Pristine

Deliverable-3.3

Final specification and consolidated implementation of scalable techniques to enhance performance and resource utilization in networks

Deliverable Editor: Michael Welzl, UiO

Publication date:	30-June-2016
Deliverable Nature:	Report/Software
Dissemination level (Confidentiality):	PU (Public)
Project acronym:	PRISTINE
Project full title:	PRogrammability In RINA for European Supremacy of virTualIsed NEtworks
Website:	www.ict-pristine.eu
Keywords:	programmable/recursive congestion control, qos-aware multi-path routing, delta-q, topological addressing, multi-layer routing
Synopsis:	This document describes the theoretical analysis, simulation and final implementation of resource-utilization and performance enhancing techniques.

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

List of Contributors

Deliverable Editor: Michael Welzl, UiO
UiO: Michael Welzl, Peyman Teymoori, David Hayes
UPC: Sergio Leon Gaixas, Jordi Perello, Sergio Leon
i2CAT: Eduard Grasa, Miquel Tarzan, Leonardo Bergesio
CN: Kewin Rausch, Roberto Riggio
IMT: Fatma Hrizi, Anis Laouiti
ATOS: Javier Garcia, Juan Vallejo, Miguel Angel Puente
PNSol: Peter Thompson, Neil Davies

Disclaimer

This document contains material, which is the copyright of certain PRISTINE consortium parties, and may not be reproduced or copied without permission.

The commercial use of any information contained in this document may require a license from the proprietor of that information.

Neither the PRISTINE consortium as a whole, nor a certain party of the PRISTINE consortium warrant that the information contained in this document is capable of use, or that use of the information is free from risk, and accept no liability for loss or damage suffered by any person using this information.

Executive Summary

In this document, the "final specification and consolidated implementation" of the techniques proposed in the previous document, "initial specification", are presented. The goal is to show how the proposed techniques in the previous document have been implemented in RINA, what their performance improvement is over other similar methods (if applicable), and what future directions are. The activities performed in D3.3 are centered around three main areas: i) programmable congestion control; ii) resource allocation and iii) topological addressing to bound routing table sizes. This document also specifies the investigation results of these techniques as policies in RINA.

Table of Contents

List of Figures

List of acronyms

ACC	Aggregate Congestion Control
AE	Application Entity
AI	Application Instance
AP	Application Process
C/U Mux	Cherish-Urgency Multiplexer
CACEP	Common Application Connection Establishment Protocol
CCP	Continuity Check Protocol
CDAP	Common Distributed Application Protocol
DA	Distributed Application
DAF	Distributed Application Facility
DC	Data Centre
DCN	Data Centre Network
DCTCP	Data Center TCP
DIF	Distributed IPC Facility
DTCP	Data Transfer Control Protocol
DTP	Data Transfer Protocol
E2E	End to End
ECMP	Equal-Cost Multi-Path
ECN	Explicit Congestion Notification
EFCP	Error Flow Control Protocol
FA	Flow Allocator
FAI	Flow Allocator Instance
FIFO	First In, First Out
FQ	Fair Queuing
IANA	Internet Assigned Numbers Authority
IPC	Inter Process Communication
IPCP(s)	IPC Process(es)
IRM	IPC Resource Manager
ISP	Internet Service Provider
LAN	Local Area Network
LG	Logistic Growth
LGC	Logistic Growth Control
LIFO	Last In, First Out
MAC	Medium Access Control
MGB	Minimum Granted Bandwidth

NM-DMS	Network Management Distributed Management System
MPLS	Multi-Protocol Label Switching
MPLS-TE	MPLS with Traffic Engineering extensions
NSM	Name-Space Manager
OS	Operating System
OSPF	Open Shortest Path First
PCI	Protocol-Control-Information
PDU	Protocol Data Unit
PDUFG	PDU Forwarding Generator Policies
PFT	Protocol Data Unit Forwarding Table
PFTG	PDU Forwarding Table Generator
PoA	Point of Attachment
QoS	Quality of Service
RA	Resource Allocator
RIB	Resource Information Base
RINA	Recursive InterNetwork Architecture
RIR	Regional Internet Registry
RMT	Relaying and Multiplexing Task
RR	Round Robin
RSVP-TE	ReSerVation Protocol with Traffic Engineering extensions
SDU	Service Data Unit
SFR	Scalable Forwarding in RINA
TCP	Transmission Control Protocol
ToR	Top-of-the-Rack
WLAN	Wireless LAN

1. Introduction

RINA is a framework that allows changing specific small parts of it, which are called policies. The intention is to be able to do everything that other networks can do by only changing policies, but benefiting from a large amount of generic functionality. Some day in the future, this generic functionality could be all implemented in hardware and extremely efficient, yet allow for just the necessary amount of flexibility - the true SDN advantage. Generally, even only implementing an existing mechanism in RINA helps to show the correctness of the architecture and highlight its benefits in terms of reducing code needed to accomplish certain tasks. However, some of the contributions of WP3 are new research developments in their own right, related to the RINA and its recursive nature in various different ways. Here, we summarize the work performed in WP3.

WP3 developments fall into three categories that map to the three tasks of the work package: congestion control, resource allocation and topological addressing.

1.1. Congestion control

The congestion control work carried out in WP3 and reported in this deliverable is as follows:

- **Programmable Congestion Control**. This consists of:
  - **Aggregate Congestion Control (ACC)** - an analysis of what happens when we just plug in a TCP-like congestion control policy in RINA and then use different stack configurations. We wanted to confirm to ourselves that doing congestion control in the RINA-way (per DIF, not end-to-end) is indeed favorable.
  - **Logistic Growth Control (LGC)** - a new congestion control mechanism. We need to have a better-to-understand (model) mechanism to use in RINA than TCP/AIMD, one that also works better than TCP and can play out differently in different DIFs, depending on the DIF's abilities (our control can work with precise signaling-based feedback or just ECN marks).
  - **A Chain of Logistic Growth Controllers** - a preliminary analysis of how chains of DIFs running LGC operate. This is indeed necessary to

eventually understand the stability and overall performance of LGC in various RINA stack configurations.

- **Recursive Congestion Control** - an analysis of different ways to give feedback in RINA. In RINA, the natural way of using control loops is per DIF; the recursive nature makes it less obvious that feedback should follow the end-to-end (from the true source to the destination of the data) path of TCP, and it appears natural to investigate other, possibly more efficient (more immediate) ways to give feedback. However, little is known about how such feedback methods play out. This information is necessary for the continued design of RINA congestion control.

- **Performance isolation in multi-tenants datacentres** - a method to dynamically scale up/down the rate of admitted flows belonging to isolated tenants while guaranteeing them a "Minimum Granted Bandwidth" (MGB). We wanted to show how that functionality similar to EyeQ [EyeQ] can be efficiently and easily implemented in RINA (1000 lines of code vs. EyeQ’s 10000 lines of code, letting us benefit from the many generic things that RINA already inherently does (e.g. error handling, ..)).

1.2. Resource Allocation

The work carried out in WP3 related to Resource Allocation and reported in this deliverable is as follows:

- **Traffic differentiation via delay-loss multiplexing policies** - a method (QTAMux within RINA) that allows applications to request and receive communication services with strongly bounded loss or delay, while fully utilising the most constrained resources. RINA allows applications to provide rich QoS information via its QoS-Cubes, allowing QTAMux to fully play out its benefits over traditional methods such as WFQ (which is in use in many practical QoS oriented scenarios, e.g. for MPLS VPNs).

- **QoS-aware Multipath Routing** - an evaluation of different multipath techniques taking advantage of RINA’s built-in support for QoS. These techniques are necessary for future evaluations in WP6 related to the DC use case, where we intend to analyse benefits that are due to the rich information about traffic characteristics provided by RINA’s QoS-Cubes.

1.3. Topological Addressing

The work carried out in WP3 related to Topological Addressing and reported in this deliverable is as follows:

- **Topological addressing and routing in distributed clouds**
    - **Scalable Forwarding in RINA (SFR)**: Analysis of applying hierarchical overlay architecture to RINA. We demonstrate that hierarchical overlay applied to RINA is scalable and the ”divide and conquer” strategy of RINA helped to bound the routing table sizes.
    - **Small-world Architecture**: Design a new hierarchical routing architecture based on small-world paradigm. New pro’s and con’s will arise when mapping the hierarchy of this routing architecture on RINA’s own hierarchy, and we also want to investigate these implications in future work beyond PRISTINE; we think that this architecture is able to deal with the dynamicity issue of the distributed clouds.
- **Topological addressing and routing in large datacentres** - Evaluation of forwarding and routing solutions that benefit from the well-known topological characteristics of typical large-scale intra-datacenter networks, so as to minimize the routing and forwarding information to be stored at (and exchanged among) network devices. RINA is a programmable environment which allows us to benefit from using policies that are tailored to the specific DCN characteristics, yielding more efficient routing in terms of table size and communication overhead.

The following table states the main focus of the methods presented in this document.

Table 1. Categorizing the WP3 work in the remainder of this deliverable

Deliverable Section	**Deliverable Sub(sub)section**	**Evaluating flexibility/ benefits of RINA**	**Implementing a novel policy/ method**	**Analytical evaluation**
Congestion Control	Aggregate congestion control	x		
	Logistic Growth Control		x	

	A Chain of logistic growth controllers		x	x
	Recursive congestion control	x		x
	Performance isolation in multi-tenants datacentres	x		
Resource Allocation	Traffic differentiation via delay-loss multiplexing policies based on ΔQ	x	x*	
Topological Addressing	Topological addressing and routing in distributed clouds: SFR	x		
	Topological addressing and routing in large datacentres	x	x	

*This is the only true implementation of the ΔQ theory in a network (with one exception: there is an implementation for Asynchronous Transfer Mode (ATM) networks).

Two sections ("QoS-aware multi-path routing" and "Small-world architecture") fit none of the criteria in the table above; these contributions enrich RINA policies for further investigation in WP6 / scientific publishing in WP7 or beyond the PRISTINE lifetime.

2. Congestion control

2.1. Programmable congestion control

2.1.1. Introduction

In this section, we present two sets of Congestion Control (CC) policies for RINA. The first set is TCP-like, and designed to illustrate how CC can be done in RINA; we call it Aggregation Congestion Control (RINA-ACC) because it operates per DIF and would work on aggregates when used inside the network. Two instances of RINA-ACC in sequence behave similar to a common Performance Enhancing Proxy (PEP) called "Split-TCP" [SplitTCP], however, due to the architectural properties of RINA, it does not have side effects that TCP Splitters normally have. A complete description of this policy set and its performance results can be found in [RINA-ACC] which was published in IEEE ICC 2016. This paper is also attached to this document in Appendix [paper1]. In this section, we summarize this work.

The second CC policy set presents an advanced CC mechanism which is called Logistic Growth Control (LGC). LGC is based on the Logistic Growth (LG) function which has been proven to have favorable characteristics regarding stability, convergence, fairness, and scalability [LGm]; these are very appealing for CC. We split the design of this policy set into two steps: the first step evaluates LGC as an end-to-end congestion controller. We compare this policy set with other similar approaches. In particular, we target datacenters and show how LGC can perform compared with DCTCP CP [DCTCP] as a similar approach. We show that LGC behaves better than DCTCP, and it converges to the fair share of the bottleneck link capacity irrespective of the Round-Trip-Time (RTT). We discuss the stability and fairness of LGC using a fluid model, and show its performance improvement with simulations. Further discussion and detailed analytical results can be found in Appendix [paper2] which is under review at the time of writing of this document.

As the next step, we model CC in RINA as a chain of controllers. Since LGC is based on LG, this will be done using the food chain and predator-pray models [DynSyst].

In summary, we present

- RINA’s flow aggregation benefits for congestion control,
- A Logistic Growth congestion control policy for RINA,
- A chain of logistic growth controllers to model congestion control in RINA and evaluate the stability of this policy.

2.1.2. Aggregate Congestion Control

In the previous deliverable, D3.2, we presented a set of TCP-like policies for CC in RINA. This section summarizes the results of [RINA-ACC] which is presented in Appendix [paper1]. The main goal of [RINA-ACC] was to show that improvements that have been done to TCP such as Split-TCP on the Internet "naturally appear" with RINA without their side effects. The benefits occur just as the result of layering, as a network configuration by-product, *without changing a line of code in the congestion controller itself*.

In RINA, recursion arises from the ability to arbitrarily arrange structurally-equivalent DIFs. We also show that in RINA, each DIF can detect and manage the congestion for its resources, pushing back to higher layer DIFs when resources are overloaded. There, the “Relaying and Multiplexing” (RMT) task – another mechanism in every IPCP/DIF – is in charge of forwarding the EFCP PDUs; it can load-balance the traffic by sending it on other paths, or recursively pushback upwards to achieve in-network resource pooling.

DIF Configurations in RINA

In RINA, every function is bound to one DIF, and every DIF can have a different type of congestion control (or none at all). Depending on DIF configurations, three situations happen:

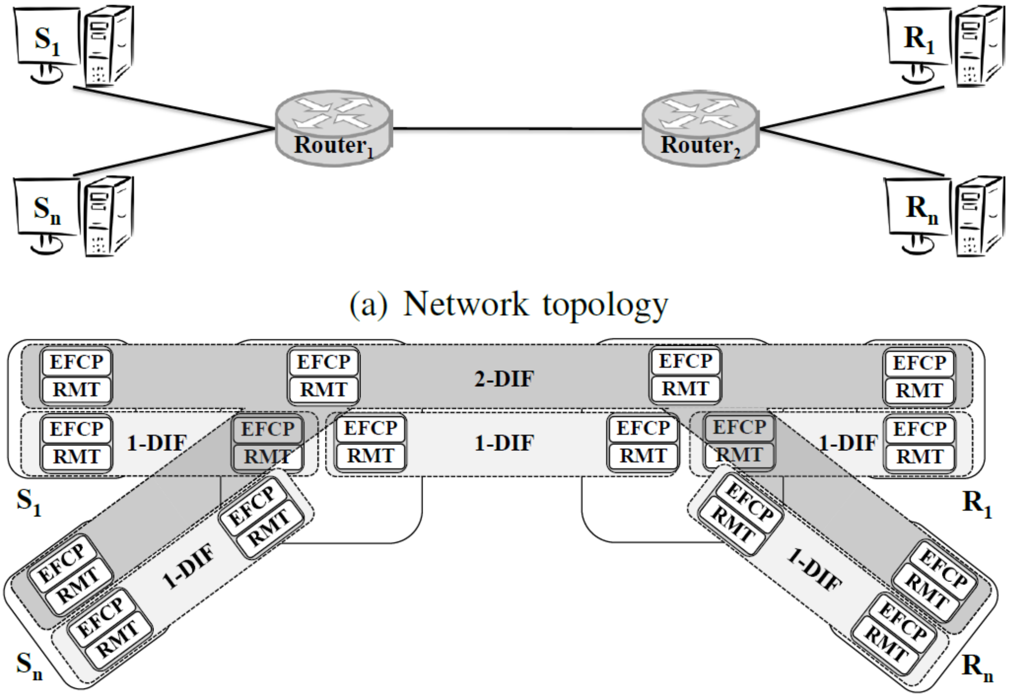
- Horizontal: Consecutive DIFs (i.e., side by side, not above each other). This is very similar to a PEP function called Connection Splitting [SplitTCP]. TCP splitters divide TCP connections by “lying” to the end systems, acting as the receiver towards the sender and as the sender towards the receiver.
- Vertical: Stacked DIFs. DIFs can be stacked above each other; an N-DIF would carry an aggregate of flows from the (N+1)-DIF sitting above it. This automatically avoids the competition between multiple end-to-end flows that occurs in the Internet today.

- Around: In-Network Resource Pooling. RINA can also react to congestion by finding and using alternative routers with lower loads at a DIF, combined with a hop-by-hop congestion control mechanism in case there is no other low-load path towards the destination; this provides dynamic routing/detouring compared with the static routing of today's Internet.

Selected Results

The following results illustrate the benefit that arise due to aggregation. They were obtained from RINASim, the simulator which was developed in the context of WP2, and compared with the TCP implementations of the INET framework of OMNeT++.

In Figure 1, senders S_1 through S_n sent a large file to receivers R_1 through R_n, respectively. The $Router_1$–$Router_2$ link was the bottleneck. For RINA, the DIF structure is also indicated in Figure 1: there were three consecutive lower DIFs and one upper-layer DIF on top. We compared RINA-ACC against the most beneficial Internet case from our previous simulations: Split-TCP, but with no aggregation.



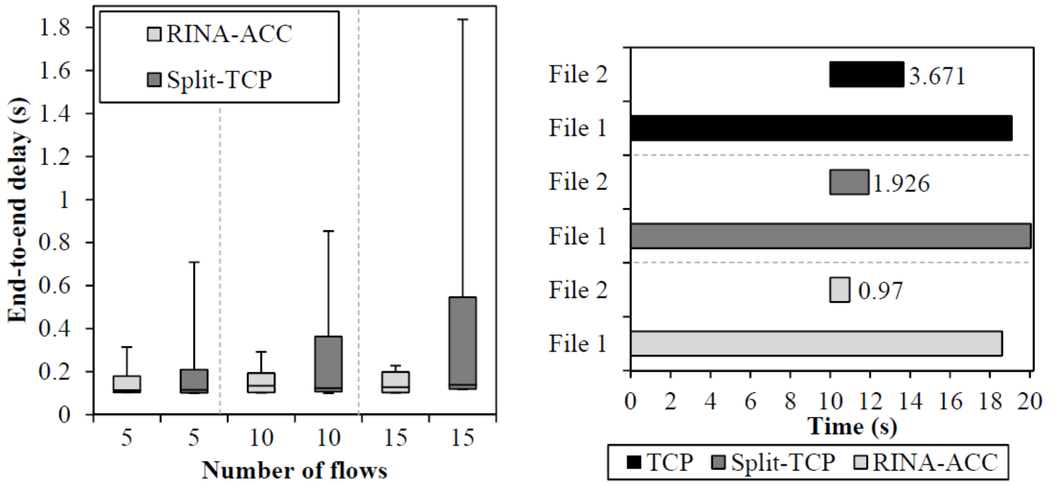
(a) Network topology

(b) RINA stack: there is one 1-DIF between every pair of adjacent nodes, and a single 2-DIF on top which connects all the nodes.

Figure 1. Network topology for multiple flows and its RINA stack

In Figure 2(a), end-to-end delay results of RINA-ACC and Split-TCP are shown in a box-and-whisker diagram; it shows the range and 10th percentile/median/90th percentile boxes of all packets in one simulation. Although the median of delay is almost the same, we observe that due to the competition among the TCP connections in the Router1–Router2 segment, some packets had much longer end-to-end delays, which also causes a higher jitter at receivers. In RINA-ACC, all traffic from the senders was carried through one flow between $Router_1$ and $Router_2$ with no competition. The effect of competition can be mitigated to some extent by employing Active Queue Management (AQM) in Split-TCP. However, AQM cannot completely resolve the high jitter problem that is due to competition.

With RINA-ACC, we see a slight reduction in peak delay as the number of flows increases because at the start, more flows translate into more packets to be sent by the aggregated flow between $Router_1$ and $Router_2$, keeping its send buffer from draining and allowing its congestion window to grow faster. This implies another benefit of ACC: flows take advantage of an already open window of the aggregate flow in their path for faster transmission and shorter delay.



(a) End-to-end delay

(b) Transmitting two files from one sender to one receiver.

Figure 2. Benefits of RINA ACC

We simulated another scenario and show the results in Figure 2(b). The network topology was the same as in Figure 1 with n = 1 (i.e., one sender and one receiver). The sender sent two files to the receiver. The transmission of File 1 with the size of 20MB started at time 0. File 2 was 500 KB, and its transmission started 10 seconds later. The horizontal axis of the figure shows time, and the bars show the start and finish times of each file for the three methods. Due to the aggregation of the second flow with the already started one in RINA-ACC, the second transfer can benefit from the large congestion window of the ongoing transmission which already has a better approximation of the available bandwidth between the two nodes.

Discussion

RINA can solve the Internet problems regarding congestion control by

1. breaking up the long control loop into shorter ones,
2. controlling flow aggregates inside the network, and
3. enabling the deployment of arbitrary congestion control mechanisms per DIF.

RINA is, therefore, an ideal vehicle for investigating drastic changes to how congestion control and in-network resource pooling could be done, and provides a suitable framework with many promising dimensions for future research.

2.1.3. Logistic Growth Control

Here, we present a new congestion controller policy. The detailed discussion is presented in Appendix [paper2]; it is an under-review paper at the time of writing the document. Our work is inspired by logistic growth in nature [LGfn]. We build upon earlier work that has found logistic growth to be a generally useful function for congestion control [LGm], and present the design and simulation-based evaluation of a new congestion controller policy in RINA.

The major reason to develop a new congestion control mechanism instead of using an existing well-known mechanism from literature is that most of these mechanisms do not converge to a fixed point but to an oscillating equilibrium (e.g. Additive-Increase, Multiplicative-Decrease (AIMD) which is used in TCP). Often (as with AIMD), even this equilibrium is dependent on the round-trip time of flows. In RINA, it is natural to me that congestion control would be applied per DIF, meaning that a congestion control mechanism must be stable in various DIF configurations (consecutive, stacked, ..). This requires using a mechanism that has well understood stability properties (as is the case for logistic growth, which is asymptotically stable).

Cornerstones of our design are:

- Similar to DCTCP, we let packets be ECN-marked (using the ECN bit in the header of the EFCP protocol) when the instantaneous queue length exceeds a threshold (which can be achieved using a special configuration of the common RED Active Queue Management (AQM) mechanism - which is also available in RINA -, and hence needs no hardware changes). However, different from DCTCP where this threshold is a function of the Bandwidth-Delay Product (BDP), our threshold is always set to only one packet, irrespective of the BDP.
- We utilize a similar method of echoing ECN (acks and delayed acks) as DCTCP. However, how sources react to ECN signals is governed by our new congestion controller.
- We do not let the queue grow, neither do we let the queue length oscillate a lot. We achieve this by using a more stable congestion controller that is based on the logistic growth function. This function has proven stability properties, and lets us attain fairness among flows irrespective of the RTT, which is not the case for TCP and DCTCP.

Congestion Controller Model

The LG function is described by the differential equation

$$\dot{N} = \frac{rN(t)(K - N(t))}{K}$$

In this function, *N* is the size of a population, *K* the so-called "carrying capacity" (the value that the equation converges to), and *r* the maximum per capita growth rate for a population (using the common terminology for logistic growth, which is most typically used to model growth of populations of species, e.g. in biology).

We consider the general Lotka-Volterra model of competition, where *S* species compete for a common limited resource according to the Logistic Growth equation

$$\dot{x_i} = x_i r_i \left(1 - \sum_{j=1}^{S} a_{ij} x_j\right)$$

r_i denotes the growth rate of species *i*. **A** = (a_{ij}) is called the community matrix where the value of a_{ij} determines the competitive effect of species $_i$ on species $_j$. We define a_{ii} = (2 *S*-1)/*S* and a_{ij} = (*S*-1)/*S*.

The equilibrium of the above system of equations is x_i = 1/*S*, and the system is globally stable (See Appendix [paper2] for a detailed discussion).

Logistic Growth Control (LGC)

We apply the above CC model to the context of a distributed congestion controller that operates at discrete time intervals.

$$x_i[n+1] = \\ x_i[n] r_i \left(1 - \frac{2S-1}{S} x_i[n] - \frac{S-1}{S} \sum_{j, j \neq i} x_j[n]\right) + x_i[n]$$

Figure 3. Growth control Equation (1)

However, in the equation in Figure 3, nodes need to know *S* which is not always applicable. Therefore, we use the following equation as the rate update rule:

$$x_i[n+1] = x_i[n] r_i \left(1 - x_i[n] - \hat{l}_i[n]\right) + x_i[n]$$

Figure 4. Growth control Equation (2)

In the equation in Figure 4, l̂ is equal to the percentage of ECN signals a source gets during the *n* th interval. We show that l̂ approximates (*S* - 1)/*S*, especially when the load is close to 100%.

We use the equation (in Figure 4) as a rate-based transmission mechanism. We also add exponentially-distributed inter-packet delay during pacing; due to the PASTA property (Poisson Arrivals See Time Averages), sources in the limit observe the same average congestion marking. This helps to improve LGC and obtain a more stable controller (congestion control policy in RINA).

The performance/convergence of LGC can be further improved by tuning the growth rate parameter, *r*; this allows to be more or less aggressive in changing the rate of a source that is getting a smaller or larger amount of resource than the fair share.

Network setup

Figure 5 shows how we suggest deploying RINA in Datacenters for this use-case: we use it as an overlay. In this case, the underlying network is IP, but we are able to exploit RINA benefits. We use LGC policies inside the hypervisor of servers. This means that all the flows in the "Inter-Server" DIF are congestion-controlled by LGC. The flows in the "Inter-VM/Tenant" DIFs are flow-controlled. All the flows benefit from the flow aggregation feature of RINA.

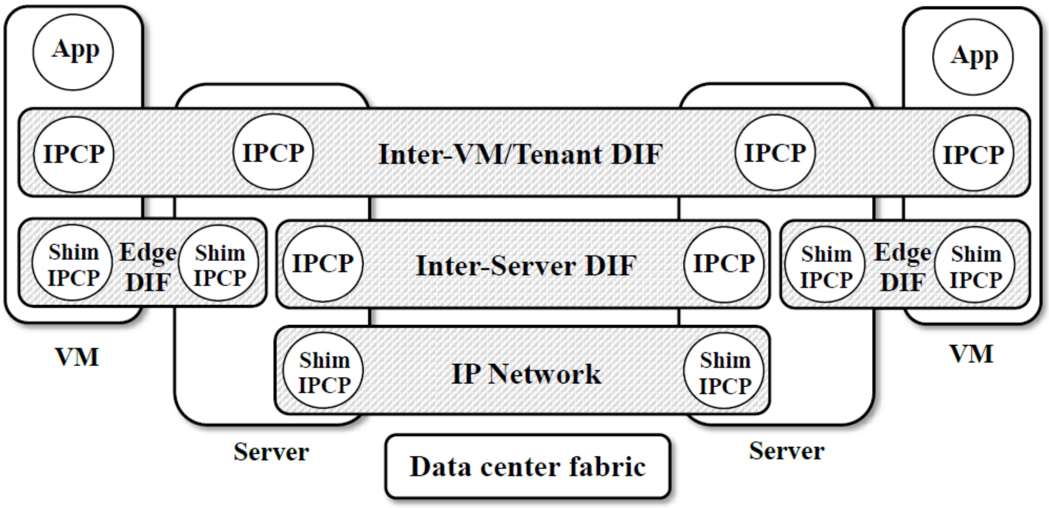


Figure 5. RINA as an overlay in a datacenter

Selected Performance Results

Focusing on the datacenter use case, in this section, we present the comparative performance of LGC with DCTCP. We chose DCTCP for comparison because it is a well-known transport protocol for datacenters, and has the same goals as LGC. We implemented both LGC and DCTCP in the INET framework of OMNeT++ to have a fair comparison.

We investigated the comparative performance of LGC with respect to DCTCP under two large-scale simulation scenarios: Clos and Leaf-Spine. In the leaf-spine topology, all the links between hosts and Top-of-Rack (TOR) switches are 10Gbps, and the other links have 40Gbps capacity. In the Clos topology, all the links in the network have the same capacity, i.e. 10Gbps.

Figure 6 illustrates the CDF of RTT in both scenarios for LGC and DCTCP. It clearly illustrates that LGC can reduce network queue sizes in both scenarios, resulting in significantly shorter delays and RTTs. We saw a comparable average throughput in these tests (see [LGm]).

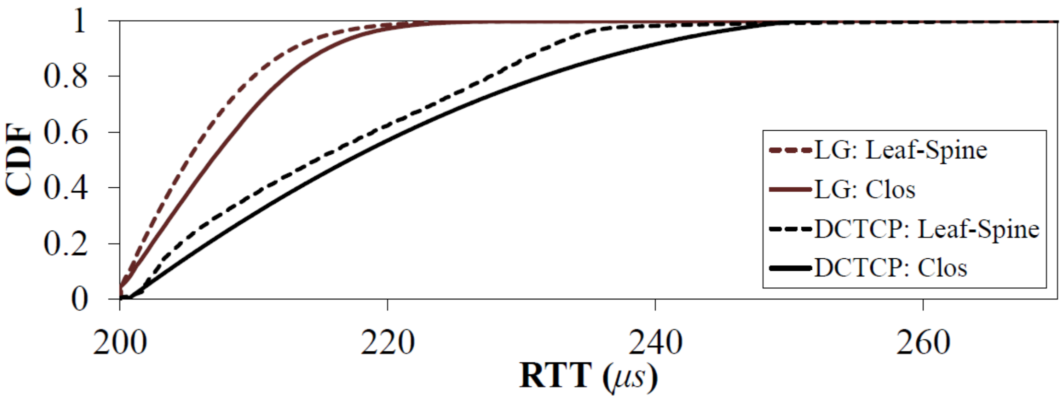


Figure 6. Cumulative distribution function of the RTT distribution under Clos and leaf-spine topologies

A Chain of Logistic Growth Controllers

An important aspect of RINA is that, depending on the number of underlying DIFs, a flow between a pair of nodes can be handled by several congestion-controlled loops. Therefore, the question is what happens if each loop of the chain is controlled by LGC; is this system stable? and how does it behave?

An interesting characteristic of the Lotka-Volterra model is that it can be used to model other types of interactions between species such as the predator-prey relationship. An extended model of the simple predator-prey is to have a chain of predators and preys: the first species is prey, consuming some limited resources, the second species is the predator for the first, but it is also prey for the third species, and so on. This illustrates a food chain model which is formalized by

$$
\begin{aligned}
\dot{x}_1 &= x_1 r_1 (1 - a_{11} x_1 - a_{12} x_2), \\
\dot{x}_2 &= x_2 r_2 (-1 + a_{21} x_1 - a_{22} x_2 - a_{23} x_3), \\
&\vdots \\
\dot{x}_j &= x_j r_j (-1 + a_{j,j-1} x_{j-1} - a_{jj} x_j - a_{j,j+1} x_{j+1}), \\
&\vdots \\
\dot{x}_n &= x_n r_n (-1 + a_{n,n-1} x_{n-1} - a_{nn} x_n).
\end{aligned}
$$

Figure 7. A predator-prey relationship between LGCs

In the final phase of the project and in work package 7, we will work on this model in the form of a scientific paper.

2.1.4. Conclusion and Future Work

In this section, we showed that in RINA, the natural way of controlling congestion is very different, where control is executed closer to the problem location. We applied a simple TCP-like policy (Aggregate Congestion Control, ACC) in a number of DIF configurations, finding that several benefits appear as a by-product of DIF configuration. In particular, aggregation of flows can yield a large benefit.

We also developed a new congestion controller policy called Logistic Growth Control (LGC), for RINA. As the first step of evaluating this policy, we implemented it in the INET framework of OMNeT++, and compared it with DCTCP. We also analyzed LGC analytically using a fluid model to investigate its stability and accuracy. Our results show that communication latency in a datacenter is greatly improved by LGC, and also that LGC achieves much better fairness between flows than DCTCP. LGC is a promising candidate for more complex control scenarios where multiple congestion controls are nested.

As future work and in the context of wok packages 6 and 7, we are further evaluating LGC in a whole RINA network (modeled as a food chain). Results will be presented in the form of a scientific paper.

2.2. Recursive Congestion Control (RCC)

2.2.1. Introduction

In RINA multiple DIF layers carry out congestion control. These DIFs can be stacked in arbitrary ways and provide more ways to use feedback than before (which of the many controllers along an end-to-end path should be notified?). This in turn raises concerns regarding stability and performance of such a system of interacting congestion control mechanisms. Our paper attached in [paper3] reports on a first analysis of feedback methods in recursive networks. In this section we give a summary of the work presented in the paper, referring the reader to [paper3] for the full details.

2.2.2. Network Model

To investigate these congestion control interactions we build a model representative of a RINA topology. Too simple a model will lack the complex control loop interactions that we wish to investigate, and too complex a model will not be solvable. Figure 8 shows the topology we use with a fluid type model. Details of the model can be found in [paper3], with experiments conducted using MATLAB SIMULINK [Simulink].

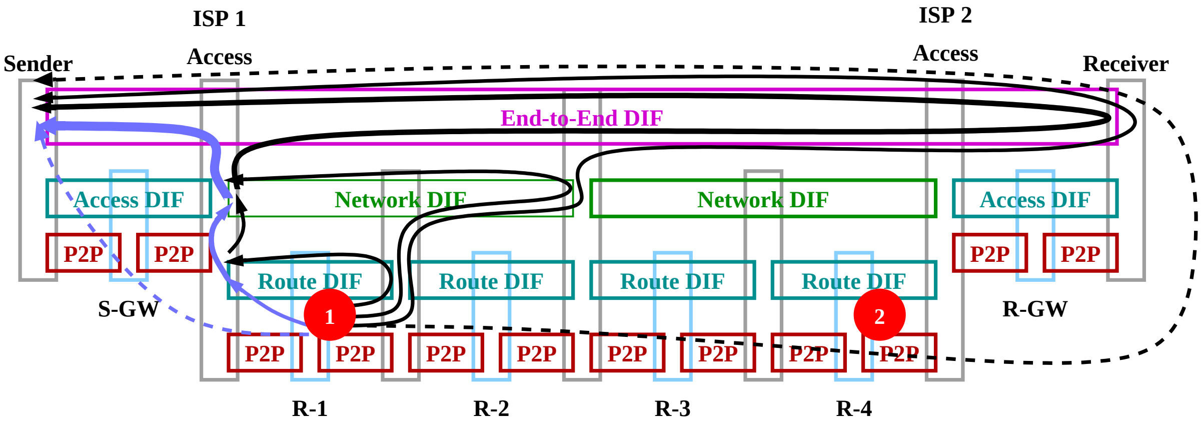


Figure 8. Feedback mechanisms being investigated

We contrast the performance of two well-known non-recursive type feedback mechanisms: (i) End-to-end forward Explicit Congestion Notification ECN like feedback [1]; (dark dashed arrow), and (ii) backward ECN (BECN) [1] like feedback (light blue dashed arrow); with three possible

[1] In our experiments congestion is a measure of queueing above a threshold, so a richer signal than standard Internet ECN.

recursive congestion feedback mechanisms: (i) Recursive forward feedback (light blue arrows), (ii) Recursive backward feedback (dark arrows), and (iii) Recursive pure push-back feedback. Pure push-back feedback follows a similar path to (iii), but only carries congestion information from the layer below, since in this model each DIF can only infer congestion from queue growth related to congestion in the DIF below. The non-recursive feedback mechanisms are tested without intermediate DIFs, while the recursive mechanism makes full use of the recursive architecture.

Apart from traffic flowing end-to-end, we inject an average of 5% random cross traffic at each or the router nodes (x-GW and R-n). Then, to investigate the effect sudden disturbances have on the stability of the system, we introduce a 50s wide pulse of cross traffic at 40% of capacity at locations identified with a circled 1 and 2.

2.2.3. Results

Our results indicate that congestion control with recursive feedback may require more buffering between end-points than end-to-end congestion control. Whether this significantly affects end-to-end latency is being further investigated.

We observe that the performance of recursive forward and recursive backward feedback has similar performance to that of the non-recursive feedback mechanisms in terms of efficiency and stability [2]. We ran experiments keeping the sender's control gains constant and changing the underlying DIF control gains, but did not observe any significant improvements when doing this (see Figure 9 for a comparison of the recursive pure push-back and backward feedback methods). Future research will look at how well the recursive architecture may perform if each DIF's congestion control was tailored for its place in the topology and individual optimised for the conditions it encounters.

[2] These comparative experiments used identical control gains for the sender and all DIFs

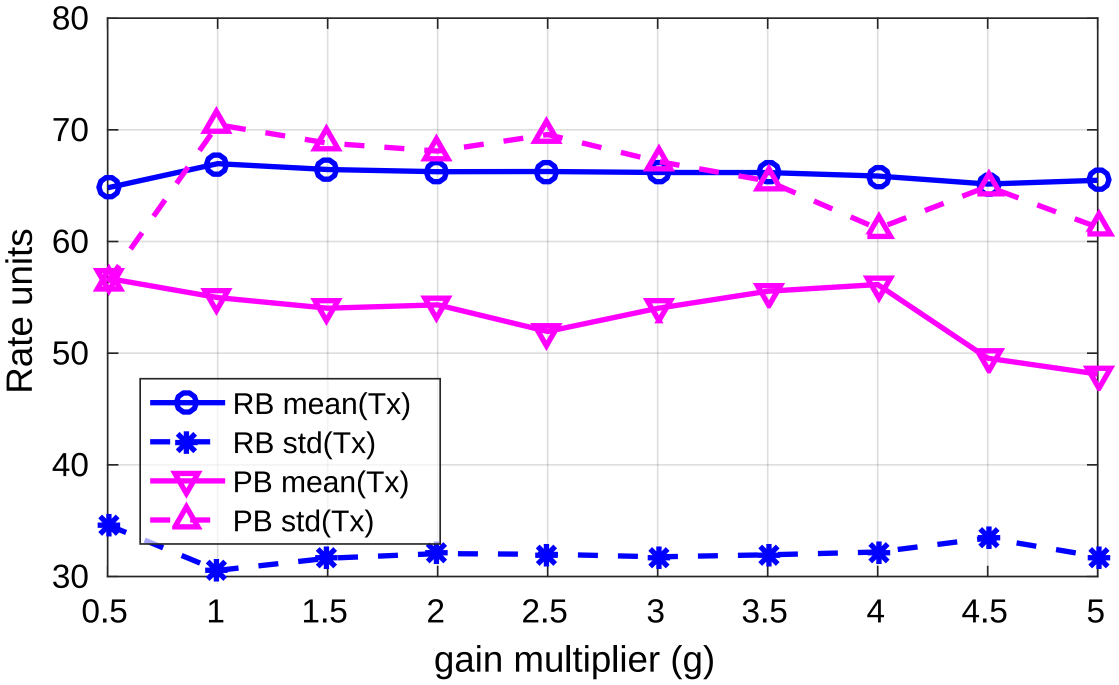


Figure 9. Comparison of the stability of the recursive backward feedback mechanism (RB) and the pure push-back mechanism (PB) when the control gains of all DIFs except the sender are varied

Of all the 5 feedback mechanisms investigated, the pure push-back mechanism performed the worst. We explored manually tuning this to obtain better results, but to no avail — except in a topology half the size and with one less layer, where some limited improvement in the very poor performance was obtained. Figure 9 illustrates the relative poor performance of the recursive pure push-back mechanism with that of the backward feedback mechanism as the control gains of all the DIFs are adjusted relative to the sender DIF. Results show the average sender rate (mean(Tx)) and the standard deviation of Tx taking into account perturbations which we produced by varying the increase/decrease gains of the congestion controllers in the lower DIFs using a multiplier.

2.2.4. Conclusions on RCC

The recursive pure push-back mechanism performs badly. This is due to two key characteristics: the delays in the signal being pushed back through the layers, and also the fact that DIF congestion controllers — and ultimately the sending sources — are unable to respond to the full extent of congestion since it is hidden from them. We find that the recursive backward feedback mechanism is the best of the three recursive mechanisms tested. This mechanism can be challenging to implement in a recursive architecture, where it may be difficult for a DIF (working on a certain aggregate of end-to-end flows) to send congestion information back to the appropriate higher DIFs and ultimately the sending sources. Recursive forward feedback can be easier to implement since feedback can follow the path of the packet up all of the layers, and performs almost as well in these tests.

Our work presented in [paper3] takes the first steps at understanding the dynamics of congestion control in recursive architectures. There are great benefits in recursive architectures, but simply bringing non-recursive control into a recursive environment is not enough. For future research in this area which will be performed in the context of WP7 we consider investigating multiple loop control theory, such as back-stepping control, as a means for developing stable efficient recursive congestion controls for RINA.

2.3. Performance isolation in multi-tenant data centers

2.3.1. Introduction

The volume at which data is now being generated or must be computed in modern data centers is constantly increasing motivating the demand for new solutions and architectures for coping with such trend. RINA provides a clean, ready to use, architecture which allows DC administration to scale up/down the dimension of their computational pool in a painless way. In addition to that, RINA also offers a way to customize the behavior of various aspects of the network (including congestion control) through the introduction of policy mechanisms.

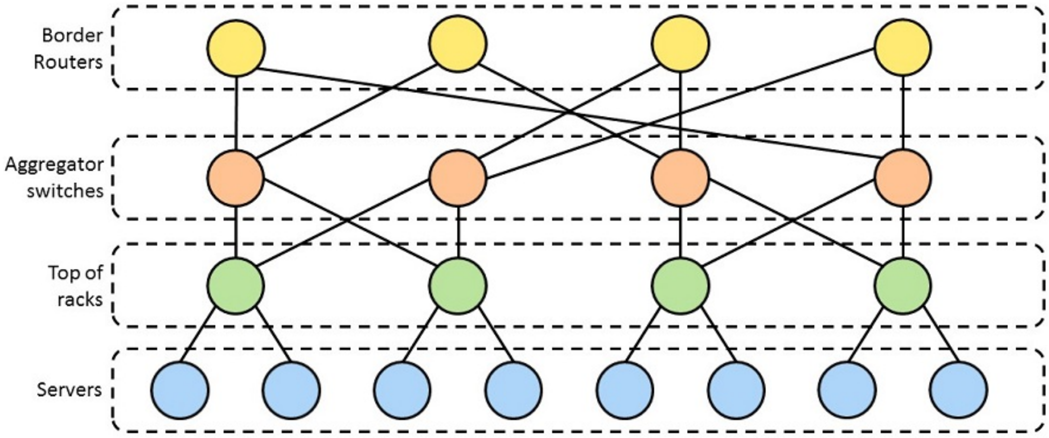


Figure 10. DC Organization

In this section, we will consider the case of a DC organized in fat-tree topology (see Figure 10) and we will enforce admission control on traffic. Note that the results presented in this section will refer purely to the congestion control mechanisms while admission control will be enforced by making sure that resources are never overbooked.

Another interesting point is that a server can generate traffic at any given time, but its overall networking resources (in terms of used bandwidth) can be under-utilized for small or even long periods. See deliverable 3.2, chapter 'Policies for performance isolation in multi-tenant data centres', for more information on the motivation of using this congestion control strategy inside datacenters (also see [EyeQ], [Riggio]). RINA is chosen as the technology to enforce such policies thanks to the scalability it offers and the possibility to introduce such behavior (using the provided SDK) by design without the necessity of complex hacks in the network stack.

2.3.2. Multi tenancy organization in DC network

During the experiments, we assume that the DC network is RINA compatible: this means that all the nodes (Servers, TOR, Aggregator switches and Core Routers) in the network communicate using RINA. A graphical representation of the setup used throughout this section is reported in Figure 11. As you can see several shim DIFs are used in order to build link layer connectivity. Then a DC fabric DIF is deployed across all the Shim DIFs. Finally, each tenant owns a tenant DIF, which provides a clean and transparent separation (see Figure 12).

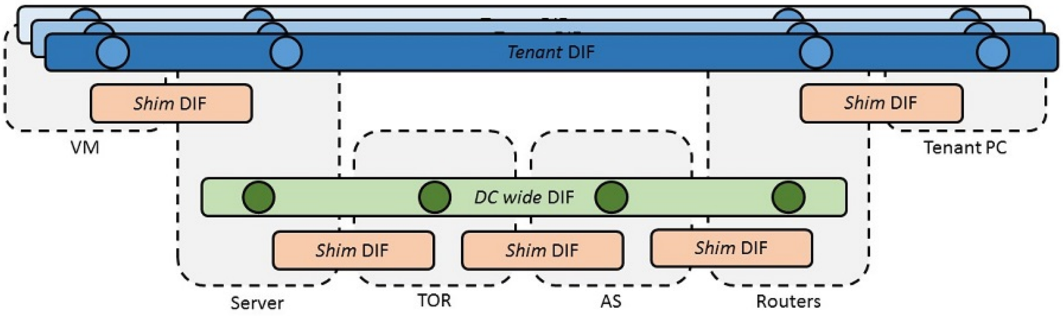


Figure 11. RINA layer organization within the DC.

Every Tenant DIF will be then competing with each other for the networking resources. Notice than upon creation of a Tenant DIF a Minimum Granted Bandwidth (MGB) parameter is used in order to specify the minimum amount of network resource capacity that should be allocated to that specified tenant. Due to the full-bisection bandwidth, it is sufficient to ensure that the sum of all MGBs assigned to each tenant does not exceed the physical bandwidth of the link between a server and the ToR switch. Note that a tenant request consists of a set of VMs and the associated MGB for each VM (we assume that all VMs use the same MGB).

A set of policies deployed at the DC DIF ensures that tenants will receive their MGB and will be able to exploit unused network resources (work preserving scheduling). Bandwidth allocation will be reset to the original MGB when needed, e.g. when new VMs join the network.

It is worth stressing that during the DIF creation phase, when a tenant requires some resources, the DC administrator must be sure that no over-subscription takes place in the node assigned to the new tenant. The sum of MGB for a node must always be smaller than or equal to the link capacity which is connecting the server to the Top of Rack switch.

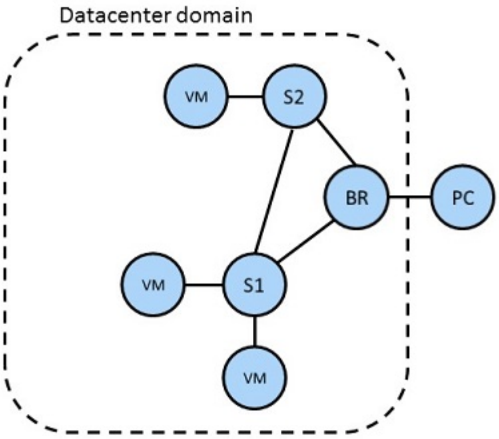


Figure 12. Network perceived by the tenant.

2.3.3. Congestion Detection

Congestion can occur at every node in the network, and it happens when the incoming traffic goes over the node’s capacity to forward it towards its destination (see Figure 13). This can happen due to sub-optimal routing decisions or due to an excess of bandwidth usage by one or more sources. When congestion is about to occur in any of the nodes of the network, we must be able to detect the congestion and react in some way in order to solve it.

We perform this operation by introducing a custom RMT policy (at DC-DIF level) which monitors the status of IPC Process queues. If, for any reason, PDUs begins to accumulate and exceed a certain threshold, then the policy starts to mark those PDUs using ECN flags. Such PDUs will then carry this information along the remaining path, triggering any other Congestion Control mechanism introduced in the IPC Processes along the path.

The lower the threshold, the lower the reaction time. Ideally, a threshold of just one or two PDUs should be used. However, we found out that due to the maturity of the IRATI codebase, setting a low value for this threshold can instead create instability. We traced back this behavior to the fact that the underlying stack does not send the pending PDUs as soon as the hit the RMT queue. Instead, PDUs undergo a small buffering that result in a bursting behavior at the wire level. This in time can lead to PDUs being ECN marked even when there is not the risk of congestion.

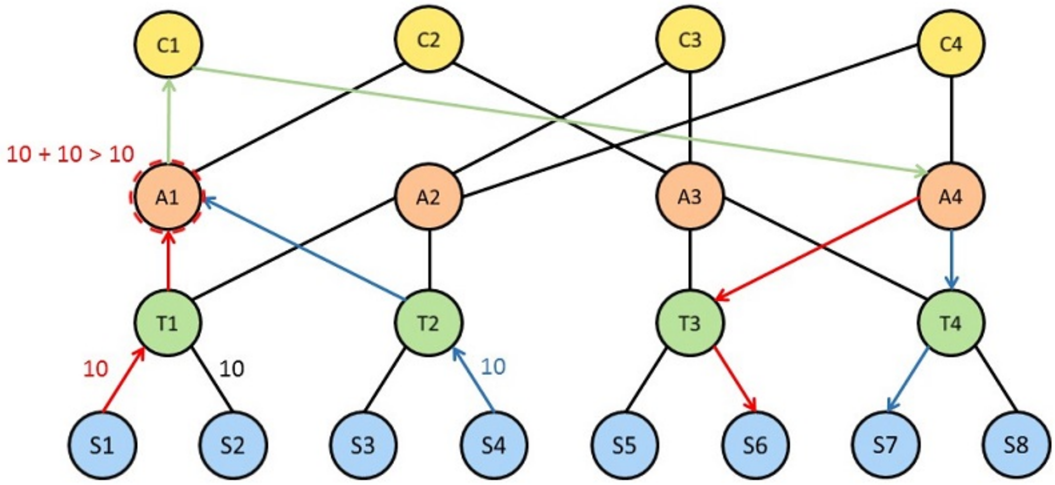


Figure 13. Two flows (red and blue) routed on the same path will cause congestion at A1. From that point on the PDU proceeding towards their destination will carry the ECN mark, which will cause the flow to react to the congestion in order to mitigate it. The green arrow indicates where the flows share the path.

2.3.4. Congestion reaction

Now that we have a mechanism which allows us to detect and mark PDUs when congestion is about to occur in the DC, what we need is something which actually reacts to this event and resolves the congestion.

Since one of the restrictions in the tenant DIF creation is to avoid oversubscription of bandwidth (a server cannot have tenants whose MGB sum exceeds the link capacity) and considered that we are using a DC topology with full-bisection bandwidth, we simply need to reset the bandwidth of the tenants with ECN marked frame to the original MGB in order to resolve congestion.

Note that we need to do this because we want (by design) to re-assign the unused bandwidth to the currently active tenants (work preserving scheduling). Otherwise, simply the fact of using a full bisection bandwidth and strict admission control would ensure the absence of congestion.

It is worth noticing also that the approach proposed in this section could also be applied to DC topologies with high bisection bandwidth (although not full bisection bandwidth). In such cases even strict admission control would not ensure the absence of congestion.

The rate control feature is implemented by introducing a custom policy in DTCP. Such policy is in charge of reacting to incoming ECN-marked PDUs and take an action in order to resolve it.

2.3.5. Flow rate enforcement

Now that the DC wide DIF can detect congestion and react to it, we need an additional mechanism which provides Congestion Reaction policies with the necessary data to understand what the Minimum Granted Bandwidth is. This is done by introducing an additional policy called Flow Allocation Rate Enforcement (FARE) to the Flow Allocation module in the RINA stack.

The FARE policy translates the average bandwidth field of the requested Tenant QoS cube into the actual sending rate property of the data transfer protocol (DTP) instance. When the DTP instance is created, the congestion reaction policy saves this value as the requested MGB. The MGB is the value to which the rate shall be rest when an ECN marked PDU is received (see Figure 14 to see MGB allocation with layering).

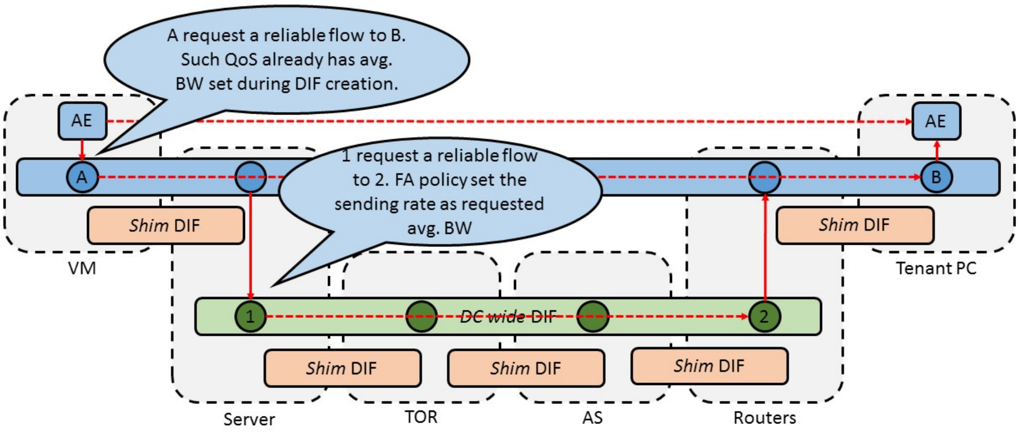

A request a reliable flow to B. Such QoS already has avg. BW set during DIF creation.
AE
AE
A
B
1 request a reliable flow to 2. FA policy set the sending rate as requested avg. BW
Shim DIF
Shim DIF
VM
Tenant PC
1
DC wide DIF
2
Shim DIF
Shim DIF
Shim DIF
Server
TOR
AS
Routers

Figure 14. How the MGB is ensured to tenants at DC-DIF level flows.

2.3.6. Route Selection

We proposed a mechanism to detect congestion, to react to it and to assign a quota of granted bandwidth to different tenants. The last step missing is taking advantage of the Datacenter network topology in order to make the most out of the full bisection bandwidth topology. We will do this by using an Equal-Cost Multi-Path routing policy, which allows the forwarding layer to be fully aware of the multiple paths available between a pair of servers.

Various forwarding strategies can be chosen in order to deliver the PDUs to their final destination within the datacenter. We will mainly focus on three different strategies(see Figure 15), which are:

- **Static path forwarding**. This policy chooses the path to take using a hash of what we call connection identifier, which contains the addresses of the destination, the QoS used and the connection endpoint ids. This means that regardless of how much time passes, a PDU with a certain connection profile (same source to the same destination, same QoS, same flow ids) will always go through the same outgoing port.
- **Flow weight forwarding**. This policy uses a greedy approach always selecting the least loaded outgoing port. Once the choice has been made, it will be remembered for the next PDUs in the flow. The selection will be considered expired after a configurable amount of time. When such timer has expired a new outgoing port will be selected.

- **Random pick forwarding**. This policy randomly select the outgoing port from the list of valid ports for each PDU.

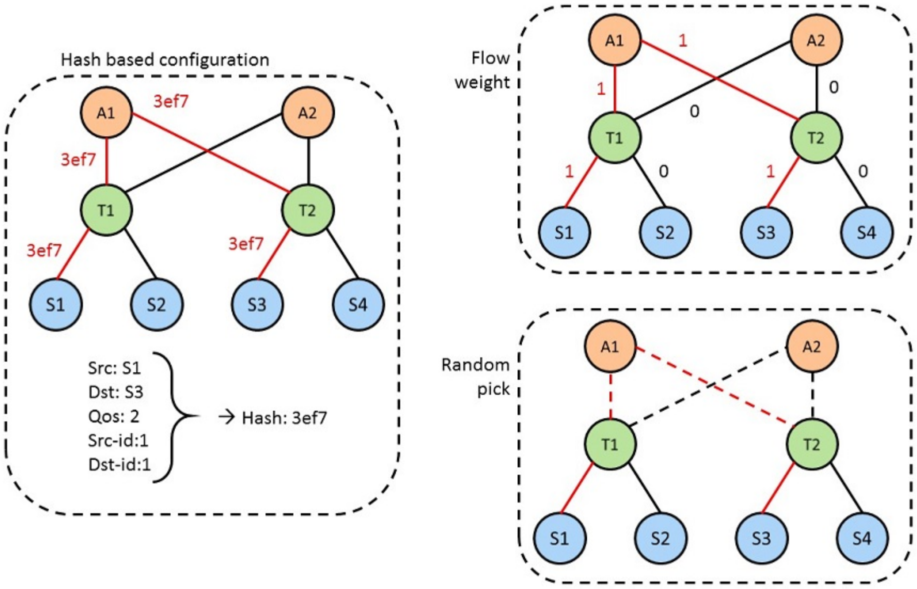


Figure 15. How different forwarding strategies handle the same flow request: hash based strategy will always route on the same port; flow weight will always choose the least loaded port; random pick will just select a random port selected from the valid ones.

2.3.7. Experimental results

Here we report on a test campaign performed using all the congestion control policies presented in this section. The policies have been implemented using the PRISTINE SDK. Measurements have been carried out over the Virtual Wall deployed at iMinds premises in Ghent. Figure 16 shows the experimental setup of the testbed. The maximum link capacity for all the links is set to 95 Mb/s.

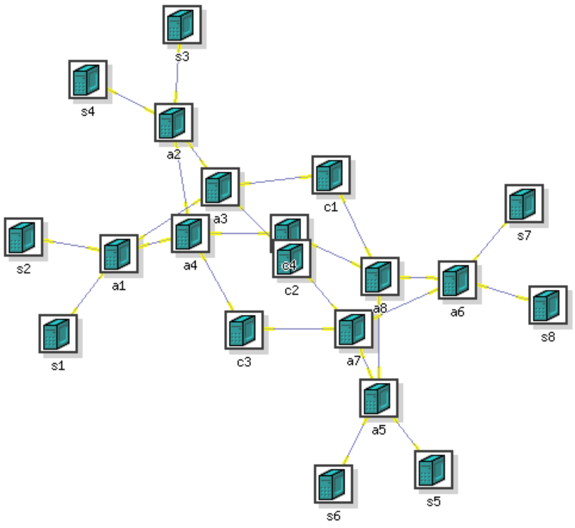


Figure 16. Configuration of nodes used in the Virtual Wall testbed.

Two PODs (sets of fat-tree like interconnected switches named 'a*') are connected by four core routers (named 'c*'). Eight servers (named 's*') provide the end points of the communication. In the following experiment, two tenants establish one flow each, from different source nodes to the same server node. The test takes place in a k=4 fat-tree shaped topology with 20 nodes arranged as two PODs (with two aggregator switches and two top of rack switches), four core routers and eight server machines.

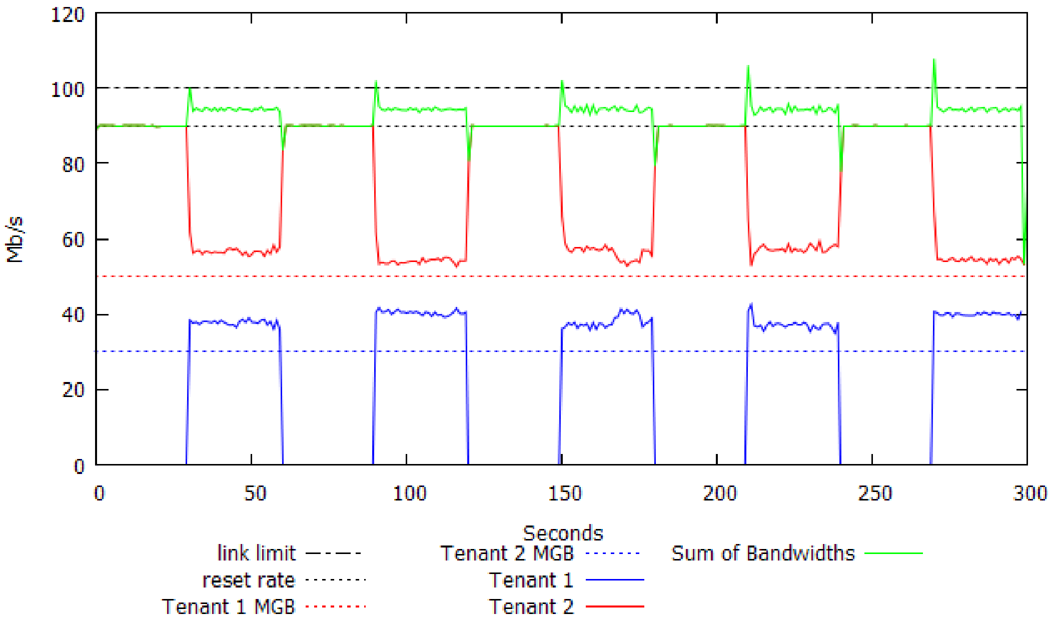


Figure 17. Experimental results. The continuous red and blue lines are tenants' instantaneous used bandwidths in Mb/s over time (seconds).

The red tenant requested a 50 Mb/s MGB while the blue tenant requested an MGB of 30 Mb/s. The link limit is set to 90 Mb/s, so any traffic which exceeds this bandwidth will create congestion in the network. At startup only the red tenant is active, and the DTCP gives it access to the whole unused bandwidth, allowing it to reach the full link speed. Every 30 seconds the blue tenant start a communication competing for the resource with the red tenant.

Since this could actually cause congestion (detected by the RMT monitor), the DTCP policy starts to react to ECN marked PDUs and starts to tune the flows down to a lower rate, following the previously explained strategy. Both the tenants will have their MGB respected, but the sum of both rates will be 80 Mb/s, leaving 10 Mb/s unused.

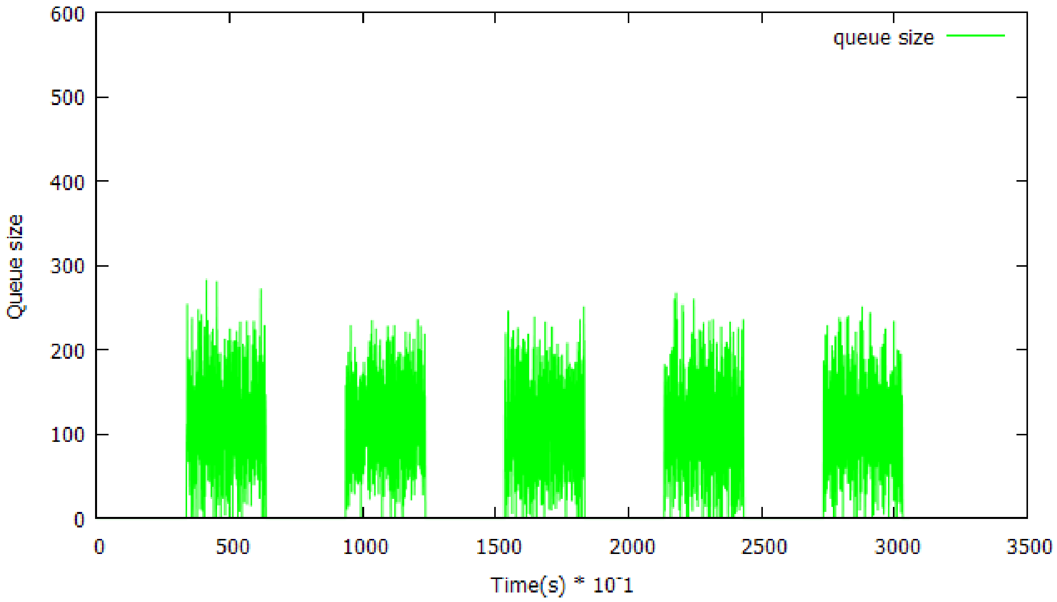


Figure 18. The state of the queue during the experiment is shown in green. The X axis shows the time of the experiment (in deciseconds) while the Y axis shows the number of PDUs in the queue.

Figure 18 shows the state of the queues in the RMT module of the IPCP. During periods when the blue tenant is not generating traffic, no congestion is detected on the node which aggregates the flow to the common destination node. When the blue tenant becomes active, PDUs start to accumulate in the queues. Very soon the number of queued PDUs will exceed the assigned threshold, and this causes PDUs to be marked with the ECN flag.

This triggers the DTCP congestion control mechanism, which lowers the rate allowing the PDUs in the queue to be consumed (the queue length arrives at 0). When the blue tenant finishes its activity, only one flow remains active in the network. As a result, this flow will be granted again full line rate.

Another preliminary test run on the same topology involved five tenants which communicate between the nodes belonging to their network with different rates and different timings. Tenants 1 and 2 have continuous communication between their server and client instances, while the remaining three tenants produce intermittent traffic (every 20 seconds for tenant 3, 25 for tenant 4 and 10 for tenant 5). Tenants 2 and 3 share the same nodes as client and server instances, while clients 4 and 5 have different client nodes but use the same server node. Tenant 1 does not share any of his nodes with other tenants.

As Figure 19 shows, tenants 1, 2, and 3 share the same path as their bandwidth utilization sum never exceeds the 95 Mb/s, which is the maximum link capacity. Tenants 4 and 5 are instead redirected to another link and they end up having a very high bandwidth when only one is active, but when both are transmitting the congestion control mechanism ends up shaping their traffic towards the minimum granted bandwidth.

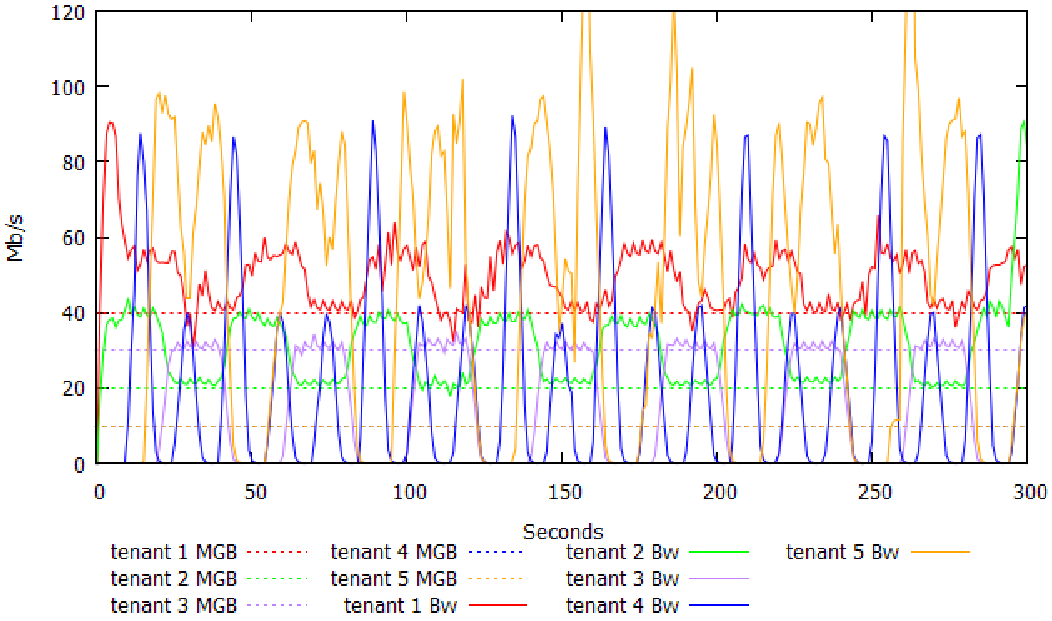


Figure 19. Congestion control mechanism when multiple tenants are active..

Figure 20 shows the queue status in the nodes where congestion is detected, which were aggregator switches a7 and a2. Aggregator switch a7 ends up reporting congestion for the first 3 tenants which were using the server nodes immediately under it (servers s5 and s6), while aggregator switch a2 was reporting the congestion for the last 2 tenants which were located in server s3 and s4.

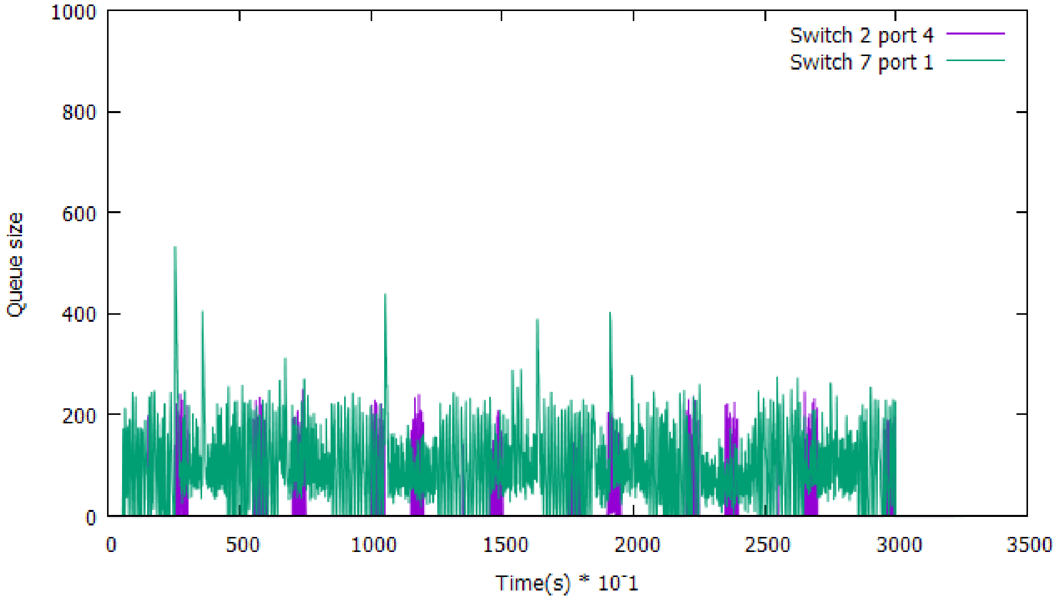


Figure 20. Status of the queues in the two congested switches during the experiment. The X axis shows the time of the experiment (in deciseconds), while the Y-axis shows the number of PDUs present in the queue.

2.3.8. Conclusions

In this section, we reported on an extended measurement campaign performed in order to validate in realistic setting the congestion control policies implemented using the PRISTINE SDK. The results of the evaluation are indeed promising.

In particular, the regular nature of the RINA architecture allows to deeply customize the behavior of the network using a few well-written policies. For example with regard to the experiments described in this section, all the policies account for less than 1000 lines of code. This includes error handling and logging but excludes network management code. On the other hand, the solution EyeQ [EyeQ] consists of more than 10000 lines of code that are distributed across the Linux networking stack. Moreover, due the invasive nature of the EyeQ solution, the actual code is very hard to maintan and distribute as a dynamic module. On the other hand, the policies described in this section can be distributed independently from the IRATI stack and combined with other policies.

3. Resource Allocation

3.1. Traffic differentiation via delay-loss scheduling policies

3.1.1. Introduction and motivation

This section describes the work carried out to accommodate the concept of DeltaQ in RINA, and how this makes RINA networks capable of providing a resource allocation framework. This framework is required to satisfy the QoS requirements established in the use cases reported in D2.1 regarding the ability of DIFs to provide flows with a given bandwidth and upper bounds on delay and loss.

In a statistically-multiplexed network, contention for shared resources is inevitable, and managing this contention is essential in order to provide consistent levels of service in terms of loss and delay for data flows. Congestion control mechanisms reported in section 2 manage the overall level of resource contention on the time-scale of end-to-end communication across a DIF. However, this cannot prevent transient periods of contention at intermediate points because of the statistical nature of the traffic. Such transient contention can produce large short-term variability in the performance of data flows. To deal with this, we apply the principle of delay-loss multiplexing within the RMT function at the DIF/IPCP, in a component called the QTAMux.

The QTAMux includes a number of elements such as C/U Mux, Policer/Shapers… The C/U Mux enables fine-grained control of the relative performance of multiple traffic classes. Policer/Shapers shapes the demand on each particular DeltaQ-quality-class to match predetermined limits, ensuring that the dynamic operation of the C/U Mux remains within its predictable region of operation. This allows absolute rather than just relative performance bounds to be delivered. For this to be effective, however, it requires information about the demand and required loss and delay targets for each class. In RINA, this information is consistently available due to the specification of a QoS Cube for each traffic flow. By using the QTAMux, we have both sufficient control and sufficient information to ensure strong bounds on the end-to-end loss and delay of data flows, including the statistical risk of not achieving such bounds due to both demand patterns and the overbooking policy of elements of the end-to-end path. Overbooking is economically necessary in many real-world use-case scenarios. In RINA, each layer knows the expectations of the applications using that layer, and the quality of service that it can expect from the layers below through recursive use of the QTAMux. Thus, the aggregated demand, the delivered performance and the statistical risk of exceeding selected bounds are composable both vertically, between layers, and horizontally within a DIF.

While it is of course possible to perform some differential treatment of packets in switches/routers, for solutions like Multi-Protocol Label Switching (MPLS)-based Virtual Private Networks (VPNs), these offer at best some minimum average rate with relative QoS differentiation, but without any control over the performance hazard implicit in overbooking. In such systems, large transient variations in the delivered performance can be minimised only by vastly under-booking the network resources, which can only be achieved in certain circumstances. By contrast, RINA using the QTAMux allows applications to request and receive communication services with strongly bounded loss and delay, while fully utilising the most constrained resources.

Thus, by building upon the work reported in deliverable D3.2 and exploiting the programmable nature of RINA, the DeltaQ concept has been translated into policies for the RMT in the two different RINA environments available: the RINA Simulator (RINAsim) and the IRATI stack. The translation has given rise to the QTAMux component, which is the subject of the following section.

3.1.2. QTAMux system description and adaptation to the RINA environment

The Cherish/Urgency (C/U) matrix, and the simple multiplexing scheme that were introduced in deliverable D3.2, have been enhanced with the addition of the Policer/Shaper (P/S). Together, the C/U multiplexer and the P/S component form the QTAMux.

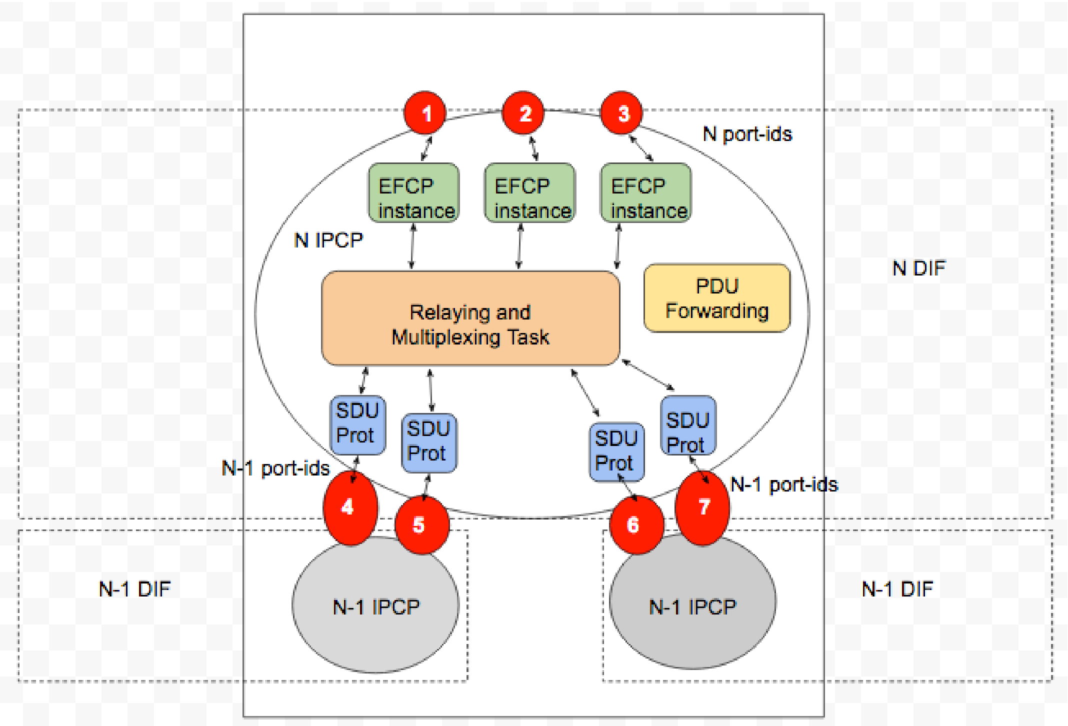


Figure 21. General structure of the data transfer parts of an IPC Process

Figure 21 shows an overview of the internal organization of the IPC Process. The most relevant component of the IPCP for the QTAMux system integration is the Relaying and Multiplexing Task (RMT). The RMT multiplexes data from several EFCP connections and incoming N-1 flows over outgoing N-1 flows. Figure 22 provides a detailed view of the flow of PDUs through the RMT:

- The RMT inspects incoming PDUs from N-1 flows and checks if they are addressed to the IPC Process. If so, PDUs are delivered to the relevant EFCP instance.
- If not, the RMT checks the PDU Forwarding table, obtains one or more N-1 ports to forward the PDU and places them in the port's output queues (outgoing PDUs from EFCP connections also follow this path).
- PDUs in output queues are scheduled according to their QoS-id and some other criteria. The QTAMux offers a way to implement this QoS-cube-based PDU scheduling for an N-1 port.

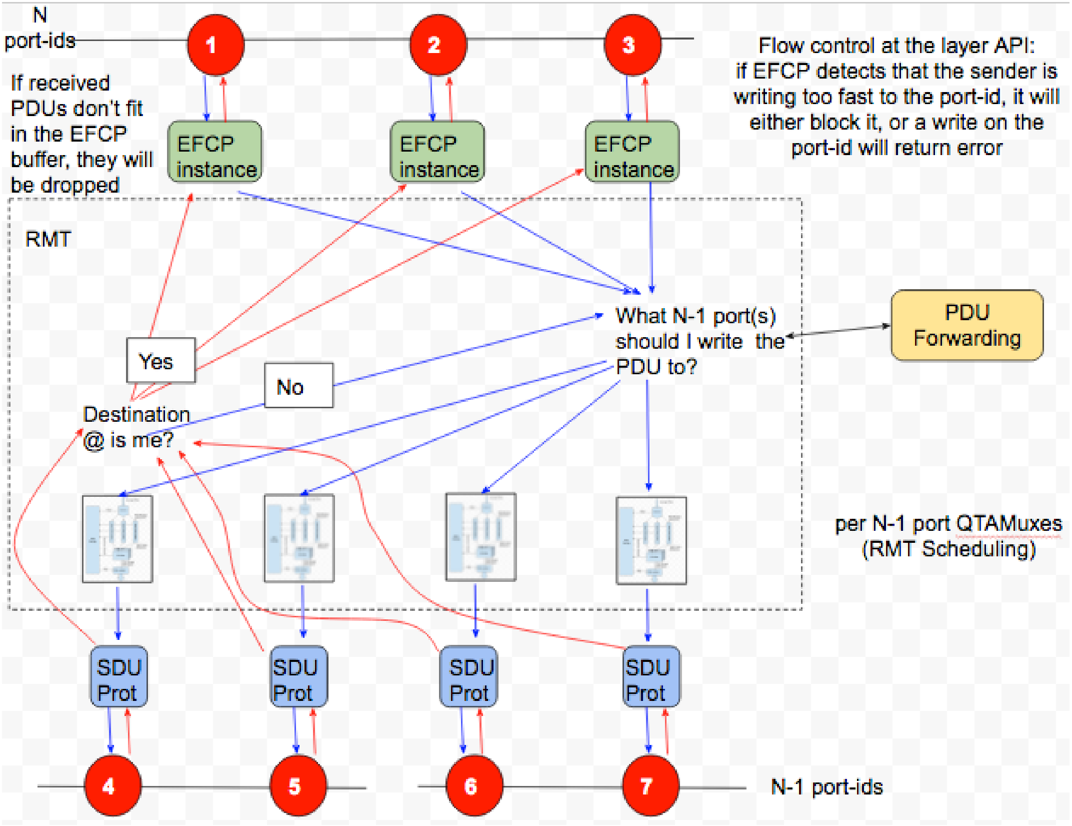

N
port-ids
1
2
3
Flow control at the layer API: if EFCP detects that the sender is writing too fast to the port-id, it will either block it, or a write on the port-id will return error
If received PDUs don't fit in the EFCP buffer, they will be dropped
EFCP instance
EFCP instance
EFCP instance
RMT
What N-1 port(s) should I write the PDU to?
PDU Forwarding
Yes
No
Destination @ is me?
per N-1 port QTAMuxes (RMT Scheduling)
SDU Prot
SDU Prot
SDU Prot
SDU Prot
4
5
6
7
N-1 port-ids

Figure 22. Group of QTA Muxes within the IPC Process

Operation of the QTAMux in RINA

In RINA, each arriving PDU belongs to one of the QoS-Cubes supported by the DIF. The QTAMux provides a mechanism to trade DeltaQ between flows of different QoS-Cubes towards a particular N-1 port. The QTAMux has three principal elements, as shown in Figure 25:

1. The Cherish-Urgency Multiplexer (C/U Mux): this trades DeltaQ (delay and loss) between the (instantaneous) set of competing streams, using loss to maintain schedulability.
2. The policer/shapers (P/S): these trade overall DeltaQ against loading factor, performing intra-stream contention for traffic in the same treatment class.
3. The stream queue: this ensures in-sequence delivery while allowing the delivered quality level to be varied dynamically.

Both the C/U Mux and the P/Ss perform matching between demand and supply - they differ in the way that they express the trades within the inherent two degrees of freedom. Note that the interaction of the QTA mux with ECN marking (both the creation of CN marks and its response to them) is for further study.

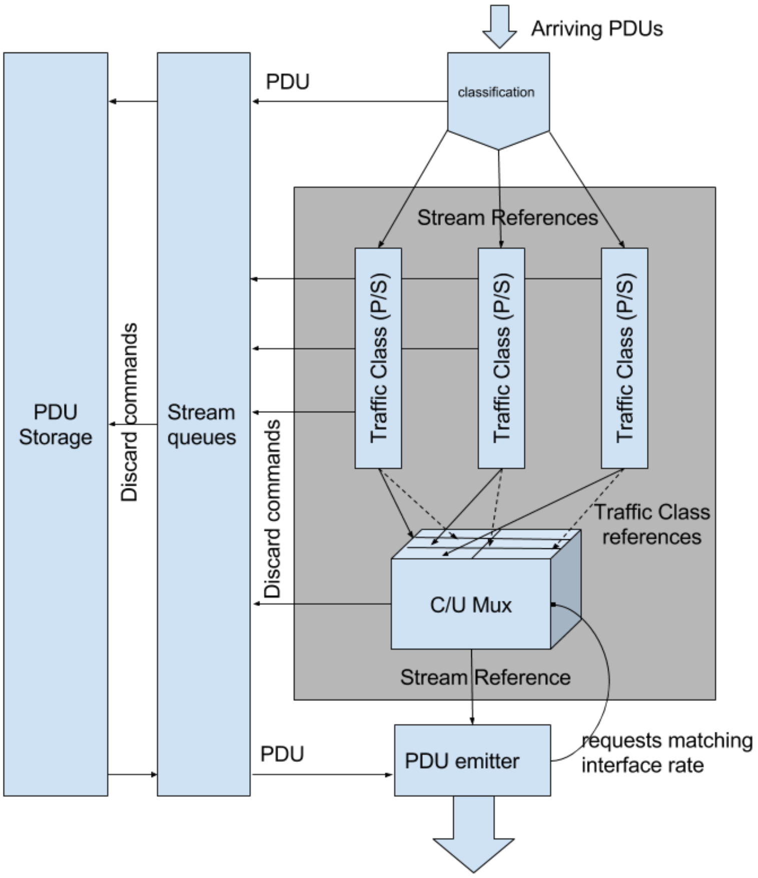


Figure 23. QTAMux block diagram

Operation of the C/U Mux

The Cherish-Urgency Multiplexer can be seen as performing inter-stream contention. Its operation is relatively straightforward to describe. Each incoming packet stream is assumed to have a pre-assigned cherish and urgency levels, derived from the PDU associations: these correspond to the stream's quality requirements in terms of loss and delay. Contention between streams for access to the outgoing port is managed by admitting packets based on their cherish classification level and servicing them based on their urgency classification level. Thus, a packet's cherish level determines its probability of loss while the urgency level determines its probable delay. Since the cherish and urgency classifications are independent of one another, the quality assigned to different streams is not limited to a traditional "priority" ordering scheme; instead, a two-dimensional classification is achieved, as illustrated in Figure 24.

Conventionally, lower-numbered cherish levels have the lower loss, and lower-numbered urgency levels have the lower delay.

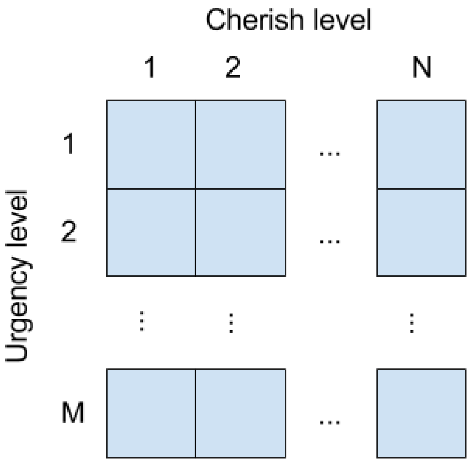


Figure 24. Two-dimensional cherish/urgency classification

[Holyer] includes examples to show how the greater control afforded by the two-dimensional classification allows the quality assigned to a data stream to be tailored to its particular loss and delay requirements. Admission control is realised by partitioning the buffer capacity of the C/U Mux and associating each partition with a subset of the cherish levels, as shown in Figure 25:

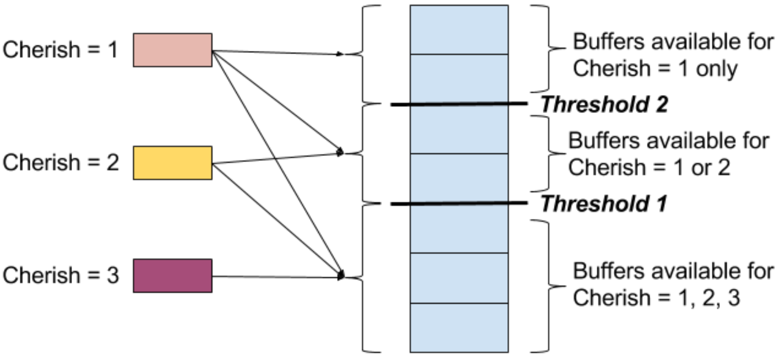

Cherish = 1
Cherish = 2
Cherish = 3
Buffers available for Cherish = 1 only
Threshold 2
Buffers available for Cherish = 1 or 2
Threshold 1
Buffers available for Cherish = 1, 2, 3

Figure 25. Partitioning of buffer space by cherish levels

A packet is admitted only if there is spare capacity in one of the partitions associated with its cherish level, as shown in Figure 26. All partitions are available to the most highly cherished packets, while packets having a lower cherish level are limited to smaller subsets of partitions. This means that highly cherished packets have a greater probability of finding available buffer resources than those having a lower cherish level, and in turn experience less loss.

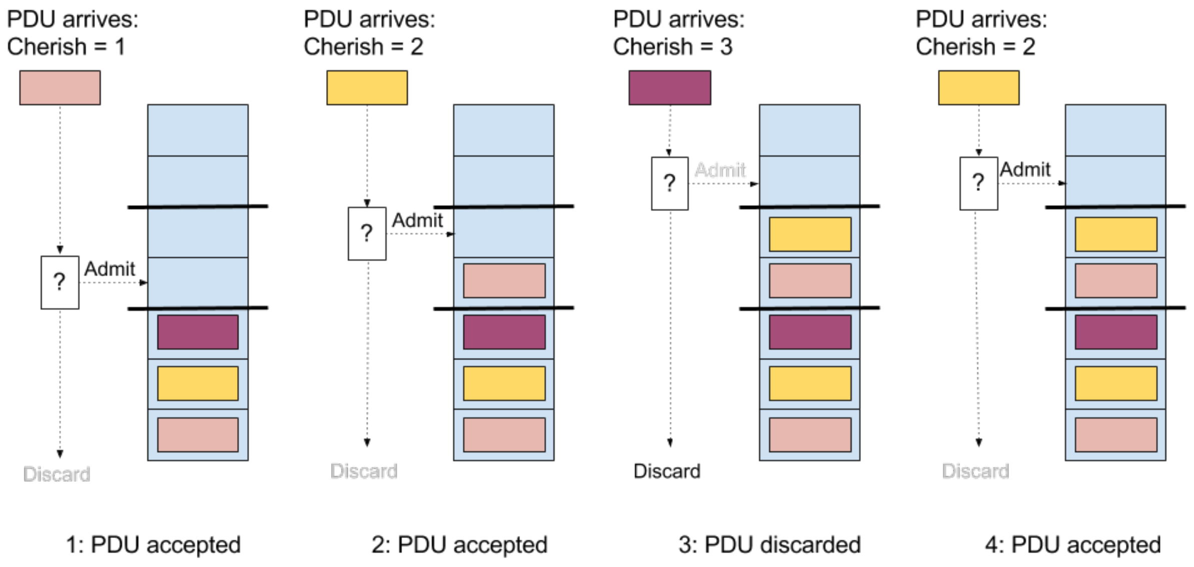


Figure 26. Admission and discarding of packets by cherish levels

Once in the Multiplexer, packets are serviced using a strict priority queueing system according to their urgency level.

There is an engineering choice to be made regarding the number of traffic classes that are supported through the QTAMux - more classes provides stronger isolation between flows but require a more complex configuration.

Operation of the Policer/Shapers

Policer/Shapers can be seen as performing intra-stream contention for traffic in the same treatment class. The operation of the C/U Mux delivers a strict partial order between the service provided to the different classes, but by itself allows traffic in classes higher in this partial order to dominate those that are lower. In order to deliver bounded quality attenuation to all classes, traffic in each class needs to satisfy certain preconditions regarding both its rate and its short-term load pattern, which are aspects of its QoS-cube. The first is regulated through policing and the second via shaping. Policer/shapers enforce these preconditions, ensuring that traffic within a class experiences contention within its allocated resource budget before contending with the traffic of other classes. Policer/shapers provide considerable flexibility in how this is done, for example, traffic that is ‘out of contract’ can be discarded or alternatively ‘downgraded’ to a lower C/U Mux class.

Operation of the stream queues

It is advantageous to decouple the matching of supply and demand (which is the function of the QTAMux) from the transmission of PDUs. In this way the QTAMux processes ‘units of demand’ (AKA ‘wantons’) that are abstracted from the specific PDUs. This has two benefits:

1. It enables an efficient implementation with minimal copying of PDUs
2. It allows ordering to be preserved even when Policer/Shapers map successive ‘wantons’ into different C/U Mux classes, where they will receive different levels of service and could overtake one another.

To achieve this we use two levels of indirection; firstly, references to PDUs belonging to a particular association are placed into a stream queue for that association; secondly references to the stream queue are passed through the QTAMux. Once the QTAMux has made its decision regarding which stream queue to service next, the PDU at the head of that queue is serviced.

3.1.3. QTAMux RINAsim simulation models, scenarios, and results

Policies

While the RINAsim makes a clear distinction between the roles of Scheduling (decide the next queue), Monitor (get required measurements) and MaxQ (decide whether to drop last PDU) policies, the use of those policies as intended does not accommodate well with the use of P/S elements, given the multiple point where dropping a PDU is possible, nor the possibility of dropping a PDU already enqueued instead of not accepting the newest one. Instead, for the QTAMux, we took an approach similar to the one commonly used in the SDK, where scheduling related policies are a unique policy with multiple hooks, instead of different policies. In order to do that, we focused most of the code of the Mux within the Monitor policy, being the only one with hooks in all the required events. Given this, the resulting policies were redefined as follows (for output only):

MaxQueue

Null policy. Should not be called, but by defaults responds with "don’t drop".

Scheduling

Called whenever a port is ready and with PDUs to be sent or for auto-scheduled calls. When called, it checks that the port is really ready (mainly as a measure for scheduled calls), then asks the monitor policy for the next queue to serve. If the monitor policy responds with a null queue, it simply aborts the current call, otherwise, the desired queue is served.

Monitor

Called whenever a port/queue is created or removed from the RMT, when a PDU is inserted into a queue and when the scheduling queries for the next queue to send. It manages all scheduling logic. Within this policy, there is a C/U Mux associated to each port, and a P/S associated to each queue.

Given that P/S are associated to queues, depending on the configuration of those, the QTAMux can be used for connectionless QoS based scheduling, connection-oriented, mixed connection/connectionless configurations, etc., all depends only on how queues and P/S are configured.

C/U Mux

Cherish/Urgency Mux with Absolute and Probabilistic thresholds. It has two hooks:

- add(Queue, Urgency, Cherish)

Called whenever a P/S decides to send a new "serve" request to the C/U Mux. It receives a Queue reference, and the urgency and a cherish levels for that reference and then takes the proper decisions. For each possible Cherish level C, the C/U Mux is configured with an absolute threshold, a probabilistic one and a drop probability. Using these thresholds, the C/U Mux decides, given its current state, to accept the queue reference or not, and in case of not accepting it, whether if the queue has to drop its first or last PDU. In case of accepting the reference, it puts it into a priority queue depending on the urgency level.

When accepting a reference, if the output port is waiting (ready without anything to send), it schedules a new call to the scheduling policy.

- nextPdu()

Called whenever the scheduling policy queries for the next queue to serve. It returns the next queue reference from the priority queue (references are served in order of urgency, then arrival). If the queue is empty, it returns a null reference.

Policer/shapers

Responsible for managing specific queues, deciding how PDUs should be dropped in case of high bursts, the cherish and urgency levels for call to the mux, PDU spacing, etc. It has two hooks:

- Inserted()

Called whenever a new PDU has been inserted into its queue.

- Run()

Called after Inserted() or auto-scheduled when the P/S decides to send the next queue reference to the Mux.

Different P/Ss can be configured depending on the requirements and restrictions of each queue.This approach of having multiple P/S instead of a unique and highly configurable module provides multiple benefits, as it allows to easily add and test new P/S and reduces the computational cost when all the capabilities are not needed. The simplest P/S works as a simple bridge between the Monitor scheduling and the C/U Mux, sending queue references to the C/U Mux at the same moment a PDU is inserted and with fixed cherish/urgency levels. The more complex one performs rate shaping with random spacing between PDUs and statistical C/U levels depending on the current input rate of the queue.

Test scenario and results

Figure 27 shows two small network examples to make a first approach to test the QTAMux policies. These examples are to check congestion control policies without needing to consider arriving flows from multiple sources. These examples, instead of using applications to produce the distinct flows, use a small module capable of injecting PDUs directly into the RMT modules. This allows to produce easily enough data to congest even high capacity links and allow for easy creation of aggregated flows. As we are only simulating part of the stack, the use of injected traffic is a valid option, considering that using applications would require to add extra DIFs and nodes to generate traffic, adding unnecessary burden.

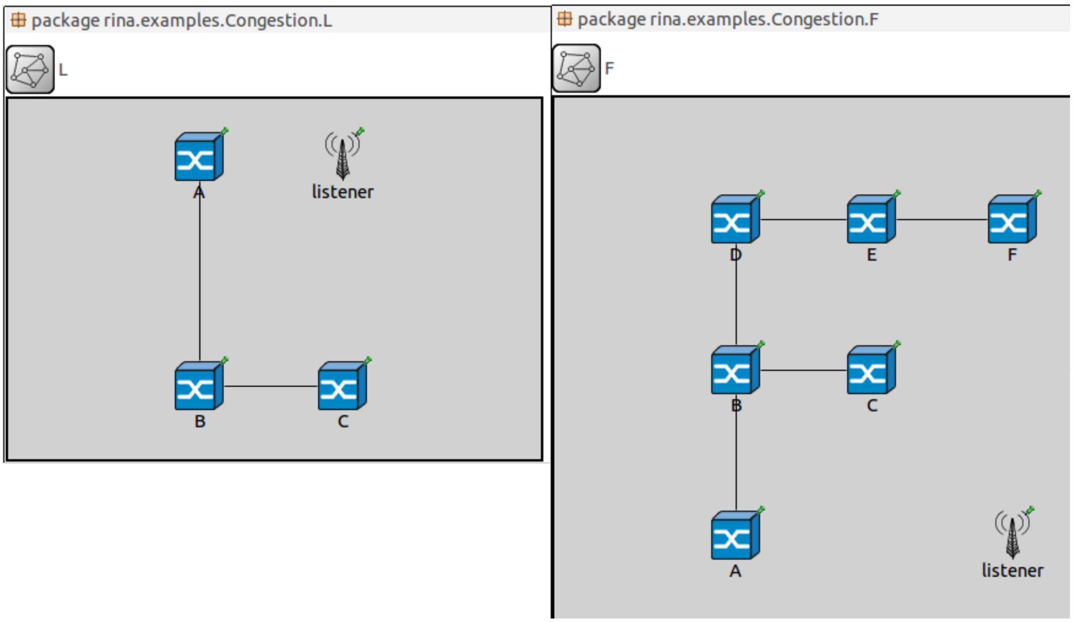

package rina.examples.Congestion.L
L
A
listener
B
C
package rina.examples.Congestion.F
F
D
E
F
B
C
A
listener

Figure 27. LF - L (left) and F (right) scenario network (omnet++)

For a bigger and more complex scenario, we have tested the scenario shown in Fig. BB and BB2, describing DIF with the same internal structure as the 10-node IP/MPLS layer of the Internet-2 backbone network [I2]. Propagation latencies are derived from the real physical distances between node locations and we assume that all N-1 flows have a capacity of 10 Gbps with a Maximum Transmission Unit (MTU) of 5KB.

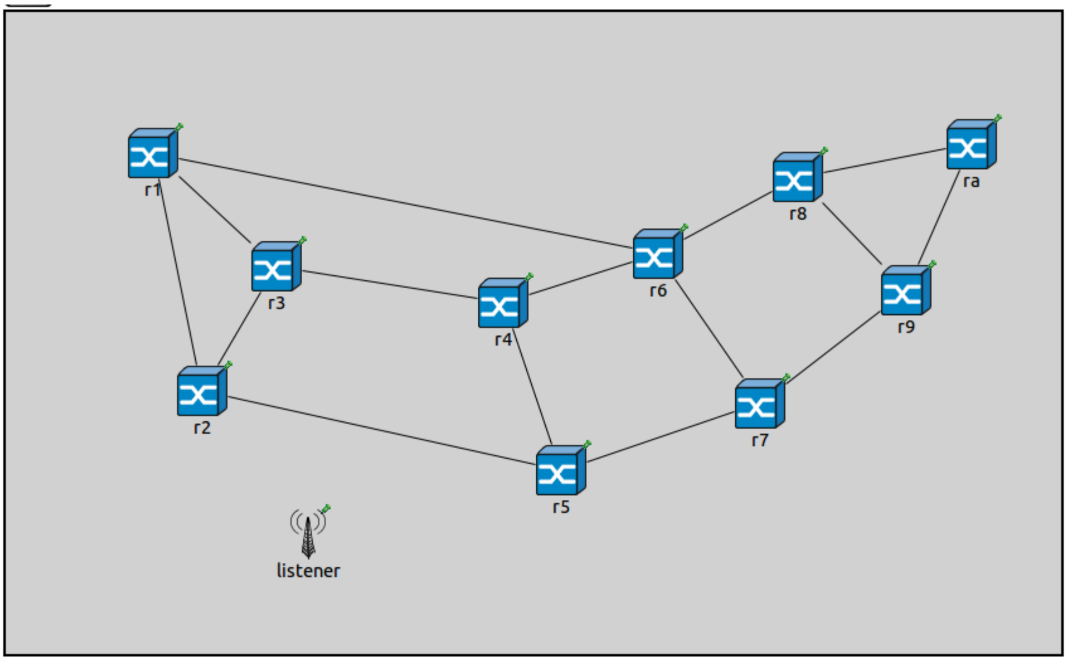


Figure 28. Backbone Scenario network (omnet++)

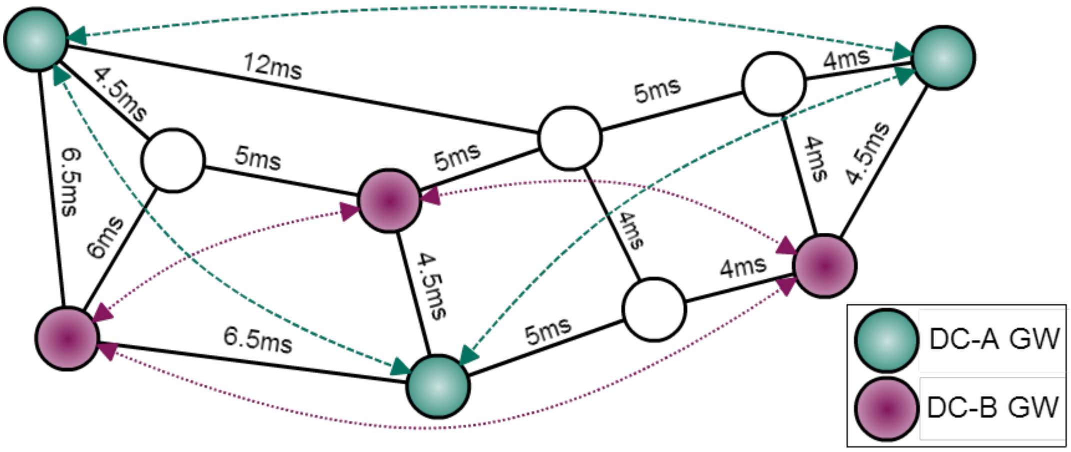


Figure 29. Backbone Scenario network with DC-GW placement

For this experiment, we considered a simple case with 4 types of flows traversing the ISP backbone network: 1) Gold (GU), urgent and lossless. 2) Silver (SN), less urgent but lossless. 3) Sensitive Best-Effort (sBE), urgent but accept losses. 4) Best-Effort (BE). In this scenario, we assume two distributed data centres (DCs), each one situated in three different geographical locations, as shown in Figure 29. These exchange two different classes of inter-DC traffic directly mapped down to the ISP Backbone network with DIF classes Gold and Silver. A 1:4 GU:SN traffic ratio has been assumed when generating the inter-DC flows. Moreover, each pair of data center locations exchanges up to 2 Gbps and 1 Gbps in DC-A and DC-B, respectively. Moreover, DC traffic has to share the available lower-level N-1 flows capacities with background traffic flows from sensitive and non-sensitive Best-effort flows (in this case with a 3:7 ratio between sensitive and non-sensitive BE traffic), so that all DIF links are filled with a similar load level. Our motivation behind this is to assume a worst-case scenario (in terms of congestion) in which all links are heavily loaded.

In this scenario, we focused on testing only the C/U Mux with the simplest P/S (mapping of QoS to C/U classes without shaping) to provide differentiation of services, ensuring some end-to-end requirements for each QoS:

1. GU traffic must experience zero losses up to at least 150% aggregated load (i.e., relative to the total link capacity).
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[3] 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.

*PST : Packet Service Time, number of packets served between the arrival and departure of a PDU into/from a queue.

To ensure that requirements would be met (with high probability), we configured the distinct thresholds for the each C/U classes analytically, resulting in the following configuration of the C/U MUx:

QoS	Heuristinc Threshold	Drop Probability	Absolute Threshold
GU	-	-	120
SN	110	0.1	120
sBE	90	0.1	100
BE	90	0.2	100

We offer a load to the DIF so that all links are equally loaded, allocating flows of 7 to 13Mbps at 100% load between the distinct pairs of nodes, and then scaling the data flow rate to the desired DIF load. For inter-DC traffic, we 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 we are interested in the degradation of sBE and BE flows, we set 30 sBE and 70 BE flows between those same pairs of nodes, enough to get comparable data with respect to the other two classes. Finally, we set 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, we are considering a worst-case scenario where no congestion control is 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 we

[3] Packet Service Time: the number of packets served between the arrival and departure of a PDU into/from a queue.

disable the congestion control in these tests, we have a scenario where only scheduling actions are taken to respond to congestion problems. Secondly, statistics are only computed for flows with multiple hops whose DeltaQ increases along their paths, whereas 'dumb' flows used to fill links are point-to point. This means that their DeltaQ 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 multihop flows.

We do not limit ourselves to only establishing that end-to-end requirements are met using the RINA + DeltaQ based policies. Instead, we 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. For the Best-Effort scenario, we use a simple FIFO queue at each node with the same 120 packets absolute threshold as the one used for the GU class. For the MPLS-based case, we configure a Weighted Fair Queuing (WFQ)-based scheduling 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, we find some interesting results in favor of RINA + DeltaQ-based policies. In RINA, we find that 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. RES-a. We can also compare these results 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 we find that, 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. In terms of latency, as variable latency is not of great importance in this scenario, we focused instead on the maximum jitter in PST experienced by each traffic class. In the RINA + DeltaQ scenario, we find that the requirements per hop are provided, 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. RES-b. 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.

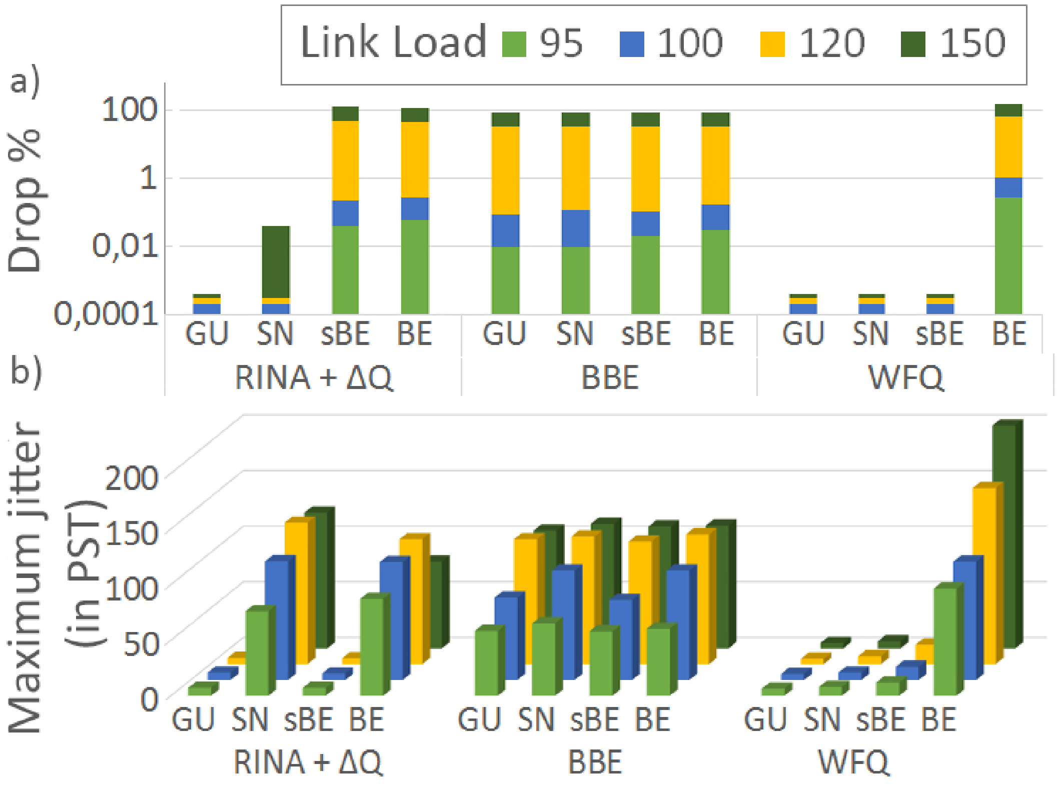


Figure 30. Average drop (a) and maximum jitter in PST (b) for GU, SN, sBE and BE flows depending on the scheduling policy used in the network

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

While we set node capacity, MTU, etc., 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.

3.1.4. Implementation of QTAMux as RMT policies in IRATI and experimentation

As in the case of the RINAsim, in the IRATI prototype, the implementation of the QTAMux with the SDK required accommodating the different components to the set of policies available.

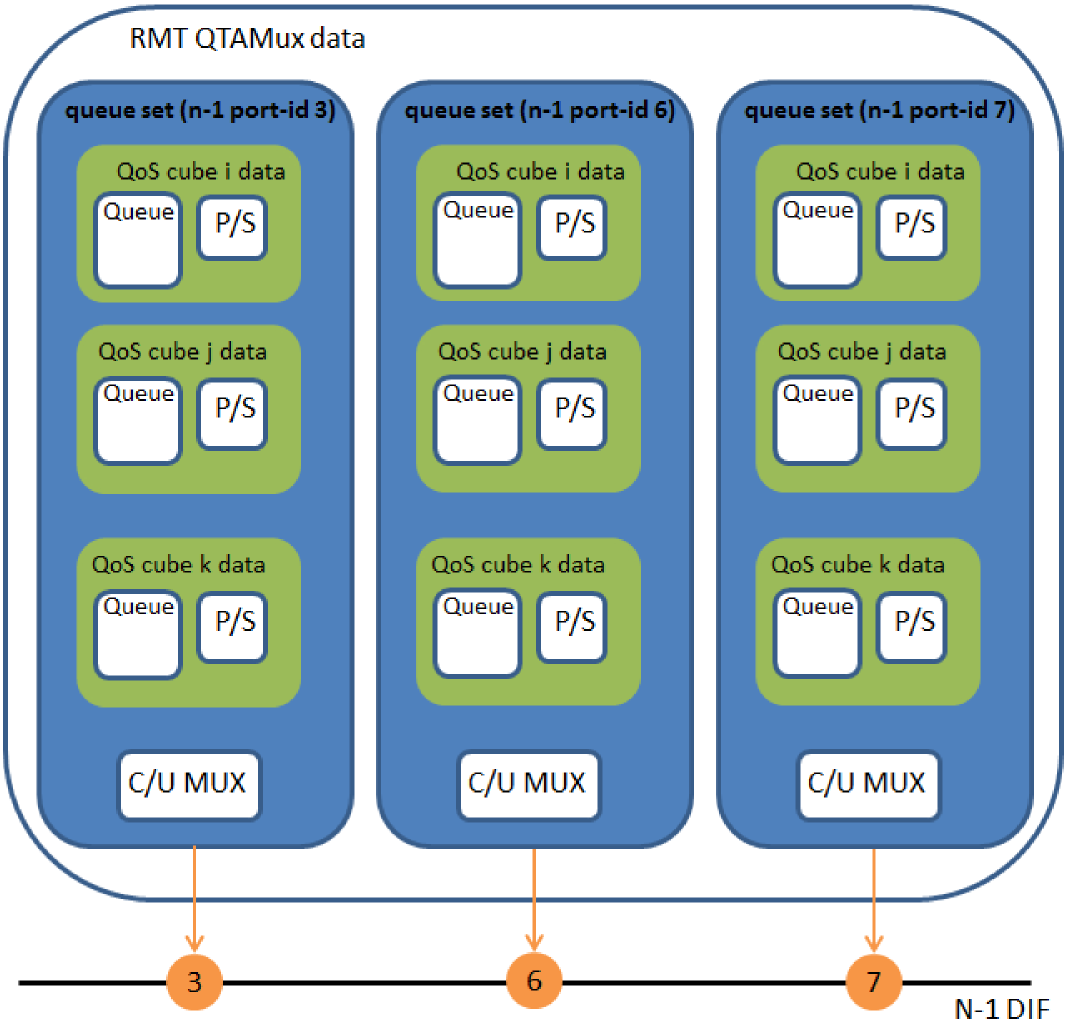


Figure 31. QTAMux block diagram as implemented in the IRATI stack

The C/U multiplexer and the P/S must be implemented using the hooks the RMT policy set offers: essentially rmt_enqueue_policy and rmt_dequeue_policy. The enqueue hook is called whenever the RMT cannot send a PDU to the layer below, i.e. when the layer below is busy and cannot take care of the PDU. This PDU is then stored in one of the RMT queues. The dequeue hook is called when the RMT is ready and requests the next packet to be sent. It is worth noting that the scheduling is a mechanism in the RMT core, thus outside of the scope of the RMT policies. This has some impact on the implementation of the QTAMux, especially in what regards to the P/S.

In the case of the P/S functionality, it has been implemented by means of the dequeue policy callback. Since the scheduling is a mechanism managed by the RMT, the only way to ensure that the bandwidth and ratio control are respected is by taking packets out of the queues accordingly to the preconditions in rate and shape. Thus, the dequeue policy is the point to add the P/S functionality. When the dequeue operation is called, the first step is for the P/S to verify the rate at which the last PDU will be served and the backlog, enforcing then the preconditions regarding rate and shape.

Regarding the C/U multiplexer, its functionality has been implemented by means of the enqueue policy callback. When a PDU arrives at the RMT and cannot be processed right away, it enters the QTAMux. The queue that the PDU will be sent to is given by the QoS cube identifier, and depending on the occupation of the N-1 port id queues, it will be decided if the PDU should be discarded.

Once the QTAMux module is loaded, the configuration of the QTAMux and its hooks are provided to the RMT component. Every time an N-1 port id structure is instantiated in the RMT, a new data structure is created, by means of the rmt_q_create_policy hook. This data structure contains the set of queues, one per QoS cube. The q_qos structure contains the queue of PDUs to be sent and the P/S data structure, plus some configuration parameters and other data used by the multiplexer and the P/S.

3.1.5. Conclusions and future work

Future networks must become demand-attentive in order to support the requirements of multiple distributed applications over the same infrastructure. Assuring QoS guarantees is a must so that each application using the network gets the quality it requires, no more and no less. With the DeltaQ resource allocation model for quality degradation, we have shown that it is possible not only to provide quality of service differentiation with SLA assurance, but doing it while also ensuring a bounded impact on less stringent services. Extending these results by including more complex and realistic scenarios and results from both RINAsim and SDK is planned to be submitted to a journal in the next months.

While the testing scenario was simple, with rather non-restrictive requirements, real scenarios are more complex and therefore require more complex configurations and more restrictive assurances. Given that, our next step will be to test a few different scenarios with a slightly more complex configuration, this time both in the RINAsim and the SDK. After that, future work in this area will be directed to providing dynamic/reactive re-configuration of nodes given changes on the network, working on QoS-aware routing policies, providing a wide range of QoS classes to ensure any type of DeltaQ requirement.

3.2. QoS-aware Multipath Routing in RINA

Existing QoS-aware multipath techniques are based on the knowledge of the characteristics of the traffic going through the network to efficiently select the best path for new flows. The RINA framework ensures the promised QoS to the applications, so implemented multipath techniques must be able to reserve necessary resources. Current state of the art has covered this problem multiple times, for example IntServ through the use of RSVP (Resource Reservation Protocol) or automatically deploying MPLS tunnels [Awduche] Although commonly used, these solutions have scalability problems because of the necessity of including additional layers on top of the traditional internet stack in order to support several levels of isolation. Moreover, such protocols face the problem of how to determinate the characteristics of each flow.

Current solutions to this last problem are often based on a Traffic Matrix (TM) that determines the characteristics of the traffic between origin and destination. However, those traffic matrices aren't trivial to obtain, they are extracted from network measured statistics or SLA with clients and usually have a time scale of weeks or months. Another current work in QoS-aware multipath can be found in [Al-Fares]. In this work, the authors needed to develop a novel technique to estimate the bandwidth that each flow will use. But those estimation presents several limitations - for example, they can't be performed to every flow, only the big ones (more than ten percent of link capacity).

Having studied the alternatives of the state of the art we can affirm that the keystone of QoS-aware multipath in RINA is the built-in support of QoS classification by design. Each RINA flow is supported by a QoS-Cube that determines its characteristics, a feature that allows knowing in advance the bandwidth used by each flow. Moreover, RINA provides for a simpler way to use multipath routing than in the internet — since RINA names the node and not the interface, there is no need for LAG. Put another way, ECMP is the same as LAG in RINA. With this information as an input, it is possible to implement the multipath policies that are described in this document.

In this section, we extend the work on multipath routing in RINA that was already described in previous deliverables. The following bullet points summarize the different multipath strategies that have been addressed.

- Simple multipath routing
  - This multipath strategy is per-hop-based and implements an equivalent routing strategy as Equal Cost Multipath (ECMP) does.
- Static QoS-aware multipath routing
  - This multipath strategy is per-hop-based and takes into account static QoS parameters (e.g. link bandwidth) to take the forwarding decisions.
- Dynamic QoS-aware multipath routing
  - This strategy is the most advanced. It is not only per-hop, but it also takes into account the full path of the flows. To take the forwarding decisions, it uses dynamic QoS information (e.g. the link utilization, congestion level, etc.)

The following subsections describe these strategies in further detail. We also explain the design of the solution that has been developed for each of the strategies. In addition, some results are included to show the testing and evaluation of the strategies' implementation. We leave the results related to performance to the later experiments to be carried out within WP6.

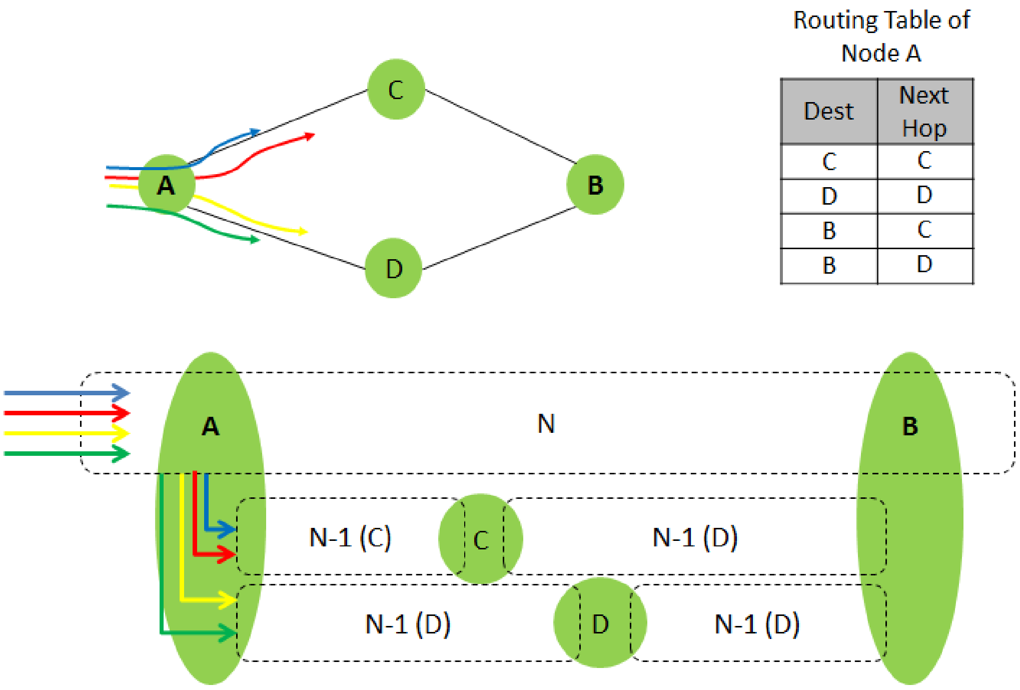
3.2.1. Simple multipath routing

This multipath strategy is a link-state strategy which load-balances traffic according to the simple multipath routing policy. The solution we have developed in this case is based on ECMP. It distributes the traffic evenly among the set of shortest paths to a certain destination. In contrast to ECMP, which distributes TCP connections, this multipath strategy distributes PDUs belonging to different EFCP connections through the different N-1 flows.

The affected RINA policies are the following (please refer to D3.2 for further details):

- PDU forwarding table generator (RA)
- Routing
- PDU forwarding policy (RMT)
- PDU forwarding table (RMT)

Figure 32 describes this strategy using a common scenario that we will also use to describe the other multipath routing strategies. In this scenario, four nodes are connected as depicted in the figure, and the necessary information for the forwarding decisions is included in tables. The traffic travels from node A to node B in all cases. The N-1 DIFs are also depicted in the scenario. For the sake of simplicity, we consider an N-1 DIF per physical link (which can be understood as shim-DIFs in our case).

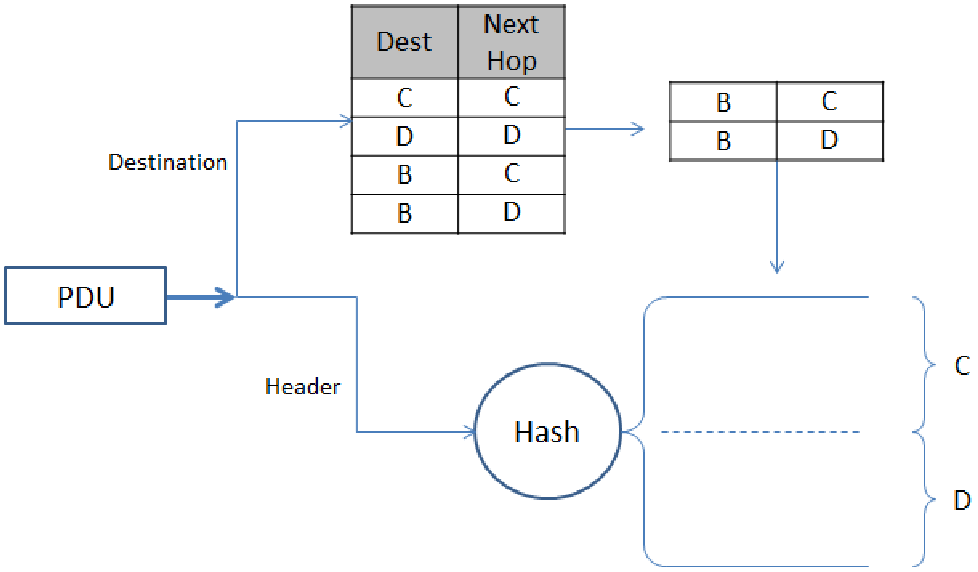


Routing Table of Node A

Dest	Next Hop
C	C
D	D
B	C
B	D

Figure 32. Simple multipath routing

The simple multipath routing strategy only considers as input the next hop to a certain destination. Based on this, it computes the forwarding decision using the same approach as ECMP — it uses a hash function to derive the output port. The following diagram shows this process from a high-level point of view:



Dest	Next Hop
C	C
D	D
B	C
B	D

B	C
B	D

Figure 33. Simple multipath routing solution

The simple multipath routing strategy steps, as depicted in the above diagram, are the following:

1. When the PDU is to be forwarded the routing table is checked and if several output ports are present for the same destination, these output ports are taken as input for step 3.
2. The header of the PDU is passed through a hash function, which generates an output according to the hash range.
3. The entire hash range is divided according to the possible output ports derived in step 1. Always in the same order as they appear in the routing table, each division is identified with one of the possible output ports.
4. The output of the hash function in step 2 is compared to the divisions of the hash range. The output port is then obtained as the port associated to the division the output of the hash function falls into.

3.2.2. Static QoS-aware multipath routing

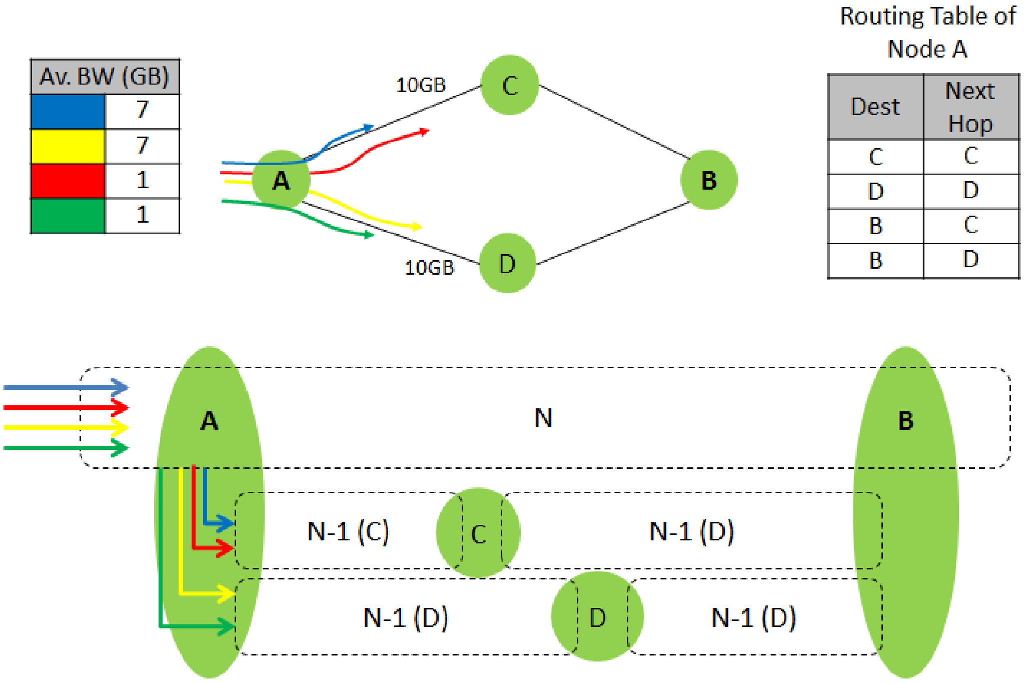
The QoS-aware multipath routing strategy considers static QoS parameters of the N-1 flows to take the forwarding decisions. In the static QoS-aware multipath case, only static link-state information is used, namely:

- Whether the underlying physical link is up or down, and
- Average Bandwidth of the N-1 flow

The affected RINA policies are the following (please refer to D3.2 for further details):

- PDU forwarding table generator (RA)
- Routing
- PDU forwarding policy (RMT)
- PDU forwarding table (RMT)

Figure 34 describes this multipath routing strategy from a high-level point of view. The average bandwidth of the different flows is contained in the first table. As we can see, the bandwidth of the N-1 flows is 10 GB and two of the flows have an average bandwidth of 7 GB. Therefore, if these two flows are forwarded through the same link, it will cause congestion. The advantage of this QoS-aware routing strategy is that this bandwidth requirement can be taken into account to take the forwarding decision, and in this case, the two 7 GB flows will never be forwarded through the same path. As the example showed in the figure, we could intelligently spread the flows, forwarding a flow of 7 GB and a flow of 1 GB through a link of 10 GB, which will avoid congestion problems.



Av. BW (GB)	
🟦	7
🟨	7
🟥	1
🟩	1

Routing Table of Node A

Dest	Next Hop
C	C
D	D
B	C
B	D

Figure 34. Static QoS-aware multipath routing

Figure 35 depicts a diagram of the solution we have implemented for this routing strategy. The steps it consists of are the following:

1. A temporary cache stores the headers of the PDUs and the output ports of the flows that have already been forwarded before. In this way, when a PDU is to be forwarded, it is checked against the cache so that PDUs belonging to the same flow are always forwarded through the same path. Two things may happen:

    a. If the PDU's header has a match in the cache, it is forwarded through the output port stored in the cache. In this case, the forwarding process ends here and no further steps are carried out on that PDU.

    b. If the PDU's header does not have a match in the cache, the forwarding process continues in the next point (step 2)

2. The QoS-cube Id is taken from the PDU's header and the average bandwidth of the flow is checked.

3. The possible output ports for the PDU's destination are extracted from the routing table.

4. The bandwidth of the N-1 flows associated to the corresponding output ports are also extracted from the stored information (this information is assumed to be known for now, further details about how this information is extracted will be introduced in the combined experiments of WP6).

5. All this information is passed as input to the forwarding function, which determines the forwarding port according to the implemented algorithm.

6. The derived port is stored in the cache along with the PDU's header so that subsequent PDUs belonging to the same flow are forwarded through the same port without having to execute the forwarding decision process.

7. The PDU is finally sent.

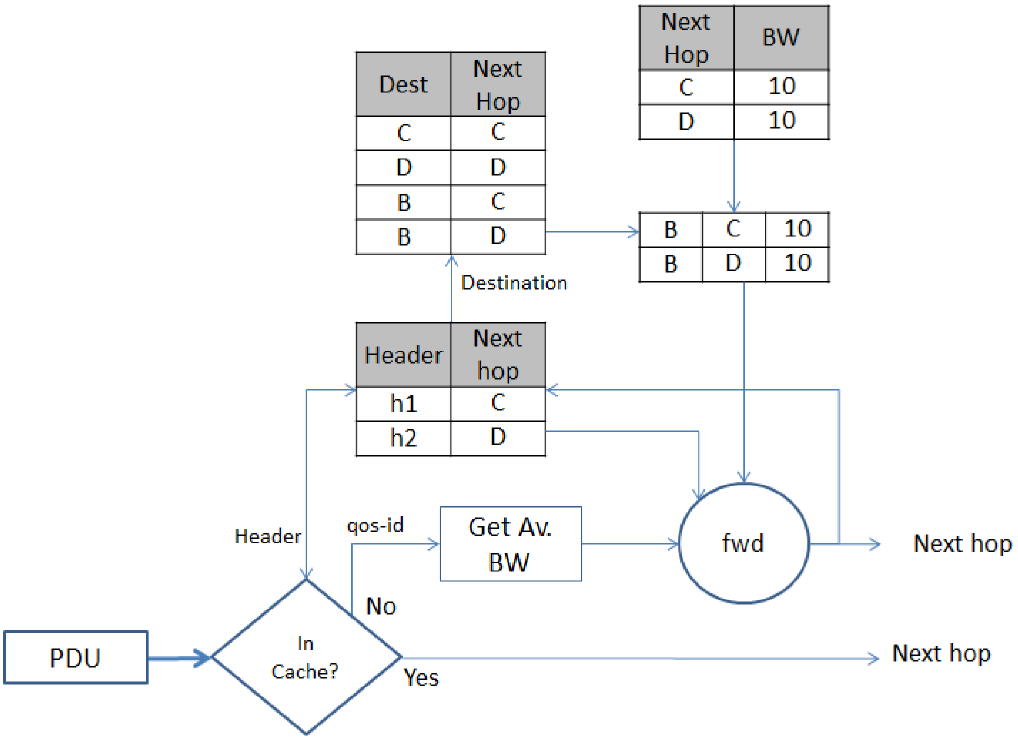


Figure 35. Static QoS-aware multipath routing solution

Many possibilities exist regarding the forwarding function. There is a wide range of possibilities when splitting traffic among different paths considering the traffic characteristics. We, therefore, propose a set of QoS-driven load-balancing algorithms that can be implemented as part of the forwarding function. These algorithms are intended for strict bandwidth requirements and are depicted from a high-level point of view in Figure 36.

We have come up with these algorithms since they offer different characteristics and suit different scenarios and purposes. In the experimentation phase in WP6 we will evaluate the different algorithms and we will analyse which ones are suitable for which scenario.

Figure 36 shows graphically the fundamental idea of each one of the load-balancing algorithms proposed. Black boxes represent the available space in each next hop and coloured blocks the requested space of the flows.

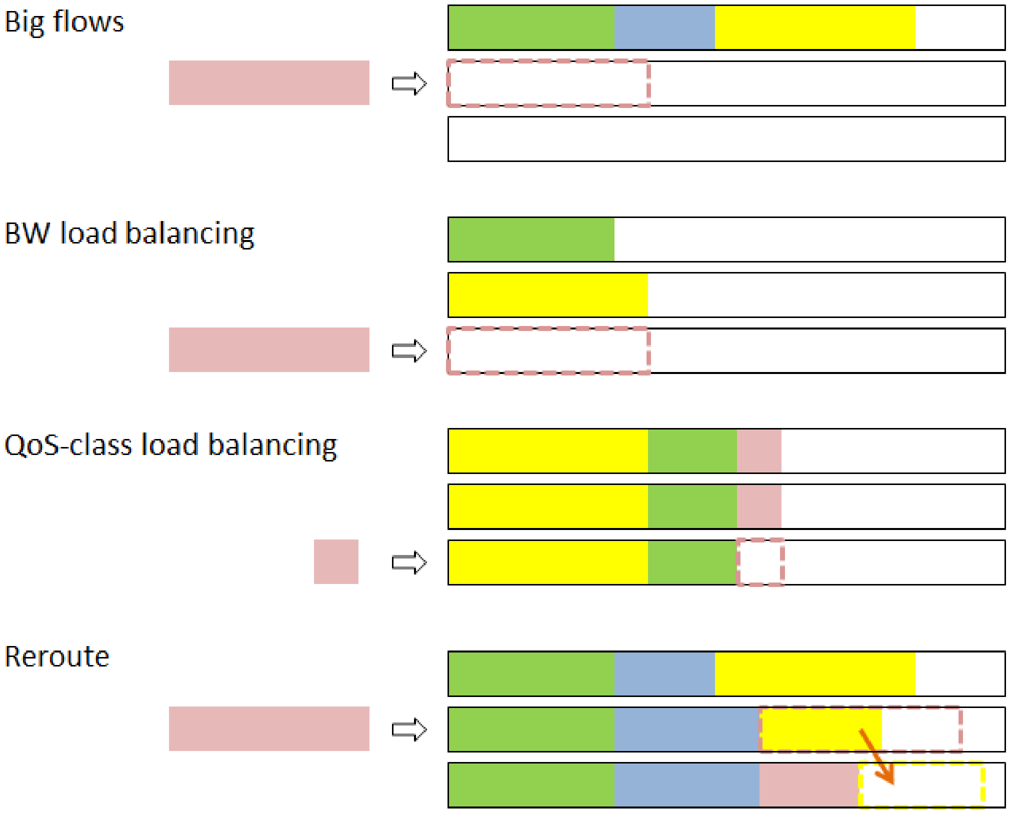


Figure 36. Static QoS-aware multipath routing algorithms

The description and objectives of the algorithms are the following:

- Big flows
    - This algorithm follows the principle of allocating new flows in the N-1 flow that offer the minimal free space, as long as the N-1 flow has enough free space to allocate them entirely. If all of them have the same free space, the decision is taken randomly. Thus, this algorithm tends to replete N-1 flows progressively while leaving big free spaces. Therefore, the advantage of this algorithm is that it can allocate arriving big flows more easily without the need of rerouting already allocated flows. The drawback of this algorithm is that some N-1 flows are overloaded while others remain free. We will study if this drawback has a real effect on performance in the experimentation phase.
- BW load-balancing
    - This algorithm follows the opposite principle than the previous one. That is, it allocates arriving flows to the N-1 flow that offers the maximum free space to allocate it. The objective of this algorithm is,

  therefore, the balance the bandwidth allocation as evenly as possible among the N-1 flows. The drawback is that when a big flow arrives, effort has to be spent in the rerouting process of the already allocated flows.

- QoS-class load balancing
    - This algorithm also pursues the goal to balance the load evenly among the different N-1 flows. For this purpose, it balances the different QoS-classes among the different N-1 flows. That is, each N-1 will try to evenly allocate the flows belonging to a certain QoS-class, in the sense that an arriving flow will be allocated to an N-1 flow that does not allocate a flow of its QoS-class, or if all N-1 flows do, to the N-1 flow that allocates the smaller number of flows of this QoS-class.
- Reroute
    - This is an algorithm that may operate in combination with the above three. The algorithm takes as input a certain space that has to be freed to allocate an arriving flow. Then, it moves (or reroutes) the already allocated flows to new suitable N-1 flows to make space for the new arriving flow. In case it's not possible to allocate the requested space, the algorithm does not carry out any rerouting.

The main process is described considering two of the above algorithm objectives ("Big flows" and "BW load balancing") as pseudo-code below:

```
ALGORITHM: Route flow
// input
newFlow // new flow to be routed
P       // set of the possible output ports to a newFlow’s destination
M[p][f]     // flow matrix mapping all flows (f) to all ports (p) in a given
 node
algorithm   // specific routing algorithm for allocating new flows to ports
            //  big flows – leave free space to allocate future big flows
            //  load balancing – load balance new flows as they arrive
// output
OP  // output port chosen to forward newFlow

BEGIN
auxPort
if newFlow needs strict BW guarantees
    for each p in P do
        if supportsQoSClass (flow, p)
        and isBetterPort (p, auxPort) then
```

```
            auxPort ← p
        end if
    end for
    if auxPort is not empty then
        OP ← auxPort
    else
        OP ← rerouteFlows(newFlow.avBW)
    end if
    updateFlowMatrix (M, newFlow, OP)
else
    // TBD: deal with no BW guarantees
end if
END
```

```
supportsQoSClass (flow, p):

if flow.AvBW < getAvailableBW (p)
    and flow.MaxDelay > SCHEDULING.getMaxDelay (p, flow.QoSId)
    and flow.Jitter > SCHEDULING.getJitter (p, flow.QoSId) then
        return true
else
        return false
end if
```

```
isBetterPort(port, auxPort):

if auxPort is empty then
    return true
end if
switch algorithm
    case big flows:
        return getAvailableBW(port) < getAvailableBW(auxPort)
    case load balancing:
        return getAvailableBW(port) > getAvailableBW(auxPort)
    end switch
// TBD: trade-off using delay and jitter statistics to select the best port?
// e.g. routing the more delay constrained flows through the less utilized
 ports.
// different possibilities are also possible like the “big flows” and “load
 balancing” for BW
```

```
rerouteFlows(BW):

switch algorithm
```

```
    case big flows:
        P ← orderDescByAvailableBW(P)
    case load balancing:
        P ← orderAscByAvailableBW(P)
end switch
// TBD: rearrange flows in M according to the desired metric (e.g.
 ascendingly by size)
M ← rearrange (M)
for each p in P do
    // auxiliary matrix to store temporal routing reconfigurations
    auxM ← M
    for each flow in M[p] do
        auxPort ← getPortToReroute (flow, p)
        if auxPort is not empty then
            auxM ← updateFlowMatrix (auxM, flow, p, auxPort)
            if getNewAvailableBW(p) > BW then
                reroute (auxM)
                return p
            end if
        end if
    end for
end for
return empty //No suitable flow was found


getPortToReroute (flow, port):

auxPort
D ← getPossibleOutputPorts (flow.destination)
for each p in D do
    if p != port then
        if supportsQoSClass (flow, p)
        and isBetterPort (p, auxPort) then
            auxPort ← p
        end if
    end if
end for
return auxPort
```

3.2.3. Dynamic QoS-aware multipath routing

The Dynamic QoS-aware multipath routing strategy uses dynamic link-state information (in addition to the previous static per-link information) to make the forwarding decisions. The used information is:

- Whether the link is up or down

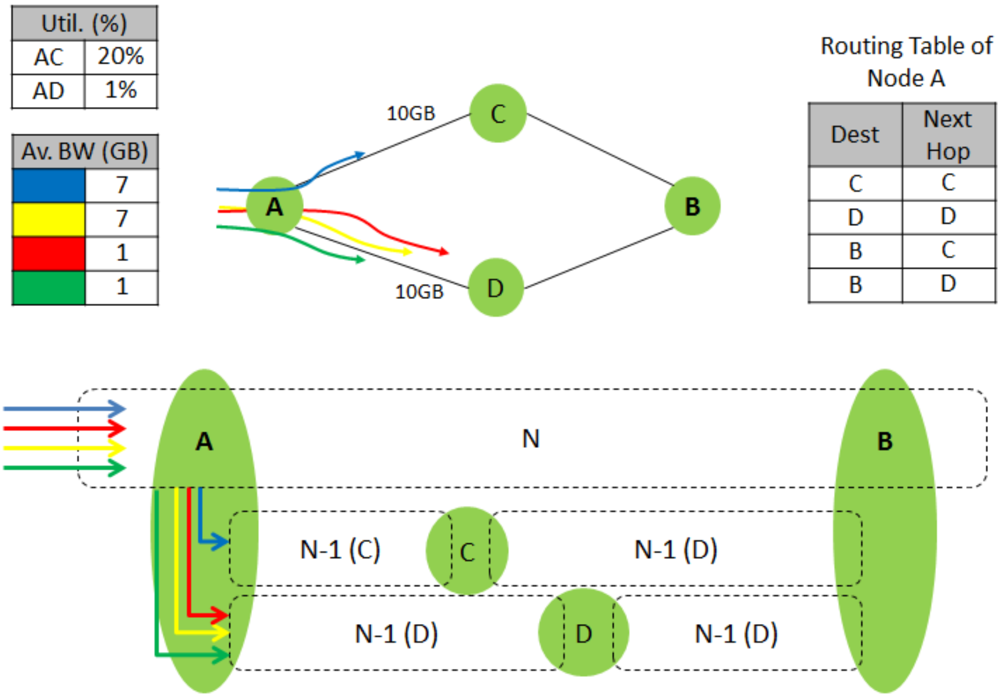
- Link bandwidth
- Dynamic monitored information
  - Instant traffic load/link utilization (where "instant" refers to the last monitored value)
  - Traffic statistics collected at the network level, such as queue states

The previously described "QoS-driven load-balancing algorithms" for the static QoS-aware multipath routing strategy are extended to consider the traffic statistics per link and spread the flows among the different paths.

The affected RINA policies are the following (please refer to D3.2 for further details):

- PDU forwarding table generator (RA)
- Routing
- PDU forwarding policy (RMT)
- PDU forwarding table (RMT)

Figure 37 describes this strategy in a similar scenario as in the previous case. The situation is the same as in the QoS-aware static strategy, but now we know extra information about the utilization of the links. In this way, we can produce a better forwarding decision since the utilization of the link AC is higher than the link AD. Then, one of the small flows is forwarded through the less utilized path, avoiding the extra congestion that could be caused in the more utilized link.



Util. (%)	
AC	20%
AD	1%

Av. BW (GB)	
🟦	7
🟨	7
🟥	1
🟩	1

Routing Table of Node A

Dest	Next Hop
C	C
D	D
B	C
B	D

Figure 37. Dynamic QoS-aware multipath routing

Figure 38 depicts the implemented solution for this routing strategy, which is equivalent to the one presented in the Static QoS-aware multipath strategy. The only difference is that now the forwarding function takes as input also the extra information regarding the utilization of the links, and the final output port is derived based on this information.

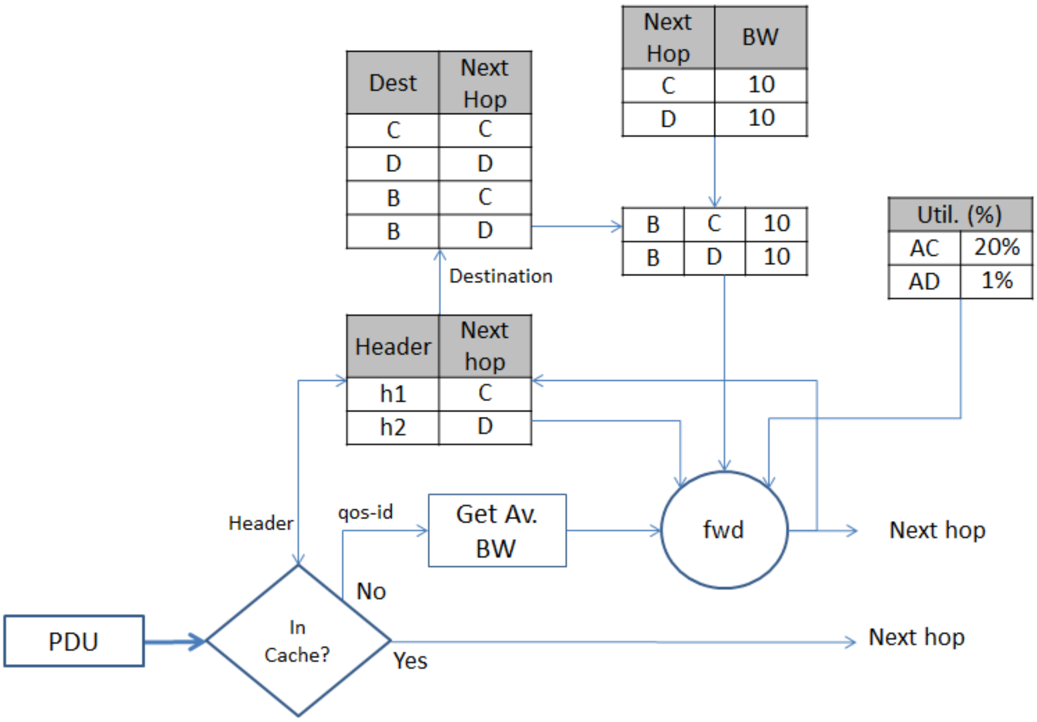


Figure 38. Dynamic QoS-aware multipath routing solution

3.2.4. Test scenario and results

This section provides validation and performance results of each of the multipath routing strategies previously described. First, a comparison of the simple ECMP routing with the more advanced QoS-aware mechanisms was done. For that, the forwarding decision algorithm that was used in the QoS-aware routing is the Bandwidth load balancing. This allows verifying the load distribution improvements that are achieved when the QoS is considered for the routing decisions. Next, the same network configuration was used to test the benefits of choosing a dynamic evaluation of the QoS versus the static approach when deciding the forwarding.

The experiments have been carried out using the RINA simulator, defining a datacenter scenario with the following topology. The DIF configuration follows the one described in [D3.2].

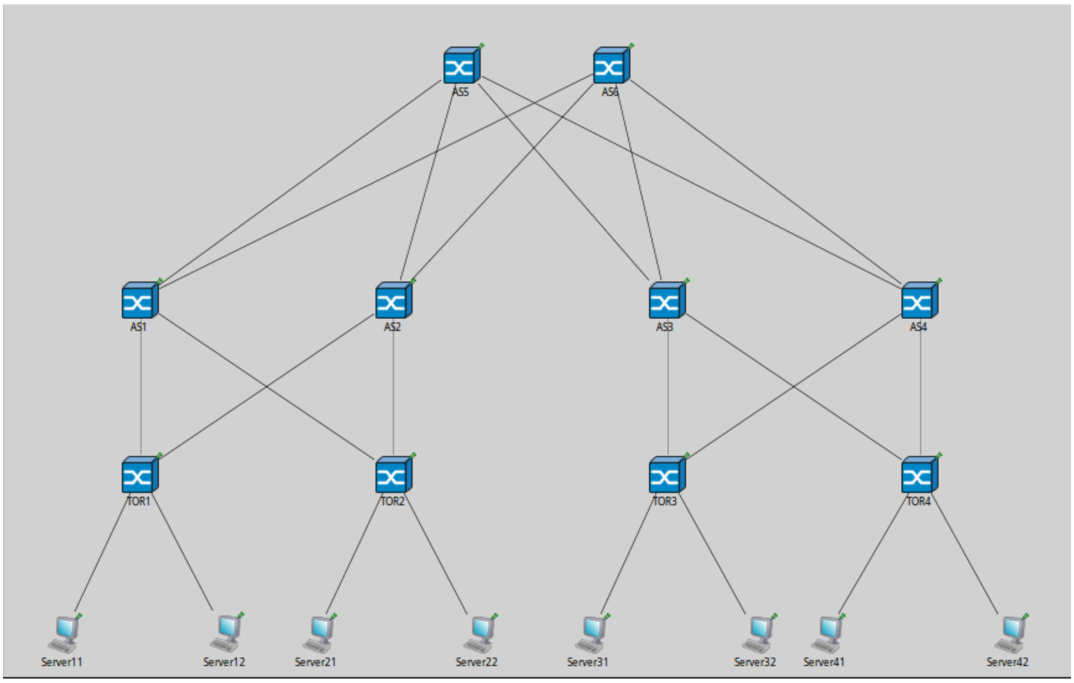


Figure 39. Multipath experiment scenario

For every simulation, the generated traffic was the same. The two servers connected to the first TOR switch (server11 and server12) are configured to send traffic at the maximum capacity of the link, defined to be 1 Gbps. This way, a non-balanced distribution of the load in TOR1 will result in a saturation of one of the upper links going to AS nodes (AS1 and AS2). The flows have been divided into three categories, attending to the QoS requirements of each one. The first QoS class (QoS1) specifies a bandwidth of 40% of the total link capacity, the second one (QoS2) requires 10% of the capacity and the last one (QoS3) is associated to 1% of the maximum bandwidth. The number of flows for each QoS class was the following: 1 for QoS1, 4 for QoS2 and 20 for QoS3, giving a total of 100% of the link bandwidth. The experiments were repeated five times randomizing the times when each flow is initiated in order to achieve non-deterministic results on every run due to the characteristics of the forwarding algorithms.

ECMP vs QoS-aware routing

As described previously, the ECMP forwarding policy is based on a hash-threshold algorithm. Thus, the forwarding only guarantees that packets of the same flow go through the same link, not taking into consideration any bandwidth requirements. Ideally, the distribution of the flows using this algorithm should be balanced between the possible paths (two in this case), however, the simulation yielded different results. Figure 40 and Figure 41 show the load distribution and the number of flows forwarded between the two ports going out of TOR1, to nodes AS1 and AS2 respectively, across five experiments.

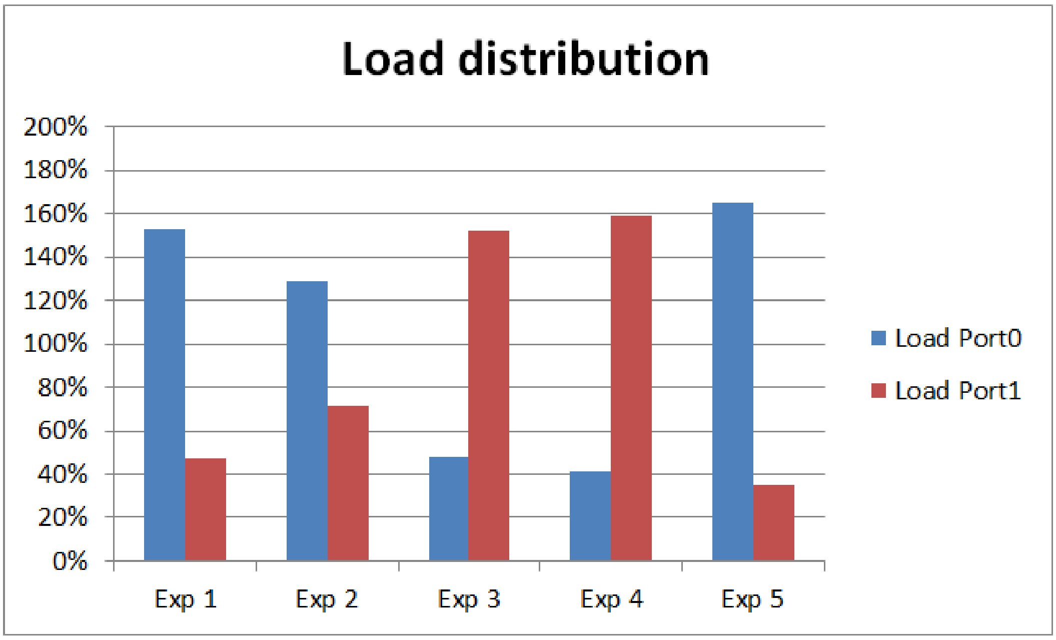


Figure 40. ECMP routing load distribution

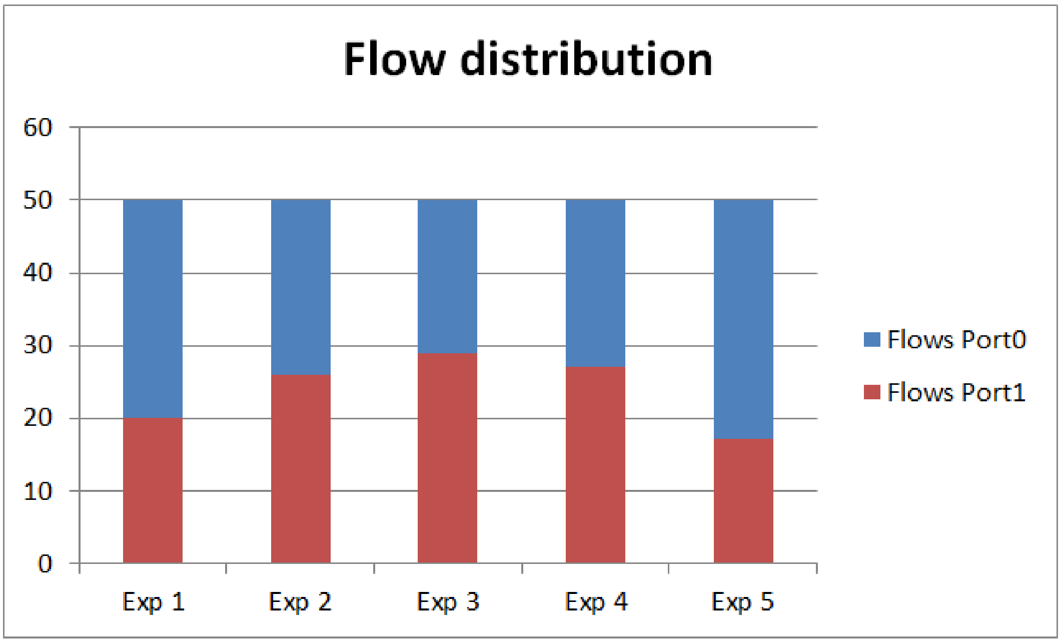


Figure 41. ECMP routing flow distribution

As it can be seen, the load distribution was not equally balanced between the ports. In every experiment, the TOR1 node sent more traffic to one of the ports that its maximum capacity, thus leading to bottlenecks in the communication due to packets waiting in queues or even packet losses depending on the traffic drop policies. With the ECMP forwarding policy, as the bandwidth is not considered in the algorithm, there were no rejected flows.

Next, the same experiments were carried out using more advanced policies that take into account the QoS of the flows.

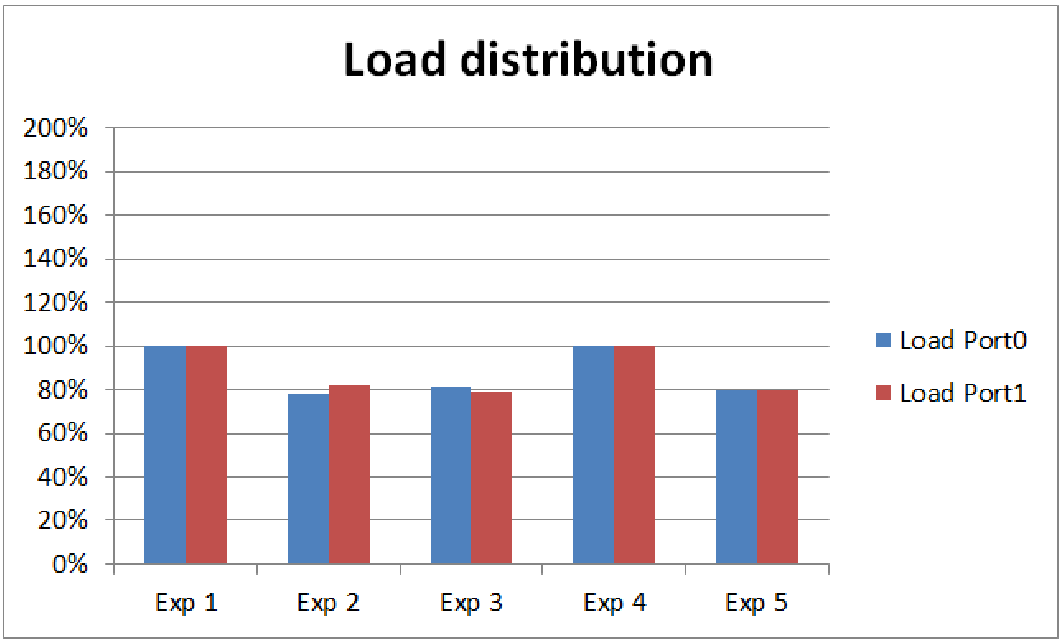


Figure 42. Static QoS-aware load distribution

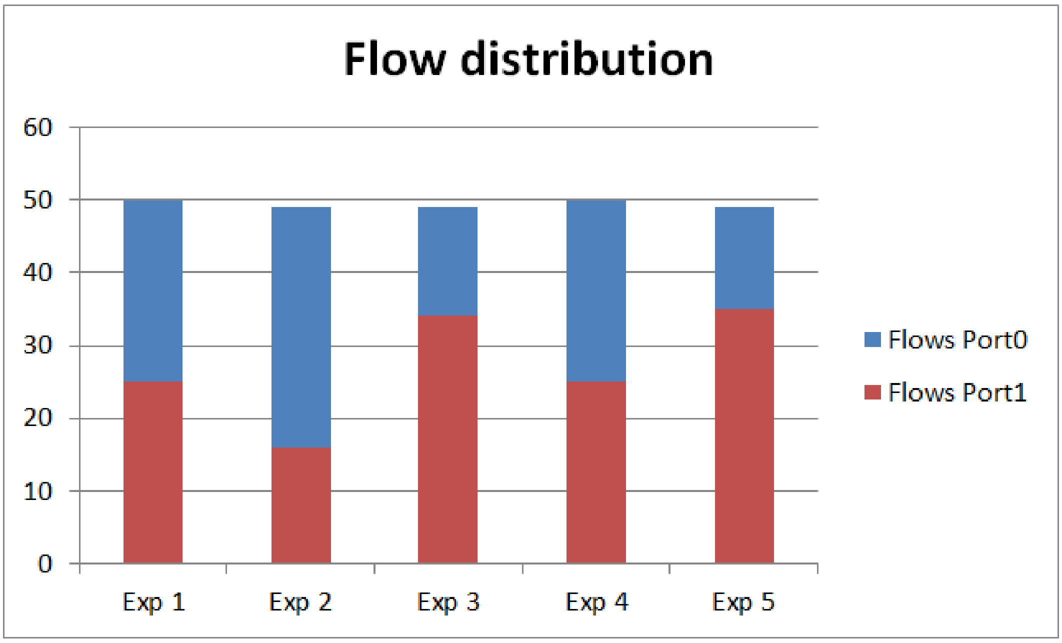


Figure 43. Static QoS-aware flow distribution

For this scenario, both the Static QoS-aware and the Dynamic QoS-aware algorithms presented similar results which are illustrated in Figure 42 and

Figure 43. This was expected due to the lack of other traffic through the links apart from the one generated by the servers connected to TOR1. The main difference between the Static and the Dynamic QoS-aware algorithms is that in addition to taking into account the forwarded flows, the latter periodically asks the scheduler to get the actual load of each port.

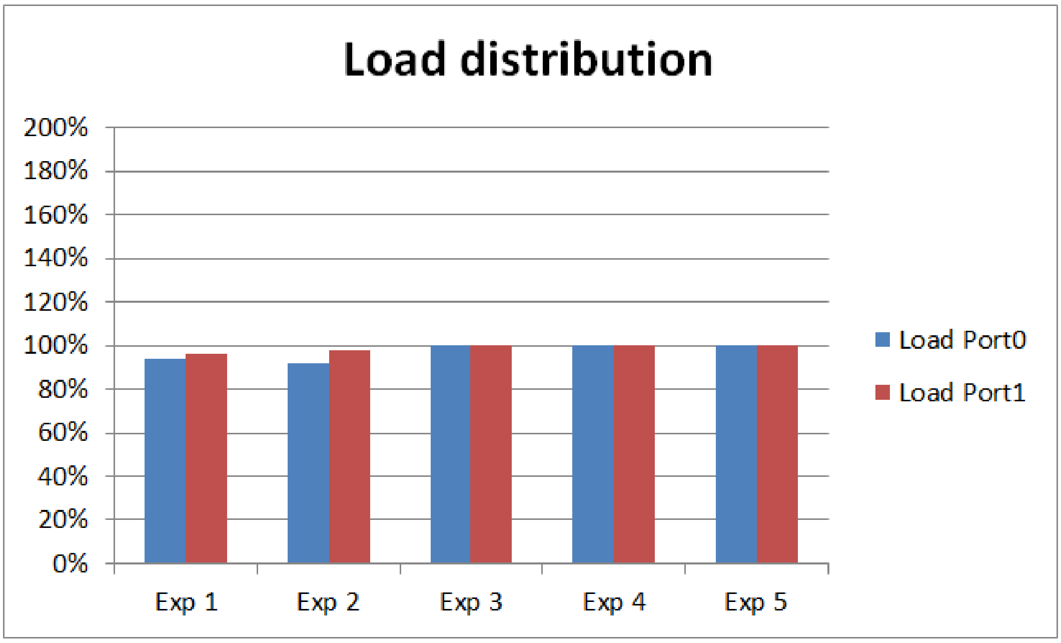


Figure 44. Dynamic QoS-aware load distribution

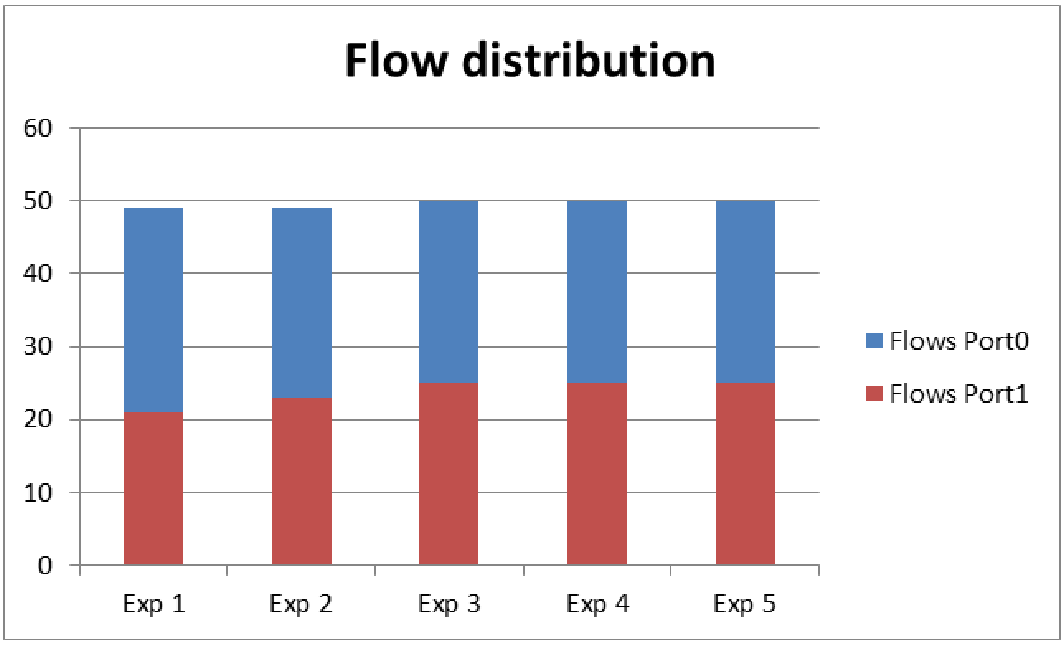


Figure 45. Dynamic QoS-aware flow distribution

Figure 44 and Figure 45 show that a much better balance between the links than with ECMP was achieved. No more traffic than the maximum capacity of a link is sent, keeping the used bandwidth between ports equally distributed. As the forwarding algorithm was focused on bandwidth load balancing, there was no reallocation of flows from one port to the other. Instead, when a flow demanded a higher bandwidth than available it was simply rejected to avoid saturating the link. This behaviour can be seen in the flow distribution graph of some experiments where the total number of packets sent does not reach the received fifty. The reason why sometimes there were packet drops is related to the random start time of the flows in the servers. If multiple small flows arrive first, they can be routed in a way that reduces the remaining available bandwidth for the bigger ones, leading to the reject flow situation.

From the previous figures, the Dynamic QoS-aware routing achieved an almost perfect distribution of load and the number of flows when compared to the Static QoS-aware routing, however this behaviour has been attributed to the randomness of the flow generation, since there are other parameters in the algorithms that could have affected the load distribution under the conditions of these experiments.

Static QoS-aware vs Dynamic QoS-aware

The following experiments show the improvements of the Dynamic QoS-aware routing vs the static approach. While the Static QoS-aware registers the QoS of all the forwarded traffic for each available port, which is then used to better distribute the load, the dynamic algorithm takes also into account direct information from the scheduler function. Periodically, the dynamic algorithm checks the bandwidth utilization reported by the scheduler against its own records, updating the latter accordingly in case there are discrepancies. By doing this, the forwarding algorithm is able to know any sudden congestion that may occur in the links due for example to best effort traffic bursts or failures in the links that may affect their maximum capacity. Therefore, the dynamic approach is more resilient to unexpected changes in the network to take better forwarding decisions.

For this experiment, a best-effort traffic demanding an 80% of the total bandwidth was injected in the links connecting nodes AS1 and AS5 and nodes AS2 and AS6. Figure 46 shows the results for the Static QoS-aware routing.

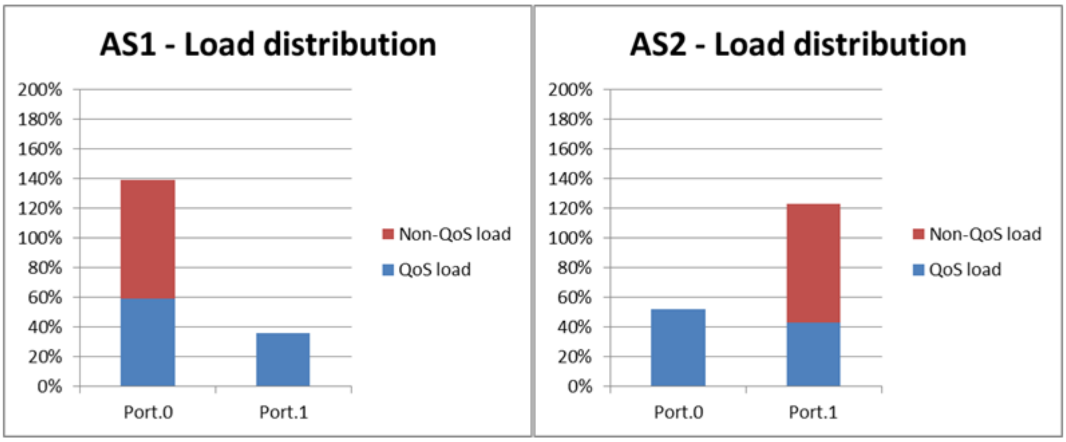


Figure 46. Static QoS-aware routing load distribution with best effort traffic

As it can be seen, the forwarding algorithm distributed more or less equally the load between the output ports without considering the non-QoS traffic. This resulted in the saturation of the links through which the best effort flows were being transmitted while the other link still had enough available bandwidth.

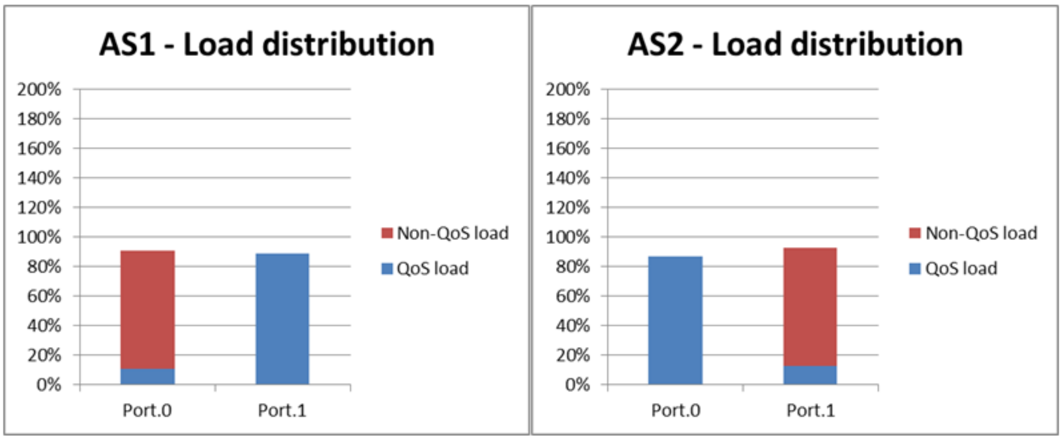


Figure 47. Dynamic QoS-aware routing load distribution with best effort traffic

As expected, for the same scenario, the dynamic algorithm was able to detect the non-QoS load that kept the links at a high load. The results shown in Figure 47 imply that the algorithm balanced the traffic correctly between the two output ports, thus avoiding any saturation.

3.2.5. Conclusions and future work

The previous experiments have shown the improvements that can be achieved by using advanced forwarding strategies. With the simplest case, ECMP, the packets of the same flow are routed through the same path without any consideration regarding the required bandwidth, which may cause link saturation. The hash-threshold algorithm focuses on statistically distributing the number of flows evenly when a large quantity of them is measured.

The next step consisted of making the routing policies aware of the QoS of the flows. This enhancement has been possible thanks to the built-in support for QoS classification of flows in RINA by means of QoS-cubes. By defining and assigning a specific bandwidth to flows in a QoS-cube, the forwarding algorithm is able to better distribute the traffic among the possible paths. Therefore, the bandwidth requirements of each flow can be fulfilled at the same time the load of the links is kept balanced. Both the static and the dynamic approach were designed to reject incoming flows that cannot be sent due to unavailable bandwidth, guaranteeing no congestion in the links. The Static QoS-aware algorithm, although it balances the traffic, is weak against non-QoS classified flows such as best effort traffic, or against unexpected errors in the links. This is so because the algorithm only has information about the flows that have already routed, their associated QoS-cubes and the QoS-cubes of the N-1 flows. So it can’t react if the real state of the network isn’t going according to the QoS-cubes, for example, an error link, or if the QoS–cubes don’t have strict bandwidth requirements, for example, best effort traffic. In order to calculate the best path for load balancing. In the dynamic case, periodic checks with the scheduler are done, providing a more accurate view of the current status of each link. Because of this, the best routing results in terms of load balancing across multiple paths have been achieved with the Dynamic QoS-aware strategy as expected.

RINA policy-based architecture has proven to be very useful for implementing and experimenting with the different multipath routing policies described in the document, as they can be developed as individual units that can be set active or inactive with the modification of a single line in a configuration file.

Further improvements to the multipath routing algorithm will be studied in the context of WP6. These include the use of a central manager able to select the optimum path taking into account the end-to-end status of the network in combination with more advanced scheduling policies.

4. Topological addressing

This section outlines the various approaches to topological addressing used within the use-cases.

4.1. Topological addressing and routing in Distributed Clouds

4.1.1. Introduction

In the last few years, cloud computing has gained considerable interest from researchers, industries and standardization bodies. This promising technology has enabled the deployment of a large set of use case scenarios that were not economically feasible in traditional infrastructure settings (e.g., big data analytics, mobile clouds and High Performance Computing applications). Cloud computing technology has introduced a new computing model in which resources (i.e., storage and CPU) are made ubiquitously available as general utilities that can be used in an on-demand style and at very low costs. When the computing resources are distributed in different geographical regions, we call this scattered deployment "Distributed Clouds".

Distributed Clouds can directly reach users due to their distributed infrastructure which makes large-scale applications possible to deploy. Distributed Clouds usually rely on resources where dedicated facilities are deployed in traditional datacenters - but there is also a different kind of distributed cloud, which consists mainly of resources scattered in offices, costumers' homes and/or data centers participating in the cloud services in a voluntary fashion. This new concept of distributed/decentralized clouds paves the way to the development of more scalable, resilient and flexible clouds. Accordingly, robust and resilient networks in terms of availability, routing and security are needed to cope with the evolution of the cloud systems.

VIFIB [Vifib] is an example of these decentralized "volunteer" clouds. Mainly, it has been proposed to protect critical corporate data against possible downtime or destruction. Moreover, it is designed to enable the deployment and configuration of applications in a heterogeneous environment. By hosting computers in many different locations and maintaining a copy of each associated database at three or more different distant sites, the probability of failure of the whole infrastructure becomes extremely low. In case of a disaster or downtime of a server, the data is replicated and the cloud continues to operate. The VIFIB system is based on a master and slave design. The master controls the different computers that run slaves. In terms of networking, the master and the slaves at different locations are interconnected through multiple IPv6 providers. In order to guarantee high reliability, VIFIB uses an overlay called re6st [Re6st], which creates a mesh network of OpenVPN tunnels on top of several IPv6 providers and uses the Babel protocol [Babel] to compute the routes between nodes. Despite its effectiveness, re6st has certain limitations, mainly related to scalability and security. The re6st overlay creates a flat topology that does not scale in case of large networks, which is an important concern for the future of the Distributed Clouds.

In this deliverable, we study issues related to scalability and dynamism in Distributed Clouds, specifically considering the case of the VIFIB system. We first highlight the issues and limitations related to the networking architecture of the VIFIB system. Then, we propose new solutions based on the Recursive InterNetwork Architecture (RINA). RINA, by its design, is better suited to handle large networks and provides interesting benefits compared to the current over-IP solutions, such as enhanced security or extended programmability. The objective of this section is to demonstrate how our RINA-based solution outperforms the re6st overlay and gives better results.

The next section gives some background on the VIFIB Distributed Clouds and its challenges. Then, we address the application of RINA for efficient management of routing in the VIFIB system. Some evaluation studies that have been conducted using RINASim simulator are provided. Finally, we conclude and provide directions for further research.

4.1.2. Characteristics and Requirements of the Distributed Clouds Use Case

Here we give some background on the VIFIB system. Furthermore, we discuss the issues and challenges related to its networking system: the re6st overlay. VIFIB is a decentralized cloud system, also known as "resilient computing". Resources are scattered in computers that are located in customers' homes and in offices. The VIFIB system is based on a master and slave architecture. Each VIFIB node allocates 100 IPv6 addresses and 100 IPv4 addresses. Each service running in the computer is attached to a dedicated IPv4 address, as well as a globally routable IPv6 address. All services are interconnected across VIFIB nodes using "tunnels" that redirect local IPv4 to global IPv6, encrypt flows and redirect IPv6 to IPv4.

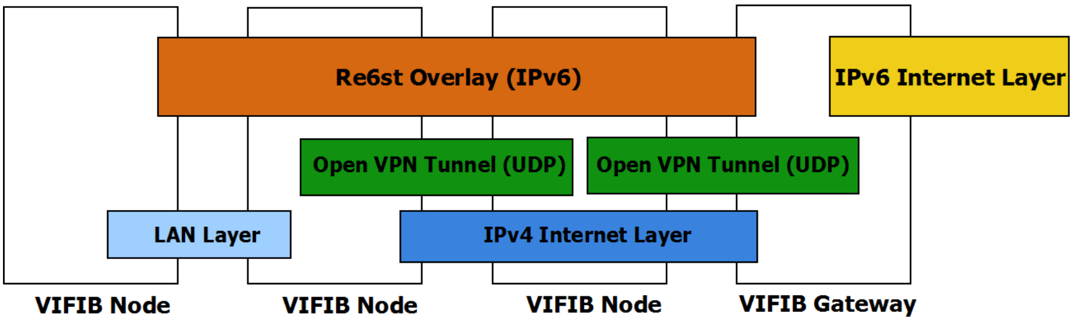


Figure 48. Architecture of the re6st overlay.

Two services running in different locations, compatible with IPv6 or not, can be interconnected through a secure link. Tunnels between computers change every 5 minutes. To minimize latency, VIFIB implements a strategy that uses the best possible tunnels according to a heuristic. The least used tunnels are the ones replaced first but always maintaining a certain compromise between resilience and low latency. This secure, resilient and low latency overlay network is called re6st [Re6st] (see Figure 48). The re6st overlay creates a mesh network of OpenVPN tunnels on top of several IPv6 providers and uses the Babel protocol to calculate the best routes. Babel [Babel] is a distance-vector routing protocol based on the Bellman-Ford algorithm. The re6st overlay organizes nodes in a flat random graph that constructs a robust network structure with a small diameter in order to minimize the latency between nodes. Since the routing tables of the overlay are under the control of the VIFIB system, the overlay can recover faster from a link failure than other algorithms used by Internet providers.

The system should ensure high privacy and resilience in order to avoid losing connectivity. Despite its robustness, re6st still presents some weakness related mainly to scalability and security. The Babel protocol relies on periodic routing table updates which result in high traffic generation. Moreover, no hierarchical routing is considered so the routing table size could be huge, especially in large scale environments. Another aspect is related to security: a malicious node could flood the network with bad routes and lead to routing problems. Tunnels are created randomly; therefore, despite their ability to achieve resilience, they are not used in an efficient manner. They might consume extensive resources even if they are not being used. In this work, we address the scalability and dynamic issues in the VIFIB system and propose to apply RINA to enhance its performance.

4.1.3. RINA to bound Routing Table Sizes for Distributed Clouds

In this section, we investigate the application of RINA to the Distributed Clouds, and more specifically, the VIFIB system. We will focus on the system architecture from a routing and addressing point of view. In the following, we will address the global architecture of DIFs ensuring the exchange of data between the DAPs and we introduce our solutions for routing and addressing: Scalable Forwarding with RINA (SFR), in a first step. Then, we present our second contribution that addresses the challenge of the dynamic nature of Distributed Clouds: Small world network overlay.

SFR: Scalable Forwarding in RINA

SFR is a solution that has been introduced in the Deliverable 3.2 [D3.2]; in this deliverable we give an overview of this approach and present the new evaluation results that were published in [SFR].

Proposed Approach

In Distributed Clouds, an increasing number of users would affect the performance of the cloud services, as resources are very limited while the requirements from the applications are growing. Accordingly, a solution to support scalability is needed in such a large scale environment. In this work, we propose to adopt the divide and conquer concept inspired by RINA in order to have a hierarchy of smaller clouds provide connectivity between the pairs of the system in an efficient way.

The main idea of our proposal is to divide the cloud into groups or regions. These groups are created and managed by the authorities based on a specific criterion, e.g. the group size, the country and/or the ISP membership. Furthermore, connectivity between the groups is ensured by inter-connecting a set of VIFIB nodes of each group. This set of VIFIB nodes, namely "Group Leaders", is elected to act as relays between the groups and to form specifically what we call the inter-groups. At the same time they preserve their membership to their original groups. In order to further scale, this "logical" organization could be repeated recursively adding other levels that will be forming a logical hierarchy. To avoid link failure problems due to the bandwidth limitation of the nodes, several VIFIB nodes could be elected from the same region as Group Leaders, providing more resilience. The way these Group Leaders are chosen needs further investigation.

Figure 49 illustrates an example of two levels of Inter-Groups hierarchy. GroupS is the group where the originating VIFIB node A belongs. Group D is the group containing the destination VIFIB node (F). Figure 50 represents this scenario in RINA logic. We assume that for each region a DIF is created to manage connectivity inside the group. Consequently, Each VIFIB node has at least one IPC Process in the groups of the overlay (the lower level of the hierarchy). Some of the VIFIB nodes that we called "Group Leaders" will also have IPC Processes in the inter-groups on the upper logical levels apart from the IPCPs belonging to the Group DIF. Suppose that VIFIB node A in Group S is the source node and node H in Group D is the destination. Node A has one IPCP connected to the Group S DIF which connects to node B that acts here as the Group Leader. Accordingly, VIFIB node B has one IPCP within Group S and one additional IPCP within InterGroup1_1 that connects it to node C in the scope of the InterGroup 1_1. Node C has three IPCPs: One within Intergroup 1_1, one within InterGroup2_N and at the same time one within Group1 in the lower group DIFs. Node E has two IPCPs: One within Intergroup 1_N at level 1 allowing connection with the node D. Moreover, it has in particular one in GroupD where the destination VIFIB node F belongs. Node E will use the Group D DIF to reach directly the destination.

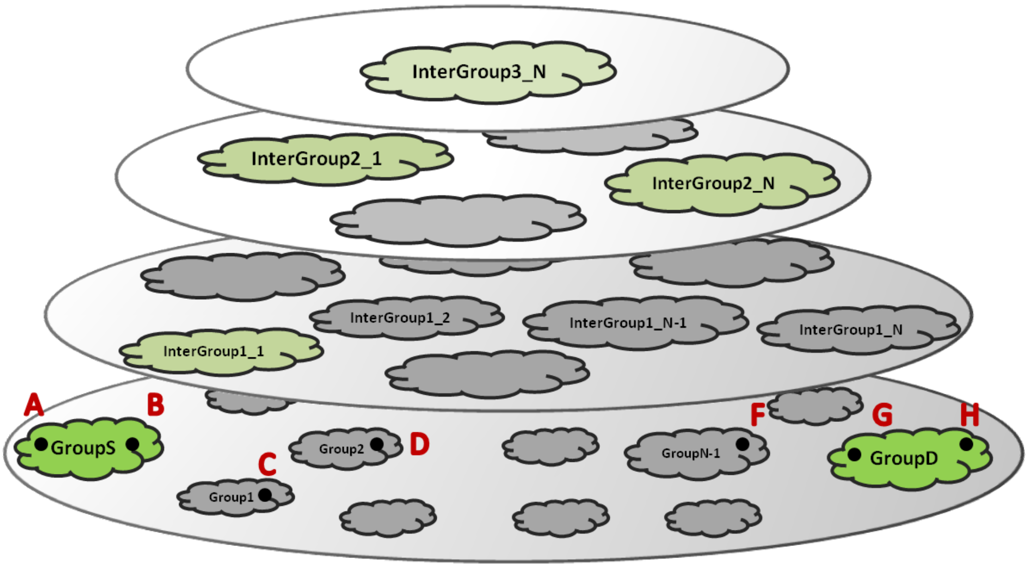


Figure 49. Example of SFR hierarchy with three levels.

In Figure 50, we illustrate the DIF architecture of the overlay cloud in the considered example. Services provided by the distributed cloud system are deployed using "App-DAFs". An App-DAF is a collection of Distributed Application Processes (DAPs) that will be sharing information (DAP 1 and DAPN in the example). These DAPs use specialized overlay "Tenant App-DIFs" that are tailored to the needs of the App-DAFs. A Tenant App-DIF is designed to connect DAP1 and DAP2 in order to support their communication process. On the other hand, the tenant Cloud DIF is designed to adapt to the dynamic network connectivity. Especially, for Distributed Clouds where VIFIB nodes could act as border routers and at the same time as customers applications, the tenant Cloud DIF ensures scalability and flexibility. It maintains a global view of the network to dynamically manage the possible suppression/appearance of the lower DIFs structure which could be very frequent in the Distributed Clouds scenario. At the bottom, each group is mapped to a DIF which is created to manage connectivity inside the region.

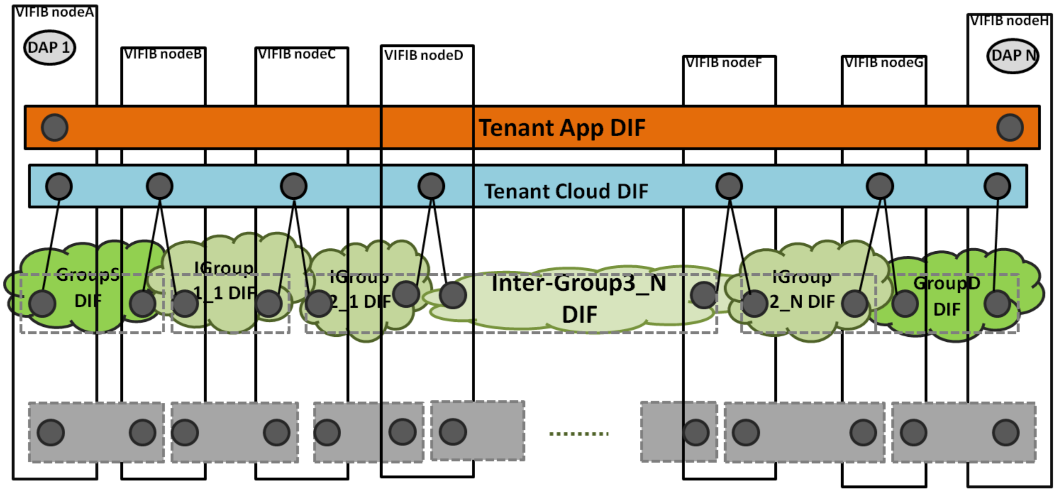


Figure 50. DIF Architecture.

To summarize, there are four types of DIFs:

- Tenant App DIFs: DIFs that provide the direct connectivity between hosts. Mainly they are used by the customers of the Distributed Cloud system. These DIFs are directly supported over a tenant Cloud DIF.
- Tenant Cloud DIFs: Medium-sized DIFs that provide connectivity between VIFIB nodes from different regions. These DIFs could be created dynamically on demand in order to adapt to the frequent change in the network connectivity.

- Inter-group DIFs: Small DIFs that provide connectivity to Group Leaders of some Group DIFs.
- Group DIFs: Small DIFs that provide high connectivity and low latency between the VIFIB nodes of the same small region.

Performance Evaluation

In this section, we evaluate the performance of our SFR scheme for Distributed Clouds. We assess the benefits of the application of RINA to the VIFIB System in terms of limiting the routing table size. Moreover, we perform a comparison of SFR with a simple Distance Vector routing protocol in order to show how it outperforms the current routing architecture that the VIFIB System is using. In the following, we introduce the simulation setup and present the results of our experiments.

Simulation Scenario

We have conducted a set of experiments to analyse the performance of our proposal. We have used RINASim. It is a simulation platform implementing the RINA architecture in Omnet++. It is intended to enable the study of RINA architecture and also to perform simulation experiments with RINA applications. Figure 51 shows the scenario that we set up in RINASim. It consists of a medium size network of 120 nodes, divided into four regions, each containing 30 VIFIB nodes including the Group Leader. All the nodes inside the regions are randomly randomly interconnected and connected to the Group Leader. All the Group Leaders are interconnected. The DIF architecture is organized as follows:

- A Group DIF is constructed to regroup all the VIFIB nodes inside each region.
- Three inter-group DIFs are designed to interconnect region 1/2, region 2/3 and region 3/4.
- A cloud tenant DIF contains all the nodes that are communicating.

In RINASim, several policies have been implemented in order to handle routing within RINA networks. For example, "SimpleDV", a distance vector routing policy, is used in the scope of each DIF. In this scenario, we consider that VIFIB nodes use a ping application to communicate where the maximum packet size is set to 1500 bytes.

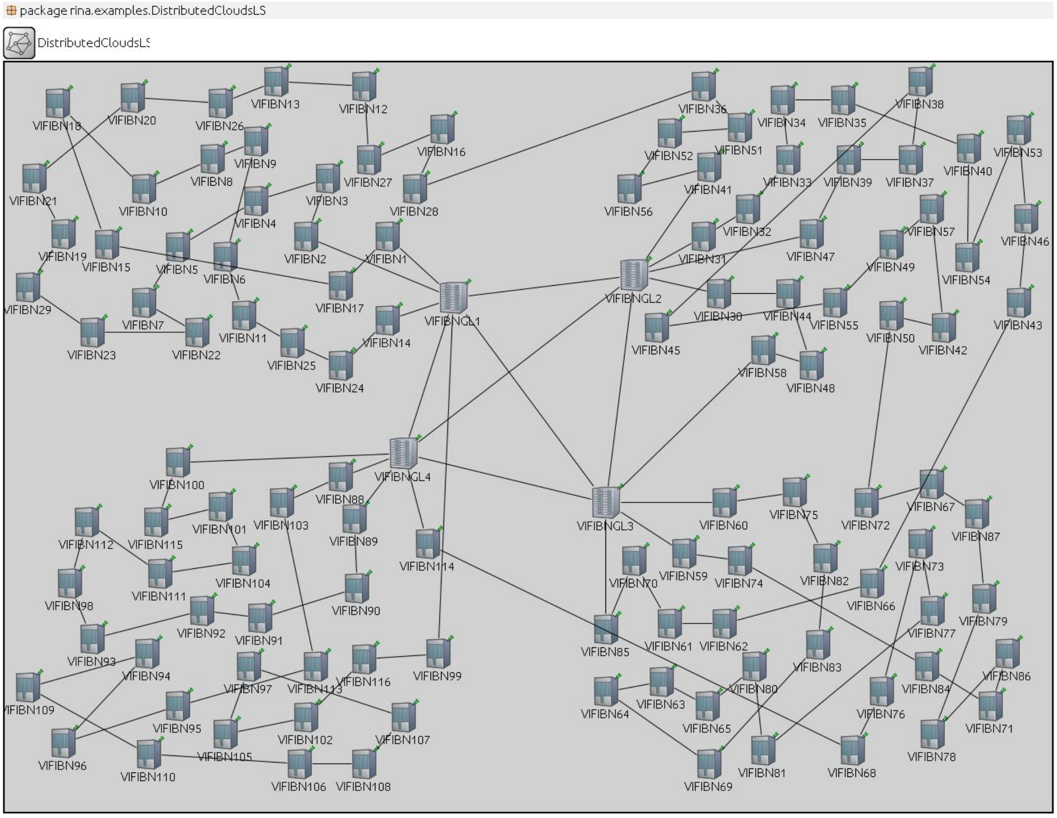


Figure 51. Distributed Clouds simulation scenario.

Simulation Results

In this section, we demonstrate the benefit of applying RINA to Distributed Clouds with results obtained by simulations. Figure 52 illustrates the PDU forwarding table size with regards to the simulation time. It provides a comparison of SFR with a distance vector routing protocol.

The distance vector routing protocol used in this comparison is similar to the routing protocol used in the re6st architecture of the VIFIB distributed cloud system. We observe that SFR shows better results. In case of SFR, as expected, the routing table size does not exceed around 30 entries which corresponds to the number of VIFIB nodes in the regions. Only Group Leaders will have additional entries corresponding to the links between other Group Leaders covering the inter-group DIFs. We can see that in case of the distance vector protocol, the PDU forwarding table size grows to around 120 entries corresponding to the whole network size. The benefit shown here is mainly due to the beneficial impact of the use of RINA in the forwarding scheme and especially its "divide and conquer" strategy that helped to bound the routing table size.

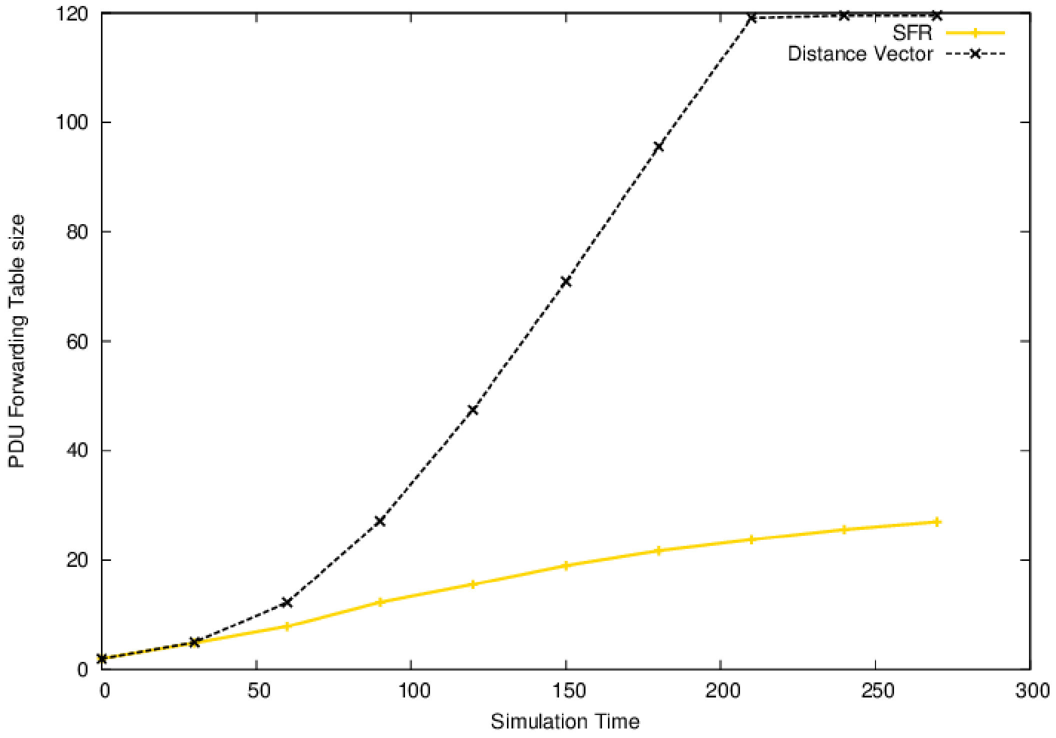


Figure 52. The variation of the PDU forwarding table size over time.

We have also assessed the dynamic creation of tenant cloud DIFs. Figure 53 depicts the PDU forwarding table size in tenant cloud DIFs considering several flows (involving 5, 12 and 21 nodes). The size of the forwarding table is plotted with respect to the simulation time. We can see from this figure that the size of the forwarding table is proportional to the number of nodes constructing the flow. Only nodes participating in the communication have entries in the forwarding table of the tenant cloud DIF. The red line in the figure represents the results for a flow between only five nodes, so we can observe that at the end of the simulation the forwarding table of each node is filled with only five entries. So, only nodes that actively communicate appear in the forwarding table. Accordingly, we can efficiently manage the use of cloud DIFs as they are created when needed and on demand. We conclude that dynamically managing the tenant DIFs better limits the forwarding table size and thus achieves more flexibility and scalability.

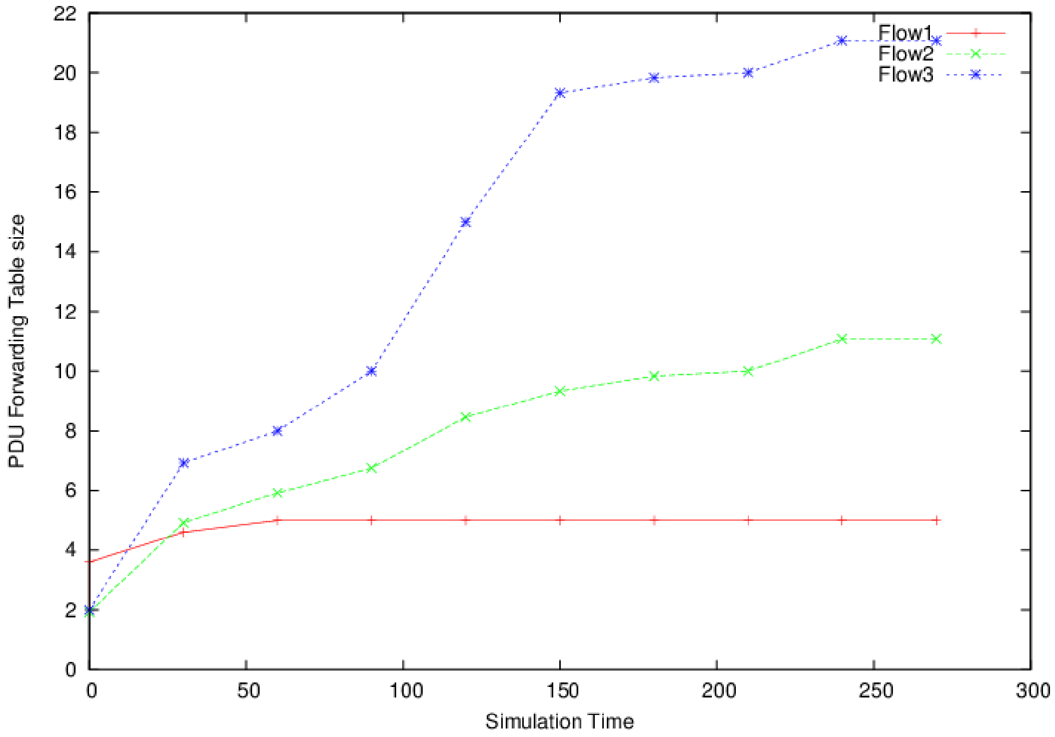


Figure 53. The variation of the PDU forwarding table size over time.

Small World Overlay Architecture for Efficient Forwarding in RINA

In the last section, we addressed the scalability challenge of the Distributed Clouds use case. There is still another issue that needs to be considered as well: VIFIB clouds are very dynamic as each node can leave/join the network whenever it wants. This is supposed to have a great impact on routing and thus on applications. In this section, we address the need to manage this dynamic behaviour while ensuring scalability at the same time.

Proposed Approach

In this section, we introduce a new solution for efficient forwarding in Distributed Clouds. It aims at constructing a small-world network overlay. Efficient forwarding is ensured by building a small-world-like structure in the cloud. Small-world networks have the advantage of ensuring a low average path length and a very high clustering degree. This implies small latency between nodes and more efficient routing. This architecture should be adaptive to the dynamic nature of the Distributed Clouds.

Small world property

In small-world networks [Milgram][Watts], most pairs of nodes have the shortest path between them. Moreover, thanks to the high clustering degree, each node in the network is connected to some neighbouring nodes. This is defined by long and short links as depicted in Figure 54. Long links define the connection between "distant" nodes. Short or "Cluster" links identify the nodes that are "closely" connected to each others and regroup them into clusters. Distributed Clouds naturally adopt the "small-world" behaviour in order to improve the forwarding decisions in such an environment. While the application of small-world graphs to networks is by itself not novel [Tie][Tianbo][Ken], these systems require to maintain a reasonably homogeneous cluster size (often achieved by dynamically creating or destroying clusters when nodes join or leave). In RINA, however, it makes sense to map clusters to DIFs in order to fully benefit from its recursive nature (e.g. for management). We therefore plan an investigation of the overhead associated with such dynamic DIF creation / destroying as future work.

In order to construct a small-world architecture, we use the Chinese Whisper algorithm [Biemann] to create these clusters.

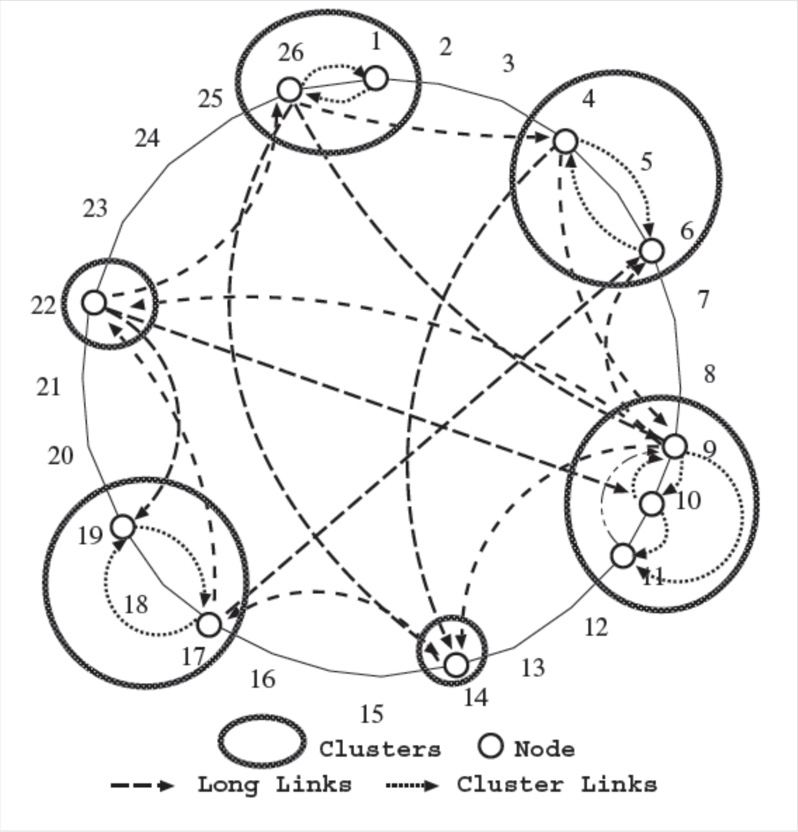
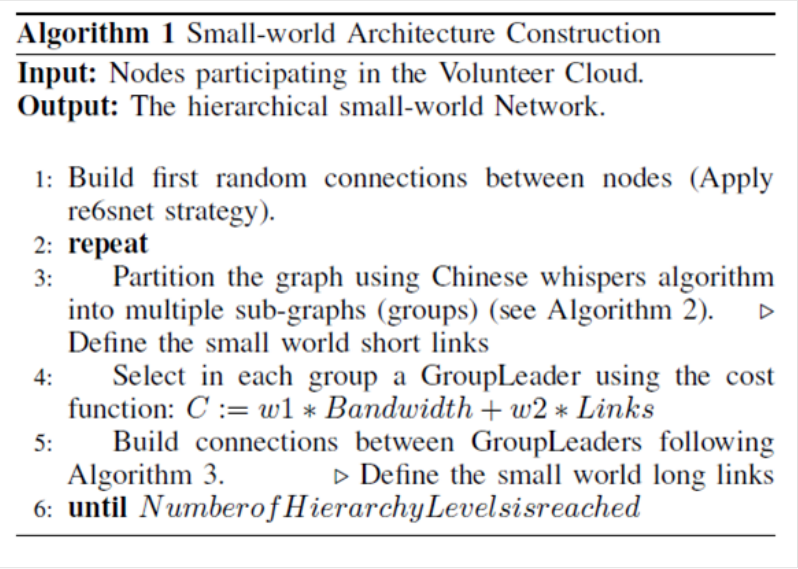


Figure 54. A small-world topology.

Building the Small-World Network

To come up with an effective routing solution for volunteer clouds, the scalability requirement has to be taken into account. Here we propose an algorithm that fulfills this requirement. First, featuring the small-world network properties, our algorithm can ensure efficient routing as short links between most pairs in the cloud is guaranteed. Second, hierarchy levels are used to ensure scalability. Our proposal is based on the idea of building a hierarchical small-world network. Clusters are formed by applying the Chinese Whispers algorithm, which will build short small-world links. Moreover, using a cost function, long links are built between the Group Leaders. Figure 55 shows the algorithm of our small-world based approach. To build a small-world-like topology, our algorithm first constructs random connections between the nodes (this is approximately the concept used within the re6st protocol). Then the partition algorithm Chinese Whispers (see Algorithm 2) is applied to build the groups in the overlay. Once we have these clusters, an election procedure is triggered to elect the Group Leader for each cluster. A cost function is used depending on the node capacity (bandwidth) as well as its number of links. The long links are then created between the Group Leaders.

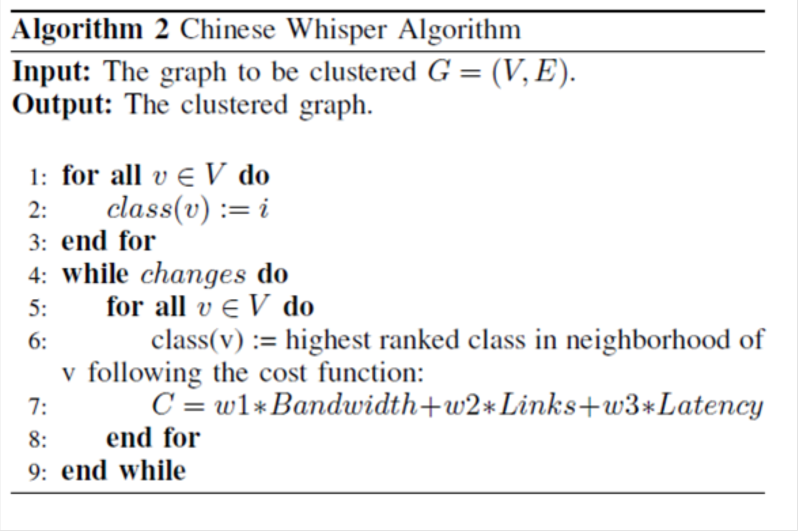


```
Algorithm 1 Small-world Architecture Construction
Input: Nodes participating in the Volunteer Cloud.
Output: The hierarchical small-world Network.

Build first random connections between nodes (Apply re6snet strategy).
repeat
    Partition the graph using Chinese whispers algorithm into multiple sub-graphs (groups) (see Algorithm 2).  ▷ Define the small world short links
    Select in each group a GroupLeader using the cost function: $C := w1 * Bandwidth + w2 * Links$
    Build connections between GroupLeaders following Algorithm 3.  ▷ Define the small world long links
until $NumberofHierarchyLevelsisreached$
```

Figure 55. The small world Architecture Construction.

Chinese Whisper (see Figure 56) is a randomised clustering algorithm that clusters undirected, weighted graphs. The advantage of this algorithm is that it has a time-linear (to the number of edges) complexity [Biemann]. Also it showed good performance when applied to a small-world like network [Biemann]. The algorithm starts by assigning classes to each node in the graph (lines 1-2). Then, for a given number of local updates, each node inherits the predominant class in its neighbourhood (lines 4-8) according to a cost function. Originally, the algorithm runs until it reaches a stable state where there are no changes in the clustering result (line 4). However, a fixed number of iterations can also be defined.



Algorithm 2 Chinese Whisper Algorithm

Input: The graph to be clustered $G = (V, E)$.
Output: The clustered graph.

```
1: for all $v \in V$ do
2:     $class(v) := i$
3: end for
4: while $changes$ do
5:     for all $v \in V$ do
6:         class(v) := highest ranked class in neighborhood of
           v following the cost function:
7:         $C = w1 * Bandwidth + w2 * Links + w3 * Latency$
8:     end for
9: end while
```

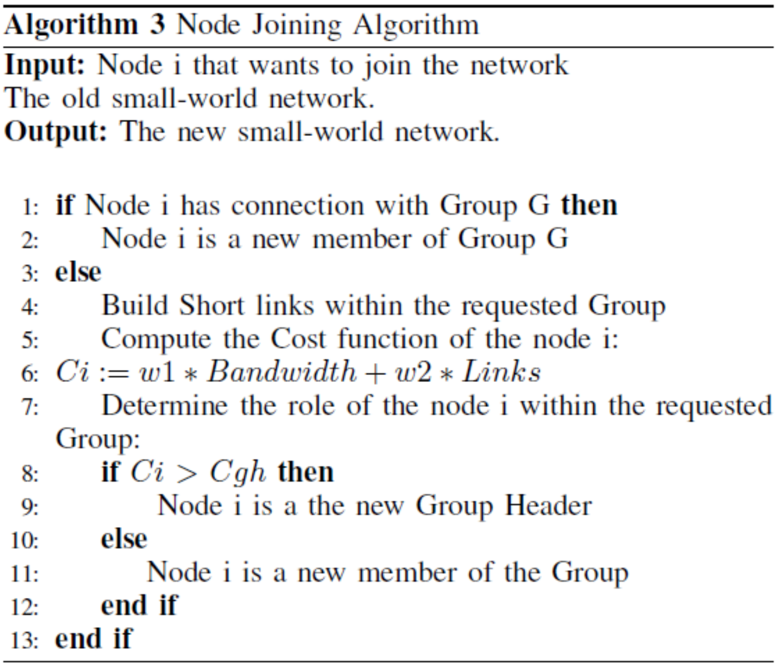
Figure 56. The Chinese Whisper Algorithm.

Dynamic Behaviour in Distributed Clouds

One key characteristic of Distributed Clouds that has to be considered is the high churn of its topology, i.e., nodes can join and leave the cloud at any time. In order to meet these requirements, we propose an algorithm that is intended to manage the forwarding architecture in a dynamic manner. The different groups and the Group Leaders' election are adapted according to the topology change. So we need to consider nodes leaving, joining and node failure in the overlay network.

- Node Joining

The algorithm of this procedure is illustrated in Figure 57. When a new node wants to join the network, the algorithm makes sure to maintain a small-world structure of the network. If the node has already at least one connection with any node in the network, it will be assigned to the group where this given node belongs. Otherwise, new short links should be built with the selected group or cluster. Then, the role of the new node should be designed (it is either a Group Leader or not). This is done by computing the cost function as in Algorithm 1. If the new node has the more significant cost function, it is elected as a new Group Leader, otherwise, it is defined as a member of the cluster.



Algorithm 3 Node Joining Algorithm

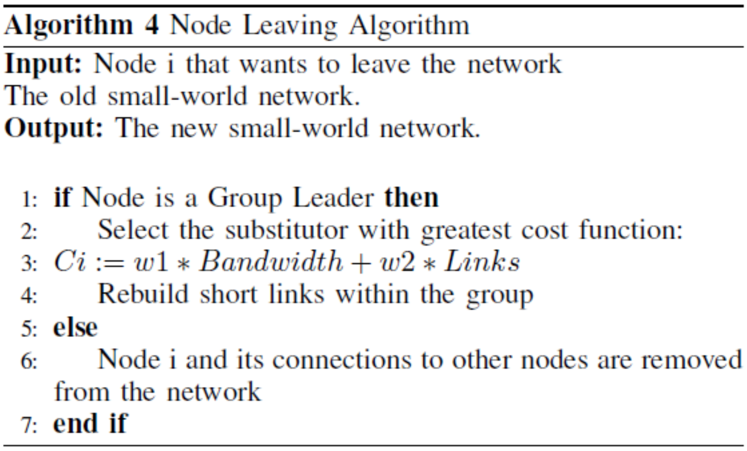
Input: Node i that wants to join the network
The old small-world network.
Output: The new small-world network.

```
if Node i has connection with Group G then
    Node i is a new member of Group G
else
    Build Short links within the requested Group
    Compute the Cost function of the node i:
$Ci := w1 * Bandwidth + w2 * Links$
    Determine the role of the node i within the requested
Group:
    if $Ci > Cgh$ then
        Node i is a the new Group Header
    else
        Node i is a new member of the Group
    end if
end if
```

Figure 57. The Node Joining Algorithm.

- Node Leaving

The procedure of leaving the small-world network is illustrated in Algorithm 4. Again, we try to keep the structure of small world whenever there is a change in the network. If the node is just a member within the group without extra functions, the procedure will be performed without any impact on the group. However if the leaving node is the Group Leader, a "substitutor" must replace it in the group. The node with the greater cost function will do the job.



Algorithm 4 Node Leaving Algorithm

Input: Node i that wants to leave the network
The old small-world network.
Output: The new small-world network.

```
if Node is a Group Leader then
    Select the substitutor with greatest cost function:
$Ci := w1 * Bandwidth + w2 * Links$
    Rebuild short links within the group
else
    Node i and its connections to other nodes are removed from the network
end if
```

Figure 58. The Node Leaving Algorithm.

Performance Evaluation

In this section, we provide some preliminary evaluations of our small-world scheme for managing forwarding in Distributed Clouds. We perform a comparison of our approach with a simple random topology scheme that is inspired by the re6st concept (of the VIFIB system). In the following, we introduce the simulation scenario and provide a simulation result.

Simulation Scenario

In our simulations, we have used RINASim. The scenario used is depicted in Figure 59. It is a small scenario composed of 10 nodes. First, the links between nodes are built in a random way following the VIFIB system principle. Our small world algorithm is then applied to build the clusters and elect the groups leaders. Figure 56 shows that there are 4 elected Group Leaders. A ping application is used to create flows between nodes. 10 random flows are triggered and the end-to-end latency is measured.

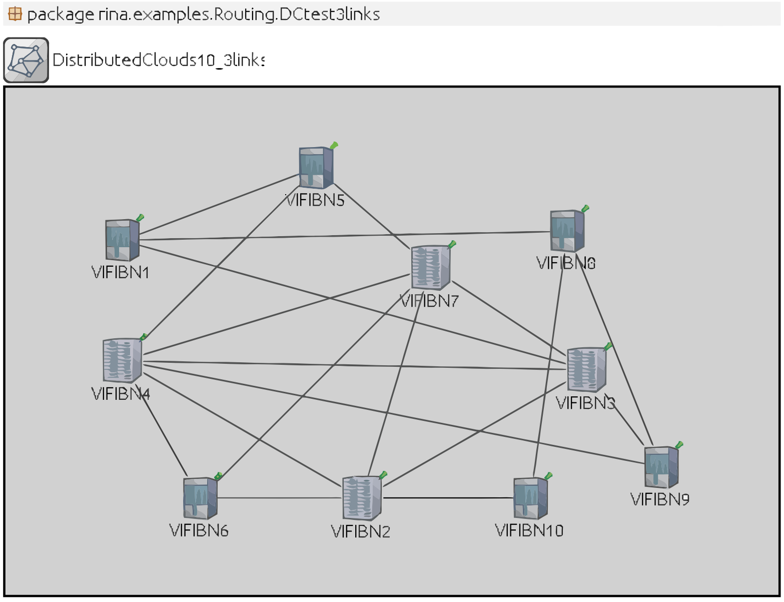


Figure 59. Simulation Scenario: small network.

We perform a modification in the network in Figure 59 (the configuration of links related to latency), then we re-run our small world algorithm. A new small-world topology is generated (illustrated in Figure 60). We measure the average latency again.

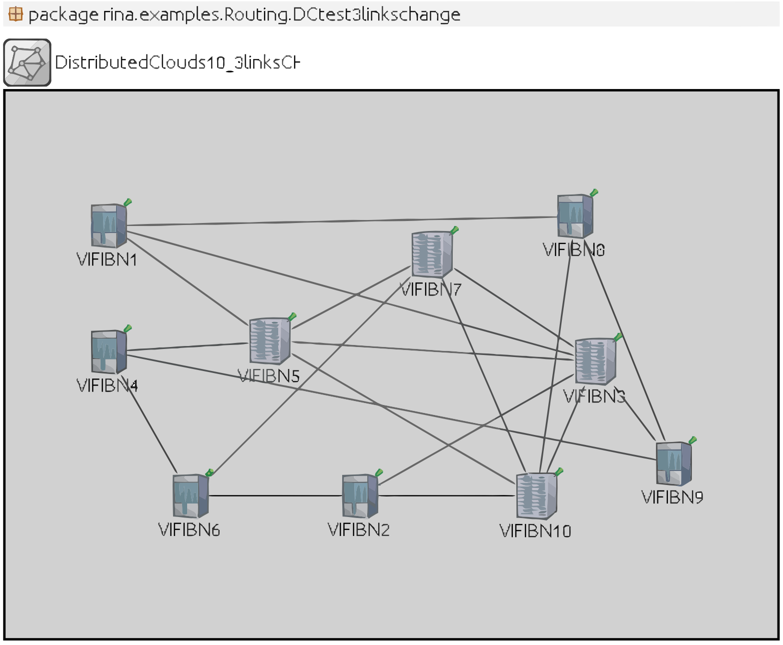

package rina.examples.Routing.DCtest3linkschange
DistributedClouds10_3linksCH
VIFIBN1
VIFIBN8
VIFIBN7
VIFIBN4
VIFIBN5
VIFIBN3
VIFIBN9
VIFIBN6
VIFIBN2
VIFIBN10

Figure 60. Simulation Scenario after change.

Preliminary Simulation Results

Figure 61 illustrates the results obtained after measuring the average latency. We deduce that our small world approach performs better than the random scheme of the re6st protocol. The first two bars correspond to the topology in Figure 56. The second bars represent the average latency for the topology after change (Figure 60). These results prove that our scheme ensures better latency than the random scheme. This shows that small world networks match Distributed Clouds very well.

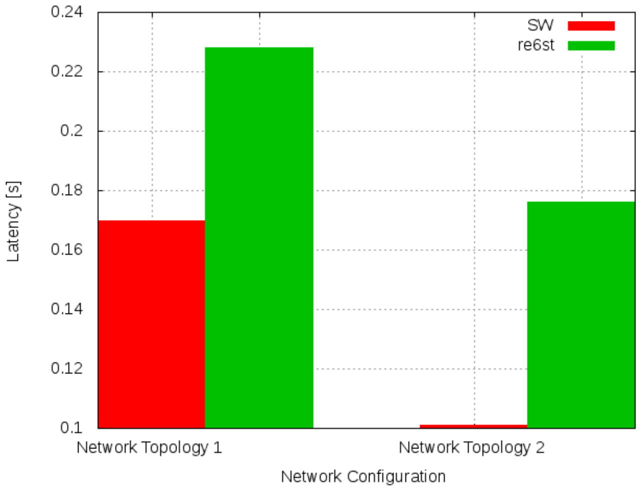


Figure 61. Average latency with respect to network topology.

4.1.4. Conclusions and Future Works

In this chapter, we presented generic architectures for routing and addressing tailored to cope with the Distributed Clouds requirements, most notably in terms of scalability and dynamicity. We have identified the limitations and issues of the currently implemented solutions for the distributed cloud use case (VIFIB). We then described SFR and the small-world architecture - two solutions that we have designed to benefit from RINAs recursive nature. The obtained simulation results show that both schemes achieve their design goals by limiting the routing table size and giving good latency results compared to the concept used currently by the VIFIB system. We plan to work further on the small-world architecture (in particular the dynamic behaviour) and produce more advanced results. Accordingly, in the context of WP7 of PRISTINE, we intend to submit a research paper.

4.2. Topological addressing and routing in large-scale datacentres

4.2.1. Introduction and motivation

Looking for superior efficiency, uptime and scalability, nowadays' commercial Data Centers (DCs) tend to rely on well-defined leaf-spine Data Center Network (DCN) topologies that not only offer low latency and ultra-high bandwidth for server-to-server communications, but also enhanced reliability against multiple concurrent failures. Examples of this reliance are Google's and Facebook's DCN topologies, available in [Arjun] and [Alexey] respectively. Moving toward the future Internet of Things (IoT) and 5G network scenarios, a plethora of emerging innovative cloud services are expected to proliferate. This will put stress upon current DCs, requiring them to grow even larger in terms of computing resources. However, common routing and forwarding solutions do not scale well for these types of networks, resulting in large forwarding tables (at least in the order of several tens of thousands of entries in highly-optimized configurations [Arora]) and a heavy table maintenance burden (information exchanged to populate routing tables and re-converge upon failures). Although this problem was identified a long time ago, the TCP/IP protocol suite—not being designed for cloud networking—limited the improvements that were achievable by the solutions proposed in the literature [Bari]. In contrast to the rigidity of the TCP/IP protocol stack, the clean-slate RINA brings a programmable environment where routing and forwarding policies in forwarding devices can be fully configured by the network administrator. This opens the door to the deployment of policies tightly tailored to the specific DCN characteristics inside a RINA-enabled DC. The RINA based solution outperforms solutions based on TCP/IP, whose protocols were optimized for the delivery of a best-effort Internet with an arbitrary topology—a very different environment to that of a DCN. The policies proposed here make use of DCN topology knowledge to forward packets to the closest neighboring device on the route to their destination based on rules. In the non-failure scenario, this approach only requires the storage of forwarding information to each adjacent neighbor (compared to traditional forwarding tables, which may contain up to one entry per network node). When there are route failures in the DCN, some forwarding rules may not succeed to deliver packets to the destination. This is the only time when additional forwarding information is required, consisting of a few exceptions overriding those rules that are stored at forwarding devices.

4.2.2. Rules and Exceptions

We focus on a RINA deployment inside a DC following the DIF setup depicted in Figure 62 Such a RINA-enabled DCN network is partitioned into three main types of DIFs, each with a different scope: i) a single DC-Fabric DIF, acting as a large distributed switch; ii) a DC DIF that connects all servers in the DC together under the same pool; and iii) multiple tenant DIFs, isolated and customized as per the requirements of the different tenants.

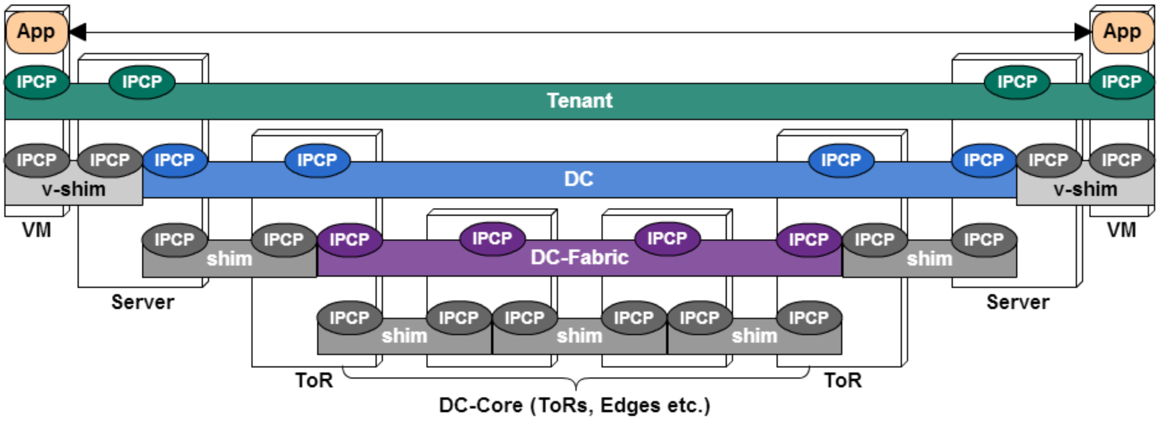


Figure 62. DIF setup inside a DC between Virtual Machines (VMs) running in DC servers. The DC-Fabric DIF (violet color) is the focus of this work.

We focus on the DC-Fabric DIF, that is, the one providing connectivity between the Top-of-the-Rack (ToR) switches and between the edge routers and ToR switches. We considered DC-Fabric DIFs following the topologies of the large Google and Facebook DCNs shown in Figure 63[4] and Figure 64[5]. Google uses a unique plane of spine switches interconnecting all pods and edge planes in the DCN, thus offering multiple equal cost paths between each pair of ToRs and edges, even under multi-failure scenarios (see Figure 63). In the Facebook DCN, fabric switches of a pod connect to a distinct spine set that provides connectivity to all other pods and to edge nodes (see Figure 64). Again, high redundancy is introduced to survive multiple concurrent failures across the DCN.

[4] extracted from reference [Arjun]
[5] extracted from reference [Alexey]

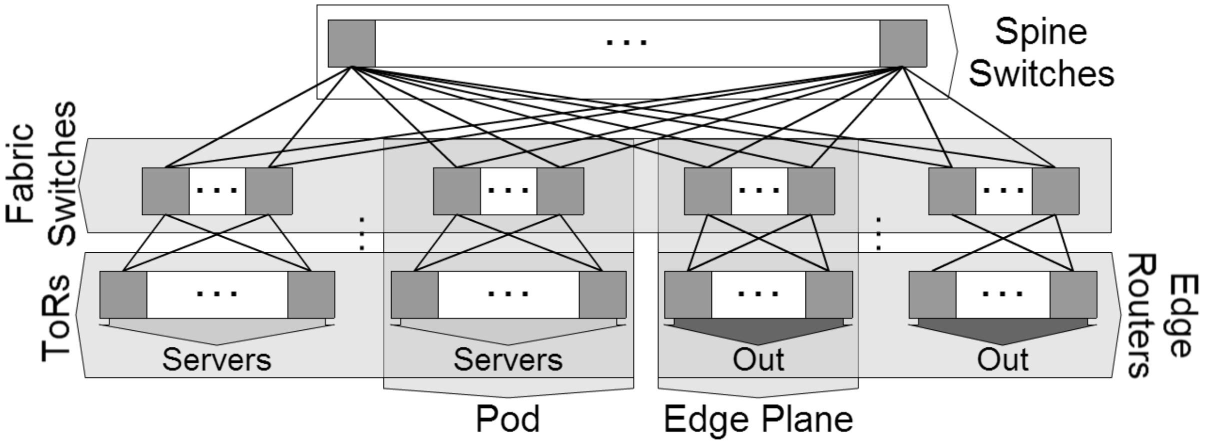


Figure 63. Google’s DCN topology

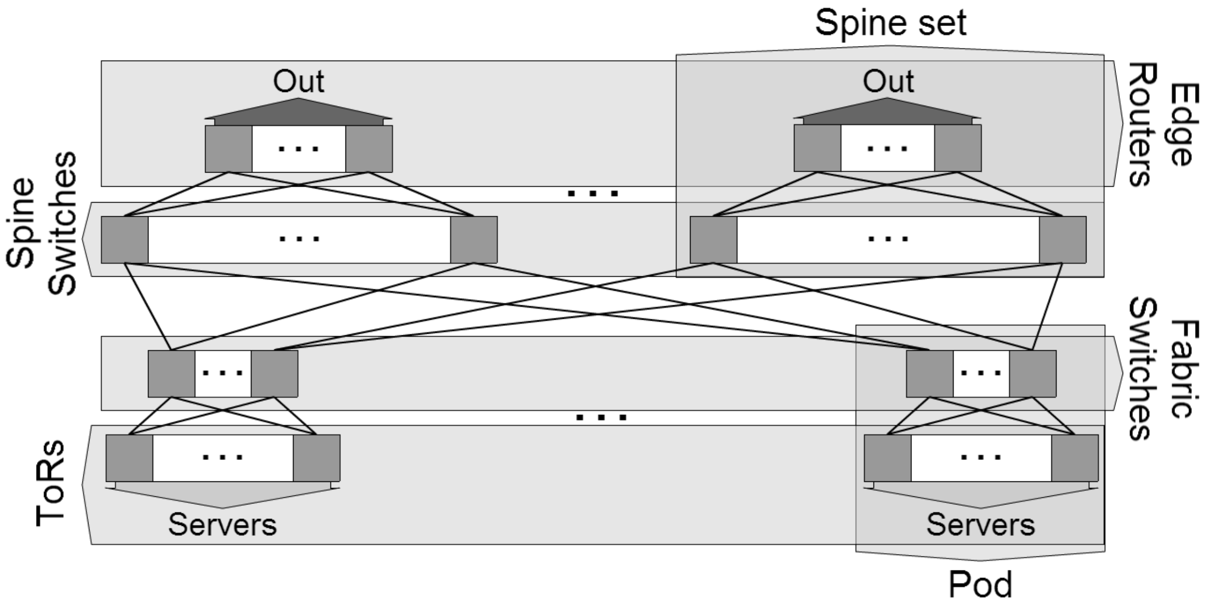


Figure 64. Facebook’s DCN topology,

Forwarding

An interesting point about DC networks is that they follow highly structured topologies. This allows us to describe the full network graph with only few parameters. For example, a DC-Fabric DIF following Google’s DCN topology can be described using only 6 parameters: Number of spine nodes (s), number of pods (P), number of edge planes (E), number of fabric nodes per pod/edge plane (f), number of ToRs per pod (t) and number of edges per edge plane (e). In a similar way, a DC-Fabric DIF following Facebook’s DCN topology can be described using 5 parameters: Number of pods (P), number of fabric nodes per pod and spine sets (f), number of ToRs per pod (t), number of spine nodes per spine set (s) and number of edges per spine set (e).

In order to profit from such well-defined topologies, it is important to assign node addresses so that any node in the network can be located with a minimum extra knowledge. Focusing on a DC Fabric DIF with the same connectivity as that of the Google's DCN, we use addresses in the form <A.B>; where "A" represents the group and "B" the node within this group. For example, having group "0" as that of the spine-plane, address <0,X> refers to the spine X. To differentiate between fabric nodes and ToRs/edges within pods, we reserve the first node ids for the fabric switches (1 to f) and the rest for ToRs and edge routers. For example, in a DC-Fabric with f = 4, address <9.2> would be that of fabric node 2 in pod 9, while address <9.5> would be that of ToR/Edge 1 in pod 9.

Conversely, for a DC Fabric DIF showing the connectivity of the Facebook DCN, we adopt a different approach. In this case, we use addresses in the form <A.B.C>; where "A" represents the type of node (ToR 0, Fabric 1, Spine 2 and Edge 3), "B" the group id (pod or spine-set), and "C" the node id within that group. For example, addresses <0.7.1> and <1.7.1> identify ToR 1 and Fabric 1 both at pod 7, addresses <2.3,1> and <3.3.1> identify Spine 1 and Edge 1 both at spine-set 3.

Being aware of the specific DCN topology (from the set of parameters), and the location of the nodes within it, means only forwarding entries to adjacent neighbours need to be stored at each DCN node. As a result, only simple forwarding rules are required. In order to quickly access neighbour information, we assign a locally unique identifier (Neighbour-Id) to every neighbour, abstracting its real address in a direct way. By using Neighbour-Ids as an index, we can store all neighbour nodes' information, including port address or status, in a direct access structure. Forwarding rules use Neighbour-Ids to define the set of valid neighbours to reach any destination across the DCN.

Given the nature of the communications inside a DC (over the DC-Fabric DIF in a RINA-enabled scenario), only end-to-end flows between ToR switches, and between ToR switches and edge routers will be established, most probably in a dynamic way. Therefore, forwarding rules only need to consider ToR switches and edge routers as possible destinations. This requires only a small set of rules where we either go towards a specific node or set of nodes (e.g. to reach a ToR from a Fabric node we either go down directly to that specific node if we are located in the same Pod, or go up to any Spine node).

These simple forwarding rules work without problem during normal operation, however when failures occur across the DCN, the simple forwarding rules may fail in directing a packet to its destination. Thus, we require additional forwarding rules to handle these failures. We term these additional forwarding rules exceptions. These exceptions are similar to traditional forwarding table entries, but are only required in certain failure scenarios. Like primary forwarding entries, exceptions can be directed not only to specific nodes, but also groups of nodes (e.g., an exception that applies to all Pods in the DCN except a specific one). Moreover, the total number of exceptions required to ensure robust communication in the advent of failures tends to be considerably smaller than the number of similar entries required in a traditional forwarding table (at most the same in the very worst case), as most communication across the DCN will remain unaffected by specific link or node failures. Only storing exceptions to primary rules when failures occur can yield a large reduction in terms of memory usage compared to a traditional forwarding table.

In addition, in Fabric switches, which tend to have a large list of valid neighbours to reach most destinations, we instead store the list of neighbours that are invalid to reach a destination. Forwarding entries in Fabric switches can be interpreted as, “To reach X go UP/DOWN without using Y”.

Given rules, exceptions and information about directly connected nodes, the full forwarding function can be seen in Algorithm: Forwarding function.

Algorithm: Forwarding function

```
Forward(addr) {
  if addr is Connected Neighbour → Forward (addr)
  if addr is not ToR or Edge → Unreachable
  if addr match any exceptions → Exception (addr)
  else → Rule (addr)
}
```

Routing

While the described forwarding policy does not require knowledge of distinct routes when all works well, it does require knowledge of failed routes to destinations and how to alternatively reach them. Hence, the routing policy has to provide enough information to populate such exceptions. While a simple link-state or distance-vector routing protocol could be used to determine exceptions to the primary rules in failure scenarios, we can compute them more efficiently by exploiting the complete DCN topology knowledge that nodes have. Indeed, there is no need for nodes to propagate the state of operational resources across the network, only the state of those experiencing failures. To this end, we propose a link failure routing policy, a variation of link-state routing based on failure propagation. Instead of having all nodes propagate their full neighbour table, only failed links are propagated (the rest are assumed to be working). This results in a large reduction in the information exchanged and stored at network nodes. Although we could use the DCN topology and failed links' knowledge to compute the forwarding exceptions using a Dijkstra's routing algorithm, such an approach has a significant computational cost and does not scale well. Instead, we found that with a list of failures we could restrict our search to problematic locations and compute the exceptions directly—so long as some constraints on valid paths are considered. These constraints on valid paths are required to reduce the complexity of the algorithms anyway. Even so, constrains are thought taking into account that there is a high number of available paths towards any ToR and Edge node, being constrains only a way to limit the depth of the algorithm. Also, it needs to be considered that paths not supported with those constraints would imply to use longer paths, preferring to have unreachable destinations and possible movements of VMs . For example, from a spine switch we only consider a Pod reachable if we are connected to at least one of its fabric switches, and a specific ToR if we can reach it with an optimal path or sub-optimal path with at most 2 extra hops (those within the Pod).

Algorithms to compute exceptions aim to direct the search toward failures that may require an exception while discarding the rest. For example, at spine nodes failures between other spines and fabric nodes are not considered as those do not affect the feasible paths, given the imposed constraints.

While such algorithms are fully dependent on the topology and require some constraints, they yield a significant improvement both in time and memory usage against traditional route computation. Indeed, there is no need to compute and store reachability information to all destination nodes, but only to the ones unreachable through the primary rules.

As an example, Algorithm 2: Pseudo coded exception computation example shows a possible way to compute exceptions in fabric nodes in a Google-based DC from a spine node. Within the pseudo-code, we considered ToR and Edges as the same type of node for the sake of simplicity.

Algorithm 2: Pseudo coded exception computation example

```
Parsed data and functions used:
unreachableGroups ← groups with all fabric neighbours unreachable
NodeFails ← ToR/Edge disconnected from some fabrics
unreachableNodes ← ToR/Edge disconnected from all fabrics
FabricFails ← Fabric with problems reaching some ToR/Edge
Reachable (A.B) ← Check if neighbor A.B can be reached
reachableAtGroup(A) ← Fabrics nodes reachables at group A (A.*)
reachablebNodeFrom (A.B) ← ToRs/Edges reachables from fabric (A.B)
reachablebFabricFrom (A.B) ← Fabrics reachables from ToR/Edge (A.B)

Algorithm:
Exceptions = Ø
if I am disconnected then return Exceptions
GroupsWithProblems = Ø
for each A.B in FabricFails do
 if Reachable(A.B) then GroupsWithProblems.add(A)
for each (A) in GroupsWithProblems do
 validPorts = Ø
 for i = 1..f do
 if Reachable(A.i) and A.i # FabricFails then
 validPorts .add (A.i)
 if validPorts == Ø then unreachableGroups.add(A)
 else Exceptions.add(A, validPorts )
for each (A) in unreachableGroups do E.add(A.0, Ø)
for each (A.B) in unreachableNodes do
 if (A) # unreachableGroups then Exceptions.add(A.B, Ø)
for each A.B in NodeFails do
 if A # unreachableGroups then continue
 if (A.B) # unreachableNodes then continue
 myReach = reachableAtGroup(A)
 dstReach = reachablebFabricFrom(A.B)
 if myReach # dstReach then continue
 if myReach ∩ dstReach != Ø then
 Exceptions.add(A.B, myReach ∩ dstReach)
 continue
 reachDst = Ø
 for each node (A.B’) in dstReach do
 reachDst .add(reachablebNodeFrom (A.B’))
 reachNei = Ø
```

```
  for each node (A.B’) in myReach do
  reachNei .add(reachablebNodeFrom(A.B’))
  validPorts = Ø
  if reachNei ∩ reachDst != Ø then
  validPorts .add(A.B’)
  Exceptions.add(A.B, validPorts )
return Exceptions
```

While Algorithm 2: Pseudo coded exception computation example describes a procedure to compute all the required exceptions at spine nodes, the complexity of it is limited by the number of concurrent failures, rather than the network size. In addition, while this algorithm takes a stateless approach, more efficient versions can take into account the previous status of the network and only compute exceptions that could be affected by the latest changes. For example, if the link between ToR switch <1.5> and fabric switch <1.1> goes down, in most cases we will not recompute all the exceptions to reach any node, but only the exceptions required to reach ToR <1.5> and other ToR switches in that Pod that could already show problems.

All this greatly reduces the complexity of computing exceptions when compared to that of computing traditional forwarding tables, allowing for faster response to network failures.

4.2.3. R/E RINAsim policies

The GitHub repository contains older forwarding policies in "DIF/RMT/PDUForwarding/SimpleDCForwarding" for forwarding on DC-Fabric DIFs following the Facebook’s DC topology. While the policies under development implement the described ideas and some of their improvements in a better way, older versions more simply illustrate the benefits of rule-based forwarding. This policy works together with the forwarding generator policy at "DIF/RA/PDUFG/SimpleDCGenerator" and routing policy at "DIF/Routing/DCRouting". These two policies, while not following the described ideas for generating exceptions, show lower complexity. Instead of exchanging all link information, nodes only exchange failure and recovery information among themselves. Nodes use this information to compute the routing table using a simple Dijkstra algorithm, instead of the one described before.

"Examples/Routing/DDC" (also shown in Figure 65) and "Examples/Routing/BigDC" contain two different examples of DCNs employing a rule-based forwarding and the reaction upon failures. They facilitate the comparison of the first improvements made by these policies, showing a great reduction in the forwarding table size.

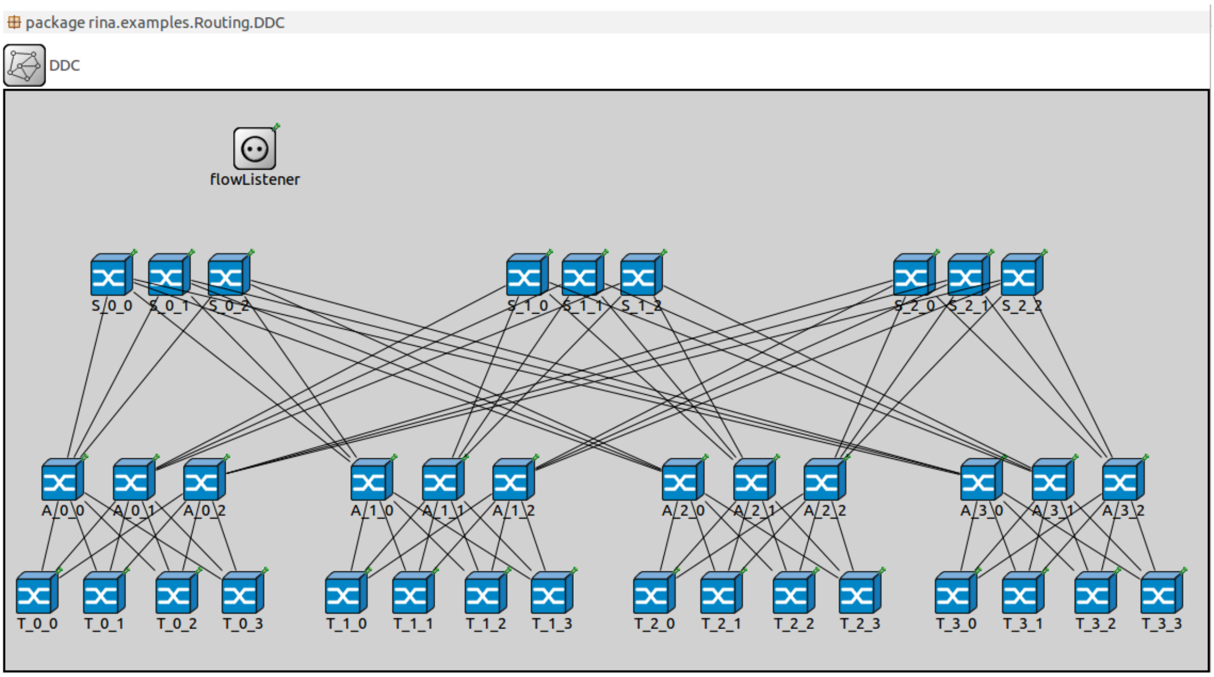


Figure 65. Network of Routing/DDC scenario

4.2.4. R/E scenarios and results

Current routing and forwarding solutions for IP impose many restrictions that the RINA architecture already overcomes. For example, for addressing DCN devices, we are not forced to use 4 or 16-byte addresses as imposed by IPv4 or IPv6 respectively, but can use scenario-specific addresses. Public addresses of servers/VMs are not propagated inside the routing updates, reducing the communication cost of the routing protocol. To show these benefits, we quantitatively evaluate the performance of the proposed routing and forwarding policies against that of currently available solutions for the same purposes but operating in a RINA environment.

To analyse the number and size of traditional forwarding table entries vs. rules plus exceptions in our policies, we have considered two different DC-Fabric DIFs: 1) DIF-Go (reproduces the Google's DCN topology), and 2) DIF-FB (reproduces that in the Facebook's DCs). Table 2 give the parametrization of both DIFs, taking the number of pods (P) as the base parameter. The expressions to determine the rest of parameters (as a function of P) allow us to obtain similar configurations as those reported for the real DCNs.

Table 2. DETAILS OF THE DCN-FABRIC DIFS

Pods (P)	P
ToRs per pod (t)	P/2
Fabric switches per pod/edge plane (f)	Log3(P)
DIF-Go - Edge planes (E)	P/4
DIF-Go - Edge routers per edge plane (e)	P/2
DIF-Go - Spine switches (s)	P
DIF-FB - Edge routers per spine set (e)	P*P/8f
DIF-FB - Spine switches per spine set (s)	P/2
Total number of ToRs	**P*P/2**

First, we compared the number of entries in a forwarding table against the number of neighbour entries plus exceptions for large-scale DCNs in distinct failure scenarios. We fix P=100 for the first tests, resulting in DCNs of 5000 ToR switches. We perform our tests for 0, 1, 2, 5 and 10 concurrent failures (randomly chosen among node and link failures). In Table 3 we can see the relative number (in %) of required entries in each case against the total number of DCN nodes. With our policies (named as Exceptions in the table), we require at most one exception per failure (as expected), thus, most of the stored entries are related to adjacent neighbors. With traditional forwarding tables (FWT), we assume that ToR and edge addresses can be aggregated at pods and edge planes/spines. After performing 50.000 tests for each number of failures, we found that the number of entries can be lowered by 11 to 18% in a traditional forwarding table (with respect to the size of the DCN). With our approach, however, we get table sizes of 0.21 to 0.27 %, being most of the stored entries for directly connected network nodes.

Table 3. AVG. ENTRIES VS. MAX ENTRIES (%) GIVEN N FAILURES

Method \ Failures	0	1	2	5	10
Exceptions	0.21	0.22	0.23	0.24	0.27
DIF-FB	**11.4**	**12.1**	**12.8**	**14.8**	**18.1**

DIF-Go	11.6	11.9	12.3	13.2	14.8
DIF-FB	**11.4**	**12.1**	**12.8**	**14.8**	**18.1**

We are also interested in comparing the amount of forwarding data stored and how the policies scale. We focus on both the number of entries and the number of port references stored in those entries. Using the same parametrization based on the number of pods (P), for values 32, 64, 128 and 256, we tested both policies in scenarios with between 1 and 10 concurrent failures. In addition, for forwarding tables, we limited the number of stored port references to 16, being a common limit in ECMP implementations. Figure 66 shows the average number of entries and stored ports in DIF-Go and DIF-FB for different P sizes (number of pods in the DCN). In the Figure 66, we can see how, the number of forwarding entries and their size grow steadily with P. In contrast, our proposed solution only needs to store adjacent neighbours' information plus exceptions, remaining the number of forwarding entries almost constant as the size of the DIFs grows up.

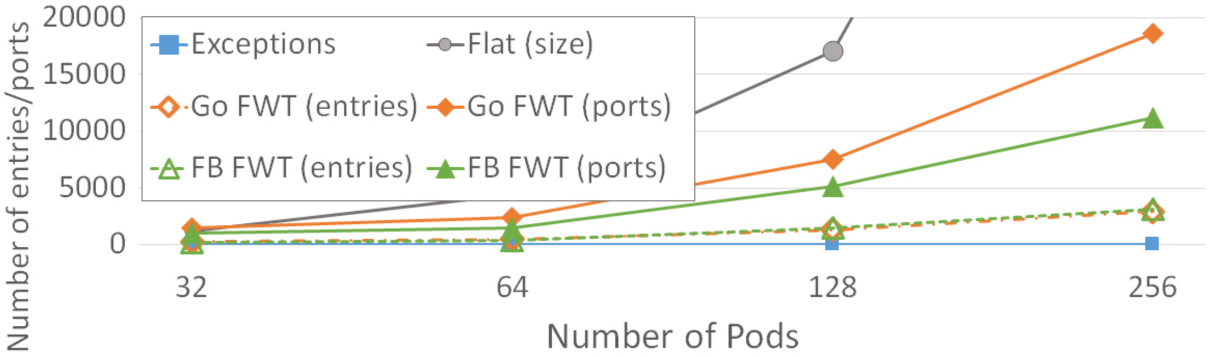


Figure 66. Avg. number of entries and stored ports given the number of pods

4.2.5. Conclusions and future work

The programmability of RINA opens the door to routing and forwarding functionality improvement in front of current solutions for IP, even when traditional forwarding tables are used. Moreover, it also allows new and more efficient policies to be used, more closely related to the real network graph and requirements. The proposed policies leverage previous knowledge about the topology in order to reduce both the data storage requirement at nodes and the communication cost to share the required routing information. The obtained results show really good improvements, usually only needing to store information about nodes that are directly connected, something that should be stored in any case.

Anyway, we still envision room for further improvement. Firstly, while new policies have been presented, we found some similarities between them and existing solutions that we could exploit to provide more generic solutions applicable to a wider range of network scenarios. Instead of policies entirely tailored to specific scenarios, we could design them more generically, letting only some (configurable) parts of them to be scenario dependant. A clear example can be a forwarding policy based on rules and exceptions. This policy is very similar to any forwarding policy that makes use of a simple forwarding table to find the first match between a destination address and a forwarding entry. In our case, though, we can introduce the grouping of neighbour nodes and the use of rules when no matchings are found. Therefore, a generic forwarding policy based on groups, exceptions and rules could be indistinctly used, e.g., to perform simple flat routing or even the topological routing strategies presented and evaluated in this section. Having a generic solution would not only reduce the complexity of adapting it to different scenarios, but also provide a base for compatible hardware implementations, a requirement for fast forwarding performance.

More generic forwarding policies also will allow research on whether centralizing part of the routing functionality can improve network performance, reduce the communication cost, and move part of the computation burden from forwarding devices. The performance of such solutions could then be compared to that of existing solutions like SDN, without the limitations imposed by TCP/IP.

References

[Alexey] Alexey Andreyev, "Introducing data center fabric, the next-generation Facebook data center network", available online at: https://code.facebook.com/posts/360346274145943/introducing-datacenter-fabric-the-next-generation-facebook-data-center-network/

[Al-Fares] Al-Fares, M. Radhakrishnan, S. Raghavan, B. Huang, N. Vahdat, Amin, "Hedera: Dynamic Flow Scheduling for Data Center Networks" NSDI. Vol. 10. 2010.

[Arjun] Arjun Singh, et al., "Jupiter Rising: A Decade of Clos Topologies and Centralized Control in Google's Datacenter Network". In SIGCOMM, London, United Kingdom, August 2015.

[Arora] D. Arora, T. Benson, J. Rexford, "ProActive routing in scalable datacentres with PARIS". In DCC 2014, Proceedings of the 2014 ACM SIGCOMM Workshop on Distributed Cloud Computing.

[Awduche] D. O. Awduche, "MPLS and traffic engineering in IP networks" Communications Magazine, IEEE, vol. 37, no. 12, pp. 42-47, 1999.

[Awduche-Agogbua] D. O. Awduche and J. Agogbua, "Requirements for traffic engineering over MPLS" 1999.

[Babel] J. Chroboczek, "The babel routing protocol", RFC 6126 (Experimental). Inter-net Engineering Task Force,, 2011, available online at:http://www.ietf.org/rfc/rfc6126.txt.

[Bari] M. F. Bari, R. Boutaba, R. Esteves, L. Z. Granville, M. Podlesny, M. G. Rabbani, Q. Zhang, M. F. Zhani; "Data Center Network Virtualization: A survey"

[Biemann] Biemann, C. (2006): "Chinese Whispers - an Efficient Graph Clustering Algorithm and its Application to Natural Language Processing Problems". Proceedings of the HLT-NAACL-06 Workshop on Textgraphs-06, New York, USA.

[DCTCP] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan, "Data Center TCP

(DCTCP)", in Proc. ACM SIGCOMM, New Delhi, India, 2010, pp. 63–74.

[DynSyst] S. Sternberg, Dynamical Systems. Courier Corporation, 2010.

[D3.2] Pristine Consortium, "Initial specification and proof of concept implementation of techniques to enhance performance and resource utilization in networks".

[EyeQ] "EyeQ: Practical Network Performance Isolation at the Edge", imalkumar Jeyakumar, Stanford University; Mohammad Alizadeh, Stanford University and Insieme Networks; David Mazières and Balaji Prabhakar, Stanford University; Changhoon Kim and Albert Greenberg, Windows Azure, 10th USENIX Symposium on Networked Systems and Implementation.

[Holyer] J. Holyer, "A Queueing Theory Model for Real Data Networks." Pages 59–70 of: Thomas, N., & Bradley, J. (eds), UK Performance Engineering Workshop. July 2000

[I2] Internet2 website, available at http://www.internet2.edu

[Ken] Ken Y. K. Hui, John C. S. Lu and DavidK.Y.Yau, "Small World Overlay P2P Networks". IWQOS 2004. The 12th IEEE International Workshop on Quality of Service, 2004.

[LGfn] P. F. Verhulst, "Notice sur la loi que la population poursuit dans son accroissement", Correspondance mathématique et physique, vol. 10, pp. 113–121, 1838.

[LGm] M. Welzl, "Scalable Performance Signalling and Congestion Avoidance". Norwell, MA, USA: Kluwer Academic Publishers, 2002.

[Milgram] Milgram S. "The small world problem". Psychol. Today 2, p60-67. 1967

[Re6st] "Nexedi re6st resilient overlay mesh network", https://www.erp5.com/NXD-re6st.Two.Page.

[Riggio] Roberto Riggio, Francesco De Pellegrini, and Domenico Siracusa, "The Price of Virtualization: Performance Isolation in Multi–Tenants Networks", in Proc. of IEEE ManFI 2013.

[RINA-ACC] Peyman Teymoori, Michael Welzl, Stein Gjessing, Eduard Grasa, Roberto Riggio, Kewin Rausch, Domenico Siracusa: "Congestion Control in the Recursive InterNetworking Architecture (RINA)", IEEE ICC 2016, Kuala Lumpur, Malaysia, 23-27 May 2016.

[SFR] F.Hrizi, A.Laouiti, H.Chaouchi. "SFR: Scalable Forwarding with RINA for Distributed Clouds", NOF 2015, 6th International Conference on the Network of the Future, Sept. 30 2015-Oct. 2, 2015, Montréal, Canada.

[Simulink] The Mathworks, Inc., Natick, Massachusetts: MATLAB SIMULINK version 8.7 (R2016a) (2016)

[SplitTCP] J. Border, M. Kojo, J. Griner, G. Montenegro, and Z. Shelby, "Performance Enhancing Proxies Intended to Mitigate Link-Related Degradations", RFC 3135 (Informational), Internet Engineering Task Force, Jun 2001. [Online]. Available: http://www.ietf.org/rfc/rfc3135.txt

[Tianbo] Tianbo Lu and Binxing Fang and Yuzhong Sun and Xueqi Cheng and Li Guo, "Building Scale-Free Overlay Mix Networks with Small-World Properties", IEEE Computer Society, 2005.

[Tie] Tie Qiu, Diansong Luo, Feng Xia, Nakema Deonauth, Weisheng Si, and Amr Tolba. 2016. "A greedy model with small world for improving the robustness of heterogeneous Internet of Things". Comput. Netw. 101, C (June 2016)

[Vifib] "Vifib web page", http://www.vifib.com/.

[Watts] Watts D. and Strogatz S. "Collective Dynamics of small world networks". Nature Vol 393. Pp 440-442. 1998.

A. Annex A

Here, we present the title and then, the content of the papers.

Congestion Control

- [paper1] Congestion Control in the Recursive InterNetworking Architecture (RINA), (published in IEEE ICC 2016, Kuala Lumpur, Malaysia, 23-27 May 2016)
- [paper2] Even Lower Latency, Even Better Fairness: Logistic Growth Congestion Control in Datacenters (accepted for publication at IEEE LCN 2016)
- [paper3] Feedback in Recursive Congestion Control, (accepted for publication at the 13th European Workshop on Performance Engineering, EPEW 2016)

Resource Allocation

- [paper4] Assuring QoS Guarantees for Heterogeneous Services in RINA Networks with DeltaQ, (under review)

Topological Addressing

- [paper5] SFR: Scalable Forwarding with RINA for Distributed Clouds, (published in Network of the Future (NoF) Conference, 2015)
- [paper6] Benefits of Programmable Topological Routing Policies in RINA-enabled Large-scale Datacenters, (accepted for publication at IEEE Globecom 2016)