knowledge of the network requirements and its traffic at any time. Using this knowledge, ΔQ-based policies have demonstrated reliable QoS assurances even in overbooked network scenarios, without relying on over-degrading low QoS traffic. ΔQ-based policies use different properties to map each QoS Cube offered by a DIF into a Cherish/Urgency matrix enforcing relative latency and loss requirements. Even so, as stated, in order to be able to provide those assurances, it is required to enforce some limitations on what the user can request.

In this regard, an important scenario to consider is that of the edges of the network (e.g. as depicted in Fig. 5.20), as it may happen that end-user's applications greedily request the highest QoS Cube for all their flows across the network. This must be avoided, as no QoS assurance would be possible otherwise (e.g. like in the current Internet). Hence,

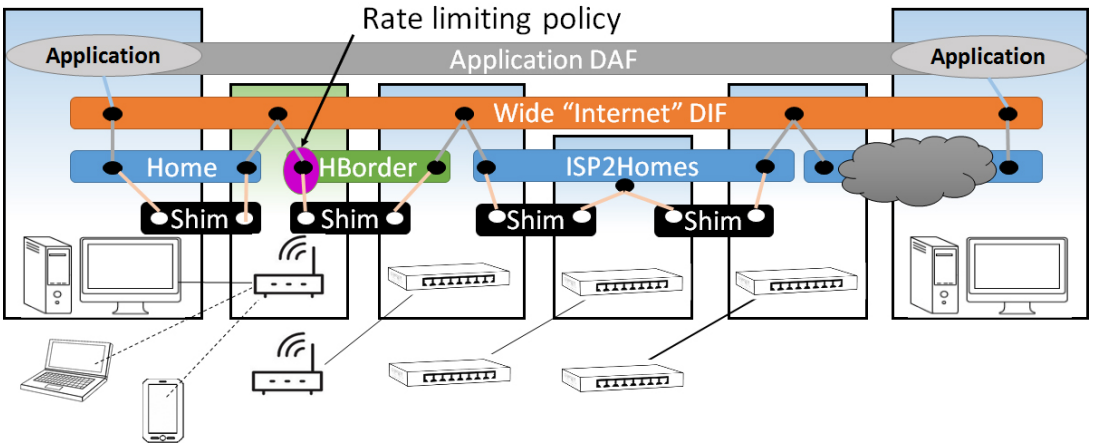

FIGURE 5.20: RINA network provider scenario structured in DIFs of increasing scope, from the physical transmission medium to the applications.

network providers must impose limitations on the quality of end users' traffic, imposing some limitations by contract in order to avoid the above-mentioned situation.

As seen in Chapter 4, RINA already provides ways for enforcing rate limits for each QoS (e.g. QoS Cubes with a maximum rate lower than the capacity of the link.). Even so, those methods act only in a per-QoS or per-flow basis, not considering the resources that other flows may be taking (or the lack of them). This ends resulting in two opposite scenarios:

- The ISP enforce restrictive rate limits to the QoS Cubes (sum of rates up to 100% of link rate).

  - Good points: The ISP ensures that the usage of high priority QoS Cubes does not surpass some strict limits.
  - Bad points: This affects the users that cannot use all their resources if not having flows of each QoS, each at the maximum allowed rate.

- The ISP enforce loose rate limits to the QoS Cubes (sum of rates over 100% of link rate).

  - Good points: Users are always capable of using all their resources.
  - Bad points: Users may use more resources than allowed if only request good QoS Cubes.

As an example, let us consider the configurations in 5.8 and 5.9, being instances of strict and loose configurations for the same small scenario, with QoS Cubes matching a 3x3 C/U Matrix.

From the point of view of the ISP, in the first case, the strict limits on the amount of traffic allowed for each QoS Cube means that it has full control on the maximum amount

Table 5.8: Strict rate-limiting scenario for 3x3 C/U Matrix

Cherish/Urgency	**Max Cherish**	**Mid Cherish**	**Min Cherish**
Max Urgency	A1 : 5%	A2 : 7.5%	A3 : 10%
Mid Urgency	B1 : 7.5%	B2 : 10%	B3 : 15%
Min Urgency	C1 : 10%	C2 : 15%	C3 : 20%

Table 5.9: Loose rate-limiting scenario for 3x3 C/U Matrix

Cherish/Urgency	**Max Cherish**	**Mid Cherish**	**Min Cherish**
Max Urgency	A1 : 5%	A2 : 10%	A3 : 20%
Mid Urgency	B1 : 10%	B2 : 30%	B3 : 40%
Min Urgency	C1 : 20%	C2 : 40%	C3 : 100%

of traffic, both in terms of cherish and urgency (e.g. at most 22.5% of traffic will be of max urgency, 32.5% of mid urgency, etc.). In the second case this control is more loose, relaying more in the expectation that the use of QoS Cubes with high requirements for either cherish or urgency will not happen at the same time (e.g. if the only max urgent traffic is non-cherished (A3) it may take up to 20%, but if all traffic is urgent (A1, A2, A3), it may take up to 35% of the link's rate).

On the other side, we have the point of view of the users. In the first case, users find the usage of the resources highly limited, being impossible to use all the resources if not having at least one flow of each available QoS Cube. This results in cases where, in order to get more resources for a flow with low requirements, multiple high-requiring flows are used (e.g. using 4 flows, A1, A2, B1 and B2, to transmit mid-urgent, mid-cherished traffic at 30% of link's rate). On the other hand, with looser limits users are capable of better redistribute their resources, being able to use their entire available link rate in a more free way.

As can be seen, both strategies have their pros and cons, but none provides a real way to consider the different dimensions of QoS requirements. Considering that, we propose an N-Dimensional rate limiting policy to control the traffic injected by the end-users into the network. The proposed rate limiting policy manages each of its N dimensions independently, setting different thresholds in the amount of resources that can be used between all flows, while allowing for low requiring flows to use the resources not used, by higher priority ones. Let us explain this with a small example. Considering 1 dimension with 3 possible priorities A, B and C, we set 3 thresholds; one for traffic of priority A, one for the joint traffic of priorities A and B, and one for the whole traffic (the last one commonly being a hard rate-limit of the N-1 flow/link). Given those thresholds, we have that traffic of priority A can use at most up to the first threshold, traffic of priority B can fill up to the second, minus the usage of the first priority, and similar with priority C and the third threshold.

TABLE 5.10: Cherish rate thresholds for 3x3 C/U Matrix scenario

Cherish	**Max Cherish**	**Max+Mid Cherish**	**Any Cherish**
Limit	*1 : 15%	*1 + *2 : 40%	* : 100%

TABLE 5.11: Urgency rate thresholds for 3x3 C/U Matrix scenario

Urgency	**Max Urgency**	**Max+Mid Urgency**	**Any Urgency**
Limit	A* : 15%	A*+B* : 40%	* : 100%

If we consider an scenario using ΔQ policies with QoS Cubes distributed within a C/U Matrix, we find a 2-dimensional rate-limiting policy, with independent limits for both cherish and urgency. In a similar way, using a Cherish/Urgency/Rate Cube (C/U/R Cube), a third extra dimension can be added to also limit the amount of traffic with high expectations of maintain their rates upon congestion. This allows network providers to impose tight limits in the usage of users, while at the same time giving them the freedom to distribute those depending on their needs. For example, considering a similar the previous scenario, the configurations stated on Tables 5.10 and 5.11 would provide tighter limits on the amount of cherished and urgent traffic respectively that the strict configuration of per-QoS limits of Table 5.8. At the same time, it would ensure that the users do not inject too much priority traffic into the network (e.g. at most 15% of traffic has the maximum urgency with respect to the strict 22.5% between QoS Cubes A1, A2 and A3), but allowing for that traffic to be freely distributed between the different QoS Cubes (e.g. that same 15% of traffic with maximum urgency could have also maximum cherish).

It has to be remarked that those limits are considered for border routers on the end-user side. This means that, while cherish and urgency levels can be taken into account while serving the PDUs leaving the nodes, their goal is not to assure ΔQ within its domain, but to enforce rate limitations based on future flow requirements along the path. Given that, a proper path selection of "Cherish and Urgency" levels that considers the whole path for each flow will be crucial to effectively provide the end-to-end QoS assurance required by upper DIFs. Luckily, RINA's flow allocation provides the required mechanisms to abstract multilevel QoS requirements, being possible to dynamically predict the end-to-end treatment of flows. It is noteworthy that, although those limits focus on two specific dimensions (urgency and cherish) in line with ΔQ, providers could define their own QoS dimensions (e.g.,cherish, urgency and packet size), requiring then to provide an appropriate mapping between upper flows and QoS Cubes on flow allocation.

5.5.2 Rate limiting policy

The proposed rate-limiting policy allows to independently enforcing different limitations in the usage of resources for the different QoS requirements. In the case of using ΔQ policies in the DIF, that translates into limiting the usage of cherished and urgent traffic. In order to enforce these rate-limits, we propose the use of a scheduling policy using a credit-based system to enforce each of the different limits on bandwidth usage. In this credit system, each byte sent, including those of the underlying DIFs' headers, cost a fixed amount of credits (K credits/byte), and, whenever the scheduling function is called, each limiting dimension will receive an amount credits ($C = \Delta t * K * \text{rate}$) calculated from the Δt from the last call and the max rate of the shim-DIF supporting the flow. Those C credits then will be divided between the different priority levels depending on their thresholds and the current amount of credits. For example, with a K of 40 credits/byte and a 100Mbps link, for a Δt of 1ms each dimension will receive $C = 10^{-3} * 10^8/8 * 40 = 50K$ credits. Considering the urgency thresholds of Table 5.11, those credits will be distributed as 15% ($7.5K$) for max priority, 25% ($12.5K$) for mid priority and 60% ($30K$) for min priority.

When considering the credit system of the scheduling policy, it is important to consider also the maximum amount of credits that each level can store. If the maximum backlog of credits for a dimension's level is too small (e.g. near the amount required for the Maximum Transmission Unit (MTU)), bursts of high priority PDUs will not be allowed (even small ones). On the other hand, if the backlogs admit too much credits, when filled (e.g. after an idle period), large bursts of priority PDUs may be sent into the network. Luckily, there is a direct relation between the maximum backlog and the maximum possible bursts (the maximum burst time is a function of the backlog, link's rate and credits gained per second). Given that relation, knowing the maximum burst that the network provider would allow is enough for configuring the different dimension's levels (preferably leaving some margin for unexpected behaviours, like random management messages).

For this policy to work, it is also important to consider the behaviour of the different nodes in the user's network. As seen in previously in 5.1.1, in order to fulfil QoS requirements it is important to have a reliable congestion control mechanism capable adapting as fast as possible to the congestion in the network. In this regard, RINA does not only allows to manage congestion at each DIF, but also, as seen in Section 4.2.2.2 of Chapter 4, it is possible to inform of congestion in a DIF either by adding some congestion flags in the headers of PDUs or by directly signalling congestion to the source those PDUs. Taking into consideration the scenario for which this policy is thought, this second approach provides a perfect solution, as the commonly short diameter of home networks allows for extremely fast reactions of congestion control mechanisms. Considering this and the size of allowed bursts, it is possible to configure the different queues and thresholds in a way permits to avoid in great measure the overuse of colliding priority traffic in the home routers (and corresponding added degradation or losses).

The pseudo-code in Appendix C describe the proposed rate-limiting policy. This policy has two main hooks, when a PDU arrives to the RMT port (function *arrival*, line 30) and when the RMT requests a new PDU to be served (function *schedule*, line 44). The *arrival* function, called with a new PDU as a parameter, search for the mapping between the PDU QoS Id and its queue (variations may use flow Id for the mapping or some mixed approach) and, depending on the current size of that, it decides if the PDU has to be stored or dropped and if congestion has to be signalized. The *schedule* function works in 4 phases First it computes the Δt from the last call and add the appropriate amount of credits to each Cherish/Urgency level. Then, it searches for the highest cherish and urgency levels with a positive amount of credits. Then, it searches for the next queue to serve within those with Cherish and Urgency levels equal or under the computed limits. Finally, if a queue has been selected to be served, the PDU is served from the queue and the used credits are discounted from the appropriated levels.

As can be seen, the *getNext* function (line 57) is undeclared within the pseudo-code. This function is used to get the next queue to serve given limitations in the allowed cherish and urgency levels. This function can have multiple implementations, depending on the order expected for queues to be served (sometimes requiring extra data). For example, a FIFO-like implementation could return the queue with the oldest PDU, or a priority-based implementation return the queue with highest urgency then cherish.

An important point for this policy is that, when selecting the valid queues to serve, it considers as valid everything under the highest priority level with credits, not considering if there are really enough credits for the next PDU, as that would prioritize queues with small PDUs and lower priority levels. In the *useCredits* function (line 108), it can be seen that not having enough will result in a negative amount of credits for the used level, after spending all the credits of higher levels. This has to be considered also when regaining credits, as simply dividing the amount of credits between the different levels in a fixed way could result in undesired scenarios (e.g. sending only data of the second priority, at a higher rate than allowed, resulting in the first priority having always a positive amount of credits when called and the second one an increasing negative pool of credits). The *addCredits* function (line 74) considers those and increases the amount of credits in a crescent way, giving away credits from higher priority levels to lower ones whenever those are in red numbers.

There are also some important points that need to be clarified. This proposed policy is thought to be used between end-users and network providers, commonly at home-routers, meaning that both the number of lower DIFs and the size of their headers are known. Within this policy, all N flows are stored in different queues depending of their priority levels (e.g. a queue for each C/U level is using ΔQ), those then are aggregated in the same N-1 flow between the end-user and the network provider (this flow expected to be one of the shim-DIF connecting those two). Finally, although other N-X flows may traverse the lower DIFs (e.g. management flows), the use of such flows may affect the behaviour of

this policy as seen in section 5.2.3), potentially resulting in undesired bursts of priority traffic that will be dropped at the provider's side if this is not taken into consideration.

5.5.3 Numerical results

To assess the proposed rate limiting policy, we have conducted experimental evaluation using the RINA SDK delivered by the FP7 PRISTINE project [104]. We a built the small point-to-point RINA network test-bed depicted in Fig. 5.21. In that scenario, we have two nodes, A and B, emulating a home router and its ISP gateway. To recreate the scenario, we used two laptops using the latest version of the RINA/IRATI stack [118] over a Debian 8 system with kernel 4.9. These two nodes, were connected using a 1Gbps Ethernet link on which a vlan, with its rate limited to 100Mbps, was configured as per connect the two nodes in the RINA environment (using a vlan-ethernet-shim). Over the shim-DIF connecting the two nodes, we set a normal DIF (Home2ISP) providing QoS support. In that DIF, on aggregated flow for each available QoS Cube is allocated. In this DIF, node A is required to ensuring that the different rate-limits are achieved. Finally, we set another normal DIF on top (Net) that would mimic an "Internet" DIF providing communication between applications in both sides of the network.

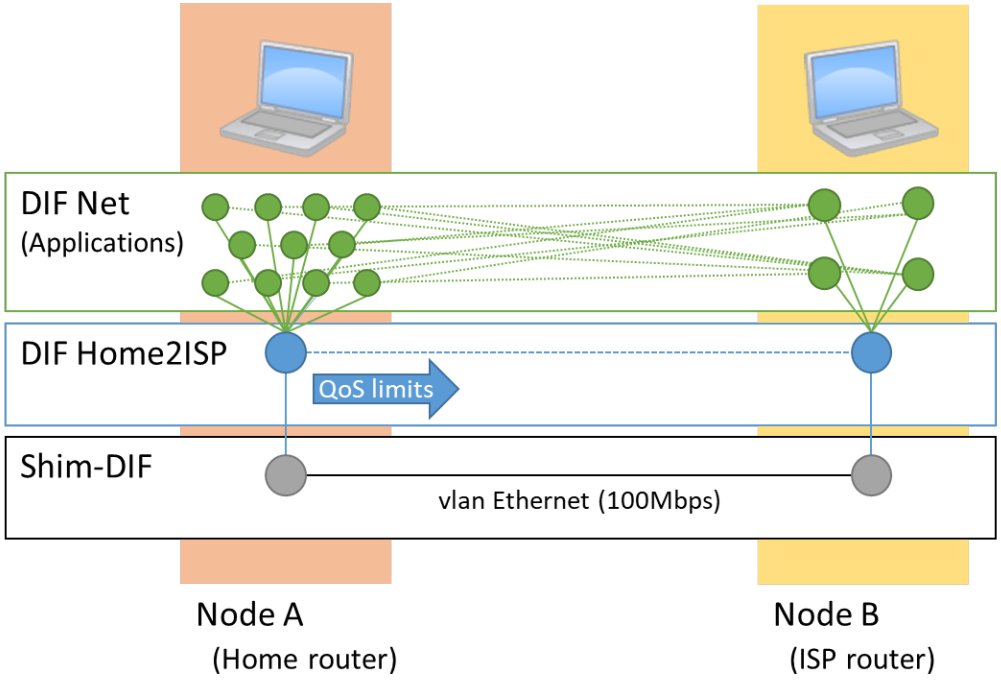

FIGURE 5.21: DIF layering of the proposed home-to-ISP scenario.

For the different experiments, we defined the seven QoS Cubes in Table 5.12, based in a 3x3 C/U matrix. Then, we defined the imposed the rate-limits Table 5.13, limiting the amount of traffic that node A can inject into the network up to each cherish/urgency level. As mentioned before,the mapping between upper flows and QoS Cubes should consider

how these flows are routed across the DIFs to effectively ensure the QoS end-to-end requirements. However,for these tests,we considered a straightforward mapping between application requirements to QoS Cubes (i.e., C/U matrix cell),leaving the end-to-end QoS assurance consideration out of consideration.

TABLE 5.12: Defined QoS Cubes for tests

Cherish/Urgency	**Max Cherish**	**Mid Cherish**	**Min Cherish**
Max Urgency	A1	A2	-
Mid Urgency	B1	B2	B3
Min Urgency	-	C2	C3

TABLE 5.13: Proposed policy : ΔQ Thresholds (Mbps)

Dimension\Level	**Max**	**Max + Mid**	**All**
Cherish	*1 : 15 Mbps	*1 + *2 : 60(+45) Mbps	* : 100(+40) Mbps
Urgency	A* : 15 Mbps	A* + B* : 60(+45) Mbps	* : 100(+40) Mbps

Once the scenario, QoS Cubes and limitations were decided, our first goal was to assert the behaviour of the proposed rate-limiting policy within the specified environment. As to avoid stationary scheduling states, for these tests we used an application that sent packets at constant intervals, but with the size of those varying between a min and max size, all maintaining an average rate of 1Mbps (including all headers). Then, we run multiple experiments were the traffic matrices were configured as to reach in average 100% of the acceptable rates. The first goal of these experiments was to assert that rate limits were enforcer correctly within the network. In order to check that, we used a similar credit based mechanism to the packets received at node B in a post-execution run on a tcpdump of the incoming port. The results of the tests were as expected, asserting that the policy maintained the outgoing traffic under the expected limits. It has to be noted that, while we used a similar method to check the incoming traffic as in the rate-limiting policy, the maximum credit on that was set slightly higher as to amount to the fact that the Ethernet ports have their own internal queues to improve performance, as they run independent to the CPU.

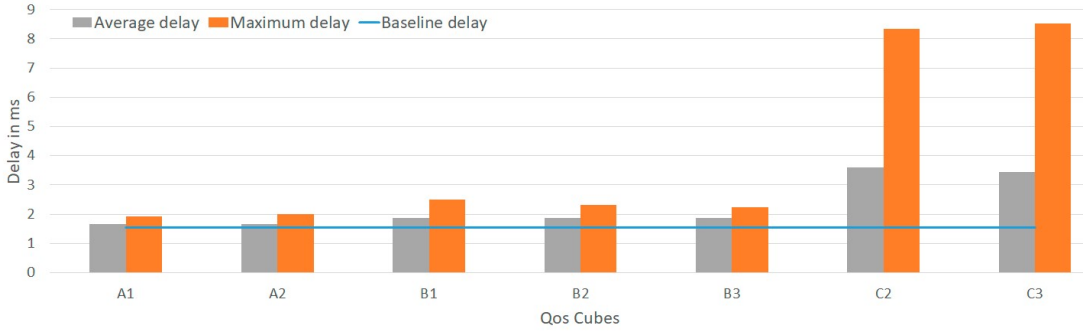

FIGURE 5.22: Comparison of average and maximum delay for the different QoS Cubes.

In addition, we wanted to ensure that the priorities defined by the C/U matrix in Table 5.12 were fulfilled. In that regard, in Fig.5.22 we can see a comparison between the average and maximum delay suffered by the flows of each QoS Cube. As can be seen, the urgency order is maintained in both average and the worst cases. In order to emphasize the effects of the scheduling policy, we also compared it to the average delay in an uncongested scenario, where we ensured for queues to be empty between packet and packet. In comparison with the baseline, we see that urgent QoS cubes (A1 and A2) get almost no delay in average, with its maximum growing up mostly given collisions with packets of the same priority or small bursts. Similar behaviour can be seen in the mid-urgent flows (B1, B2 and B3), but with slightly higher waits in queues given their lower priority. In contrast, non-urgent flows suffer from high delays in comparison. This is as expected, and works as a measure to avoid losses due to the small overbooking of the network (e.g. in the worst cases, we could have burst of up to 120% of the link rate). While the delays are high, it has to be noted that we are considering an overbooked low-rate link in those tests. In that regard, if we consider the preceding packets in queue instead of the time spent there, non-urgent packets only wait for 25 preceding packets in average, 90 in the worst case (100 packets was the drop threshold for non-cherished flows, meaning 0 loses in the tests).

TABLE 5.14: QTAMux RINA policy : Rate limits per QoS Cube (Mbps)

Cherish/Urgency	**Max Cherish**	**Mid Cherish**	**Min Cherish**
QTA:Max Urgency	A1 : 5 Mbps	A2 : 10 Mbps	A3 : –
QTA:Mid Urgency	B1 : 10 Mbps	B2 : 15 Mbps	B3 : 20 Mbps
QTA:Min Urgency	C1 : –	C2 : 20 Mbps	C3 : 20 Mbps
DS	1 : 15 Mbps	2 : 45 Mbps	3 : no-limit

Once the behaviour of the proposed policy is validated, next we want to compare it against the main QoS scheduling policy in RINA, namely, the QTAMux (QTA), as well as to a DiffServ-based policy (DS) [147]. In order to do that, we set a scenario where limits per QoS and limits per quality could be compared, using the same testbed described in Fig. 5.21 and QoS Cubes defined in Table 5.12. For the proposed rate-limiting policy (R-lim), we consider the same limits for cherish and urgency levels described in Table 5.13, and for the QTAMux and DiffServ we consider the limits per QoS Cube described in Table 5.14. It has to be noted that those limits are only a possible configuration for this scenario (ISPs should freely decide or modify the limits they impose to their clients). Besides, we consider three types of traffic:

- Voice flows: Based on G.722 [162]. Constant interval between packets, but size varies between voice and silence periods. Urgent but admits some losses, minimum A2.
- Video: Based on YouTube HD and fullHD qualities [163]. MTU size packets with varying bitrate. Mid urgent, but requires to avoid losses, minimum B1.

- Data: P2P like flows. MTU size packets at maximum rate possible. Non-urgent and can withstand losses, minimum C3.

Given these applications and flow constrain, we constructed our scenarios in a way that the maximum number of voice, video and data flows could be supported without surpassing the limits. With that into consideration, we constructed our traffic matrices as follows:

- R-lim and DS scenarios:
  - 150 voice flows with QoS Cube A2
  - 3 FullHD with QoS Cube B1
  - 4 HD flows with QoS Cube B1
  - 12 5Mbps P2P flows with QoS Cube C3
- QTA scenario:
  - 25 voice flows with QoS Cube A1 and 95 with A2
  - 1 FullHD flow with QoS Cube A1 and 1 with B1
  - 4 HD flows with QoS Cube B1
  - 3 5Mbps P2P flows with QoS Cube B2, 4 with B3, 4 with C2 and 4 more with C3.

Before presenting the results for this scenario, first, it has to be noted that, in this scenario, we have the requirement to maintain the same QoS Cube between layers. This is important, as in this case the DS policy does not degrades the packets that goes over the rate-limit, but drop them as, otherwise, those would regain their priority when reaching their destination. With that in mind, we can realise from the construction of the scenario itself that requirements are better translated into QoS Cubes in the R-lim and QTA scenarios, as those can differentiate not only by cherish, but also by the urgency of flows. In addition, the fewer restrictions in the R-lim scenarios remove the need for differentiating traffic with the same requirements, increasing the amount of flows that can successfully be accepted in the network.

Regarding the network utilization, in Fig. 5.23, we can see a comparison between the amount of traffic successfully sent in the network for each application, as well as the overall link occupation in each case. As can be seen, the amount of successfully sent data of voice and video flows policy results slightly higher with the R-lim and DS than with QTAMux policy. In contrast, data flows are boosted in the QTA scenario. This was something predictable, as less voice and video flows can be successfully accepted with the requirements of the QTA scenario. In addition, in Fig. 5.24, we can see a comparison between the amounts of traffic used by for each QoS Cube in each scenarios.

These results, mainly describing the traffic matrices used, they highlight the need of a fair rate-limiting policy. In summary, as traffic could not use the QoS Cube that better adapts to its requirements in the QTA use case, we end in a scenario where 60% of the outgoing traffic ends assigned to QoS Cube providing a better than required service.

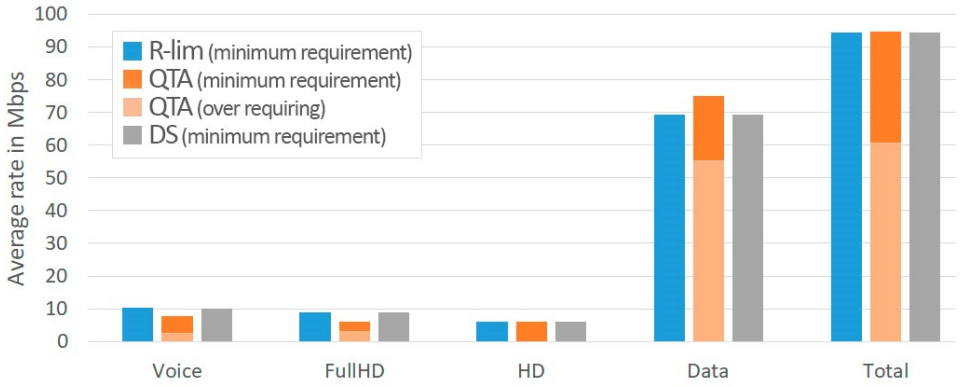

FIGURE 5.23: Comparison of average rate in Mbps by application.

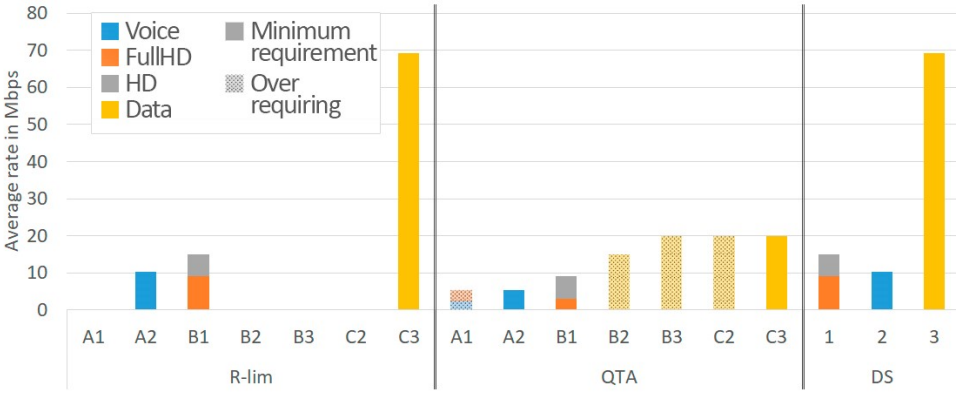

FIGURE 5.24: Comparison of average rate in Mbps by QoS Cube.

Besides the problems that an unfair rate limiting policy imposes to the ISP, a strict rate limitation also affects the service that applications eventually receive. Indeed, when imposing too strict rate limits, we enforce an artificial differentiation between flows with the same requirements. Fig. 5.25 shows a comparison between the service received by each application. In the QTA scenario, voice flows get more or less the same service in each case (all have the same urgency). However, we see oscillations in video flows, where FullHD urgent flows experience a smaller delay than the rest, similarly to that experienced by voice flows. In contrast, mid-urgent flows get slightly higher average delay and an extra 0.5 ms of maximum delay in comparison. In a similar way, we can see how Data flows suffer large variations, near to 1 ms, between the maximum delay of those assigned to QoS Cubes B2/B3 and C2/C3. In comparison, in the R-lim scenario, we see all flows of each application receiving similar services (as expected), but more important, in all suffering lower delays (both in average and in worst case) than the flows sharing the same QoS Cube in the QTA scenario. In contrast to the two ΔQ-based policies, when

using the DS policy, flows do not experiment any visible differentiation in terms of delay, resulting in a best-effort scenario.

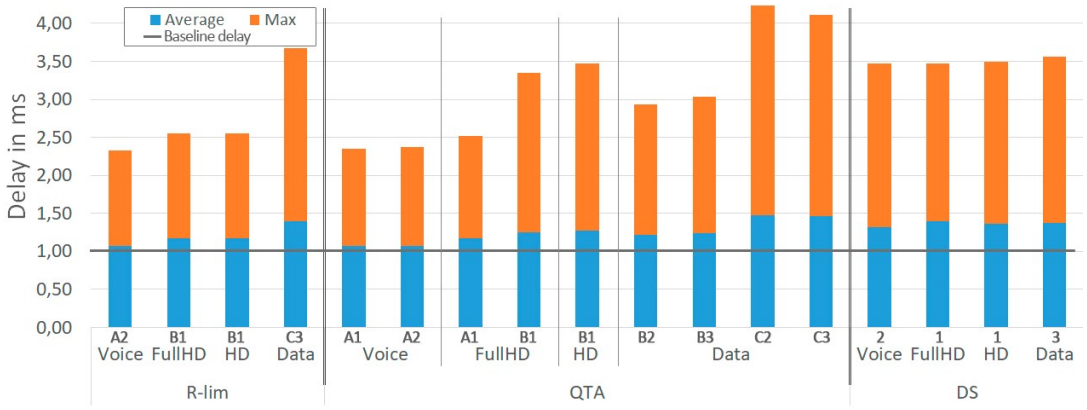

FIGURE 5.25: Comparison of average and maximum delay by application and QoS Cube used.

In this second scenario, we did not consider a traffic matrix as tight to the rate limits as when testing the rate limiting policy. Instead, we considered the use of traffic patterns based on current applications, each with its own QoS requirement, in a scenario close to congestion. There, the maximum rate would be that imposed by the rate limits (with high probability), something that could be policed within RINA's flow allocation. While, at first sight, working under the maximum rate would result in a scenario without too many collisions, it has to be considered that burst of flows arriving from different applications can be common in this scenario. This is a similar scenario to that in a usual home nowadays, as the number of connected devices keeps increasing.

Finally, with respect to this particular testbed and the obtained results, some particularities have to be considered. Firstly, the testbed used as a shim-DIF a 100Mbps vlan over a 1Gbps Ethernet [26] link. This has some peculiarities with respect to using the Ethernet link directly at its max rate. In one hand, we have to consider the slightly larger headers of the Ethernet frame due to use of vlans. On the other hand, given that the vlan works at 1/10th of the Ethernet link capacity, the inter-frame delay used to separate Ethernet frames does not affect us, as all packets are served with higher separations. Secondly, while we are emulating routers, we are doing it using machines running a complete server, while at the same time having multiple active applications. This affects negatively to all networking processes, as those have to compete for the CPU time with other non-related processes.

5.5.4 Conclusions

In order to allow for end-users to request differentiated QoS treatment for their flows, it is imperative for Internet service providers to enforce upper limit the amount of high

priority traffic that users can send over its network. In this section, an end-user rate limiting policy based on the ideas of ΔQ is proposed and experimentally evaluated. This policy limits urgent and cherished traffic independently, not only succeeding in avoiding end-users filling the network with high priority traffic, but also doing this in a user-friendly way, allowing them to use the available capacity in the way most suited for their needs.

With this policy and the control of the NMS, it is easy for network providers to limit what their users can send in a graceful way (as the user knows its limits, the access point only discards if the user breaks the contract). In addition, given the control of the NMS in RINA DIFs, such limits do not need to be fixed, but can be dynamic, for example depending on the current network usage or time frame. This allows, for example, increasing the amount of priority traffic that users can send at night, when the provider's network gives service to a lower number of end users.

While the proposed policy focuses on the priority contention of outgoing flows, something required for avoiding greedy users, it does not considers the assurance of QoS requirements in an end-to-end basis. In fact, this policy bases on the inherent recursivity of RINA, capable of providing means to assure QoS requirements on the end-to-end path in view of the guarantees provided by lower layers. However, while RINA provides the means to effectively translate specific end-to-end requirements into the most suited QoS Cubes at any level, here, a more straightforward approach, focused only on the limited scope of the proposed policy, has been considered. In this regard, it is left for future work to propose and test the joint work of RINA's flow allocation policies and rate-limiting policies.

The proposed policy gracefully solves the main issue of end-user rate limiting to allow a global differentiated QoS treatment. Even so, there is room for improvement (e.g. comparing different multiplexing solutions). In addition, given its recursive layering, RINA enables fast congestion control policies that could profit from the interaction with the rate limiting policy at home routers. This could allow for a distributed congestion control within home networks, mitigating some potential concurrency problems between multiple flows of different clients.

5.6 Chapter summary

In contrast with the current TCP/IP Internet, RINA defines a networking environment that provides a full support for QoS. While QoS solutions for TCP/IP exists, those provide only a limited QoS differentiation, something that is not always enough to fulfil the QoS standards of RINA. With that in mind, in this chapter it is disused the use of ΔQ scheduling in RINA in order to provide better QoS assurances into the network. The QTA Mux scheduling policy for RINA is introduced, a ΔQ-based policy that manages both intra-flow and inter-flow contention, providing enhanced QoS assurances for the flows traversing the network.

Given that, in order to provide reliable QoS assurances, a good understanding of the networking environment is required. Some of the key point that may influence the behaviour of flows in the network are pointed: the expected usage of the network, and how the different flows interact; the physical properties of the network (latency, link rate, nodal degree, etc.); and where and how scheduling decisions are done.

In order to analyse the behaviour of the proposed policies, comparisons of the behaviour within a backbone network of a simpler version of the QTA Mux, configured with QoS assurances in mind, with common QoS differentiation solutions are provided. Preliminary results showing a clear benefit of QoS assurance with respect to simple QoS differentiation, then the behaviour of the QTA Mux in a production-like environment as been asserted. For that, the proposed policies have been tested in a small scenario, emulating the everyday more typical scenario of cloud computing. With results within the expected range, those tests asserted the implementation of the QTA Mux policy for production environments.

Finally, QoS assurances require of some control in the usage of resources, as otherwise greedy users would always require the best service for all flows. In that regard, in this chapter a new rate-limiting scheduling policy is proposed, also based on the ΔQ framework. As the results suggest, this new policy not only ensures that the amount of high priority traffic injected in the network does not surpass the imposed limits, but also does that in a user-friendly way.

Chapter 6

Forwarding and Routing in RINA

The scalability of networks is one of the biggest concerts during the network design, especially when considering large networks. A bad design may result in low performance, unexpected problems (both of configuration and by its latter grown), or extra costs (requirement of more powerful hardware, maintainable, energy usage, etc.). Sometimes, the networking model itself limits the available options, leaving only sub-optimal solutions to consider. The IPv4 addressing space and the use of BGP for sharing AS prefixes is a clear example of this, with important consequences like the major Internet collapses on 2014 [4, 5, 41], when the number of IPv4 prefixes exceeded the limits of older routing devices. In that case, given the nature of the current TCP/IP Internet and the exponential grown of the network, the only possible solution was to use more expensive hardware with higher memory, not solving the problem, but delaying it until new hardware is required again.

Unlike the current model, RINA is thought with high concerns about the scalability of the network. Given its recursive layering in DIFs, RINA provides a mechanism to easily divide the network into smaller and more manageable domains. As each DIFs only has to concern about its own networking domain, this remove the necessity of storing large amounts of information related to what is outside of that domain. This itself removes a large number of constrains in the requirements of hardware. For example, a similar scenario as that of the BGP collapse would be highly improbable within a RINA network, as the "BGP DIF" would only have to route between the different ASs, instead of consider all the different prefixes owned by each AS.

The use of RINA not only reduces the scope of policies (reducing their scalability constrains), but it also leaves opened the gates to further improvements, as now policies are capable of taking fully profit from the different topologies of networks, and network designers have less limitations to enforce some topologies to them. This chapter discuss different forwarding and routing solutions to enhance the scalability of the network in diverse ways, all of those taking profit from one or another of the particularities of the RINA environment. In Section 6.1 a new RINA forwarding policy is proposed, though specially for networks following some topological properties. With the use of this policy, different

properties inherent in of the network topologies can be used to perform forwarding decisions, while at the same time it may work as a traditional forwarding table solution. In Section 6.2, the use of such policy is considered in different data centre scenarios, profiting from their topological properties. Given those properties, the appropriate configurations is provided, resulting in great scalability improvements in the network. Section 6.3 analyses the use of compact routing solutions in large networks and propose and how those could be implemented in a RINA environment. Section 6.4 provides a brief study of the benefits of connectionless routing with QoS path selection, both for the scalability of the network in terms of resources as for the fulfilment of QoS requirements. Finally, Section 6.5 resumes the key ideas of this chapter.

6.1 Rules and Exceptions Forwarding policy

Thanks to the use of policies, RINA opens the path to more configurable and scalable networks, capable to better adapt to any networking environment and requirements at lower cost that common solutions. In this regard, routing and forwarding are two of the networking tasks more affected by the use of policies, as well as the addressing of nodes within DIFs. As stated in Chapter 4, addressing within RINA DIFs is based on Jerry Saltzer's idea [8] of clearly separate naming (who), addressing (where) and routing (how), so a node name does not implies its address nor the address how to specifically reach it. Therefore, it is an environment where names uniquely distinguish nodes within a DIF, but being location-independent. Each node has also a synonym or address, that is location-dependent, and that provides information on where that node is with respect to an abstraction of the connectivity graph of the DIF, without impairing how to get there.

This section builds extensively on this property, proposing a replacement forwarding policy for traditional forwarding tables, the Rules and Exceptions forwarding policy, or R&E forwarding policy. Whenever the network graph contains some topological properties that helps to know where and how to reach nodes given their location, the R&E forwarding policy allows to minimize the amount of forwarding information stored in nodes, and the computation overhead of forwarding decisions. The proposed policy, briefly described in the Pseudo-code 6.1, takes profit from the programmability of RINA policies to allow the configuration of forwarding rules based on the network topology and the use of dynamic exceptions for whenever something fails in the network.

```
1 Forward(addr)
2     if Connected_Neighbour (addr)
3         Forward_to (addr);
4     else
5         Exception e = Search_Exception (addr);
6         if e != null
7             Use Exception (e);
8         else
9             Use Rule (addr);
```

PSEUDO-CODE 6.1: Rules and Exceptions policy pseudo-code

6.1.1 Rules

The primary element of the proposed R&E forwarding policy are the "rules". Forwarding rules are simple computations that, given the expected topology of the network, the location of the current node in it and that of the destination node (given by its address), provide a list of valid neighbours to where a PDU can be forwarded in order to successfully reach that destination node. Rules are designed for the expected topology of the DIF, meaning the non-failure scenario, given that, rules are neither affected by changes in the network graph nor can route around failures by themselves (except for those designed to provide secondary routes to neighbour nodes in case of link failure). Nonetheless, for destinations where the primary rules are valid, they provide fast forwarding decisions with minimal information.

While rules are used to elect the possible valid N-1 ports to where forward a PDU, the specific N-1 ports are hidden from rules. Instead, rules use an abstraction of N-1 ports based on the position where the N-1 port pointers are stored within an ordered array of N-1 ports. Considering that, what rules return is not really the list of valid N-1 ports to use, but the list of indexes for those ports (or a range of indexes). Given that list, the forwarding policy then select one of the ports referred from it, skipping those with null pointers. This has multiple advantages as rules are not affected by changes on the flows, meaning that, for example, a N-1 flow can be removed or replaced upon failure without affecting the primary rule.

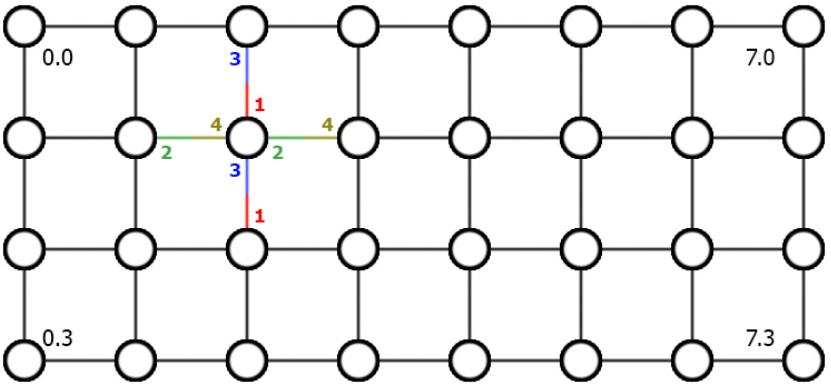

FIGURE 6.1: Simple example of topological routing. 2D 4x8 mesh.

The simplest way to see how these rules work is with a simple example. Let us consider the small network at Fig. 6.1. This network describe a small 2D mesh of 32 nodes where the addressing space is conformed of address of the form $x.y$, being x and y the horizontal and vertical coordinates respectively. Given this network, and knowing the

current and destination nodes addresses, it is trivial to know which of the underlying N-1 flows traverse a path to the destination node with minimum cost (shortest path). If the x coordinate of the destination is bigger than that of the current node going to the right will always get it closer to the destination and if it is lower going to the left will get it closer instead, and similar for the y coordinate.

The configuration of the policy takes profit from the regular topology of the network. In order to store the N-1 ports, those are assigned an index based on the relative position of the destination node. In this case, indices are assigned in a clockwise order: 1 for up, 2 right, 3 down and 4 left. For the special nodes on the extremes the same idea is used, but, in this case, leaving empty pointers whenever there are no neighbour nodes. Given this ordering of N-1 ports, the forwarding rules for a node $X.Y$ would be something as simple as those described by the pseudo-code 6.2. Note that, while in this case the same rules apply to all nodes, that does not need to be always true and different rules may be required for different locations.

```
1  Rule (a.b)
2      list valids;
3      if(a < X)
4          valids.push (4) ;
5      else if(a > X)
6          valids.push (2) ;
7      if(b < Y)
8          valids.push (3) ;
9      else if(b > Y)
10         valids.push (1) ;
11     return valids;
```

PSEUDO-CODE 6.2: Rules pseudo-code for the 2D 4x8 mesh at node $X.Y$

In order to do a lookup for the next hops, the policy first checks if the destination is a connected neighbour. If not, it searches if an exception is present; if so, the exception is executed to forward the PDU; if not, the default rule is applied. Although this pseudo-code seems more complex than a simple lookup of an entry in a forwarding table, its primary benefit comes from the fact that searches are only of neighbours plus a small number of exceptions (if any), while rules may consist only of few instructions, taking profit from the topology of the network graph.

6.1.2 Exceptions

Upon failures in the network, some of the primary rules may become invalid to reach a specific destination (or set of destinations). In case of a failure in the same node, if more than one N-1 port is eligible, the rules are not affected, and the forwarding policy will simply select one of the live N-1 flows from those referred by the rules. In other cases, a simple redefinition of rules should be enough to avoid non-valid paths. Even so, in

most cases, but that is not always possible or desirable (rules implemented in hardware, high extra complexity vs. small gain, etc.). In such cases, specific exception for these destinations may be recorded.

These exceptions, work similar to traditional forwarding entries, checking for entries matching the destination address. Even so, their encoding has some small peculiarities, taking profit from the storing of ports already used for rules. First, exceptions does not store the pointers itself to N-1 ports, but work with the indexes of those ports. Given this, exceptions do not need to be over-written if an N-1 flow is replaced and do not become invalid either if one of their ports goes down. In addition, there are 2 different types of exceptions, encoding the list of valid indexes in 2 different ways. In one hand, with "SIMPLE" encoding, exceptions store the list of indexes for the valid N-1 ports to use. On the other hand, with "REVERSE" encoding, exceptions store the list of indexes for those N-1 ports that cannot be used to reach the destination (the benefits of this will be seen latter with the addition of "groups"). Given these, encodings, empty exceptions also have special meanings, as an empty list in SIMPLE encoding has the same meaning as "unreachable" and with "REVERSE" encoding it has the meaning of "use any".

With that in mind, only storing exceptions to primary rules upon failures can yield a large reduction in terms of memory usage and computational cost. For failure-less scenarios, that is clear, as those do not require any exception and may work only with rules. But, this is also true in failure scenarios. Since most communications across the network remain unaffected by specific link or node failures, the number of required exceptions tend to be considerably smaller than the number of entries that a traditional forwarding table would require. Still, even in the worst imaginable case, having a similar usage, the number of exceptions at most would be the same of that of forwarding entries.

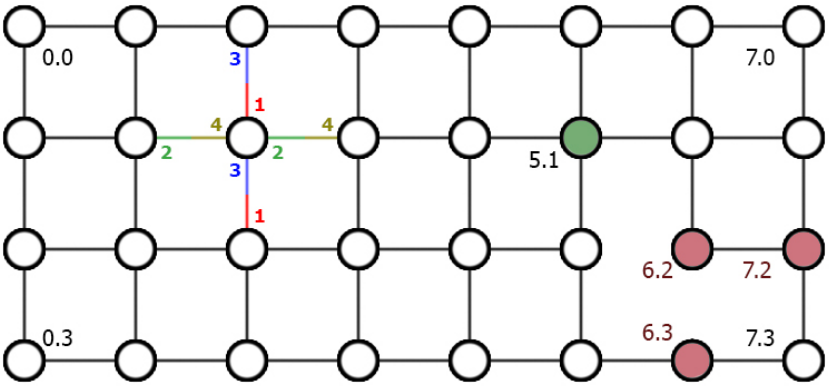

FIGURE 6.2: Simple example of topological routing with failures. 2D 4x8 mesh, 2 links down.

To see how exceptions work, let us extend the previous example. Fig. 6.2 shows a failure scenario where two links are down (links connecting nodes 5.2 with 6.2 and 6.2 with 6.3). In this scenario, the primary rules used for forwarding PDUs may provide non-optimal forwarding paths, or even paths that do not arrive to the destination. It is for those

destinations that specific exceptions are required, in order to overwrite the decision of the primary forwarding rules. In this example, let us focus on node 5.1. While most of the destinations can be reached without problems with the use of primary rules, there are 3 nodes for which the primary rules may give invalid next hops (6.2, 6.3 and 7.2). Considering that, then the exceptions table for node 5.1 would be simply:

- 6.2 — SIMPLE {2}
- 6.3 — SIMPLE {3}
- 7.2 — SIMPLE {2}

As can be seen, only 3 exception entries are required, plus the information about the neighbouring nodes, to be able to forward correctly towards any node in the network. Farther from the failures, that number is reduced even more, having cases like the node 0.0 where no exception is required.

6.1.3 Groups

When dealing with networks with a large nodal degree, it is common to have large groups of N-1 flows equally good to reach most of the destinations. A simple example of this is having multiple N-1 flows between the same pair of nodes for protection and increased capacity, or the use of multiple intermediate aggregator switches in networks like those used in data centres. In the first case, sometimes those multiple N-1 flows can be simply aggregated as a unique flow (e.g. port trunking [152] or link bundling [153]) Even so, that is not always the best solution nor even a valid one (e.g. non-aggregable N-1 layers, limitations with multipath as all N-1 flows between the same pair of nodes is being treated as a unique one, etc.), and direct control from by the hands of the forwarding policy is preferable.

In order to reduce the complexity of both rules and exceptions, the R&E forwarding policy uses groups of N-1 flows in order to provide an enhanced version of this link aggregation. These groups can be seen as ordered arrays of pointers to N-1 flows that can be used by both rules and exceptions to easily define the set of valid ports to reach a destination, without requiring generating each time the full list of valid ports. As for simple link bundling, a group can be defined as the set of all flows directed to the same destination node. That is not its only use, as, unlike link bundling and similar aggregated solutions, an N-1 port can be part of more than one group simultaneously. Actually, the default array of N-1 flows used until now by rules and exceptions is no more than the default group, or group 0.

While the main definition of groups is based on what the different rules expect (although extra groups can be used), their population is performed by the same forwarding generator policy in charge of populating exceptions, being the main link between rules and exceptions and the specific N-1 ports in use. The usage of groups also has provides some

improvements to both rules and exceptions, being possible for example to return a full group as the result of a rule (instead of generating every-time the full list of valid ports) or encode exceptions as "use any port of that group, but that one" (with the join use of groups and REVERSE exceptions).

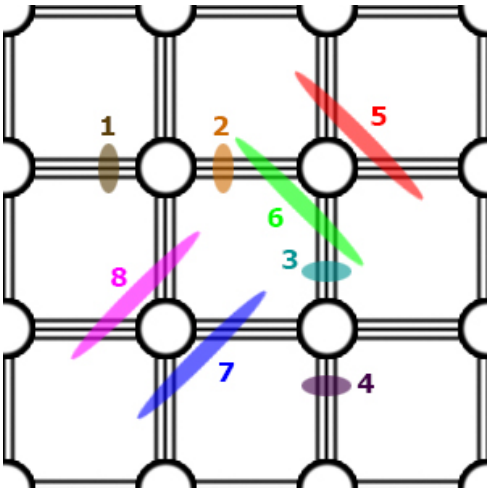

FIGURE 6.3: Extended example of topological routing. Protected 2D mesh with triple links.

Let us expand the previous example with the addition of groups. Fig. 6.3 shows an extended version of the previous 2D network where each link has been replaced by a triplet of links to enhance protection. Considering this new network, the first approach is to group all N-1 flows with the same destination, creating this way groups 1 to 4. Now, considering the previous rules used (Pseudo-code 6.2), it can be seen that that there are mainly 8 different forwarding options defined. If the destination node is either in the same column or row, go in a straight line towards the destination node, with only one possible next hop (in this case those corresponding to the same groups from 1 to 4 or left, right, up and down). Otherwise, there are always two different next hops among which to choose, being that also the most common case (right or up, left or down, right or down, and left or up). Then, as there is no restriction for N-1 flows to be in only one group, 4 new groups (5 to 8) are defined to match these new options.

```
1  Rule (a.b)
2      if(a < X)
3          if(b < Y)
4              return group(8)
5          else if(b > Y)
6              return group(6)
7          else
8              return group(1)
9      else if(a > X)
10         if(b < Y)
11             return group(5)
12         else if(b > Y)
13             return group(2)
14         else
```

```
15              return group(7)
16      else if(b < Y)
17          return group(3)
18      else
19          return group(4)
```

PSEUDO-CODE 6.3: Rules pseudo-code for the protected 2D 4x8 mesh at node $X.Y$

Given the definition of groups, the forwarding rules could be redefined as expressed in Pseudo-code 6.3. While seaming more complex than the previous rule, their complexity is similar, being mostly a different way to implement the two-dimensional switch between the relative vertical and horizontal coordinates. On the other hand, the use of groups provides some benefits, like removing the requirement to generate the list of valid N-1 flows on each call and the abstraction between directions and N-1 flows (being possible to add or remove flows between nodes without redefining the existing rules).

6.1.4 Extensions

So far, the complete R&E forwarding policy has been presented, with some exemplary configurations for the defined scenarios. However, most implementation decisions are left open, allowing the policy to be extended in multiple ways. For example, given the fast computation requirements of forwarding policies, they could be easily implemented as programmable hardware. A benefit of this approach is that, while taking advantage of the network topology knowledge, it is generic enough to allow using the same hardware independently of the scenario (even as a replacement of a traditional forwarding table), only relying on configurable rules. This makes the presented policy an interesting substitute of traditional forwarding tables for future RINA-based hardware.

When it comes to a potential implementation, multiple extensions can be considered, either to enhance its behaviour or to extend the range of supported scenarios. An example can be load-balancing decisions between all available neighbours. It may be important for a flow to always follow the same path if possible. To this end, two options can be considered, either storing the results in a small cache to follow always the same paths, or taking load-balancing decisions based on a hash value of the flow identifier. Another extension that can be considered is the addition of QoS metrics into rules and exceptions, something to consider for scenarios where different metrics are considered.

6.1.5 Conclusions

The Rules and Exceptions forwarding policy presented in this section provides an extensible replacement for the current forwarding tables capable of adapt and take profit from the structure of different network topologies. Matching the programmability of RINA,

the R&E forwarding policy provides a configurable environment that permit to greatly reduce the computational cost of forwarding decisions and the memory usage required to store forwarding information in networks with a well-defined topology.

In this section have already seen a small sample of what the proposed R&E forwarding policy is capable in networks following well-defined topologies. This will be extended in Section 6.2, where the R&E forwarding policy will be extensively used in order to provide a complete topological forwarding and routing solution for data centre networks with minimal cost.

6.2 Topological Routing for Data centres

Seeking the highest efficiency, uptime and scalability, nowadays' commercial Data Centres (DCs) tend to rely on small variations of well-defined leaf-spine Data Centre Network (DCN) topologies. These topologies offer low latency and ultra-high bandwidth for server-to-server and server-to-edge communication, and also enhanced reliability against multiple concurrent link and node failures across the DCN. The reported Google's and Facebook's DCN topologies, available in references [136] and [56] respectively, are good examples of this tendency.

With the increasing usage of cloud computing and moving towards future Internet of Things (IoT) [155] and 5G network scenarios [156], a plethora of emerging new cloud services are expected to proliferate. To properly face them DCs will be required to grow even larger in terms of computing and storage resources. Leaf-spine DCN topologies can be scaled to accommodate these requirements. However, routing and forwarding solutions in DCNs, typically based on the TCP/IP protocol suite, do not scale well. That is, large and unmanageable forwarding tables (at least in the order of several tens of thousands of entries in highly-optimized configurations [138]) should have to be properly handled at DCN nodes. Moreover, IP routing protocols incur in a high communication cost (information exchanged to populate routing tables and re-converge upon changes in the DCN topology). Such limitations of the TCP/IP protocol suite for efficient routing inside DCs have been known for long time, being not specifically designed for well-defined (e.g. leaf-spine) DCNs and very inflexible for improvements [130].

In contrast to the rigidness of the TCP/IP protocol suite, the clean-slate Recursive InterNetwork Architecture (RINA) [133, 134] enables a programmable environment where Quality of Service (QoS), forwarding, routing or security policies can be freely configured by the network administrator. This opens the path to the deployment of policies tightly tailored to the specific characteristics and needs of any network environment. For example, in a RINA-enabled DCN, forwarding and routing policies can be programmed for superior scalability in leaf-spine topologies, outperforming solutions based on TCP/IP, whose protocols were designed to deliver traffic over any arbitrary topology in a best-effort

manner. This section aims to quantifying the benefits that RINA can bring in large-scale DCs thanks to its programmable behaviour, giving the possibility to deploy customized forwarding and routing policies. In particular, the use of the previously presented R&E forwarding policy is proposed in order to increment scalability while profiting from DCN topology knowledge to forward PDUs to any (or a subset only) of the closest neighbour devices to their destination based on programmable rules. In the non-failure scenario, this approach requires a minimal amount of information to be stored at any forwarding device as only adjacent neighbour reachability is necessary, in contrast with the high number of entries required in current DCs.

This is a a great improvement with respect to traditional forwarding tables in IP networks, which may contain even more than one entry per network node (e.g. nodes with private DC-address plus multiple public addresses for the Virtual Machines). When failures occur in the DCN some forwarding rules may fail to successfully deliver packets to destination. In these cases, few exceptions overriding those rules need to be stored at forwarding devices. This is the only time when additional forwarding information is required.

In addition to the huge reduction on forwarding table size, it is also illustrated how the knowledge of the DCN topology characteristics, which can be summarized by merely a few parameters in most common cases, can substantially reduce the routing communication cost, as well as the path computation burden. Indeed, when all nodes know the DCN topology characteristics, network changes due to failures and repair actions are the only information that must be disseminated. Considering this, routing policies are also proposed, reducing the amount of information shared between nodes to a large extent.

Additionally, with the rules-and-exceptions policy, routing information is only needed for the computation of new exceptions (or to remove them when a failure has been repaired), instead of the full forwarding table. This allows bounding the computation of exceptions to only destination nodes in the neighbourhood of failures, which results in a computational cost dependent on the number of concurrent failures in the DCN, rather than its size. This work present our previous work in [139] and [131], where the R&E forwarding policy was first presented, contemplates different leaf-spine and clos DCN variations and quantifying the benefits of the forwarding and routing policies in a significantly broader way. Instead of the highly tailored forwarding policies presented in [139], the more generalized R&E forwarding policy was proposed in [131], allowing its implementation in generic hardware and enabling its deployment in any network that could benefit from the approach proposed here. Finally, in addition to the extended distributed routing approach, a centralized approach to exceptions computation is also introduced in this work.

6.2.1 Related work

DCN topologies have been always designed to ensure the maximum cost-efficiency. Even so, common routing and forwarding techniques cannot take profit from their specific topological characteristics. To address these issues, deterministic routing [157] was the scheme initially deployed in many DCs. In such a scheme, nodes are assigned addresses based on their specific topological properties, so that the route between any pair of nodes is known beforehand and does not change over time. These routes are usually encoded in the same packets, in the form of a bit-stream or coordinates (e.g. see [158]). While highly scalable, this rigid scheme has two major drawbacks: the lack of automation in defining addresses, and thus the setup of routes, and the inability to leverage multiple paths (given the identity relationship address=path), preventing any possible recovery actions upon DCN failures.

In fact, any node in a DCN typically communicates with only a small sub-set of neighbours. This may become a favourable scenario for using solutions based on source routing. These solutions can alleviate both routing and forwarding requirements, as only paths to this small sub-set of neighbours have to be computed. Within this family, the valiant routing scheme [159] was proposed as a solution to overcome the shortcomings of deterministic routing, bringing multipath support and load balancing, while still being highly scalable. In this scheme, for a communication between any pair of nodes, a random intermediate address A is firstly selected, and the path is composed by routing packets from source to A and then from A to the destination. This multipath approach provides an easy and direct solution for avoiding non-valid routes in failure scenarios, simply replacing the intermediate node when either the source-to-A or A-to-destination sub-paths became invalid. However, this is achieved at expenses of longer paths, while still lacking an automation process for addressing and having load balancing restricted to the selection of the intermediate node A.

While the valiant routing scheme does not take full profit from the high connectivity of DCNs, alternative source routing solutions also exist to this end. For instance, the Line Speed Publish/Subscribe Inter-Networking (LIPSIN) [160] provides a more robust solution for DCNs with support for multipath, once the flow has been allocated. Unlike in the valiant routing scheme, LIPSIN assigns unique names to the different links in the network (as directed links). When allocating a flow, the forwarding tree from source to destination is computed. Given this tree, a Bloom filter [161] is computed, containing the links in the tree. This is added to the header of the packet, and each node in the path selects the next hop from one of the links within the encoded bloom-filter. To avoid false-positives produced by bloom-filters, when allocating a flow, special entries are added to the nodes in the tree that could generate invalid paths. LIPSIN solves some of the shortcomings of the valiant routing scheme regarding to load balancing. However, it also introduces additional complexity, like the requirement of adding extra entries at intermediate nodes to avoid false-positives.

Currently, given the low-cost of IP-based commodity servers and existing forwarding devices, many large scale DCs have adopted more generic solutions based on the TCP/IP protocol suite. To mitigate the inherent limitations of IP routing solutions, initially designed for an Internet with arbitrary topology, modifications to link-state and path-vector routing have been introduced in order to better accommodate to more specific scenarios. For example, Facebook uses BGP-4 to disseminate routing information in its DCNs [146], which was initially designed for Internet backbone networks. In this way, the need for an address per interface (as commonly required in IP) is avoided by assigning an Autonomous System Number (ASN) per node, while routing on the one node instead of the interface [154]. Nonetheless, designed for more dynamic and heterogeneous networks, BGP-4 suffers from many limitations when facing highly regular DCN topologies with high nodal degrees (e.g. path exploration upon failures, manual configuration of timers, the need to setup TCP connections between any pair of connected ASNs, and so on) Eventually, these schemes imply a high communication cost and require many entries in the routing and forwarding tables to take optimal routing decisions and allow for route recovery upon failures.

Moreover, Software Defined Networking (SDN) [86, 87] is an approach that has been spreading lately, especially in tightly managed networks. SDN builds upon the separation of transport and control planes, enabling programmable networks and flexible management, which allows administrators to better adapt to their particular network requirements. With SDN, most forwarding decisions are centralized, requiring only a few nodes to know the state of the full network. Given the large number of nodes in DCNs, this centralized approach has been lately considered as a substitute of more traditional distributed approaches. For example, Google DCs use a SDN-based approach to control packet forwarding within the DCN [136]. Although this strategy allows taking efficient decisions at low communication cost, it also has multiple drawbacks, like a: full dependency on centralized management to perform any forwarding decision for new flows, or; potential scalability issues since the computational cost of computing centralized decisions increases with the network size or introducing single points of failure.

Among these approaches for DCN routing, each solution has its pros and cons. Source routing solutions, like valiant routing or LIPSIN, provide forwarding decisions that do not require almost any forwarding information to be stored in the network nodes (except some exceptions), but increase the complexity of flow allocation and path recovery. In addition, they require considerably longer packet headers to encode the path information, which has an impact on the resource usage. In contrast, IP-based solutions rely on the use of forwarding tables and generic routing solutions (e.g. BGP-4), resulting in solutions that do not take profit from the network topology. These generic approaches can benefit from cheap commodity hardware, but result in costly routing operations and large forwarding tables. Finally, SDN-like solutions take a more centralized approach where only few powerful nodes manage the entire network. This allows for a more precise management of the network and removes most of the information required at intermediate nodes.

However, they also show potential scalability issues, since everything is managed by the central authorities.

In contrast to the reviewed routing solutions, this section takes a different approach based on the previously proposed Rules and Exceptions forwarding policy. It assumes that the entire DCN is already known thanks to their topology and that addressing schemes can easily gives the location of nodes in that topology, something that can easily be achieved in RINA as will be discussed latter. Taking some knowledge as granted, the amount of information stored at nodes becomes minimal, only requiring storing exceptions to the primary forwarding rules when the DCN topology changes for any reason (e.g. a link or node failure). In order to compute these exceptions, either distributed or centralized routing solutions can be used. These solutions can also take profit from the static knowledge of the network, resulting in fewer and smaller routing updates (lower communication cost) and simpler computations.

6.2.2 Scenarios under study

While having the same two protocols and mechanisms at each layer (i.e., DIF), it is possible to configure each DIF instance with policies specifically tailored to its scope (operating environment). This enables an easy and cheap deployment of scenario-optimized solutions, outperforming any generic solution. In this section, we investigate the benefits of a RINA deployment inside a data centre, following the DIF setup depicted in Fig. 6.4. Such a RINA-enabled DC is partitioned into three main types of DIFs, covering three different scopes:

1. One DC-Fabric DIF, acting as a large distributed switch that connects all ToR switches and edge routers.
2. One DC DIF, connecting all servers in the DC as a single pool of computation and storage resources.
3. Multiple tenant DIFs, isolated and customized as per each tenant requirements.

Note in the figure that underlying point-to-point links are abstracted as shim (or v-shim) DIFs. As seen before, this allows abstracting any legacy technology or physical media (e.g. Ethernet, hyper-visor VM communication [23], etc.). In this case, these shim-DIFs are also the ones that perform any kind of link aggregation (port trunking, link bundling, etc.), provide reliable communication between both pairs of nodes and inform of the status of those links to the DIF Management System and upper DIFs. That meaning that, while considering routing and forwarding solutions, we will not require to check by ourself the status of the underlying links, reducing the complexity of the different policies in use.

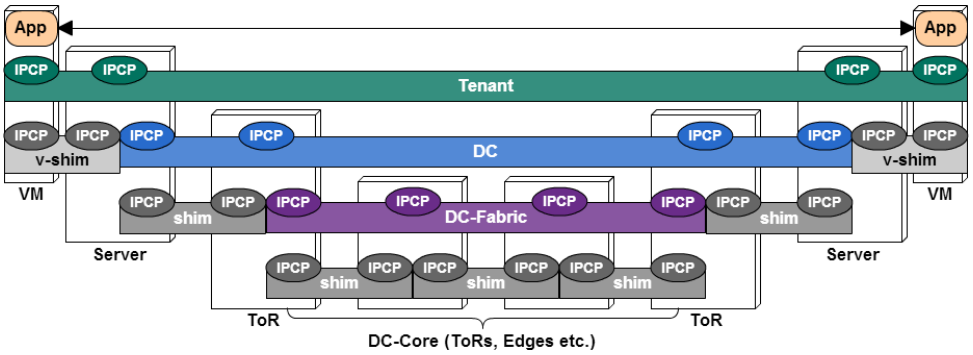

FIGURE 6.4: Example of recursive DIF layering in a typical DCN.

In DC and Tenant DIFs, there is only one "eligible" path between any pair of IPCPs, making forwarding decisions straightforward and routing unnecessary (e.g. to go from server A to server B in a distinct rack in the DC DIF, traffic must be forwarded to its ToR switch, next to the ToR switch of the rack where server B is located, so as to finally deliver it to server B). In contrast, DC-Fabric DIFs are specifically designed in a way that there exist multiple redundant paths between any pair of nodes, in order to ensure resiliency upon multiple concurrent failures, as well as effective load balancing. Therefore, DC-Fabric DIFs tend to follow well-designed topologies with certain properties. However, in a much more flexible and efficient way than what would be achieved with the TCP/IP protocol suite.

Taking these limitations as a motivation, in this section we focus on quantifying the benefits that forwarding and routing policies specifically tailored to the DCN topological characteristics can bring into DC-Fabric DIFs in a RINA-enabled DC. To this end, we contemplate five different DC-Fabric DIFs, mimicking the topological characteristics of a nowadays' widely accepted generic DCN design (clos and leaf-spine) or specific DCN design solutions made publicly available by large corporations, as Google or Facebook, and also a variation of one of them (modified clos).

6.2.2.1 Generic leaf-spine (GLS) DCN

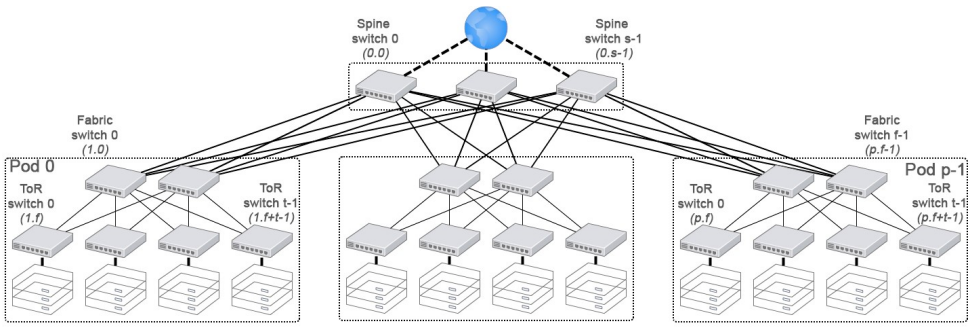

FIGURE 6.5: Generic leaf-spine (GLS) DCN topology.

The leaf-spine topology (Fig. 6.5) is one of the simplest and more straightforward DCN topology available nowadays. In this topology, we have pods forming a full bipartite graph, with ToR switches at one side and fabric switches at the other. Then, another bipartite graph connects all fabric switches with the spine switches, acting as edge routers at the same time. Interestingly, DCNs following this topology can be fully described by only 4 parameters:

- p : Number of pods in the DCN
- t : Number of ToR switches per pod
- f : Number of fabric switches (aggregators) per pod
- s : Number of spines switches (edge routers) in the DCN

Hence, we propose the use of the following possible addressing scheme in a DC-Fabric DIF following this topology where addresses encode the type of node, as well as its location:

- Spine switch : $0\ .\ spine_{id}$
- Fabric switch : $(1 + pod_{id})\ .\ fabric_{id}$
- ToR switch : $(1 + pod_{id})\ .\ (f + tor_{id})$

With this addressing scheme, for example, the address of the ToR switch 5 of pod 3, when having 4 fabric switches per pod, would be 4.9. It has to be noted that, in this and following scenarios, we take the approach of consider all identifiers (for pod, ToR, etc.) starting at 0, hence why pod 3 takes addresses $4.*$ instead of $3.*$, as it is really the 4^{th} pod in the DC.

We could have used any other topological addressing scheme, if it had simple relations between address and location. Even so, we decided on this one as, not only by its simplicity, but also for the fact that addresses can be encoded with only $[log_2(1 + p) + log_2(f + t)]$ bits. With that in mind, we get that, even for DCNS with twice the number of pods and ToR switches per pod as nowadays large-scale DCNs, we only require merely 2 bytes for addressing within the DCN, an important reduction compared to the 4 and 16 byte-length addresses of IPv4 and IPv6, respectively. In addition, these 2 byte addresses match perfectly one byte per address coordinate, allowing for improved performance.

In case that an increase on the number of fabric switches per pod is planned in the near future, the expected value of f should be considered. This will allow a graceful upgrade, without requiring full node renaming in the DC-Fabric DIF, although this could still be easily managed in RINA.

While this topology is widespread, it is hardly scalable to nowadays' DCN sizes. Since the number of edge routers (spine switches) rises as the network grows up, this requires increasing the node degree of fabric switches dramatically.

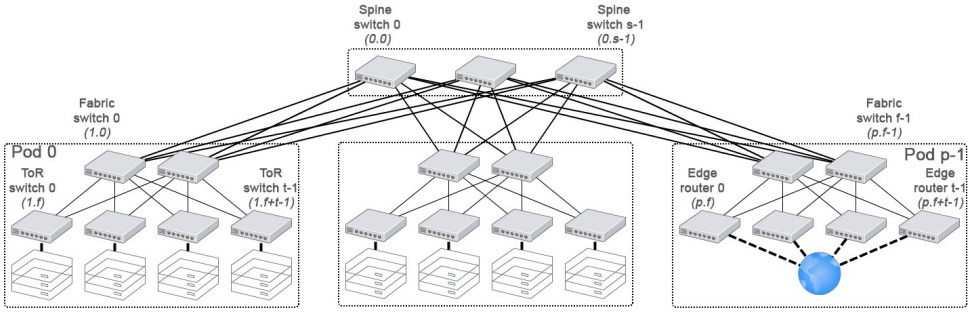

FIGURE 6.6: Google's (GO) DCN topology.

6.2.2.2 Google's (GO) DCN

As reported in [136], Google has decided to deploy a modified version of the generic leaf-spine topology in its DCNs (Fig. 6.6). With the same connectivity between ToR, fabric and spine switches as in the generic leaf-spine DCN topology, Google moves edge routers to their own pod-like sets, instead of locating them at the spine switches. This modification entails some benefits and drawbacks against traditional leaf-spine DCNs. Firstly, it solves the scalability problems of the leaf-spine topology, as it moves edge routers out of spine switches. In addition, it fosters load balancing and relieves the responsibility of ensuring reliability from the high loaded spine switches. However, this comes at the cost of slightly increasing the path length of such flows to/from the exterior of the DC premises.

The parameters describing this topology are the same as for the generic leaf-spine DCN. Moreover, the addressing scheme proposed before also fits this DCN topology.

6.2.2.3 Facebook's clos based (FB1) DCN

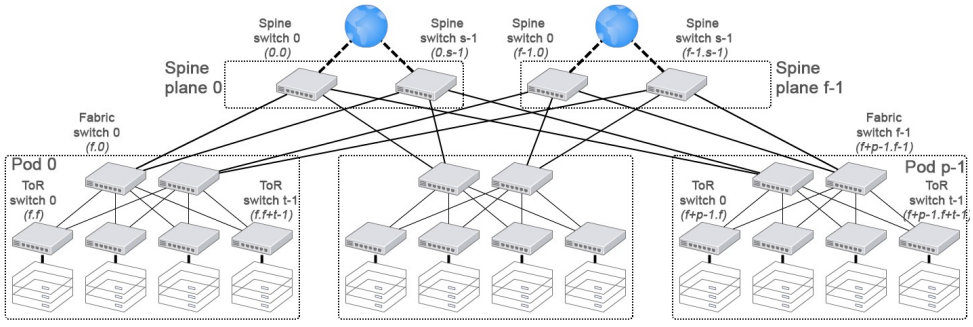

FIGURE 6.7: Facebook's (FB1) DCN topology.

In contrast to Google, Facebook [56] deploys a clos DCN topology in its DCs (Fig. 6.7). In this case, pods follow the same bipartite graph as generic leaf-spine DCNs. However, instead of a unique plane of spine switches connecting all fabric switches, multiple spine planes are deployed, one per fabric switch in the pods (each fabric switch is connected to

one and only one spine plane). Compared to the generic leaf-spine topology, clos DCNs foster better scalability, allowing to increase the number of fabric switches per pod without requiring an important upgrade in terms of ports at spine switches. DCNs following this topology can be described by 4 parameters, similarly as the generic leaf-spine DCN:

- p : Number of pods in the DCN
- t : Number of ToR switches per pod
- f : Number of spine planes = fabric switches per pod
- s : Number of spine switches (edge routers) per spine plane

Given this parametrization, we propose to use the following addressing scheme in the DCN (similar to that proposed for leaf-spine topologies). As before, these addresses encode both the type of node and its location:

- Spine switch : $spine\_plane_{id}$. $spine_{id}$
- Fabric switch : $(f + pod_{id})$. $fabric_{id}$
- ToR switch : $(f + pod_{id})$. $(f + tor_{id})$

With this addressing scheme, for example the address of the ToR switch 5 of pod 3, if having 4 fabric switches per pod/spine sets, would be 7.9. Remember that pod, ToR, etc. identifiers start at 0, hence pod 3 being the 4^{th} one and ToR 5 the 6^{th} one in the pod.

Note that these addresses can be encoded with only $[log_2(f + p) + log_2(f + t)]$ bits. If an upgrade to increase the number of spine planes is already planned in the near future, the f value should be set as expected in order to avoid a full DCN renaming process in the DCN (i.e., as also suggested for the GLS DCN).

6.2.2.4 Previous Facebook (FB2) DCN

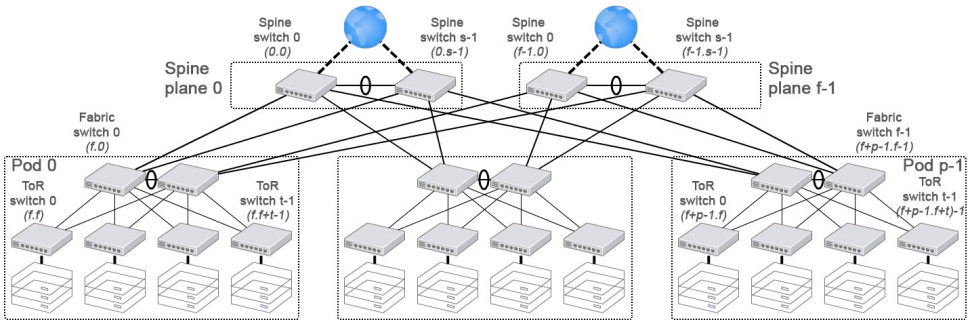

FIGURE 6.8: Previous Facebook's (FB2) DCN topology.

The DCN topology used at previous Facebook's DCs [137], depicted at Fig. 6.8, follows the same clos topology as for the current ones, but with an added extra layer of protection against failures. In this variation of the clos topology, all fabric switches within a pod are connected describing a ring topology, and the same is done for spine switches at spine planes. These extra links should not be used in non-failure scenarios. Conversely, they

enable short secondary paths between ToR and spine switches when link that connects the spine switch with the fabric switch of their pod fails, which would otherwise require re-routing the traffic across other pods. These links increase the reliability of the network and avoid using resources of other pods upon failures. Nevertheless, they are protection resources that may remain unused most of the time.

Having this DCN topology almost the same structure as FB1, it can be described with the same parameters and the same addressing scheme can be used.

6.2.2.5 Modified clos (MC) DCN

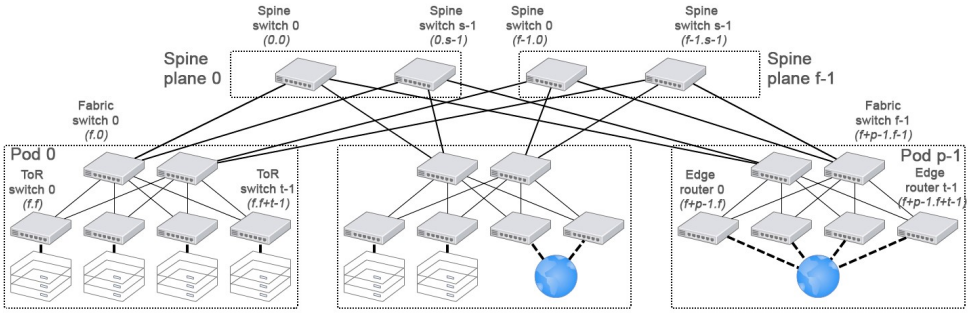

FIGURE 6.9: Modified clos DCN topology.

This modified clos DCN topology (Fig. 6.9) follows the idea of Google to move edge routers at the same level as ToR switches, while taking advantage of the enhanced scalability (upgradability) that the clos topology provides. However, it also carries some of their drawbacks. For instance, it solves one of the Clos topology main problems, namely, the loss of a direct route between a spine switch (that also acts as edge router) and all ToR switches of a pod, when the link between the spine switch and the connected fabric switch of the pod fails. In addition, it yields enhanced load balancing for outgoing flows. As a drawback, the path length of these flows is slightly increased.

This DCN topology can be described by the same parameters as for FB1 DCN. Moreover, the same addressing scheme there can be used.

In addition to the benefits of moving edge routers at the ToR level, given that RINA allows for a clean separation between DIFs, addresses in the DC-Fabric DIF are not related to those used in the upper DC DIF. That removes the need of grouping edge routers (to be able to aggregate public IP addresses), thus making possible to place some edge routers directly at pods, with the benefits that a maintaining most outgoing flows within the same pod would enable.

6.2.3 Topological forwarding. Using the R&E forwarding policy

A key requirement of any forwarding policy is the ability to quickly decide the neighbouring node (or nodes) to which a packet has to be forwarded in order to reach its destination. Traditional forwarding tables, while allowing the aggregation of destination addresses per entry with address masks, do not scale well as the network size grows up. In DCNs deploying the TCP/IP protocol suite, this is even more problematic, as the relation between nodes and addresses requires a whole bunch of public addresses for both tenants with their own IPs and for enabling the mobility of servers and VMs.

RINA inherently removes the need of extra addresses, as no public address is required (IPCP addresses can be independent, i.e., private within each DIF). Nevertheless, the use of conventional forwarding tables in IPCPs with a flat addressing scheme (per DIF) would not solve the problem of the scalability of regular DCN topologies as the ones reviewed previously. Luckily, programmable forwarding policies in RINA are not restricted to a traditional forwarding table. Instead, any forwarding function capable of quickly performing accurate forwarding decisions can be used. In this regard, the Rules and Exceptions forwarding policy, introduced in Section 6.1 provides a minimalistic forwarding function that can take profit of the specifics of such DCN topologies, and be used as an upgrade of generic forwarding tables.

Being aware of a specific leaf-spine or clos DCN topology like the ones presented before, and the location of the node (encoded in its address), merely storing forwarding information to neighbouring nodes is enough for the R&E forwarding policy to forward any PDU to its destination with few simple forwarding rules. When failures occur across the DCN, some of the routes may become invalid, but in those cases, exceptions to primary rules can be used to overwrite those primary rules. Even so, the number of exceptions should be considerably smaller than the number of entries in a traditional forwarding table, as many communications across the DCN remain unaffected by specific link or node failures.

Considering the use of the R&E forwarding policy within DCNs, next we present the different configuration for both leaf-spine and clos DCNs.

6.2.3.1 R&E configuration. Groups

In order to reduce the complexity of rules and exceptions, the R&E forwarding policy uses groups of nodes in order to perform an abstraction of neighbour node addresses and ports and remove the need of creating large list of valid N-1 ports in scenarios with large nodal degrees. The definition of these groups, used by rules and exceptions to easily define the set of valid ports to reach a destination, is a key point for the optimal behaviour of the process of forwarding decisions. Tables 6.1 and 6.2 depicts possible definitions of groups for the leaf-spine and clos DCNs respectively.

TABLE 6.1: Definition of groups for leaf-spine DCN topologies (GLS and GO)

At ToR switches	A: Fabric switches valid to reach nodes in the same pod. B: Fabric switches valid to reach nodes in other pods.
At fabric switches	A: ToR switches. B: Spine switches.
At spine switches	A: Fabric switches ordered by pod_{id} followed by $fabric_{id}$ position = $fabric_{id} + f * pod_{id} - 1$

TABLE 6.2: Definition of groups for clos DCN topologies (FB1, FB2 and MC)

At ToR switches	A: Fabric switches valid to reach nodes in the same pod. B: Fabric switches valid to reach nodes in other pods, ordered by $fabric_{id}$ position = $fabric_{id}$
At fabric switches	A: ToR switches. B: Spine switches.
At spine switches	A: Fabric switches ordered by pod_{id} position = $pod_{id} - f$

It has to be noted that, apart from these groups, there is the default neighbour group (group 0) In this case, for spine switches, group A is a synonym of this default group 0, but is left here as a clarification for the further defined rules. It has also to be reminded that, while in some cases the order within a group does not matter, in others can be really important, as rules may point to specific positions or ranges within groups. In any case, it is possible to have null positions within a group. In those cases, null positions will be simply skipped when executing rules or exceptions.

6.2.3.2 R&E configuration. Rules

The key elements of the R&E forwarding policy are the forwarding rules. Rules should be simple computations that, given the expected topology of the network, current location and destination address, returns the list of valid neighbours to where forward a PDU, if it can be forwarded. Being designed for the non-failure scenario, rules are neither affected by changes in the network nor can route around failures. Nonetheless, being that not the common scenario, they provide fast forwarding decisions with only minimal information on the neighbouring nodes.

Tables 6.3 and 6.4 shows the rules used in the leaf-spine and clos DCNs respectively to forward PDUs toward a destination address $a.b$, returning in each case the valid list of neighbours either as a whole group (GROUP X), a range within a group (RANGE $X[min, max]$) or a unique node in a group (NODE $A[index]$).

TABLE 6.3: Definition of rules for leaf-spine DCN topologies (GLS and GO)

At ToR switches	if $a = pod_{id}$: return GROUP A else : return GROUP B
At fabric switches	if $a = pod_{id}$: return GROUP A else : return GROUP B
At spine switches	if $a = 0$: return GROUP A else : return RANGE $A[(a-1)*f, a*f-1]$

TABLE 6.4: Definition of rules for clos DCN topologies (FB1, FB2 and MC)

At ToR switches	if $a = pod_{id}$: return GROUP A else if $a < f$: return NODE $A[a]$ else : return GROUP B
At fabric switches	if $a = fabric_{id}$: return GROUP B else if $a = pod_{id}$ or $a < f$: return GROUP A else : return GROUP B
At spine switches	if $a < f$: return GROUP A else : return NODE $A[a-f]$

As can be seen, forwarding rules are really simple, requiring only a few lines to define how to route packets to the whole DCN. The defined forwarding rules makes full use of the previously defined groups in Tables 6.1 and 6.2 to decide the list of valid neighbours to reach a destination. Specifically, we can obtain two kinds of decisions from them: either to use any node from a group, or a range of nodes from a group (a specific neighbour if the range has length 1). In fact, as rules do not know the content of groups, nor are affected by changes in them (recall that any null position in a group is simply skipped when the rule is executed), this has some really great advantages, as a great number of routes that become invalid upon failures can be avoided with only small changes in the definition of groups. An example of this is the definition of groups for ToR switches. There, both defined groups are the same if there are no failures, but if something fails close to the node, (e.g. a fabric switch losses its connectivity with all spine switches), one or both can be re-defined to exclude the affected neighbours, without affecting the existing rules in any way.

Note that the presented parametrization of the DCNs assume the same number of ToR and fabric switches at each pod, and number of spine switches in spine planes (if there is more than one). If that is not the case, while the proposed solutions would still be valid, possible modifications on rules might be required to accommodate such changes. For example, at spine switches in leaf-spine DCNs, in case of having a varying number of fabric switches per pod, we could define f as the maximum number of fabric switches among all pods, filling the unused positions with null pointers (remaining then valid the current rules). Alternatively, if the hardware memory allows it, it would be simpler to

define a group for each pod and replace the rule as "Any fabric switch of group pod_{id}". Also, note that, as rules contemplate non-failure scenarios, these are not affected by failures that do not affect primary paths.

6.2.4 Routing. Computing exceptions

Upon failures in the DCN, some of the primary rules may become invalid to reach a specific destination (or range of destinations). Sometimes (mainly with nearer failures), a simple redefinition of a group should be enough to avoid failed paths, but that is not always possible. In such cases, we may find a still unreachable destination, so we need to record a specific exception for this destination (or destinations, e.g. a pod). As those exceptions are only required upon certain failure scenarios, its number is considerably smaller than the number of entries that a traditional forwarding table would require, since most communications across the DCN remain unaffected by specific link or node failures. In addition, we can also reduce even more the amount of required exceptions, if we consider that end-to-end flows provided by the DC fabric DIF are only between ToR switches or Tor switches and Edge routers, not requiring then exceptions to other type of nodes.

The use encoding of exceptions using groups also has a great improvement in the amount of information that those have to store. Specially, given the high number of available paths within the DCNs, the use of group reverse exceptions has a huge effect in reducing the size of exceptions in nodes where most of the available neighbours are still valid to reach a destination, allowing the generation of exceptions such as "To reach X use any of G except Y".

Now let us see a small example to see how the Rules and Exceptions forwarding policy works in a simple DCN environment. Let us consider the small network in Fig. 6.10, a minimum example of a Modified Clos network with few failures in it.

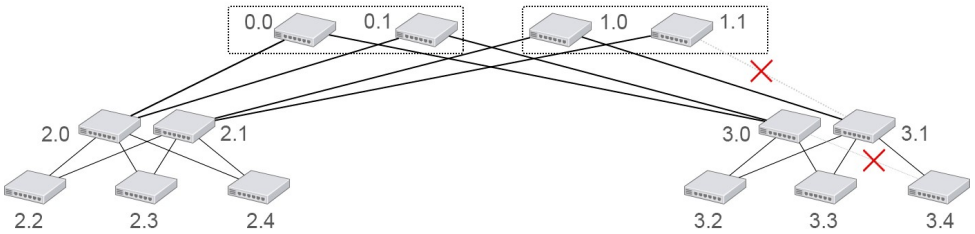

FIGURE 6.10: Modified clos DCN topology with multiple failures.

First, let us see how the failures affects to the definition of the neighbour groups of some nodes. In this case, any node of pod 0 (2.∗) neither any of the spine plane 0 (0.∗) is affected as they are far from the failures. On the other hand, all nodes with link failures (1.1, 3.0, 3.1 and 3.4) get some of all their groups modified, in this case removing the disconnected neighbour from them (e.g. node 3.4 redefines groups A and B as $\{3.1\}$.

In the case of exceptions, the failures affect to more nodes. In this case, we see that ToR switches both pods require exceptions to ToR 3.4 (except for itself), in case of ToRs in pod 0 (2.2, 2.3 and 2.4) directing to 2.1 and for those ToRs in the same pod, directing to the fabric switch 3.1. At fabric switches we get a more varied scenario. In this case, node 2.0 may get either an unreachable exception for destination 3.4 or one using any ToR neighbour to reach it (Group A), as ToR 3.4 is disconnected from its fabric in that spine plane. On the other hand, at 2.1, the failure between 3.0 and 3.4 does not affect, but, as the fabric switch 3.1 is disconnected from the spine switch 1.1, it requires an exception to the whole pod 1 ($3.*$) directing to 1.0. In contrast, fabrics switches in pod 1 (3.0 and 3.1) do not require any exception thanks to their modified groups of neighbours. Finally, spine switches do not need to store any exception. The reason of leaving spine switches without exceptions, is that, for a rule to become invalid, the path itself has to be down, so the previous nodes in the path would not forward the PDUs to that spine in the first place.

Considering the state of the forwarding policy in this scenario, let us see how forwarding would be done between some ToR nodes, in the same or different pods:

From 2.2 to 2.3:

- At 2.2 (ToR). Destination in same pod, use any of group A $\{2.0, 2.1\}$.
- At 2.0 or 2.1 (Fabric). Destination is a connected neighbour, forward.

From 3.2 to 3.4:

- At 3.2 (ToR). There is an exception to node 3.4, forward to 3.1.
- At 3.1 (Fabric). Destination is a connected neighbour, forward.

From 2.2 to 3.2:

- At 2.2 (ToR). Destination in another pod, use any of group B $\{2.0, 2.1\}$.
- If chosen 2.0
  - At 2.0 (Fabric). Destination in another pod, use any of group B $\{0.0, 0.1\}$.
  - At 0.0 or 0.1 (Spine). Forward to Fabric switch of destination pod $\{3.0\}$.
  - At 3.0 (Fabric). Destination is a connected neighbour, forward.
- If chosen 2.1
  - At 2.1 (Fabric). There is an exception to pod $3.*$, forward to 1.0.
  - At 1.0 (Spine). Forward to Fabric switch of destination pod $\{3.1\}$.
  - At 3.1 (Fabric). Destination is a connected neighbour, forward.

From 2.2 to 3.4:

- At 2.2 (ToR). There is an exception to node 3.4, forward to 2.1.
- At 2.1 (Fabric). There is an exception to pod $3.*$, forward to 1.0.
- At 1.0 (Spine). Forward to Fabric switch of destination pod $\{3.1\}$.
- At 3.1 (Fabric). Destination is a connected neighbour, forward.

While exceptions are useful to avoid failure paths re-route PDUs for secondary ones, we still require a mean to compute those exceptions. In this regard, routing policies are responsible for providing enough information to populate the exceptions to primary forwarding rules. This does not substantially differ from what would be done with a traditional forwarding table. In order to compute exceptions, we could use a generic link-state or distance-vector routing protocol, and simply compare the resulting entries with those that forwarding rules would provide. Even so, in the same way as rules allow us to reduce the amount of information required for forwarding, we can also enhance routing by exploiting the complete knowledge about the DCN topology that all nodes have. Indeed, there is no need for all nodes to propagate the state of operational resources across the network or to compute routes to all nodes, but only propagate and compute failure information. To this end, we propose a distributed and a centralized routing policy that take advantage of the topology of the DCN and failure information to reduce both the computational and communication costs required to compute the exceptions to primary rules.

6.2.4.1 Distributed routing

Generic link-state algorithms assume no knowledge about the network graph and even from the expected connectivity in the current nodes. Therefore, nodes need to share all their connectivity information in order to compute how packets should be forwarded. When considering a network where, in the failure-less scenario, the network graph is completely known beforehand by all nodes, then these restrictions disappear, becoming only necessary to disseminate network changes. In order to propagate failure information, we propose a variation of link-state routing, based on the propagation of failed links solely.

In this routing policy, we assign to each link a unique name that also provides us an easy way to locate the link's owners (e.g. link name = *src.dst*). In order to differentiate link's updates, each link maintain an update-sequence-Id, synchronized between remote endpoints, and incremented each time the link status change from ON-OFF and vice versa. When a link status change occurs, both endpoints propagate the new status to all their neighbours that, not knowing the new status already, continue propagating it until all nodes are aware of the change. Therefore, there is no big difference with any generic link-state routing. The policy propagates link instead of node status, halving the amount of updates in the best case (i.e., the status of one link instead of two nodes is propagated). However, the bigger improvement comes from the fact that we only need to propagate and store failure-related information. Hence, the network can run without specific routing, provided that (single- and multi-) failure situations are not too common over time.

Nevertheless, there exist some requirements for the proposed routing policy. The first one is that, in order to avoid all nodes sending updates of disconnected neighbours when starting the network, it is required to have a small delay between the initialization of

the network and the bootstrapping of the routing policy (at least enough to give a node enough time to connect to all neighbours). This should not be a problem, but something to be taken into account.

Another issue is to decide at what moment an old update can be discarded. This is the case when a failure is repaired and an ON update is propagated in the network to notify the change in the topology; as the network is back to the no-failure state, this information can be discarded. In order to allow all nodes to get this ON update, this information is stored during a limited period of time. Nonetheless, it could happen during a failure that a node or a group of nodes remain disconnected from the rest of the network, which cause that an update never reaches them if dropped. In this case, there are two possible solutions: nodes with one or more failures may not drop old updates, avoiding ON updates to be lost before reaching all the network; or a reactive approach can be taken, in which a new update is propagated when an old (outdated) update arrives at one of its source.

With the expected knowledge of the network graph and shared knowledge about link failures, computing either the forwarding table or, in this case, the exceptions, may be something as trivial as using the Dijkstra algorithm to compute the shortest path to each destination. However, we are only interested in computing those exceptions for currently unreachable destinations. Instead of re-computing the entire forwarding table using Dijkstra we propose algorithms that take advantage from the DCN topology knowledge, focusing the search on such problematic DCN regions (where unreachable destinations are), allowing to add some routing restrictions if desired.

```
1  Clean old exceptions (e.g., all from pod if has new changes)
2  Parse and sort new link failures:
3      Pod -> {ToR switch -> Fabrics switch, Fabric switch -> ToR switch, Fabric switch -
       >     Spine switch}
4      Fabric switch -> {Pod -> Spine switch, Spine switch -> Pod}
5  Initialise groupA and groupB as a list of alive neighbours.
6  Check in-pod failures (if changes in pod):
7      Remove from groupA all fabric switches of the same pod with problems reaching all
       other ToR switches or edge routers.
8      For Each other ToR switch or edge router with problems in the same pod, create
       exception if cannot be reached through all groupA.
9  Check out-pod failures (for changes in pods):
10 For each alive neighbour:
11     If disconnected from all spine switches, remove from groupB and check next.
12         Check pods with problems at the same fabric switch and mark as unreachable
       through the current neighbour if there are no shared spine    switches available
       or the destination fabric switch is not connected to any ToR switch or edge router
       .
13     Create pod exceptions to pods unreachable by all fabric switches in groupB.
14     For each pod with problems at Tor switches or edge routers:
15         Initialize valids as groupB or the list of valids if it has an exception.
16         For each ToR switch and edge router with problems in the pod:
17             Generate an exception if cannot be reach from all valids for pod.
18 Update groups A and B in forwarding policy as groupA and group B respectively.
19 Encode exceptions and replace update the forwarding policy.
```

PSEUDO-CODE 6.4: Description of distributed routing exception computation mechanism at a ToR switch in the MC topology

In Pseudo-code 6.4, we find a simple example of how exceptions could be computed at ToR switches and edge routers in a DCN describing the modified clos topology (with similar algorithms being possible for other nodes and scenarios). In order to compute exceptions, we first do a fast pass cleaning the previous state, clearing old exceptions (e.g. towards pod new failure/recovery) and restarting neighbour groups, as well as parsing the known failures. Then, we search for reachability problems within the same pod (if any failure), removing from group A those neighbours disconnected from the rest of the pod and creating exceptions to ToRs with failures. Then, if there are failures outside the pod, first we check if our neighbours are connected to the other pods (removing them from group B if not), and then, for each pod with failures, search for problems reaching either the whole pod or specific ToRs in it. Finally, with the groups and exceptions recomputed, we update the forwarding policy with the new information.

In this example, we can see that, while algorithms required in these cases are more complex than general solutions and are completely dependent on the topology as well, they significantly reduce the computational cost of computing exceptions. Another benefit of the algorithms used to compute exceptions is that they can be designed with some routing rules in mind. For example, considering that only flows to ToR switches and edge routers will be established (apart from the point-to-point ones), we can only contemplate exceptions toward entire pods and specific ToR switches and edge routers. This way we can skip exceptions to fabric and spine switches, while at the same time avoiding paths through fabric switches not connected to any ToR switch or edge router.

In order to compute exceptions in a fast way, those algorithms do not consider to find the best path for all possible failure scenario, but limit the possible resulting paths to only those close to the optimal ones. This constrain mean that, in a very infrequent case where no good path would remain alive, no other path would be eligible. Such constraints may be restrictive in some cases (e.g. the pseudo-code on Fig.6.4 considers only optimal paths, while it could have also considered sub-optimal paths within the same pod). However, they still do not represent a true reduction of the forwarding capabilities of the DCN, given the available high connectivity. In contrast, these constraints allow us to define what we consider as invalid paths, namely, paths for which their performance would be under our expectations, introducing unacceptably large latency and extra bandwidth consumption. Instead, in those really improbable cases, we consider that two nodes are unreachable between them and that, being more cost-effective in those cases to simple move a virtual machine to a reachable node.

6.2.4.2 Centralized routing

Following the SDN concept [86], which fosters centralisation of network control to relieve the computational requirements of the forwarding devices, we also propose a variation of the previous routing policy that takes advantage from the computational resources available in the DC to centralize the computation of exceptions. In Fig. 6.11, we can see a small example of this approach that could be deployed in a DCN describing the MC topology.

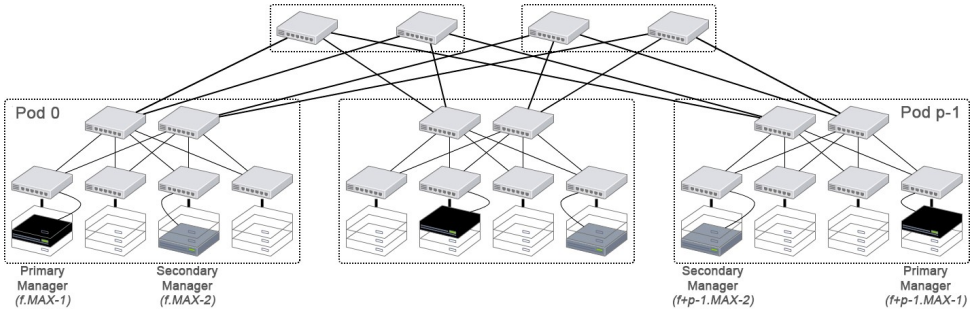

FIGURE 6.11: Location of managers for centralised routing in a DCN describing the MC topology.

In this case, we have two or more servers per pod acting as primary (P) or secondary (S) routing managers. These managers are the ones in charge of configuring groups and exceptions for all ToR switches, edge routers and fabric switches in the pod, while the rest of nodes only set-up their forwarding policies and perform quick modifications to groups (e.g. when a link goes down, before advertising the event to the managers).

Managers are assigned addresses in the form of "$pod_{id}.manager_{id}$", similar to ToR switches, so that forwarding toward them can be done by forwarding rules. In addition, an anycast address "$pod_{id}.MAX$" may also be used as an alias to the primary manager (or the next reachable secondary one), so that the destination of routing updates do not depend on the specific addresses of the managers. As nodes within the pod cannot rely on the managers to compute the paths towards them, a distance-vector routing algorithm, with its scope limited to that of the pod, is used to compute the paths to all managers of the pod, and specific exceptions are added if needed.

When a node in the pod detects a failure or recovery event, it informs the pod primary manager about the change. Once the manager gets the update, it synchronizes its knowledge with the rest of managers in the same pod. Then, it computes a baked update with information about how to reach ToR switches and edge routers, whose reachability is affected, and sends it to the primary managers of other pods. Finally, it computes and propagates the exceptions for all the nodes in the pod. Fig. 6.12 depicts a simplified version of the message propagation between nodes and managers in case of a failure.

When a failure situation is detected in spine switches they inform the managers of all connected pods, but no exception is computed for them. This is a small simplification,

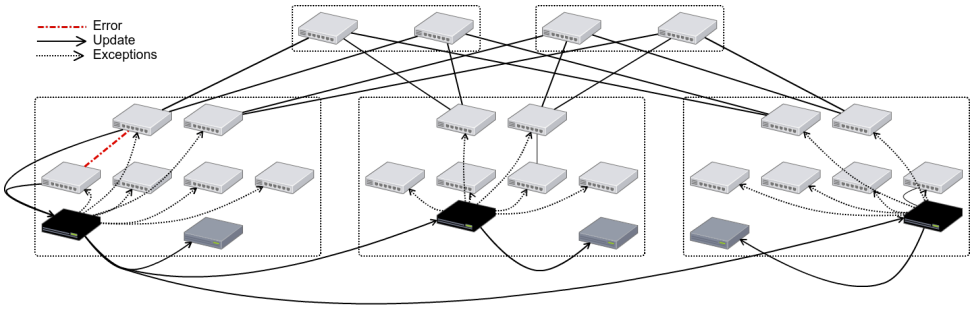

FIGURE 6.12: Example of routing update process in a DCN describing the MC topology (centralised routing).

based on the fact that there are no end-to-end flows directed to any spine switch (apart from possible ACKs to routing updates). Given that, any possible exception at a spine switch would only be used for a small period of time before a new stable state that avoiding its usage is established. This also avoids possible forwarding loops, while reaching that new stable state.

In addition to the benefits that moving most of routing computation to a few servers entails in terms of requiring less computational resources at forwarding devices (thus making them cheaper), this solution employs available resources. Also, given that updates only need to be shared between managers, and that not all failure situations require new exceptions, the total number of routing messages ends being greatly reduced with respect to any distributed solution, where any change could end being propagated through the entire network.

```
1  --Baking updates--
2  From failures at fabric switches
3      List of fabric switches disconnected from all spine switches and or ToR switches
       and    edge routers.
4      List of failures towards spine switches from connected fabric switches, if any.
5  Compute the distance matrix of nodes in the pod:
6      For ToR switches and edge routers without the same distance to all connected
       fabric     switches, share the distance towards all the pod fabric switches.
7
8  --Computing exceptions--
9  For each fabric switch:
10     Check connectivity with ToR switches and edge routers and compute group A.
11         If not disconnected, add to default A.
12     Check connectivity with spine switches and compute group B.
13         If not disconnected, add to default B.
14 For each ToR switch and edge router:
15     Compute groups A and B from the neighbours with minimum distance to any of default
        A and B respectively.
16
17 =Compute in-pod exceptions=
18 Exceptions from the distance matrix comparing to groups.
19
20 =Compute out-pod exceptions for pods with failures=
```

```
21 For each pod with some failures:
22     Compute the list of fabric switches that reach a connected fabric switch at the
       destination pod.
23     If all disconnected, create an unreachable pod exception at all nodes and go to
       next problematic pod.
24     For each ToR switch and edge router:
25         Check if neighbours of group B takes the same minimum distance to reach any of
        the fabric switches connected to the destination pod, if not, create an exception
       .
26     For ToR switches and edge routers with failures at destination
27         Compute the distance from our fabric switch to the node from connectivity and
       shared distances.
28         Add unreachable exception at fabric switch if shared distance for it is
       INFINITE.
29     From each ToR switch and edge router:
30         Check if neighbours of group B take the same minimum distance to reach the
       node (distance to fabric switch + from fabric switch to node), if not, create an
       exception.
```

PSEUDO-CODE 6.5: Description of centralized routing exception computation mechanism at a pod manager in a DCN describing the MC topology

In Pseudo-code 6.5, there is a small example of the baked data shared with managers in other pods and how exceptions and groups can be computed after changes in a DCN describing the MC topology (similar algorithms could be used for the other topologies). In this example, each manager manages a unique pod and therefore shares failures and compute only exceptions for its nodes. To compute exceptions, it first computes the default groups A and B for ToR switches given the connectivity of fabric switches and specify them for each ToR depending on their failures. Then, given the already computed distance matrix, it searches for exceptions within the same pod and, for each other pod with known failures, it searches for disconnected fabric switches to those pods and use the distance between ToRs and fabric switches in each side to compute exceptions to the other pod and its ToRs. Regarding the updates, we share baked information about the connectivity of edge routers, ToR and fabric switches in the pod. Instead of the raw list of failures, we pre-compute if there are unreachable fabric switches and how far are ToR switches and edge routers from fabric switches. Having these baked exceptions per pod allows us to greatly reduce the computational cost compared to having to consider all the network connectivity at the same time, computing exceptions to each pod once at time.

Unlike with distributed routing, here we relax the restrictions on valid paths. While we do not allow to use intermediate pods when forwarding, we allow the use of any secondary path within both the source and destination pods. Even so, as exceptions are extracted from already computed distance matrices of pods, the resulting complexity is not higher than considering only optimal paths. In addition, as problems in one pod do not affect the way to reach other pods, only exceptions to the affected pod have to be recalculated upon changes.

Note that in this scenario we assumed ToR switches (and therefore servers) at each pod. If not, this solution would be still possible by adding a manager server to those pods with only edge routers therein. This happens in DCNs describing the GO topology. Another solution could be to have manager servers connected directly to fabric switches and interconnected between them, as can be seen in Fig. 6.13, where few primary and backup managers can be used to manage the full network in a scalable way. This solution does not use computational resources initially available in the DCN, but works well with any of the considered topologies, while at the same time reduces the cost of routing updates. In this case, the number of managers can be reduced, as each manager can compute the exceptions of more than one pod. Moreover, only status updates from fabric switches are required to know whether a link or node is down (considering in this case as down a fabric switch non-reachable through any manager). As an additional note since the management layer would be something fully separated from the DCN forwarding, it is not required for managers to have its own addresses within the DCN address space, but can work in a smaller, disjoint management DIF.

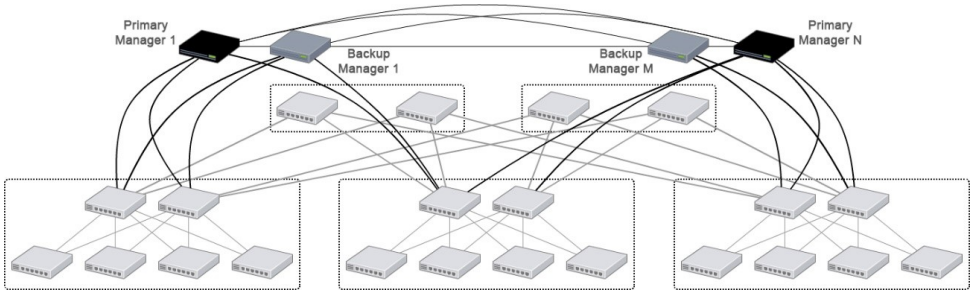

FIGURE 6.13: Alternative manager placement for centralized routing at a DCN describing the MC topology.

6.2.5 Numerical results

Current forwarding and routing policies based on TCP/IP impose multiple limitations that the use of the RINA architecture already solves. For instance, in RINA we are not forced to use 4 or 16-byte addresses as imposed by IPv4 and IPv6, respectively, but can use scenario-specific addresses, resulting by itself in both smaller memory usage and routing updates. In addition, as RINA use a recursive layering and a naming scheme that differentiates node location from address, hence, address of servers and virtual machines do not need to be known and do not affect the routing in the Fabric DIF. This facilitates multi-homing and mobility at the same time that reduces the scope of routing within the DCN. While the benefits of adopting RINA are enough to contemplate its usage inside DCs, we also want to quantitatively evaluate the scalability of the proposed forwarding and routing policies against that of currently available solutions migrated as-is into RINA. Being the use of topological solutions, in special one using the previously presented R&E forwarding policy, the main benefit of this, we firstly compare the number of entries that would be required in a traditional forwarding table and their size, against those required

using our R&E solution. In order to perform this comparison, we considered all DCN topologies presented in Table 6.5, all of them representing large DCs with 8192 ToR switches and 2048 edge routers.

TABLE 6.5: Assumed values for the parameters describing each of the considered DCN topologies

Scenario	p	t	f	s
Generic Leaf-Spine (GLS)	128	64	4	2048
Facebook old and new (FB1 and FB2)	128	64	8	256
Google (GO)	160 (128+32)	64	4	128
Modified Clos (MC)	160 (128+32)	64	4	64

Note that a configuration like the one used for the GLS-based DCN is quite unrealistic, being impossible to scale that topology in terms of edge routers without incrementing dramatically the size of fabric switches (number of ports). Even so, we keep this scenario in the results, as it provides us an illustrative example, where fabric switches have many neighbours.

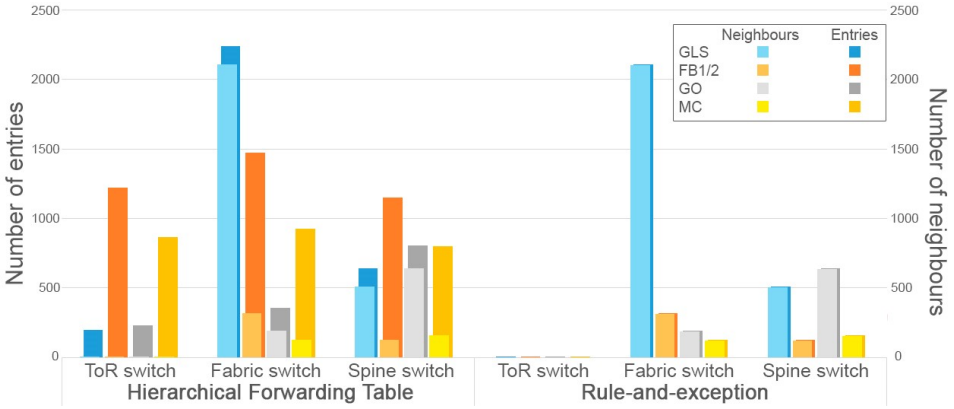

FIGURE 6.14: Number of neighbours vs. required entries in the different DCN topologies without failures.

Fig. 6.14 shows a comparison of the number of entries required in the non-failure scenarios for the different node locations and forwarding policy in each of DCN topology. We considered 2 forwarding functions:

- Hierarchical forwarding table: Stores one entry per aggregated set of addresses (i.e., superior scalability than a flat addressing scheme with an entry per address in the network).
- Rule-and-exception: The proposed policy, which stores one entry per neighbour and group.

In each case, we considered the entries required for neighbour nodes and those used for forwarding packets to remote (non-neighbour) nodes or groups of nodes (e.g. pods). It

can be seen at first sight that almost no information has to be stored at ToR switches when using our R&E policy, compared to the required entries with hierarchical forwarding tables. This is really interesting, as they are the forwarding hardware most used in DCNs. For fabric and spine switches, these reductions in the number of stored entries is not so remarkable, due to the large number of neighbours that these nodes have in some of the considered DCNs (e.g. in the GLS topology, fabric switches have more than 2000 neighbours each). Even so, it has to be noted that in this case, all nodes encounter reductions from hundreds to thousands of stored entries.

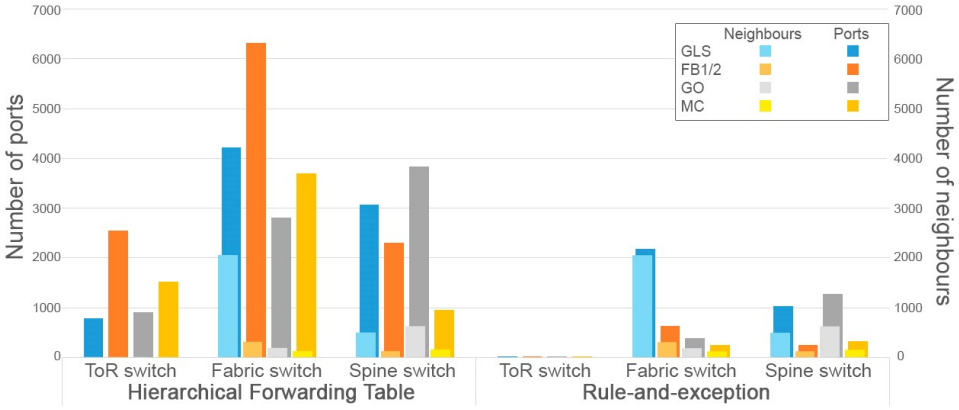

FIGURE 6.15: Number of neighbours vs. required stored ports in entries in the different DCN topologies without failures.

As shown in Fig. 6.14, the proposed R&E forwarding policy lowers the number of entries to be stored at DCN devices significantly. However, this evaluation does not show their real memory requirements. To illustrate this, Fig. 6.15 compares the number of ports stored among all entries required in the non-failure DCN scenarios, independently of the entry encoding used. As can be seen, while it is true that the number of stored entries between hierarchical forwarding tables and the proposed policy is more or less similar, memory requirements differ substantially. While the R&E forwarding policy requires to store only neighbouring node information (storing them at most once per defined group), in forwarding tables, apart from neighbour information, we store not only one port per destination, but some of the neighbours that can be used to reach it. As the number of ports eligible to reach a destination may be really huge (e.g. any spine switch can be used to reach other pods from a fabric switch), we have imposed a limit of 16 ports per entry, limiting the size of forwarding entries. Even with the limitation of storing at most 16 ports per entry, which has a negative impact on load balancing, the difference between forwarding tables and rules is clear in this case.

When considering failure scenarios, the R&E forwarding policy uses either the modification of groups or exceptions to avoid the existing failures. In Fig. 6.16, we depict the relative number of entries, with respect to the number of pods, for an MC-based DCN of 160 pods (a relation 4 ToR switch per edge router) in different failures scenarios with 0, 1, 2, 5 and 10 failures each (either node or link failures). In fact, at most one exception per failure is required by the R&E policy, resulting in a negligible increment

in stored forwarding information, something ensured by the special connectivity of DCN topologies. Instead, with hierarchical forwarding tables, some failures can prevent the aggregability of some groups of addresses. As a result, we can encounter some noticeable size increments of the hierarchical forwarding tables.

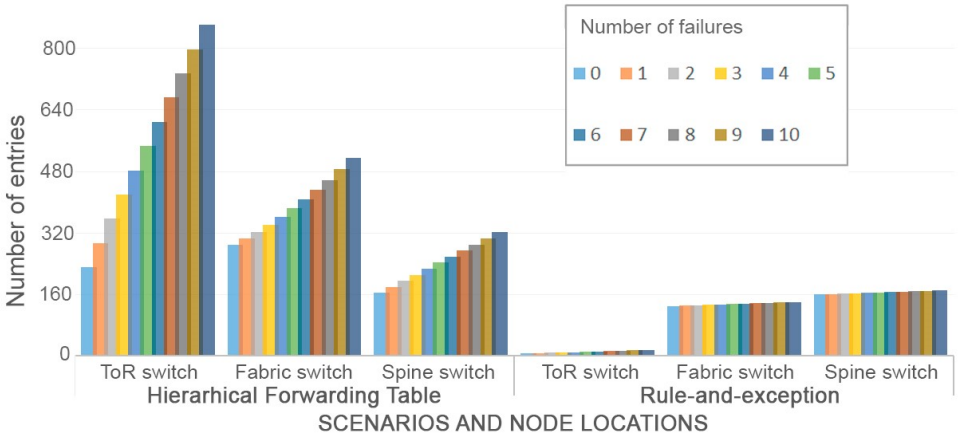

FIGURE 6.16: Comparison between the average number of entries per forwarding node in scenario MC with 1 to 10 concurrent failures.

In addition to the number of entries and table size, we are also interested in comparing the scalability of the proposed R&E forwarding policy as the DCN grows up. Fig. 6.17 shows the average number of entries and stored ports in the GO and MC-based DCN scenarios (with similar results for other leaf-spine and clos variants). In these scenarios, we considered the same parametrization as before, but varying the number of pods (having each pod in average $64 * 4/5$ ToR switches and $64/5$ edge routers). We considered a non-failure scenario and, for the scenarios using hierarchical forwarding tables, the same limit of at most 16 stored ports per forwarding entry, typical of ECMP. In the figure, we can see how, while the aggregation of addresses implies a notable improvement with respect a flat solution, the number of forwarding entries and their size grows steadily with the number of pods. In contrast as the R&E policy only requires the storage of adjacent neighbour's information, it remains almost constant as the size of the DCN grows up.

Although the benefits of our proposal against hierarchical forwarding tables (with or without aggregation of addresses) are good enough to justify topology dependent policies, the computational and communication costs of searching exceptions, given the specific number of failures, should be also considered. Regarding the latter, we have a clear improvement in the sense that, as nodes know the DCN topological characteristics, they can take some knowledge as granted. With this knowledge, we can avoid most of the initial propagation of routing information flooding that any common link-state or distance-vector routing protocol needs for populating nodes' routing (and forwarding) tables. In addition to this, TCP/IP solutions tend to require some type of refresh of routing information to ensure that the knowledge is updated. In this regard, as RINA DIFs can itself provide reliable communication between nodes and our policies work based on link status (with has to be synchronized between both extremes), routing refreshes become unnecessary,

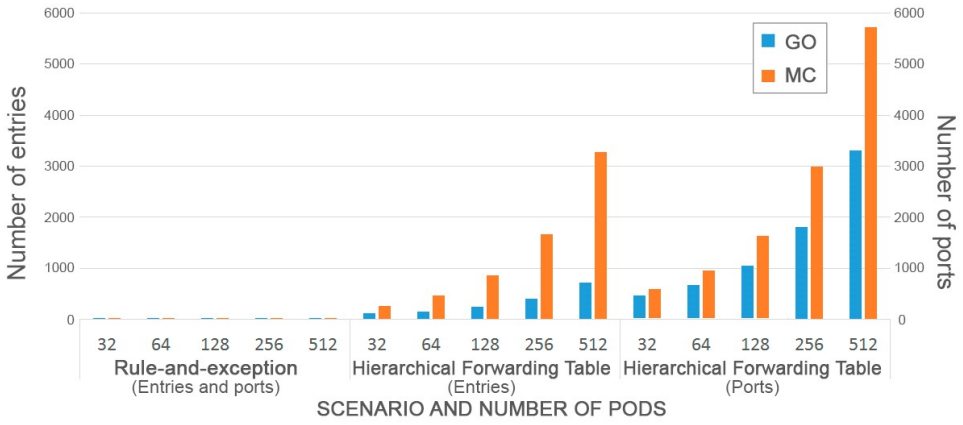

FIGURE 6.17: Average number of entries and stored ports for different number of pods in GO and MC-based DCN.

both in distributed and centralized approaches. Given that, only in case of failure and recovery events should routing updates be propagated. In this regard, the distributed approach shares a similar cost as any other link-state protocol, having to propagate the new update to all the network, although the number of messages can be split in half as the same update is propagated from both extremes of the link failure, instead of the link state at both extremes. In the centralized cases, it all depends of the approach used when locating manager servers. A quick approximation would require a status update per manager plus an exception update for any node requiring a re-population of exceptions. Anyway, the size of routing messages, both in the distributed and centralized approaches, should not exceed one packet in most if not all cases.

Finally, in terms of computational cost, in order to validate our proposed approach to computing forwarding exceptions, it is important not to exceed the cost of traditional solutions based on Dijkstra routing algorithm. We take as an example the pseudo-code proposed for computing exceptions at ToR switch and edge routers in the MC topology in the distributed approach (Pseudo-code 6.4) and the one for computing exceptions in the centralized case of the same scenario (Pseudo-code 6.5). To simplify the results, we consider the same parametrization described in Table 6.5, and scaling the DCN based in the number of pods (p). We also consider the number of failures (r) as a parameter to describe the complexity of each approach.

With Dijkstra-based approaches, the complexity of computing exceptions would grow linearly with p, as the number of nodes does the same. For the proposed approaches, centred on the failures in the network, there are two main scenarios possible: processing a failure in the same pod or in another one. If the failure is in the same pod (probability $1/p$) we need to compute exceptions for all failures in the DCN, with a linear cost in the number of failures. If the failure affects a different pod (probability $1 - 1/p$), then we need to compute exceptions only to that specific pod, being that a near constant function. Since the number of concurrent failures in these types of networks is small by design, we found that those bounds carry a great improvement. Moreover, we should consider that

the probability of having to take the most complex route upon a failure when computing new exceptions reduces as the DCN grows. In addition, since these networks tend to operate in the non-failure scenario most of the time, this represents a big performance improvement as we only require the constant cost of checking that there are currently no failures in the network.

6.2.6 Conclusions

This section extends the use of the R&E forwarding policy for RINA-enabled large-scale DCNs proposed. These policies take the knowledge of the DCN topologies in order to provide a superior efficiency and scalability, achieving fast and successful forwarding decisions in non-failure scenarios, with merely requiring information about neighbouring nodes. Upon DCN link or node failures, forwarding exceptions are computed and stored at forwarding nodes to override decisions of primary rules that have become temporally invalid. To minimize the size of stored exceptions, the R&E forwarding policy allows a reverse encoding, storing only a list of unreachable neighbours for forwarding, instead of the full list of valid ones. This yields a significant improvement in memory usage, given the large number of redundant paths in DCNs and similar networks.

Additionally, it also proposed complementary routing policies taking advantage from the known topologies and not having to compute the full forwarding table in order to reduce largely the communication and computational cost of routing. These routing policies, instead of sharing all connectivity information, disseminate only failed link information, largely reducing the communication cost. The obtained results illustrate the scalability of the proposed topological forwarding and routing policies. The interested reader can experiment with the proposed policies in the online tutorial available in [116].

6.3 Compact Routing solutions

Large networks like the current Internet have multiple scalability-related problems related to the common use of flat routing schemes, requiring increasingly more and more resources (computational cost, memory usage, routing updates, etc.). RINA, with his dynamic and recursive layering, bets for a new Internet solution based rather in the use of multiple "smaller" networks (sometimes created on demand) that would interconnect only users when connectivity is required, instead of providing a global connectivity. Even so, a "small" network in comparison to the current Internet may still be a large network in comparison to anything else.

Looking back to the previous sections on topological routing (6.1 6.2), there are multiple cases where the size of the network may not impose a scalability problem by itself, as sometimes there are more graceful forwarding and routing solutions than simple flat

routing ones. On the other hand, while it is possible to provide smart solutions that use the topology, the topology also has to be benevolent toward routing solutions, and any random environment cannot take profit from those kind of solutions.

In large networks like the Internet, building the network from scratch with a suitable topology for smart routing is not commonly a viable solution. First, in addition to the huge cost of replacing the already existing infrastructure, if dealing with networks that has to cover large areas of land, the topology of the network would be already restricted by that of the land (e.g. neighbouring nodes in the network graph will commonly be also near physically). In addition, when considering the management of those large networks, it has to take into account also that does networks are not always a unique large domain, but instead tend to be formed by multiple smaller networks, each managed by a different player. Moreover, the constant increase of players and the different changes in their connectivity makes impossible to centralize the management of those large networks and enforce any kind of network topology.

Given the difficulties to use any other kind of routing within large networks (at least those described previously), most common production environments use one of the multiple generic solutions for routing in flat (or almost flat) networks. While those solutions work as intended, the lack of a central management capable of organize the network forces them to put all the weight of routing in increasingly large routing tables.

6.3.1 Fundamentals of compact routing

In contrast with generic solutions that simply compute the best paths between any pair of nodes in any network (e.g. the shortest path) at the cost of large routing tables, compact routing schemes are strategies that allow more scalable networks providing a trade-off among:

- Routing Table Size
- Address and Header Size
- Stretch

In order to fulfil its goal of route packets between any node a and b, compact routing solutions do not use only the information on forwarding tables, but take profit from additional information on the network or the destination node. The use of this extra information, commonly encoded either at the same node address or as additional headers, while slightly increments the size of the different packets traversing the network, allows for solutions that require only sub-linear routing tables, resulting in an important push towards the scalability of the network. However, compact routing solutions do not only suffer from slightly longer packets, but, in order to reduce routing table sizes, these

solutions do that at the expenses of not ensuring that packets will follow the best path, but only that they will arrive to their destination. Even so, while not ensuring the use of the optimal paths, compact routing solutions ensure a bounded maximum stretch in the network, ensuring that the resulting paths would not be too distant to those of flat routing solutions.

There exist different approaches when it comes to compact routing solutions [141]. For instance, most compact routing schemes work defining spanning trees within the network, each with a different root, and route packets based on the relative distance within those trees of neighbour nodes and the destination. Other solutions, instead base their behaviour in the definition of clusters of nodes (in a similar way as to how path-finding algorithms for GPSs works). As those compact routing solutions do not use addresses in a traditional way, given the tight relationship between addresses and nodes in the TCP/IP model, those solutions do not fit right within current production environment. In contrast, RINA, with its support for a complete separation between node name, address and route, and the open environment provided by the use of programmable policies, provides a perfect environment for the use of compact routing solutions.

6.3.2 Landmark based routing schemes and RINA

Given the possibilities that RINA gives to the implementation of compact routing solutions in production environments, this section shows how a simple landmark [142] based routing, i.e. a compact routing scheme based on landmarks and clusters of nodes, could be implemented as a primary routing mechanism in a DIF. Landmark routing schemes are simple compact routing solutions that mimics how routes are communicated in real life. Whenever a packet is close to its destination, landmark routing gives the direct path towards it (flat routing in the proximity). Otherwise, when the same packet is far from the destination, landmark routing gives the route towards a well-known location, a "landmark", close to its destination. In that case, there are two possible outcomes, a packet may arrive first to its landmark and from that go to the destination, or in its path to the landmark it may arrive to a node that knows how to reach the destination and change the route without passing by the landmark. This has some benefits with respect to similar multilayer solutions (e.g. adding intermediate DIFs for landmark-to-landmark communication and landmark vicinity and use landmarks as border routers) in the sense that it is common to never reach the designed landmark in most communications, being that simply used as a reference point.

Considering the implementation of this routing scheme in a RINA DIF, only few policies have to be taken into account: a DIF management system that provides the address and landmark of destination nodes; a forwarding policy that routes to either the destination node or its landmark; a pair of routing and forwarding generator policies that populates the forwarding policy.

First, let us consider how the landmark information would could be encoded in the PDUs headers. Given that RINA provides a complete separation between node name and addresses, instead of adding extra headers fields, it is easier to simply use the addresses of nodes to encode both destination and landmark information. As RINA gives high freedom to the configuration of addresses, this can be easily done with the use of addresses in the form $landmark_{id}.destination_{id}$. As RINA does not put any impediments to the re-addressing of nodes or multi-homing, a node address do not only may change in time (e.g. leaving the landmark proximity after a network change), but can have different addresses, effectively using multiple landmarks at the same time. As a note, destination and landmark identifiers can be randomly distributed, as long as they are unique. In addition, both destination and landmark identifiers can be part of the same address space, being then the $landmark_{id} = destination_{id}$ for the landmark nodes, even so, given the smaller number of landmarks (of the order of $\sqrt{N}$), having landmark identifiers in a secondary address space could reduce node addresses to 3/4s of its length.

When it comes to the sharing of node information between nodes, RINA already hast the means by default. As stated in Chapter 4, the establishment of a RINA flow begins with locating the destination node given its node name (application name and instance). This process can be done in multiple ways depending on each DIF, but the main approach is to maintain a distributed database of node name to node addresses and query to it. With this distributed database, whenever a node enters or leaves the proximity of a landmark, it only has to register or de-register the address related to that landmark ($landmark_{id}.destination_{id}$), as would be done with any other routing scheme. This by itself solves the problem of mapping nodes to the landmark based addresses, as whenever a source node wants to establish a new flow, it only needs to query the list of locations for the destination node an use the one closest to it (e.g. use the address with the closest landmark). On the other side, as the source node already knows all its addresses without requiring external information, it can use the one using the nearest landmark as default address. As a note, given that landmark nodes always maintain a fixed address (as they are their own landmark), those become also potential candidates for the location of the servers for the distributed database.

Knowing the address ($landmark_{id}.destination_{id}$) of nodes, forwarding policies within the DIF can be simple, and most of the default policies could work without problems. For example, the same R&E forwarding policy introduced in Section 6.1 could be a valid policy for this scenario, considering that using empty rules it would work similar to a common forwarding table. In this case, only two type of entries would be needed: Neighbour entries ($*.destination_{id}$), that direct towards close nodes; Landmark entries ($landmark_{id}.*$), that direct towards landmarks. In this case, the order of the entries would be important, as neighbour entries should be checked before landmark ones.

Finally, as the forwarding table would need to be populated, some kind of routing policy should be in place. For this routing scheme, it is necessary to distribute routing information on landmarks along the entire network, but information about the rest of

nodes only up to a certain distance. In that case, a small modification of distance vector-routing could be used, having different infinite measures for landmarks and the rest of the nodes in the network (e.g. network diameter for landmarks but 3 hops maximum for non-landmarks). With those modification, all nodes would learn how to reach all the landmarks in the network, while at the same time avoid storing information from nodes outside their proximity. It has to be noted that, in this case where the vicinity of nodes has a fixed diameter, that has to either ensure that all nodes are close enough to a landmark (in order to be in its vicinity). If that is not possible, variable reach for the vicinity is also possible, for example having each node setting the maximum path length towards it as the distance to its closest landmark.

Fig. 6.18 shows a complete example of how compact routing can be used in a RINA scenario. First, some nodes in the network are designed as landmarks, in this example, nodes 0(A), 6(B), 5(C) and 7(D). The different nodes in the network then initialize their routing module and compute the paths to all nodes within its reach and all the landmark nodes in the network. For example, node 1 has nodes 0 and 2 within reach, then it stores an entry for node 2 (node 0 not needed as it is already a landmark). Those also computes an entry for all and each one of the landmarks in the network. Now, all nodes in the network (landmarks skipped for simplification) register to the nearest landmarks, being those at the same time servers for the distributed database of nodes to names, and those same landmarks take care of distribute the information between them.

When the node 1 wants to allocate a flow towards the node 4, it first query 0 for the list of locations of 4. 0 responds with all the known addresses for 4 ($B.4$ and $C.4$), and as the distance between 2 and B is less than towards C, it decides to reach 4 using $B.4$. Then, 1 send a flow allocation request from $A.1$ to $B.4$. While the expected path between 1 and 4 is then supposed to go to B and then to 4, when arriving at node 3, it already knows the path to 4, so the flow is re-directed using a more direct path. In the other direction then happens something similar, as when the flow reach node 2 it already knows how to go to node 1 so the path is also re-directed. Note that, while in the first case, node 3 is also using B as a landmark, in the second case, nodes 1 and 2 do not share a landmark, but both are at reach from each other. In the same way, it could happen to arrive at a node using the same landmark than the destination node, but that does not have it at reach.

6.3.3 Conclusions

One of the worst hindrances for the scalability of networks is a bad planning of it (or no planning at all). In networks like the Internet, where multiple factors lead to that situation (e.g. network has to "follow" the land, multiple players, etc.), it is important to improve the reduce the cost of scalability in any possible way. As routing solutions that take profit from the topology of the network are not possible in those networks, the focus goes to less optimal solutions that, unfortunately all have some kind of drawback. In this

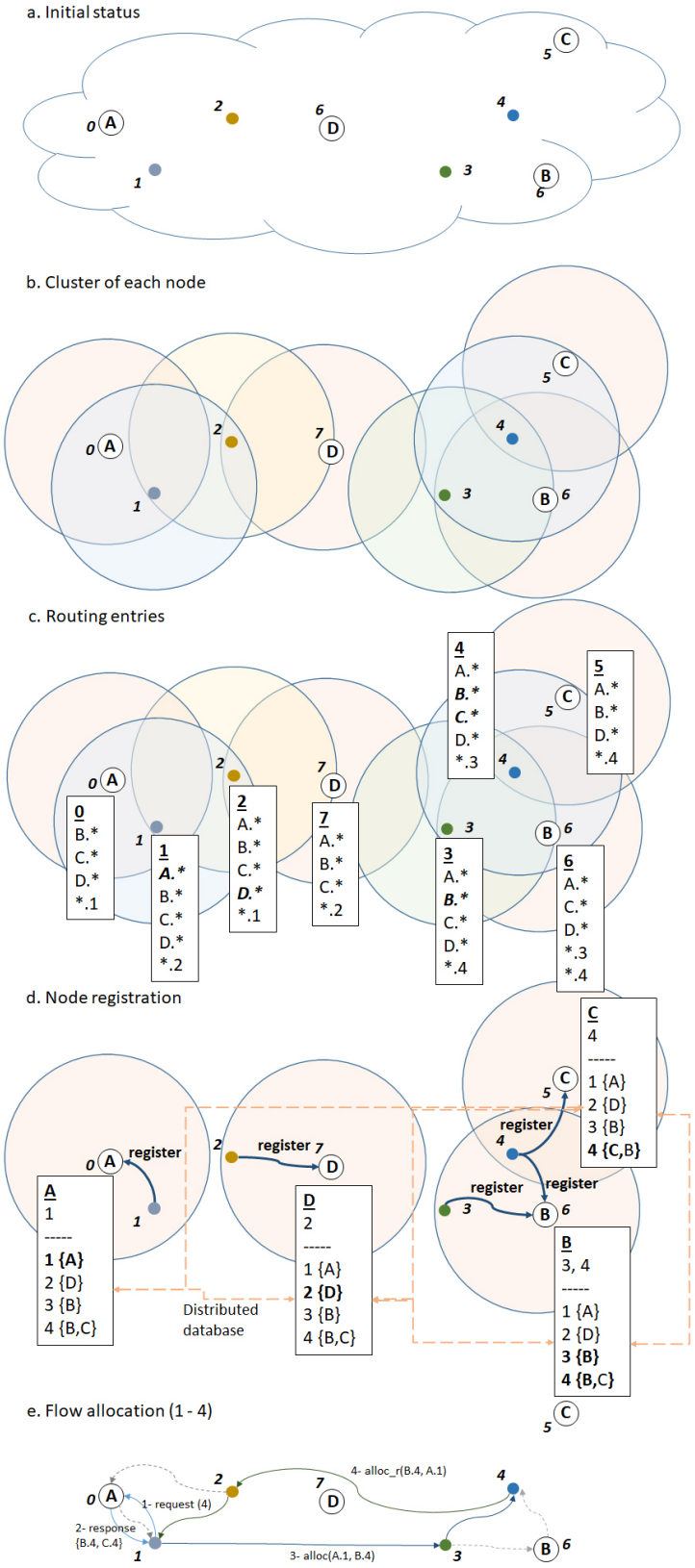

FIGURE 6.18: Example of DIF using a landmark routing scheme.

regard, in contrast with the current production solutions that rely mostly in brute force (scalability at the cost of memory and computing resources), this section reviews the use of a different family of routing solutions, the family of compact routing solutions. Those, instead of putting all the weight of scalability on routing tables, does that providing a trade-off among routing table size, stretch and PDU header size.

While the mentality of RINA is to divide the network in multiple layers of DIFs to gain scalability, in contrast with the current Internet (everything in the same layer), it is important to also see how those solutions could be implemented in RINA. In this regard, this section shows how one of these compact routing solutions, landmark routing, could be implemented in a RINA DIF. Unlike with the TCP/IP Internet model, RINA does not put any impediment to the implementation of those kind of solutions, allowing easily implementing and using new policies adapted to the requirements of each DIF. Instead, the default configuration of RINA provides by itself an environment friendly for these solutions, especially thanks to the separation of node name and addresses, and its flow allocation process.

6.4 QoS and Path selection

Providing a replacement to the current TCP/IP Internet model, the Recursive InterNetwork Architecture (RINA) proposes a new recursive and programmable networking model. With a recursive layering divided in networking scopes, complete layer configurability and full QoS support, RINA not only adapts to the current networking requirements, but also provides an environment capable of evolving with them. As seen before in Chapter 5, with the use of ΔQ policies, RINA is capable of provide enhanced QoS assurances, with a tight control on the degradation that flows suffer when traversing the different devices on the network.

Moreover, DIFs share all the same API, making it easier to ensure QoS guarantees, as the smaller and more manageable environments of lower layers can easily be informed of the requirements of upper layers, in addition to be able to inform about any problem preventing to meet those requirements. Taking advantage of that dynamic knowledge, network managers can improve and easily automatize the configuration of each layer to get its full potential.

As adaptable as RINA's ΔQ policies may be, that only allows network administrators to provide an smart sharing of the degradation the flows suffer at traversing the different networking devices. While that may be enough for small networks, it may not be enough for larger ones,where flows between forwarding devices (either direct links or flows through other DIFs) may produce multiple times as much degradation as scheduling on the forwarding devices themselves. In this context, this section focus on the assurance

of QoS guarantees in long-haul RINA networks, where traffic flows have to traverse large physical distances end-to-end.

6.4.1 Path selection and QoS

As commented above, in large networks degradation suffered between forwarding devices may be multiple times larger than that suffered at those devices themselves. That is a common case in long-haul transport networks where the propagation delay can arise as the main factor in latency degradation. This delay not only can be substantial, but also can largely vary depending on the chosen path, up to the point that the path traversing less nodes (the shortest path), may not correspond to the one ensuring the smallest latency (the fastest path). As this propagation delay cannot be completely avoided, as even the smaller links would impose some latency, the goal is to minimize it, for example considering other metrics more related to it (e.g. link distance). Doing it for all flows, however, would not only increase the length (in hops) of some paths, but also potentially increase also the amount of flows traversing the network links comprised in fastest routes across the network (typically those in the network central region), which would lead to congestion and excessive packet losses. Indeed, there exists a trade-off between delay, losses and bandwidth utilization, which is the rationale behind ΔQ. More than that, given the specific requirements of a QoS Cube, it is easy to know if a given path would assure or potentially fail to support it, considering not only the path latency but also the maximum degradation that any hop in the path would add upon congestion.

An optimal approach to QoS forwarding should consider all candidate flow setups in a connection-oriented way, where all flows with similar requirements between the same pairs of nodes are aggregated over the same path, so that their QoS requirements are ensured even in the worst congestion scenario. This approach is not always viable however, not only due to the high complexity necessary to contemplate all possible candidate paths, but also due to the number of forwarding entries required in connection-oriented scenarios. Considering that, an intermediate approach is proposed, where, for each QoS Cube, each destination is reached by a forwarding tree ensuring that the required QoS levels in its paths from all other nodes are fulfilled. To avoid adding too much degradation to the shortest flows, limits to the additional degradation that flows can suffer are also imposed, limiting that way the degradation with respect to their optimal path. Limiting forwarding to only these valid trees ensures that QoS requirements are met for all flows, while requiring only simple forwarding entries per destination and QoS Cube. Furthermore, as multiple valid trees are available for each destination at the embedding stage, rather than only the best ones, the solution space is enlarged, allowing for a more optimized usage of resources.

To compute a solution for this problem, first is needed to have the set of valid trees for each pair of destination and QoS Cube. As these sets can be large depending on the network, it is possible to use instead only a small fraction of those, selected at random.

For that, the algorithm depicted in 6.6 can be used to compute forwarding trees for each destination that ensure QoS requirements. In addition, considering the expected traffic matrix, those trees also give the required capacity for each link in the network.

```
1  Start a new empty tree with the destination node as root.
2  While all nodes are not in the tree:
3      Compute the distance* from the root node to the remaining nodes not yet in the
       tree.
4      From the remaining nodes, select one of the farthest (n).
5      Compute the valid paths** from the node n to the root.
6      Select one path at random and add it to the tree (all nodes in the path are now in
        the tree).
7
8  * Distance computed with different metrics depending on QoS requirements, commonly a
       joint metric contemplating hop count and latency. The current tree is taken into
       account, not considering paths that generate cycles in it.
9  ** Any path that extends the current tree (without generating cycles), meeting the QoS
        Cube requirements and under the allowed degradation from the optimal path.
```

Pseudo-code 6.6: Description of SA metaheuristic for computing sub-optimal solutions

Then, in order to solve the problem at hand, here it is proposed the use of the Simulated Annealing (SA) meta-heuristic depicted in Fig. 6.7. In this SA, the solution space considered is that of all combinations of tree-sets that contain one valid tree for each destination and QoS Cube. Given a solution, its *N-neighbourhood* is compressed by those solutions that result in replacing up to N trees in it. Then, the weight of a solution is computed as the capacity required in the most loaded link plus α times the total capacity requested, for a small value α. α, k_{max} and max_{neis}, as well as the temperature function $temp(k)$ being all dependent on the size of the network and supported traffic.

```
1  sbest = s = random(SolutionSpace)
2  for k = 1 ... kmax
3      t = random(SolutionSpace);
4      for q = 1 ... maxneis do
5          t = best( t, random(N-neighbour(s) )
6      if t.cost < sbest.cost then : sbest = s = t;
7      else if accept (t, s, k ? kmax) then : s = t;
8  return sbest;
9
10 accept(t, s, k)
11     return (t.cost < s.cost)
12         or exp((s.cost - t.cost) / temp(k)) < uniform(0,1);
```

Pseudo-code 6.7: Pseudo-code for computing random valid trees per destination and QoS Cube

Given that the simulated annealing metaheuristic has a limited number of iteration, each with a small cost, the most expensive part of the solution is that of the computation of forwarding trees. Even so, given that those are dependent only on the network graph

(without considering the usage of flows), it is only required to compute them once, being possible to use that for different traffic matrices. In addition, given that multiple failures are not common in the kind of networks where this solution could be used, it is possible to, not only pre-compute trees in a non-failure scenario, but to have pre-computed trees for the different "1-failure scenarios". Given that, while the planning for all flows traversing a DIF is best done as an off-line solution (as dynamic changes would easily produce congestion bursts in most of the network), the proposed solution also provides a way to dynamically work around failures and big changes in the traffic matrix while maintaining close to optimal solutions.

6.4.2 Numerical results

In order to test the proposed solution, we used a 17-Node backbone network (depicted in Fig. 6.19) based on a German backbone network (DTAG). We considered a uniform traffic matrix where the offered traffic between each pair of nodes is divided according to their QoS requirements. To this end, only their latency requirements are considered for simplicity. The resulting traffic profile is as follows: 10% of QoS Cube A (minimum latency); 30% of QoS Cube B (low latency); and 60% of QoS Cube C (no strict latency requirement). Instead of fixing the amount of traffic and minimizing overbooking, we have considered the opposite case, scaling rates to the capacity of the most loaded link.

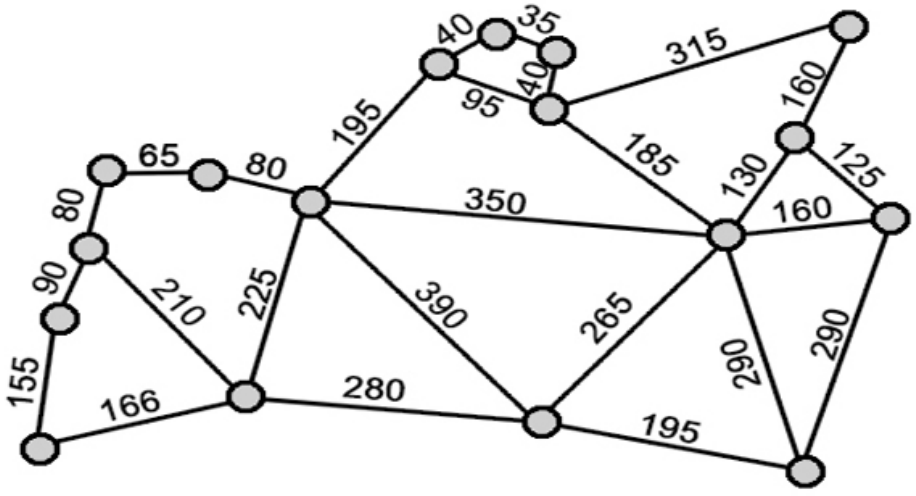

FIGURE 6.19: 17-Node backbone network based on German backbone network (DTAG)

First, we aim to analyse how simple QoS differentiation behave against giving the same treatment to all QoS classes (either using the shortest or fastest paths). We considered three metrics, one per QoS Cube. For QoS Cube A we consider distance (in Km) as metric. For QoS Cube B we consider hops + distance/20 as metric. Finally, for QoS Cube C consider the number of traversed hops as a metric. With these metrics, distance-vector routing is used to select the best paths for each QoS Cube over the network. As comparison, we considered two solutions using only the shortest (hops) and fastest

(distance) paths, respectively. Fig. 6.20 shows the required capacity on each link, as well as the worst case and average for each solution, relative to the usage in the most loaded link in the shortest path solution. The first notable result is that, while fastest paths improve the delivered QoS, this is achieved at expenses of requiring a higher capacity on some links, especially those covering smaller distances (note that the peak is not even on one of the most loaded links in other solutions). On the other hand, the separation per QoS, not only provides better services to the more requiring flows, but also reduces slightly the load on the most used link. This has an interesting meaning as, while specific to this network and the traffic matrix, results imply that considering QoS does not have to worsen, but could even improve the usage of the network.

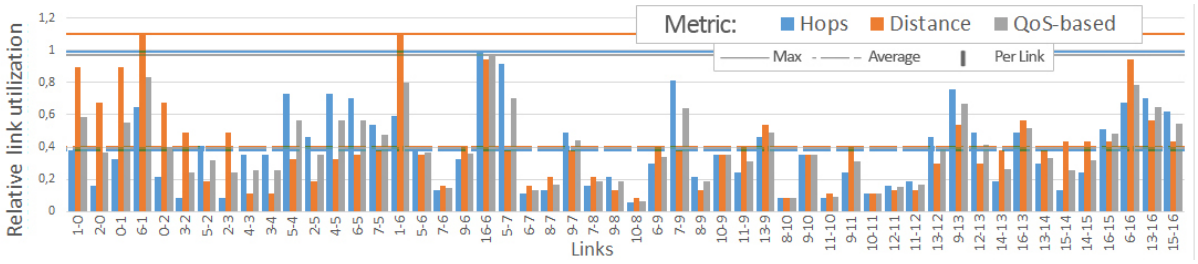

FIGURE 6.20: Link usage comparison between metrics: hops, distance and the described per-QoS metric.

After establishing the benefits of using different metrics per QoS, we aim to quantify how the freedom to decide whether to allow path degradation (within QoS requirements) can improve the network usage, identifying at the same time the drawbacks of the proposed approach with respect to a traditional connection-oriented one. We compare the results from the previous QoS-based metric with those of connection-oriented and the proposed approach, both with different accepted path degradation for QoS Cube C. In this regard, Fig. 6.21 shows significant improvements, greatly reducing the required capacity on the most loaded links when accepting either 1 or 2 extra hops. Given these results, for example, if we could provide 100Mbps between peers using the first solution, this would mean that one extra hop for QoS Cube C could improve the load to 120Mbps per flow, 150Mbps if we allowed 2 extra hops instead. At the same time, this is done without dramatically affecting the average load, as seen in Fig. 6.22, meaning that only few flows need to be degraded to reach such network performance improvements. However, allowing higher degradations, while still provides some improvement, results in a huge increment of the average utilization, as a high number of flows needs to be degraded to reach that improvement. In addition, comparing connection-oriented and tree-based solutions, we see that both have the same worst cases, only differing in a slightly higher average load with the use of trees.

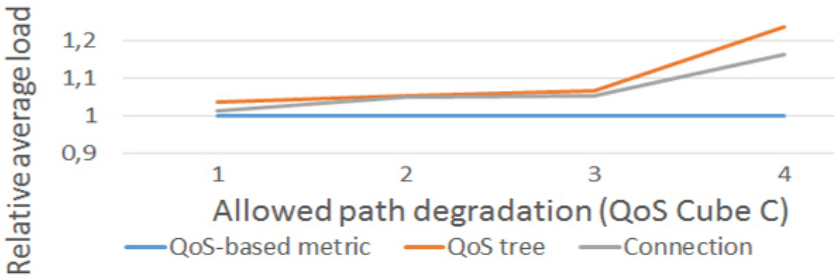

FIGURE 6.21: Comparison of worst link utilization between QoS-based metric, QoS trees and connection-oriented approaches

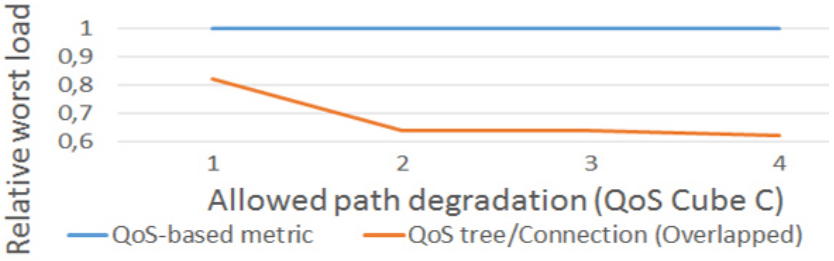

FIGURE 6.22: Comparison of average link utilization between QoS-based metric, QoS trees and connection oriented approaches

6.4.3 Conclusions

When considering QoS differentiation in an Internet-wide network, it is important to provide quality guarantees in each networking level. When using ΔQ within a RINA scenario, QoS guarantees can already be given based on the internal scheduling that the different devices perform at a PDU level, but that is not always enough to ensure QoS guarantees on large scale. When covering large distances, this has a great importance given the large path latencies that does paths entail.

Looking to a possible future, where the current TCP/IP Internet is finally replaced with a smarter network model with a full support for QoS (e.g. RINA), here the use of metrics per-QoS in long-haul RINA networks is discussed, further improving the assurance of QoS guarantees that RINA and ΔQ can provide to the network. In addition, a heuristic for path selection has been proposed, resulting in paths that minimize the load of the most constricted nodes in the network, while providing paths that ensure QoS guarantees under normal utilization. Even so, this is only a first approach to provide QoS assurances within all networking levels, and extra considerations would need to be taken if production environment where considered, including the effects of changes in traffic patterns (e.g. day-time vs. night-time) or mechanisms for quick path recovery under failures.

6.5 Chapter summary

In addition to the use of common routing solutions (e.g. link-state or distance-vector routing), the complete architecture of RINA, its full separation of names and addresses, and it complete programmability by the means of policies, provide an environment that friendly accepts most of the solutions that are not viable with the current TCP/IP Internet model. Considering that, this chapter studied some of the different possibilities that RINA opens to the field of routing and scalability.

First, the new Rules and Exceptions (R&E) forwarding policy has been presented, a replacement for the common forwarding tables policies that, in addition to the common behaviour of forwarding tables, allows to use programmable forwarding rules dependent on the network graph, opening the path to any routing solution not completely dependent on forwarding entries.

With the R&E forwarding policy as a base, a complete topological solution for data centre routing has been proposed, considering different data centre network topologies. In addition to the required forwarding rules, two kinds of routing policy (a distributed and a centralized one) have been proposed for these solutions, taking into account common knowledge on the topology of the network, instead of using a brute force approach. Finally, the requirements of those policies have been compared with that of current solutions, exhibiting the proposed solution a great improvement in its scalability.

For scenarios where taking profit from the network topology cannot be done to that extent, other kind of routing policies have been also considered. In this case, this chapter briefly analyses the use of compact routing solutions in RINA scenarios. Given that compact routing solutions rely in extra information on the location of nodes or how to reach them, they do not stick well with the current TCP/IP model and its more straightforward approach (e.g. to transmit between A and B you do not need to allocate a flow). In contrast, given its separation between node name and addresses, and its process of flow allocation, RINA provides an environment where compact routing solutions can be easily deployed and the extra information required can be automatically shared when required. This is seen with an small example of how one of these compact routing solutions could be implemented in RINA. Specially,a landmark-based routing solution could be implemented in RINA with only a few changes in a distance-vector routing policy, while profiting from most management functions already in RINA.

Finally, the other side of network scalability is also considered: the usage of resources and how well a routing solution can provide the required QoS requirements. In that regard, different connectionless and connection-oriented approaches for path selection have been analysed. In addition, this work proposed a mixed connection-oriented/connectionless approach that could result in more QoS-friendly paths while reducing the network usage. The obtained results showed that, as expected, connection-oriented based approaches

provided a greater performance than connectionless ones, but at the cost of most requiring hardware. In contrast, the proposed solution resulted in performance close to that of the connection-oriented solutions, but having its cost greatly reduced thanks to its connectionless side.

Overall, this chapter has shown multiple possibilities of what RINA can provide to different kinds of networks. Even so, those are not the only possibilities of this architecture, but only a small sample of what can be done.

Chapter 7

Conclusions and future work

The current Internet, based on the TCP/IP Internet protocol suite lacks the means to provide a scalable networking environment with support for QoS. Based on a fixed stack, TCP/IP has been patched once and again in order to try to surpass the limitations that the model itself carriers. Even so, the current Internet is one with huge scalability problems (e.g. massive routing tables in core routers, depletion of IPv4 address pool, requirement to end-users to use NAT, etc.) and fails to accommodate the varying QoS requirements of current distributed applications (e.g. there is no support for stating QoS requirements).

Trying to solve those problems present in the current Internet, the Recursive InterNetwork Architecture (RINA) presents itself as a new clean-slate solution for networking based on the root of networking. RINA proposes a different networking stack, following the different networking domains of the network rather than a functionality-based like the one proposed by TCP/IP. Being defined by the networking domains, the layers of RINA, or Distributed IPC Facilities (DIFs), are not bound to specific functionality, but can be configured to perform any networking function required within its domain, with its behaviour decided by the mean of policies. This way, RINA advances towards the original concept of Inter Process Communication (IPC), defining any data transfer process between applications and devices, centred in only a couple of primitive operations. Not only that, but RINA also provides a complete support for QoS in the network, not only ensuring QoS guarantees within the DIFs, but providing an extensive API that allows for applications themselves to state the requirements of their flows.

Not being constrained by the strictness of TCP/IP, RINA-based solutions can take fully profit from network properties and allow for a more specific configuration. The work in this thesis delved into the use of specific solutions for RINA networks, centred in the provision of reliable QoS assurances and the scalability of the network.

Chapter 5 introduced the use of the ΔQ framework for QoS assurance into RINA networks. Given that RINA offers a complete support for applications and IPCPs to state

their communication requirements, simple QoS differentiation could not be always enough to correctly provide those needs. In this thesis, the different key points to consider when providing QoS assurances (the traffic, the network and the policies) have been pointed, and the resulting ΔQ scheduling policies have been tested in different backbone scenarios. At that point, ΔQ scheduling policies have shown to provide more accurate QoS assurances than current QoS differentiation solutions. In addition, given that QoS assurances require some level of control over the traffic moving within the network, and greedy users may try to allocate higher priority flows when not needed, it is also proposed a new QoS rate limiting scheduling policy. While the proposed policy correctly limits the amount of priority traffic that end-users can inject into the network, it does that in a user-friendly way, allowing them to better use their contracted resources.

Chapter 6 explored the benefits of topological routing solutions for large-scale RINA networks. In this thesis, a new forwarding policy that merges traditional forwarding table decisions (exceptions) with programmable rules is proposed, reducing considerably the size of forwarding tables and the computational cost of forwarding decisions in large scales networks with well-defined topological properties. Taking profit from this policy, topological solutions for large-scale data centres are proposed, showing that, in those cases, topological routing can reduce the amount of forwarding information almost only to that of neighbouring nodes. In addition, it is also stated the possibility that RINA offers to use other types of non-standard routing solutions, like those of compact routing family. In order to do that, in this thesis it is analysed how a Landmark-routing solution could be configured within a RINA DIF, showing that the only required modification would be that of a slightly modified distance-vector routing policy and management agent. Finally, the requirements of routing for QoS constrained networks are also considered, showing that centralized connectionless routing solutions can provide almost identical results as connection-oriented configurations in general case scenarios, without the forwarding constrain of connection-oriented solution.

Further work in the research line of this thesis may follow the multiple lines of work. With respect to QoS assurance with ΔQ scheduling policies, large scale tests with real traffic would be required to assure the correct behaviour of the policies, something that was out of the reach for this research. In addition, the recursive stack of RINA open interesting possibilities for QoS assurances in unreliable networks (e.g. wireless links). Unreliable links, in special those of Wi-Fi routers such as those commonly used at end-users homes, are a great source of losses in the network, resulting in both high degradation and increased retransmission of data. In RINA, that could be easily avoided, as fast retransmission could be performed at the link level (even for low-latency QoSs). Even so, wireless technologies are not as simple as wired ones, being not only unreliable, but the link rate varying depending on the connection state (e.g. lower rate to improve reliability if there are interferences). With that in mind, it is important to consider also how RINA could accommodate these peculiarities of the wireless environment to work with policies like congestion notification, in order to correctly shape the traffic traversing the conflicting links before problems (e.g. losses) occur.

In terms of forwarding and routing solutions, multiple work lines are also open. For instance, with respect to the proposed Rules&Exceptions scheduling policy, its implementation, as well as the testing in production scenarios has been left outside of this research due to multiple constrains. Even so, its implementation could not only provide a base for different topological solutions in RINA networks, but also provide a default "forwarding table" policy to replace the current one, including the different benefits that groups and inverse encoding. In addition to its use for data centre networks, it would also be interesting to see how this policy could be used in different existing scenarios, or how non-regular networks could profit from a migration to more well-defined topologies. On the other hand, while the use of landmark-routing in RINA networks has been considered, it would be also interesting to consider other possible routing solutions, never considered in production environment, but now possible thanks to RINA.

Appendix A

RINAsim

As part of the FP7 PRISTINE[104] projects, an OMNeT++ simulator for RINA, the RINASim[113], has been developed. The RINASim has become an stand-alone framework based on the OMNeT++ discrete event simulator environment that aims to provide the community with a reliable and up-to-date tools for simulating RINA-based networks. From 2016, RINASim has been part of the official frameworks for OMNet++ [114], and, in its current state, RINASim provides an entirely working implementation of the RINA environment, containing all mechanisms included in RINA specifications.

While leaded by the Brno University of Technology in the Check Republic, we at UPC have been one of the primary contributors, not only providing new policies and use cases, but participating actively in the early development and the support to new researchers interested in experimenting with RINA. This appendix list some of such contributions.

- Separation of Forwarding-Routing into Forwarding, Forwarding Generator and Routing policies.

  We designed the first version of the current implementation of Routing to Forwarding policies, introducing the separation between routing and forwarding generator (population of forwarding) policies, as well as introducing the workflow between both policies. This allowed to easily swap partially policies depending on the scenario (e.g. swap between distance-vector and link-state routing while maintaining a generator designed for a specific scenario). In addition, we worked towards a true forwarding policy, replacing the original forwarding table implementation with a policy based one.

- Forwarding and Routing policies.

  We have contributed in the implementation of most forwarding, forwarding generator and routing policies currently present in the RINASim. Those policies include from simple static routing configurations with traditional table based forwarding policies to QoS-based multipath solutions and topological solutions.

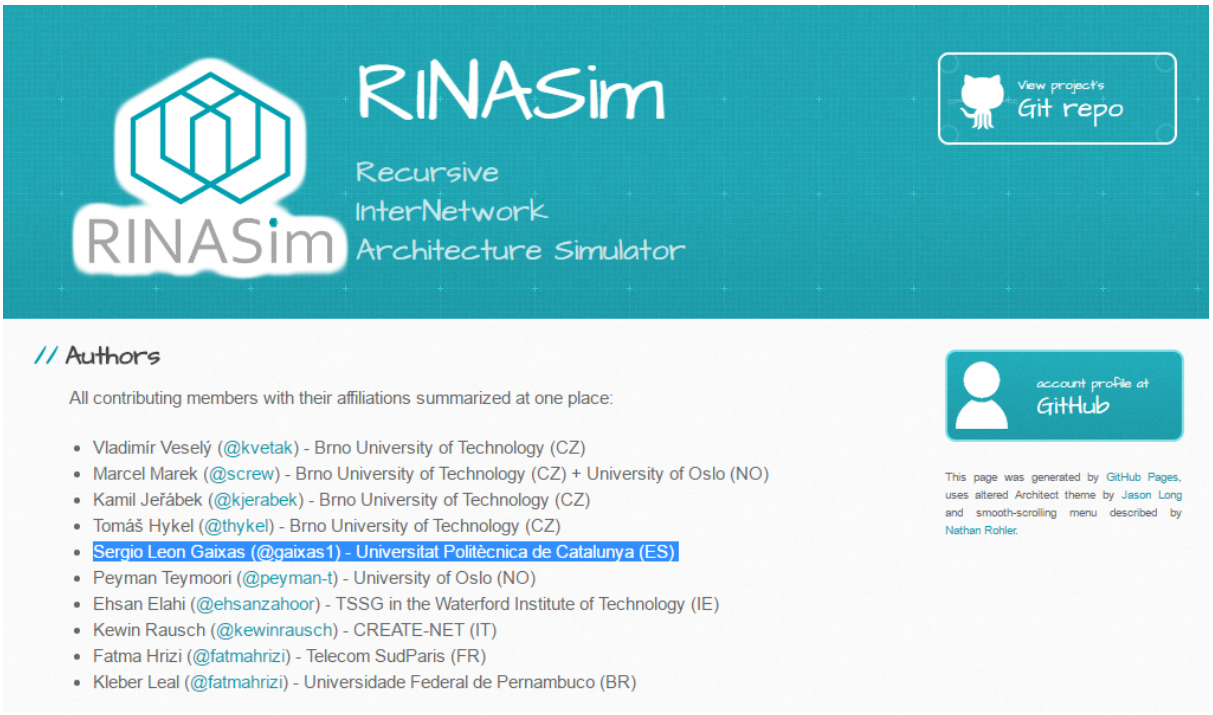

FIGURE A.1: RINASim main page.

- Scheduling policies.

  We have contributed in the implementation of a good number of scheduling policies for the RMT, being the most important the Weighted Fair Queue and ΔQ based policies, and worked towards the unification of scheduling policies into a unique policy. We also implemented the "ModularMonitor", a special policy where different sub-policies can be easily configured for specific actions (e.g. decide if drop incoming PDUs or the output order).

- Data injection and monitoring.

  As a parallel work to the default "application" based flows, we worked towards a system based on data injection. With this, we were capable of easily inject high loads on data into the network, without requiring to simulate all the RINA stack and the large number of applications and flows required in order to do that. This allows to emulate highly loaded networks, something not easily doable otherwise. In addition, we implemented multiple monitoring approaches to these "flows", being possible to, not only monitor them as if those were real flows, but also distinguish them from the rest of flows if required.

- Tutorials.

  In addition to the contributions in form of policies, we also contributed with two tutorials for ΔQ scheduling policies and topological data centre routing.

A.1 Tutorial - Assuring Absolute QoS Guarantees for Heterogeneous Services in RINA Networks with ΔQ

This tutorial, available at [115], shows how to configure and use basic ΔQ scheduling policies to provide differentiable treatment to services given their QoS. In addition, a first approach to the interaction between ΔQ policies and congestion control is shown, allowing for a reasonable overbooking of resources while maintaining QoS requirements (with the dynamic data-rate reduction of flows given based on their requirements).

The scenario used in this tutorial can be found under at `https://github.com/kvetak/RINA/tree/master/examples/Tutorials/DeltaQ_Scheduling`, and is composed by the following files:

- net.ned: Network description.
- omnet.ini: Network and overall configuration.
- QTA.xml: Configuration of the QTAMux used for ΔQ policies.
- shimqoscube.xml: QoSCubes definition for shim-DIFs.
- {cong/free}_qoscube.xml: QoSCubes definition for upper DIFs in congestion controlled and free scenarios.
- connection{shim/set3/set9}.xml: Definition of preconfigured flows.
- qosreq.xml: QoS requirements of preconfigured flows.
- data{0/1x3/1x9/10x3}.xml: Data flows definition for the different configurations.
- directory.xml: Configuration of IPCP locations

The network described in "net.ned" is a 6 nodes network describing containing the sub-set of data centre nodes as seen in Fig. A.2. In this network, the main flows to consider are those departing from node A towards nodes B and C, being those full ToR-2-ToR flows. In addition, emulating the bandwidth usage of other flows that could collide with those, multiple flows between nodes in that path are allocated.

While in all scenarios, the ΔQ policies inform of congestion in the form of ECN marking, QoSs in the Free* scenarios are configured to ignore them, resulting in high losses for low cherished flows under periods of high load. For the Cong* scenarios, instead, QoSs are configured to reduce the data-rate of the aggregated flows according to the arrival of ECN marked PDUs, resulting in low losses, even in periods of high load of flows with high priority. In terms of flows and QoS, in this scenario we considered a 3x3 Cherish/Urgency

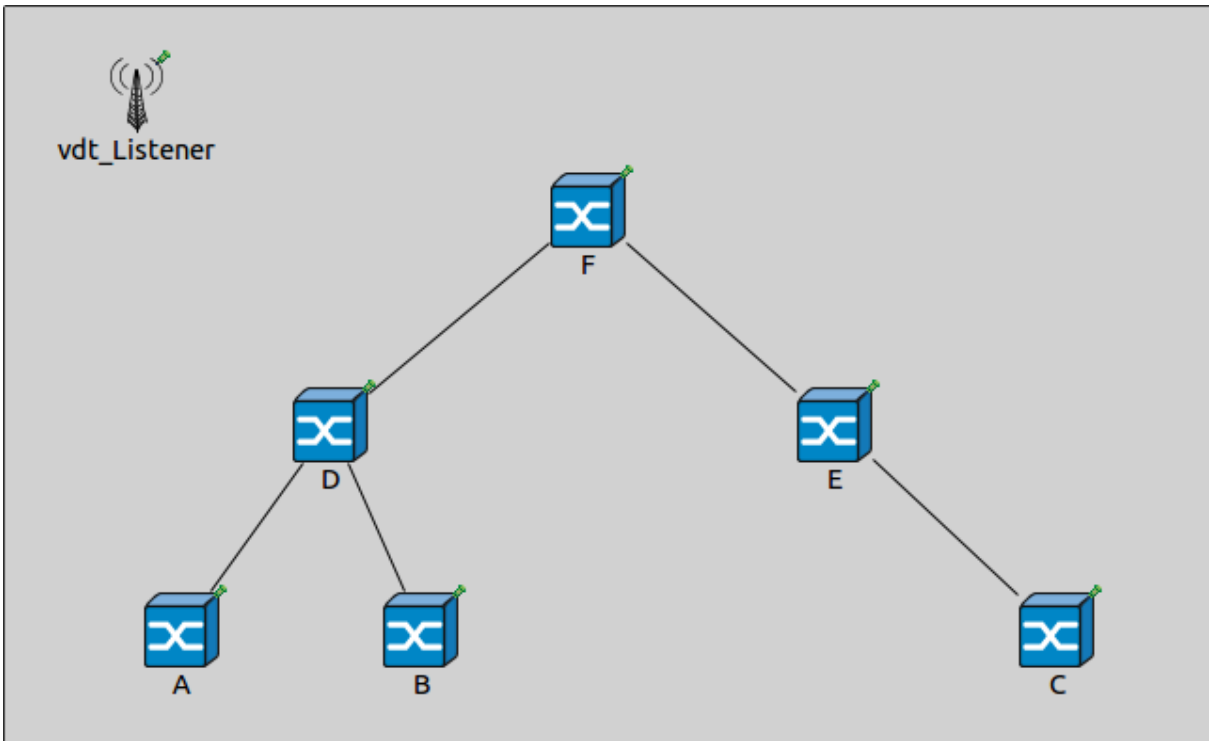

FIGURE A.2: Scheduling tutorial network.

matrix. For each position, we define the QoS identifier as A*, B* and C* from more urgent to less, and *1, *2 and *3 from more cherished to less. Given the urgency of flows, we considered 3 different of applications:

- QoSs A*: Real-time voice traffic. ON/OFF traffic with small PDUs and without retransmission.
- QoSs B*: Video on demand and web browsing. ON/OFF traffic with MTU sized PDUs and retransmission of losses.
- QoSs C*: Filetransfer. Constant traffic with MTU sized PDUs and retransmission of losses.

With those QoSs, we considered 3 configurations for each, considering flows of 1 or 10 Mbps and using the full Cherish Urgency matrix or only the triad A2, B1 and C3. In total, for this tutorial we consider 6 different configurations for the scenario:

- Without congestion control:
- Free1Mbps3QoS
- Free1Mbps9QoS
- Free10Mbps3QoS

- With congestion control:
- Cong1Mbps3QoS
- Cong1Mbps9QoS
- Cong10Mbps3QoS

The rest of the tutorial can be found at [115].

A.2 Tutorial - Topological Routing in DC

This tutorial, available at [116], shows how to configure and use topological routing and forwarding policies for data centres. In special, we show how to work with the Groups Rules and Exceptions forwarding policies (a first stage implementation of the Rules and Exceptions policy) in both distributed and centralized routing.

The scenario used in this tutorial can be found under at `https://github.com/kvetak/RINA/tree/master/examples/Tutorials/Topological_DC_Routing`, and is composed by the following files:

- net.ned: Network description.
- omnet.ini: Network and overall configuration.
- shimqoscube.xml: QoSCubes definition for shim-DIFs.
- qoscube.xml: QoSCubes definition for upper DIFs.
- shimconnectionset[Central].xml: Definition of preconfigured flows.
- qosreq.xml: QoS requirements of preconfigured flows.
- directory.xml: Configuration of IPCP locations

The network described in "net.ned" is a modified clos DC network with 8 pods with 6 ToRs each and 4 spine-sets of 3 spines each as seen in Fig. A.3. In addition, for the centralized routing configuration, manager servers have been connected to the two first ToRs of each pod.

The main objective of this scenario is to show how routing and forwarding policies behave after some flow failures. In order to do that, multiple flows are "killed" after the network is stable and, after it has been stabilized again, all ToRs try to communicate between them.

In this scenario we consider 4 different configurations:

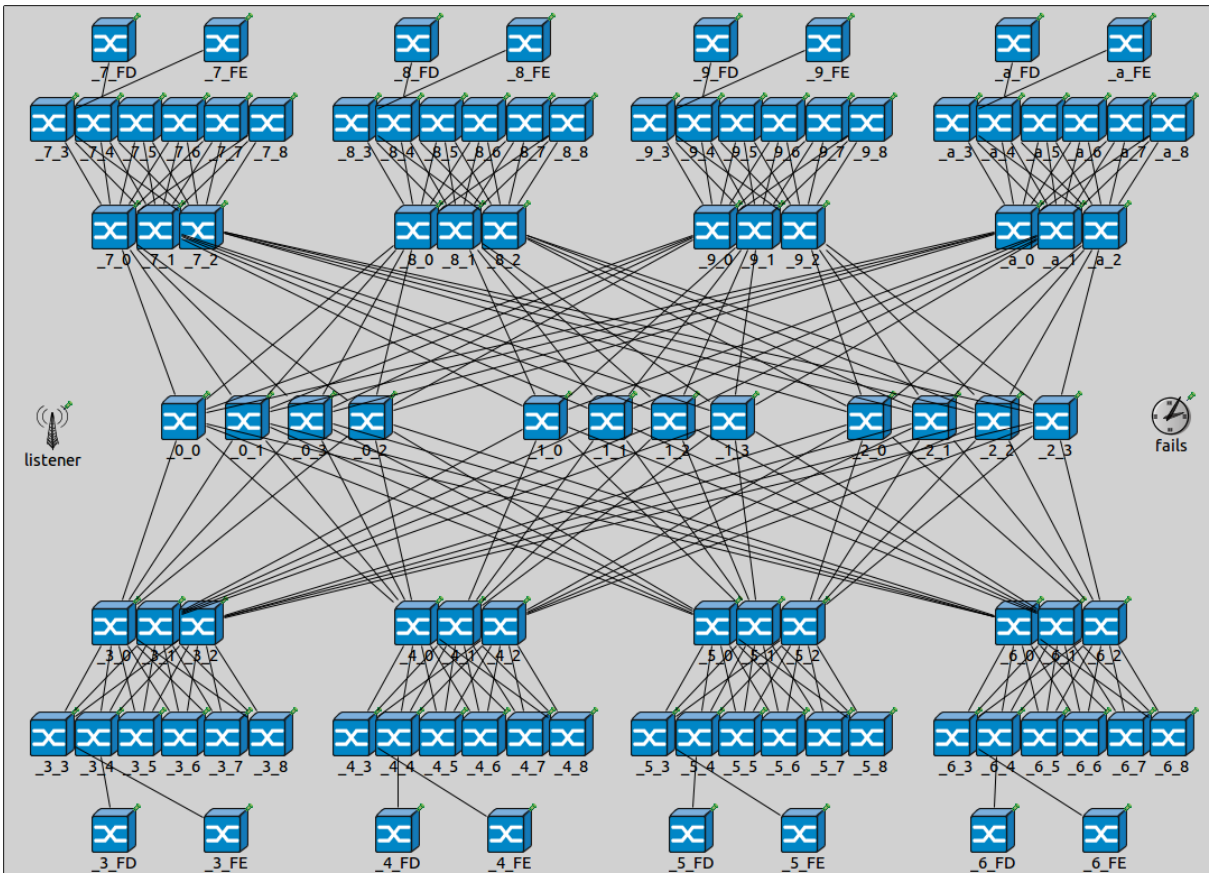

FIGURE A.3: Routing tutorial network.

- Default configuration

  Errorless configuration with a default link state routing and forwarding tables.

- GRE Static configuration "routingxceptions"

  Errorless configuration with GRE policies without routing

- GRE Distributed configuration "routingxceptionsErrors"

  Scenario with multiple random errors, GRE policies and distributed erro-based routing.

- GRE Centraliced configuration "centralized"

  Scenario with multiple random errors, GRE policies and centralized erro-based routing.

The rest of the tutorial can be found at [116].

Appendix B

RINA SDK

As part of the FP7 IRATI[117] project, a RINA Software Development Kit (SDK), IRATI, has been developed for OS/Linux systems [118]. Currently, the IRATI implementation is mature enough to allow for small and medium scale experimentation on RINA networks, although still not viable for production environments. Even so, this implementation have been constantly in development, through projects like the FP7 PRISTINE[104] or the H2020-ARCFIRE[105]. During this thesis, we have made an extensive usage of such SDK and made some contributions in the form of policies and testing applications, in addition to helping in the test of the SDK, from which some hidden bugs have been found.

The IRATI implementation provides a similar design as used in common network applications, with the use of a RINA API which closely resembles the socket API, while taking into account the differences in the naming and addressing scheme and the QoS support that RINA provides in contrast to other network architectures. The socket API is currently defined by the POSIX.1-2008 standard, something that provides multiple advantages like:

- The POSIX standards make it easier for developer to understand the RINA API and port existing applications to RINA.
- The use of file descriptors as universal handlers to interact with the RINA API makes it possible to use standard system calls (e.g. read, write, close, etc. and stream synchronisation mechanisms (e.g. select, poll, etc.).

Using this IRATI implementation, a common client-server application would have the simple work-flow depicted in Fig. B.1. While the use of POSIX sockets provides an standard way to use RINA flows, their allocation is not as simple as in the case of TCP/UDP flows (although that may change in future implementations). Currently, applications using the SDK require to listen to multiple events generated by the RINA API in order to correctly manage the register/unregister of the application in specific DIFs and the

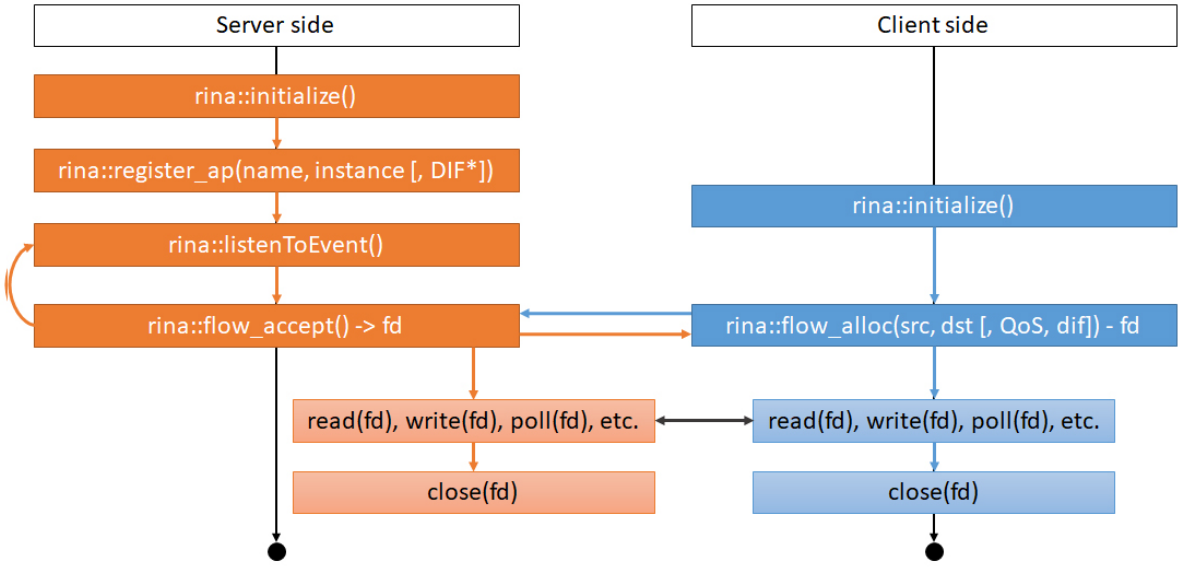

FIGURE B.1: Abstraction of the work-flow of a Client-Server application in RINA.

allocation and deallocation of flows, etc. Given this, the process required to allocate the simplest of the flows may complicate even the simpler of the applications. In order to minimize the effect of this we have designed small base classes for RINA applications, providing a minimal interface to the RINA API for applications using a client/server approach (available at[119]). Using this base, creating a server application in RINA becomes as easy as to create a new class object defining the application name and instance as well as the list of DIFs where it has to be registered. Then, the virtual function "handle_flow" will be called in a new thread with both port id and filedescriptor as a parameter whenever a new flow towards the server is allocated. On the other side, clients only require to configure the destination and their name (and optionally also instance identifiers) and a new flow towards it will be allocated, also handled by a "handle_flow" function. In addition, client applications can also specify the DIFs where RINA should search for the destination application and the QoS requirements for the flow. Using these classes, developers only require to focus on the implementation of the function handling the flow, using the same POSIX APPI commonly used with TCP/UDP. In addition to the base classes, we have also developed multiple testing applications with the purpose of generate traffic following different traffic patterns (e.g. voice, video, ping, file-transfer, etc.), as well as mechanisms to extract useful data from those flows.

In addition to the work in the SDK within the PRISTINE project, we have also implemented small scheduling policies for testing purposes (available at[120]). These policies include:

- rmt_be.

  A variant of the default best-effort implementation available in the RINA SDK. While having a similar behaviour, unlike the default implementation, this policy

does not free the allocated buffer after the PDU it contains is served, but, instead stores them for be re-used for following PDUs. Although not providing a sustainable improvement in performance, this policy greatly reduces the effects on memory fragmentation caused by the rapid allocation and release of memory in the scheduling policy.

- rmt_eqta.

  A variant of the QTAMux implementation available in the RINA SDK. This implementation extends the default implementation presented in 5, adding the possibility to link sequentially multiple PSs (in a similar way as stated in the original patent for ΔQ scheduling [121, 122, 123, 124, 125, 126]) or to completely avoid them for specific QoS Cubes. In addition, unlike the default implementation available in the RINA SDK, this implementation improves the performance of the scheduling decision with the use of direct access data-structure for the C/U Mux and the different P/Ss. Even so, this comes at the cost of removing the possibility of re-configuring the size of the C/U matrix and number of P/Ss after the initial configuration.

- rmt_rlim.

  An implementation of the rate-limiting policy defined in 5.5. In order to improve performance, this implementation sacrifices the possibility of reconfigure the size of the C/U matrix to have instead direct access structures to them. In contrast with the rmt_eqta policy, where this change has only a small impact, in this policy this allows to avoid multiple look up calls within the loops used for adding and remove credits, being able instead to maintain all the required information within a small portion of continuous data with a direct access structure.

Appendix C

Rate limiting policy pseudo-code

```
1  /*Data structures*/
2  struct Queue {
3      int cherish; //Cherish level of the queue
4      int urgency; //Urgency level of the queue
5      list<PDU> queue;   //PDUs stored in the queue
6      int congTh;  //Threshold to inform of congestion
7      int dropTh;  //Threshold to drop incoming PDUs
8  }
9  struct QoS2Q{
10     int QoSId;   //QoS id
11     Queue * q;   //Pointer to its destination queue
12 }
13 struct Dimension {
14     int  N;         //Number of levels
15     int* credit;    //Available credits for each level
16     int* max;       //Limit on credits per level
17     int* gain;      //Credits per ns gain per level
18 }
19
20 /*Scheduling data*/
21 list<Queue> queues;//Information of queues
22 list<QoS2Q> qos2Q;//Mapping from QoS to queues
23 Dimension C;   //Information of cherish
24 Dimension U;   //Information of urgency
25 int  K;        //Cost in credits/byte
26 int  hSize;    //Size of extra headers
27 int remaining; //Remaining PDUs to serve
28 time t0;        //Time of previous call in ns
29
30 function arrival(PDU pdu) {
31     //Search the queue for the pdu's QoS id
32     Queue * q = search(qos2Q, QoSId == pdu.QoSId);
33     if(q->size() > q->dropTh) {
34         signalizeCongestion(pdu);
35         if(q->queue.size() > q->dropTh) {
```

```
36              return false;
37          }
38      }
39      q->queue.push(pdu);
40      remaining +=1;
41      return true;
42  }
43
44  function schedule() {
45      //Get the time difference from last call
46      time dt = current_ns() - t0;
47      if(dt > 0) {
48          addCredits(C, dt);
49          addCredits(U, dt);
50          t0 += dt;
51      }
52      int bestC = getBest(C);
53      int bestU = getBest(U);
54
55      //Select the next queue with
56      //  cherish >= bestC and urgency >= bestU
57      Queue * q = getNext(queues, bestC, bestU);
58      if( q == NULL){
59          return NULL;
60      }
61
62      PDU * pdu = q.queue.pop();
63      int size = pdu->size() + hSize;
64      int credits = size * K;
65
66      useCredits(C, credits);
67      useCredits(U, credits);
68
69      remaining -= 1;
70      return pdu;
71  }
72
73  /* Adds credits to each dimension's level after a Dt*/
74  function addCredits(Dimension &d, time dt) {
75      int j = d.N-1;
76      for(int i = d.N-1; i >= 0; i--) {
77          int rem = d.gain[i] * dt;
78          for(; j>i; j--) {
79              if(d.credit[j] < 0){
80                  if(rem > d.credit[j]) {
81                      rem += d.credit[j];
82                      d.credit[j] = 0;
83                  } else {
84                      d.credit[j] += rem;
85                      rem = 0;
86                      break;
```

```
87                  }
88              }
89              j++;
90          }
91          if(rem > 0){
92              d.credit[i] = min(d.max[i], d.credit[i] + rem);
93          }
94      }
95  }
96
97  /*Returns the first level with credits*/
98  function getBest(Dimension d, time dt) {
99      for(int i = 0; i < d.N; i++) {
100         if(d.credit[i] > 0){
101             return i;
102         }
103     }
104     return -1;
105 }
106
107 /*Spends an amount of credits c used for level l*/
108 function useCredits(Dimension &d, int l, int c) {
109     for(int i = l; i >= 0; i--) {
110         if(c >= d.credit[i]) {
111                 d.credit[i] -= c;
112                 c = 0;
113                 break;
114         }
115         c -= d.credit[i];
116         d.credit[i] = 0;
117     }
118     if( c < 0){
119         d.credit[l] -= c;
120     }
121 }
```

Pseudo-code C.1: ΔQ rate-limiting scheduling policy

Appendix D

Published work

D.1 Publications in Journals

1. **S. Leon**, J. Perelló, D. Careglio, E. Grasa, D. López and P. Aranda. Scalable Topological Forwarding and Routing Policies in RINA-enabled Programmable Data Centres. Published in *Transactions on Emerging Telecommunications Technologies* 2017. DOI: https://doi.org/10.1002/ett.3256 (Available online). SCI index (2016): 1.535.

2. J. Gabarró, **S. Leon** and M. Serna. The computational complexity of QoS measures for orchestrations - The computational complexity of QoS measures. Published in *Journal of Combinatorial Optimization* 2017. DOI: https://doi.org/10.1007/s10878-017-0146-9 (Available online). SCI index (2016): 1.235.

D.2 Publications in Conferences

1. **S. Leon**, J. Perelló, D. Careglio and M. Tarzan. Guaranteeing QoS requirements in long-haul RINA networks. In *Proceedings of the 19th International Conference on Transparent Optical Networks (ICTON 2017)*, Girona (Spain), May 2017.

2. **S. Leon**, J. Perelló, D. Careglio, E. Grasa, M. Tarzan, N. Davies and P. Thompson. Assuring QoS Guarantees for Heterogeneous Services in RINA Networks with DeltaQ. In *Workshop NetCloud 2016*, Luxembourg, December 2016.

3. **S. Leon**, J. Perelló, E. Grasa, D. Lopez, P. Aranda and D. Careglio. Benefits of programmable topological routing policies in RINA-enabled large-scale datacenters. In *IEEE GLOBECOM 2016*, Washington (USA), December 2016.

4. **S. Leon**, J. Perelló, D. Careglio, J. Solé-Pareta and S. Spadaro. Introducing the Benefits of RINA over SDN-enabled Multi-layer Optical Networks. In *Workshop Red Temática Elastic Networks*, Cartagena (Spain), May 2016.

5. **S. Leon**, J. Perelló, D. Careglio and S. Spadaro. On the benefits of RINA over programmable optical networks for dynamic and smart resource management. In *Proceedings of the 17th International Conference on Transparent Optical Networks (ICTON 2015)*, Budapest, July 2015.

D.3 Publications under review

1. **S. Leon**, J. Perelló, D. Careglio and M. Tarzan. End-user Traffic Policing for QoS Assurance in Polyservice RINA Networks. Submitted for publication in *Springer Telecommunication Systems* Journal. SCI index (2016): 1.542. In the 2nd review round, after receiving "Minor revisions required".

D.4 Tutorials

1. **S. Leon**. Assuring Absolute QoS Guarantees for Heterogeneous Services in RINA Networks with *Delta*Q; 2016; Available at https://github.com/kvetak/RINA/wiki/Assuring-Absolute-QoS-Guarantees-for-Heterogeneous-Services-in-RINA-Networks-with-%CE%94Q.

2. **S. Leon**. Topological Routing in DC; 2016; Available at `https://github.com/kvetak/RINA/wiki/Topological-Routing-in-DC`.

Bibliography

[1] Internet Growth Statistics 1995 to 2017 - the Global Village Online; Available at http://www.internetworldstats.com/emarketing.htm; Last Accessed April 25, 2018.

[2] Internet of Things (IoT) connected devices installed base worldwide from 2015 to 2025 (in billions); Available at https://www.statista.com/statistics/471264/iot-number-of-connected-devices-worldwide/; Last Accessed April 25, 2018.

[3] IPv4 Address Depletion - Cisco; Available at https://www.cisco.com/c/en/us/about/press/internet-protocol-journal/back-issues/table-contents-37/103-addr-dep.html; Last Accessed April 25, 2018.

[4] BGP: Yet another Internet time bomb - TechRepublic; Available at http://www.techrepublic.com/blog/data-center/bgp-yet-another-internet-time-bomb/; Last Accessed April 25, 2018.

[5] What caused today's Internet hiccup — BGPmon; Available at https://bgpmon.net/what-caused-todays-internet-hiccup/; Last Accessed April 25, 2018.

[6] Networking — Definition of Networking by Merriam-Webster. Available at https://www.merriam-webster.com/dictionary/networking; Last Accessed April 25, 2018.

[7] J. Day. Patterns in Network Architecture: A Return to Fundamentals. Prentice Hall, New Jersey, United States of America; 2008; ISBN 0137063385.

[8] J. Saltzer. On the naming and binding of network destinations. RFC 1498; August 1993; Available at https://www.rfc-editor.org/info/rfc1498; Last Accessed April 25, 2018.

[9] IEEE Standards Association. Guidelines for Use of Extended Unique Identifier (EUI), Organizationally Unique Identifier (OUI), and Company ID (CID). Available at https://standards.ieee.org/develop/regauth/tut/eui.pdf; Last Accessed April 25, 2018.

[10] R. Droms. Dynamic Host Configuration Protocol. RFC 2131; March 1997; Available at https://www.rfc-editor.org/info/rfc2131; Last Accessed April 25, 2018.

[11] J. Postel. User Datagram Protocol. STD 6, RFC 768; August 1980; Available at https://www.rfc-editor.org/info/rfc768; Last Accessed April 25, 2018.

[12] J. Postel. Transmission Control Protocol. STD 7, RFC 793; September 1981; Available at https://www.rfc-editor.org/info/rfc793; Last Accessed April 25, 2018.

[13] B. Haberman. IP Forwarding Table MIB. RFC 4292; April 2006; Available at https://www.rfc-editor.org/info/rfc4292; Last Accessed April 25, 2018.

[14] K. Barraclough, P. Cripps, A. Gay, and A. Jones (2017). Token ring network. Patent number US 5553073 A; Available at https://www.google.com/patents/US5553073; Last Accessed April 25, 2018.

[15] P. Li, B. Ravindran. Fast, best-effort real-time scheduling algorithms. IEEE Computers Transactions, vol. 53, is. 9, pp. 1159-1175, ISSN 0018-9340; 2004.

[16] H. Mong-Fong, et al. An adaptive approach to weighted fair queue with QoS enhanced on IP network. Proceedings in IEEE Region 10 International Conference on Electrical and Electronic Technology, vol. 1, pp. 181-186; 2001; Phuket Island and Langkawi Island, Singapore.

[17] V. Cerf, Y. Dalal, and C. Sunshine. Specification of Internet Transmission Control Program, RFC 675; December 1974; Available at https://www.rfc-editor.org/info/rfc675; Last Accessed April 25, 2018.

[18] J. Turner. New directions in communications (or which way to the information age?). IEEE Communications Magazine, vol. 24, is. 10, pp. 815, ISSN 0163-6804; 1986.

[19] P.M.Gopal and J.W.Wong. Analysis of a hybrid token-CSMA/CD protocol for bus networks. Computer Networks and ISDN Systems, vol. 9, is. 2, pp. 131-141, ISBN 0-7803-0538-8; 1985.

[20] W. Stevens. TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms. RFC 2001; January 1997; Available at https://www.rfc-editor.org/info/rfc2001; Last Accessed April 25, 2018.

[21] K. Ramakrishnan and S. Floyd. A Proposal to add Explicit Congestion Notification (ECN) to IP. RFC 2481; January 1999; Available at https://www.rfc-editor.org/info/rfc2481; Last Accessed April 25, 2018.

[22] B. A. Forouzan. TCP/IP Protocol Suite. McGraw-Hill, New York, United States of America; 2002; ISBN 0071199624.

[23] J. Postel. NCP/TCP transition plan. RFC 801; November 1981; Available at https://www.rfc-editor.org/info/rfc801; Last Accessed April 25, 2018.

[24] The OSI Model's Seven Layers Defined and Functions Explained; Last Update 2017; Available at https://support.microsoft.com/en-us/help/103884/the-osi-model-s-seven-layers-defined-and-functions-explained; Last Accessed April 25, 2018.

[25] M. Rouse. OSI reference model - (Open System Interconnection); Last Update 2014; Available at http://searchnetworking.techtarget.com/definition/OSI; Last Accessed April 25, 2018.

[26] IEEE Computer Society. IEEE Standard for Ethernet. IEEE Std 802.3TM-2015; Available at http://ieeexplore.ieee.org/document/7428776/; Last Accessed April 25, 2018.

[27] Cysco — VoIP over Frame Relay with Quality of Service (Fragmentation, Traffic Shaping, LLQ / IP RTP Priority); 2006; Available at https://www.cisco.com/c/en/us/support/docs/voice/voice-quality/12156-voip-ov-fr-qos.html; Last Accessed April 25, 2018.

[28] C. Brazdziunas. IPng Support for ATM Services. RFC 1680; August 1994; Available at https://www.rfc-editor.org/info/rfc1680; Last Accessed April 25, 2018.

[29] J. Postel. Internet Protocol. STD 5, RFC 791; September 1981; Available at https://www.rfc-editor.org/info/rfc791; Last Accessed April 25, 2018.

[30] S. Deering and R. Hinden. Internet Protocol, Version 6 (IPv6) Specification. RFC 2460; December 1998; Available at https://www.rfc-editor.org/info/rfc2460; Last Accessed April 25, 2018.

[31] Internet Corporation for Assigned Names and Numbers - ICANN; Available at https://www.icann.org; Last Accessed April 25, 2018.

[32] Internet Assigned Numbers Authority - IANA; Available at https://www.iana.org/; Last Accessed April 25, 2018.

[33] Address Supporting Organization - ASO; Available at https://aso.icann.org/; Last Accessed April 25, 2018.

[34] Country Code Names Supporting Organisation - CNNSO; Available at https://ccnso.icann.org/en; Last Accessed April 25, 2018.

[35] Well known SCTP, TCP and UDP ports, 0 through 999; Available at http://www.networksorcery.com/enp/protocol/ip/ports00000.htm; Last Accessed April 25, 2018.

[36] R. Stewart, et al. Stream Control Transmission Protocol. RFC 2960; October 2000; Available at https://www.rfc-editor.org/info/rfc2960; Last Accessed April 25, 2018.

[37] E. Kohler, M. Handley and S. Floyd. Datagram Congestion Control Protocol (DCCP). RFC 4340; March 2006; Available at https://www.rfc-editor.org/info/rfc4340; Last Accessed April 25, 2018.

[38] S. Bensley, et a. Data Center TCP (DCTCP): TCP Congestion Control for Data Centers. RFC 8257; October 2017; Available at https://www.rfc-editor.org/info/rfc8257; Last Accessed April 25, 2018.

[39] D. Plummer. Ethernet Address Resolution Protocol: Or Converting Network Protocol Addresses to 48-bit Ethernet Address for Transmission on Ethernet Hardware. STD 37, RFC 826; November 1982; Available at https://www.rfc-editor.org/info/rfc826; Last Accessed April 25, 2018.

[40] G. Malkin. Traceroute Using an IP Option. RFC 1393; January 1993; Available at https://www.rfc-editor.org/info/rfc1393; Last Accessed April 25, 2018.

[41] G. Huston. What's So Special about 512?; 2014; Available at http://www.potaroo.net/ispcol/2014-09/512.html; Last Accessed April 25, 2018.

[42] S. Goldberg. Why Is It Taking So Long to Secure Internet Routing?. Queue - Security, vol. 12, is. 8, pp. 20; New York, United States of America; 2014; DOI 10.1145/2668152.2668966.

[43] DARPA —ARPANET and the Origins of the Internet; Available at https://www.darpa.mil/about-us/timeline/arpanet; Last Accessed April 25, 2018.

[44] G. Deloche. Implementation of the Host - Host Software Procedures in GORDO. RFC 11; August 1969; Available at <https://www.rfc-editor.org/info/rfc11; Last Accessed April 25, 2018.

[45] S. Crocker. Protocol Notes. RFC 36; March 1970; Available at https://www.rfc-editor.org/info/rfc36; Last Accessed April 25, 2018.

[46] T. Berners-Lee and M. Fischetti. Weaving the Web: The Original Design and Ultimate Destiny of the World Wide Web by Its Inventor. Harper Collins, New York, United States of America; 2000; ISBN 006251587X.

[47] World Wide Web Consorcium; Available at https://www.w3.org/; Last Accessed April 25, 2018.

[48] PSI Network; Available at http://www.psi-net.com; Last Accessed April 25, 2018.

[49] UUNET at Wikipedia; Available at https://en.wikipedia.org/wiki/UUNET; Last Accessed April 25, 2018.

[50] CERFNet; Available at http://www.cerfnet.com; Last Accessed April 25, 2018.

[51] Free Pool of IPv4 Address Space Depleted — The Number Resource Organization; Available at https://www.nro.net/ipv4-free-pool-depleted/; Last Accessed April 25, 2018.

[52] What is Default-Free Zone (DFZ)? - Definition from Techopedia; Available at https://www.techopedia.com/definition/24974/default-free-zone-dfz; Last Accessed April 25, 2018.

[53] G. Huston. BGP Growth Revisited; 2011; Available at http://www.potaroo.net/ispcol/2011-11/bgp2011.html; Last Accessed April 25, 2018.

[54] G. Huston. BGP in 2014; 2015; Available at http://www.potaroo.net/ispcol/2015-01/bgp2014.html; Last Accessed April 25, 2018.

[55] G. Huston. Addressing 2014 - And then there was 2!; 2015; Available at http://www.potaroo.net/ispcol/2015-01/addressing2014.html; Last Accessed April 25, 2018.

[56] A. Andreyev. Introducing data center fabric, the next-generation Facebook data center network; 2014; Available at https://code.facebook.com/posts/360346274145943/introducing-data-center-fabric-the-next-generation-facebook-data-center-network/; Last Accessed April 25, 2018.

[57] J. Melvin and R. Watson. First Cut at a Proposed Telnet Protocol. RFC 97; February 1971; Available at https://www.rfc-editor.org/info/rfc97; Last Accessed April 25, 2018.

[58] A. Bhushan .File Transfer Protocol (FTP) status and further comments. RFC 414; December 1972; Available at https://www.rfc-editor.org/info/rfc414; Last Accessed April 25, 2018.

[59] J. Klensin. Simple Mail Transfer Protocol. RFC 2821; April 2001; Available at https://www.rfc-editor.org/info/rfc2821; Last Accessed April 25, 2018.

[60] C. Holland. Proposed Remote Job Entry Protocol. RFC 360; June 1972; Available at https://www.rfc-editor.org/info/rfc360; Last Accessed April 25, 2018.

[61] T. Starr, J. M. Cioffi and P. J. Silverman. Understanding digital subscriber line technology. Prentice Hall PTR, Upper Saddle River, United States of America; 1999; ISBN:0-13-780545-4.

[62] P. Graham. Web 2.0; 2005; Available at http://www.paulgraham.com/web20.html; Last Accessed April 25, 2018.

[63] Smartphone at Wikipedia; Available at https://en.wikipedia.org/wiki/Smartphone; Last Accessed April 25, 2018.

[64] Social Media - Techtarget; Available at http://whatis.techtarget.com/definition/social-media; Last Accessed April 25, 2018.

[65] Online game at Wikipedia; Available at https://en.wikipedia.org/wiki/Online_game; Last Accessed April 25, 2018.

[66] Y. Rekhter and T. Li. A Border Gateway Protocol 4 (BGP-4). RFC 1771; March 1995; Available at https://www.rfc-editor.org/info/rfc1771; Last Accessed April 25, 2018.

[67] J. L. Sobrinho. Network routing with path vector protocols: theory and applications. Proceedings in the 2003 conference on Applications, technologies, architectures, and protocols for computer communications, pp. 49-60; 2003; Karlsruhe, Germany.

[68] J. Mauch, J. Snijders and G. Hankins. Default External BGP (EBGP) Route Propagation Behavior without Policies. RFC 8212; July 2017; Available at https://www.rfc-editor.org/info/rfc8212; Last Accessed April 25, 2018.

[69] P. Marques, et al. Internal BGP as the Provider/Customer Edge Protocol for BGP/MPLS IP Virtual Private Networks (VPNs). RFC 6368; September 2011; Available at https://www.rfc-editor.org/info/rfc6368; Last Accessed April 25, 2018.

[70] Introduction to EIGRP - Cisco; Available at https://www.cisco.com/c/en/us/support/docs/ip/enhanced-interior-gateway-routing-protocol-eigrp/13669-1.html; Last Accessed April 25, 2018.

[71] B. S. Davieand Y. Rekhter. MPLS: Technology and Applications. Morgan Kaufmann, San Francisco, United States of America; 2000; ISBN 1558606564.

[72] P. Mockapetris. Domain names: Concepts and facilities. RFC 882; November 1983; Available at https://www.rfc-editor.org/info/rfc882; Last Accessed April 25, 2018.

[73] P. Mockapetris. Domain names: Implementation specification. RFC 883; November 1983; Available at https://www.rfc-editor.org/info/rfc883; Last Accessed April 25, 2018.

[74] S. Castro et al. A day at the root of the internet. ACM SIGCOMM Computer Communication Review archive, vol. 38, is 5, pp. 41-46; 2008.

[75] J. Klensin. Simple Mail Transfer Protocol. RFC 5321; October 2008; Available at https://www.rfc-editor.org/info/rfc5321; Last Accessed April 25, 2018.

[76] K. Jungwon and P. J. Bentley. An evaluation of negative selection in an artificial immune system for network intrusion detection. Proceedings in the 3rd Annual Conference on Genetic and Evolutionary Computation, pp. 1330-1337; 2001; San Francisco, United States of America.

[77] Network Address Translation (NAT) FAQ - Cisco; 2014; Available at https://www.cisco.com/c/en/us/support/docs/ip/network-address-translation-nat/26704-nat-faq-00.html; Last Accessed April 25, 2018.

[78] Online all the time – average British household owns 7.4 internet devices - The Guardian; Available at https://www.theguardian.com/technology/2015/apr/09/online-all-the-time-average-british-household-owns-74-internet-devices; Last Accessed April 25, 2018.

[79] I. Asnis and V. Dham (2009). Mediated network address translation traversal. Patent number US 8149851 B2; Available at https://www.google.com/patents/US8149851; Last Accessed April 25, 2018.

[80] H. Levkowetz and S. Vaarala. Mobile IP Traversal of Network Address Translation (NAT) Devices. RFC 3519; April 2003; Available at https://www.rfc-editor.org/info/rfc3519; Last Accessed April 25, 2018.

[81] J. Rosenberg, et al. Session Traversal Utilities for NAT (STUN). RFC 5389; October 2008; Available at https://www.rfc-editor.org/info/rfc5389; Last Accessed April 25, 2018.

[82] Locator/ID Separation Protocol; Available at https://www.lisp4.net/; Last Accessed April 25, 2018.

[83] M. Boucadair and C. Jacquenet. Locator/ID Separation Protocol (LISP): Shared Extension Message & IANA Registry for Packet Type Allocations. RFC 8113; March 2017; Available at https://www.rfc-editor.org/info/rfc8113; Last Accessed April 25, 2018.

[84] V. Fuller, et al. Locator/ID Separation Protocol Delegated Database Tree (LISP-DDT). RFC 8111; May 2017; Available at <https://www.rfc-editor.org/info/rfc8111; Last Accessed April 25, 2018.

[85] A. T. Campbell, et al. A survey of programmable networks; ACM SIGCOMM Computer Communication Review, vol. 29 is. 2, pp. 7-23; 1999; ACM, New York, United States of America; 1999; DOI 10.1145/505733.505735.

[86] - B. A. A. Nunes, et al. A survey of software-defined networking: past, present, and future of programmable networks. IEEE Communications Surveys & Tutorials, vol. 16, is. 3, pp 1617-1634; 2014; ISSN 1553-877X.

[87] H. Kim and N. Feamster. Improving network management with Software Defined Networking. IEEE Communications Magazine, vol. 51, is. 2, pp. 114-119; 2013; ISSN 0163-6804.

[88] N. McKeown et al. OpenFlow: enabling innovation in campus networks. ACM SIGCOMM Computer Communication Review, vol. 38, is. 2, pp.69-74; ACM, New York, United States of America; 2008; DOI 10.1145/1355734.1355746.

[89] Open Networking Foundation; Available at https://www.opennetworking.org/; Last Accessed April 25, 2018.

[90] S. G. Low (2017). Network virtualization. Patent number US 9705756 B2; Available at https://www.google.com/patents/US9705756; Last Accessed April 25, 2018.

[91] What is Network Virtualization? – Definition; Available at https://www.sdxcentral.com/sdn/network-virtualization/definitions/whats-network-virtualization/; Last Accessed April 25, 2018.

[92] J. Doherty. SDN and NFV Simplified: A Visual Guide to Understanding Software Defined Networks and Network Function Virtualization. Addison-Wesley Professional, Indianapolis, United States of America; 2016; ISBN 0134307372.

[93] M. Chiosi et al. Network Functions Virtualisation: an introduction, benefits, enablers, challenges and call for action. IEEE Communications Magazine, vol. 53, is. 2, pp. 90-97; 2012; ISNN 0163-6804.

[94] B. Carpenter. Architectural Principles of the Internet. RFC 1958; June 1996; Available at https://tools.ietf.org/html/rfc1958; Last Accessed April 25, 2018.

[95] The Pouzin Society - Building A Better Network; Available at http://pouzinsociety.org/; Last Accessed April 25, 2018.

[96] S. Vrijders, et al. Unreliable inter process communication in Ethernet: migrating to RINA with the shim DIF. Proceedings in International Congress on Ultra Modern Telecommunications and Control Systems andWorkshops (ICUMT); 2013; Almaty, Kazakhstan.

[97] Eduard Grasa, et al. Simplifying multi-layer network management with RINA. Proceedings in TERENA Networking Conferences; 2016; Prague, Czech Republic.

[98] R. Watson. Timer-Based Mechanisms in Reliable Transport Protocol Connection Management. Computer Networks, vol. 5, is. 1, pp. 47-56; Elsevier, Amsterdam, The Netherlands; 1981; DOI 10.1016/0376-5075(81)90031-3.

[99] P. Membrey, D. Hows and E. Plugge. DNS Load Balancing. Practical Load Balancing, pp. 53-69; Apress, New York, United States of America; 2012; ISBN 978-1-4302-3681-8.

[100] N. Davies. Delivering predictable quality in saturated networks. Technical Report; 2003; Available at http://www.pnsol.com/public/TP-PNS-2003-09.pdf; Last Accessed April 25, 2018.

[101] N. Davies, J. Holyer and P. Thompson. An operational model to control loss and delay of traffic in a network switch. Proceedings in 3th IFIP Workshop on Traffic Management and Design of ATM Networks; 1999; University of Bristol, Bristol, United Kingdom.

[102] N. Davies, J. Holyer and P. Thompson. A Queueing Theory Model that Enables Control of Loss and Delay at a Network Switch. Technical Report; 1999; University of Bristol, Bristol, United Kingdom.

[103] K. Minji, M M'edard and J. Barros. Modeling network coded TCP throughput: a simple model and its validation. Proceedings in the 5th International ICST Conference on Performance Evaluation Methodologies and Tools, pp- 131-140; 2011; Paris, France.

[104] PRISTINE FP7-619305, Programmability in RINA for European supremacy of virtualized networks; Available at http://ict-pristine.eu; Last Accessed April 25, 2018.

[105] ARCFIRE H2020-687871, Experimenting RINA on FIRE+; Available at http://ict-arcfire.eu; Last Accessed April 25, 2018.

[106] Forbes - A Short History Of Bitcoin And Crypto Currency Everyone Should Read; Available at https://www.forbes.com/sites/bernardmarr/2017/12/06/a-short-history-of-bitcoin-and-crypto-currency-everyone-should-read/; Last Accessed April 25, 2018.

[107] SETI@home webpage; Available at https://setiathome.berkeley.edu/; Last Accessed April 25, 2018.

[108] SlapOS webpage; Available at https://slapos.vifib.com/; Last Accessed April 25, 2018.

[109] The Pouzin Society - Podcasts; Available at http://rina.tssg.org/; Last Accessed April 25, 2018.

[110] V. Veselý. A New Dawn of Naming, Addressing and Routing on the Internet. PhD diss., Faculty of Information Technology, Departament of Information Systems, Brno University of Technology, Brno, Czech Republic; 2015.

[111] S. Leon Gaixas, et al. Assuring QoS Guarantees for Heterogeneous Services in RINA Networks with ΔQ. Proceedings in NetCloud16; 2016; Luxembourg;

[112] L. Kleinrock. Queuing System, Vol 1: Theory. Wiley-Interscience, pp. 117. Wiley, Nueva York, United States of America; 1975.

[113] RINASim at Github; Available at https://github.com/kvetak/RINA; Last Accessed April 25, 2018.

[114] RINASim; Available at https://rinasim.omnetpp.org/; Last Accessed April 25, 2018.

[115] S. Leon Gaixas. Tutorial - Assuring Absolute QoS Guarantees for Heterogeneous Services in RINA Networks with *Delta*Q; 2016; Available at https://github.com/kvetak/RINA/wiki/Assuring-Absolute-QoS-Guarantees-for-Heterogeneous-Services-in-RINA-Networks-with-%CE%94Q; Last Accessed April 25, 2018.

[116] S. Leon Gaixas. Tutorial - Topological Routing in DC; 2016; Available at https://github.com/kvetak/RINA/wiki/Topological-Routing-in-DC; Last Accessed April 25, 2018.

[117] IRATI FP7-317814, Researching and Prototyping the Recursive InterNetwork Architecture to Support Distributed Computing; Available at http://irati.eu; Last Accessed April 25, 2018.

[118] IRATI - RINA stack at Github; Available at https://github.com/IRATI/stack; Last Accessed April 25, 2018.

[119] S. Leon Gaixas. RINA Traffic Generator at Github; Available at https://github.com/gaixas1/traffic-generator; Last Accessed April 25, 2018.

[120] S. Leon Gaixas. ; RINA, Scheduling policies at Github; Available at https://github.com/gaixas1/rinapolicies; Last Accessed April 25, 2018.

[121] N. Davies, et al. (2004). Prioritizing data with flow control. Patent number US 20040196855 A1; Available at https://www.google.com/patents/US20040196855A1; Last Accessed April 25, 2018.

[122] N. Davies, et al. (2004). Allocating priority levels in a data flow. Patent number US 20040199655 A1; Available at https://patents.google.com/patent/US20040199655A1/; Last Accessed April 25, 2018.

[123] N. Davies, et al. (2004). Data flow control. Patent number US 20040022190 A1; Available at https://patents.google.com/patent/US20040022190A1/; Last Accessed April 25, 2018.

[124] N. Davies, et al. (2004). Filtering data flows. Patent number US 20040196792 A1; Available at https://patents.google.com/patent/US20040196792A1/; Last Accessed April 25, 2018.

[125] N. Davies, et al. (2009). Prioritizing data with flow control. Patent number US 7535835 B2; Available at https://patents.google.com/patent/US7535835B2/; Last Accessed April 25, 2018.

[126] N. Davies, et al. (2009). Information flow control in a packet network based on variable conceptual packet lengths. Patent number US 7499400 B2; Available at https://patents.google.com/patent/US7499400B2/; Last Accessed April 25, 2018.

[127] S. M. Rahman. Multimedia Networking: Technology, Management and Applications: Technology. Idea Group Pub, Pennsylvania, United States of America; 2002; ISBN:1930708149.

[128] L. Leahu. Analysis and predictive modeling of the performance of the ATLAS TDAQ network. PhD diss., Universitatea POLITEHNICA Bucures, Bucuresti, Romania; 2013.

[129] D. Ahmed, et al. Language independent on-off voice over IP source model with lognormal transitions. IET Communications, vol. 7, is. 14, pp. 1449-1455; 2013; ISSN 1751-8628.

[130] M. F. Bari, et al. Data center network virtualization: a survey. IEEE Communications Surveys & Tutorials, vol. 15, is. 2, pp. 909-928; 2013;

[131] S. Leon Gaixas S, et al. Scalable topological forwarding and routing policies in RINA-enabled programmable data centers. Transactions on Emerging Telecommunications Technologies, vol. ?, is. ?, pp. ?-?; 2018;

[132] S. Kandula, et al. Dynamic load balancing without packet reordering. ACM SIGCOMM Computer Communication Review, vol. 37, is. 2, pp. 51-62; ACM, New York, United States of America; 2007; DOI 10.1145/1232919.1232925.

[133] E. Trouva, et. al. IS THE INTERNET AN UNFINISHED DEMO? MEET RINA!. Proceedings in TERENA Networking Conference; 2010; Prague, Czech Republic.

[134] J. Day, I. Matta and K. Mattar. Networking is IPC: A Guiding Principle to a Better Internet. Proceedings in the 2008 ACM CoNEXT Conference; 2008; Madrid, Spain. Article No. 67; 2008; Madrid, Spain.

[135] F. Dürr, T. Kohler. Comparing the Forwarding Latency of Open-Flow Hardware and Software Switches. Technical Report Computer Science; 2014.

[136] A. Singh, et al. Jupiter rising: a decade of Clos topologies and centralized control in Google's datacenter network. Proceedings in ACM Conference on Special Interest Group on Data Communication (SIGCOMM); 2015; London, United Kingdom.

[137] Roy A, et al. Inside the social network's (datacenter) network. Proceedings in the 2015 ACM Conference on Special Interest Group on Data Communication (SIGCOMM); 2015; London, United Kingdom.

[138] D. Arora, T. Benson and J. Rexford. ProActive routing in scalable datacentres with PARIS. Proceedings in the 2014 ACM SIGCOMM Workshop on Distributed Cloud Computing; 2014; Chicago, United States of America.

[139] S. Leon Gaixas S, et al. Benefits of programmable topological routing policies in RINA-enabled large-scale datacenters. Proceedings in the IEEE Global Communications Conference (GLOBECOM); 2016; Washington DC, United States of America.

[140] Y. Ganjali and A. Keshavarzian. Load balancing in ad hoc networks: Single-path routing vs. multi-path routing. Proceedings in INFOCOM 2004, Twenty-third Annual Joint Conference of the IEEE Computer and Communications Societies; 2004; Hong Kong, China.

[141] M. Thorup and U. Zwick. Compact routing schemes. Proceedings in the thirteenth annual ACM symposium on Parallel algorithms and architectures, pp. 1-10; 2001; New York, United States of America.

[142] P. F. Tsuchiya. The landmark hierarchy: a new hierarchy for routing in very large networks. Proceedings in Communications architectures and protocols, pp. 35-42; 1988; California, United States of America.

[143] M. Nottingham. URI Design and Ownership. BCP 190, RFC 7320; July 2014; Available at <https://www.rfc-editor.org/info/rfc7320; Last Accessed April 25, 2018.

[144] J. Postel. Internet Protocol Handbook: Table of contents. RFC 766; July 1980; Available at https://www.rfc-editor.org/info/rfc766; Last Accessed April 25, 2018

[145] T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.1. RFC 4346; April 2006; Available at https://www.rfc-editor.org/info/rfc4346; Last Accessed April 25, 2018.

[146] Y. Rekhter and T. Li. A Border Gateway Protocol 4 (BGP-4). RFC 4271; January 2006; Available at https://www.rfc-editor.org/info/rfc4271; Last Accessed April 25, 2018.

[147] P. Almquist. Type of Service in the Internet Protocol Suite. RFC 1349; July 1992; Available at https://www.rfc-editor.org/info/rfc1349; Last Accessed April 25, 2018.

[148] Amazon AWS. Available at https://aws.amazon.com; Last Accessed April 25, 2018.

[149] V. Beal. The Differences and Features of Hardware and Software Firewalls; 2010; Available at https://www.webopedia.com/DidYouKnow/Hardware_Software/firewall_types.asp; Last Accessed April 25, 2018.

[150] Queueing in the Linux Network Stack - Linux Journal; 2013; Available at http://www.linuxjournal.com/content/queueing-linux-network-stack; Last Accessed April 25, 2018.

[151] Routing Between VLANs Overview - Cisco; Available at https://www.cisco.com/c/en/us/td/docs/ios/12_2/switch/configuration/guide/fswtch_c/xcfvl.pdf; Last Accessed April 25, 2018.

[152] D. Hucaby and S. McQuerry. VLANs and Trunking; 2002; Available at http://www.ciscopress.com/articles/article.asp?p=29803&seqNum=3; Last Accessed April 25, 2018.

[153] IEEE 802.3ad Link Bundling - Cisco; 2006; Available at https://www.cisco.com/c/en/us/td/docs/ios/12_2sb/feature/guide/sbcelacp.html; Last Accessed April 25, 2018.

[154] P. Lapukhov, P. Premji and J. Mitchell J. Use of BGP for routing in large-scale data center. IETF Network Working Group, draft-ietf-rtgwg-bgp-routing-large-dc-07; August 2015; Available at https://tools.ietf.org/html/draft-ietf-rtgwg-bgp-routing-large-dc-07; Last Accessed April 25, 2018.

[155] A. Whitmore, A. Agarwal and L. Da Xu. The Internet of Things—a survey of topics and trends. Inf Syst Front, vol. 17, is. 7, pp. 261-274; 2015.

[156] J. G. Andrews, et al. What will 5G be?. IEEE J Sel Area Comm, vol. 32, is. 6, pp. 1065-1082; 2014.

[157] M. E. G'omez, P. L'opez, J. Duato. A memory-effective routing strategy for regular interconnection networks. Proceedings in the international Parallel and Distributed Processing Symposium (IPDPS'05); 2005; Denver, United States of America.

[158] S. Habib, F. S. Bokhari and S. U. Khan. Routing techniques in data center networks. Handbook on Data Centers, ch. 16; Springer-Verlag, New York, United States of America; 2015.

[159] K. Chen, et al. Survey on routing in data centers: insights and future directions. IEEE Netw, vol. 25, is. 4, pp. 6-10; 2011.

[160] J. Petri, et al. LIPSIN: line speed publish/subscribe inter-networking. ACM SIGCOMM Comput Commun Rev, vol.39, is. 4, pp- 195-206; 2009.

[161] B. H. Bloom BH. Space/time trade-offs in hash coding with allowable errors. Commun ACM, vol. 13, is. 7, pp. 422-426; 1970.

[162] G.722 : 7 kHz audio-coding within 64 kbit/s; Available at http://www.itu.int/rec/T-REC-G.722/en; Last Accessed April 25, 2018.

[163] Youtube - Live encoder settings, bitrates, and resolutions; Available at https://support.google.com/youtube/answer/2853702; Last Accessed April 25, 2018.

[164] Internet2; Available at http://www.internet2.edu; Last Accessed April 25, 2018.