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

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:

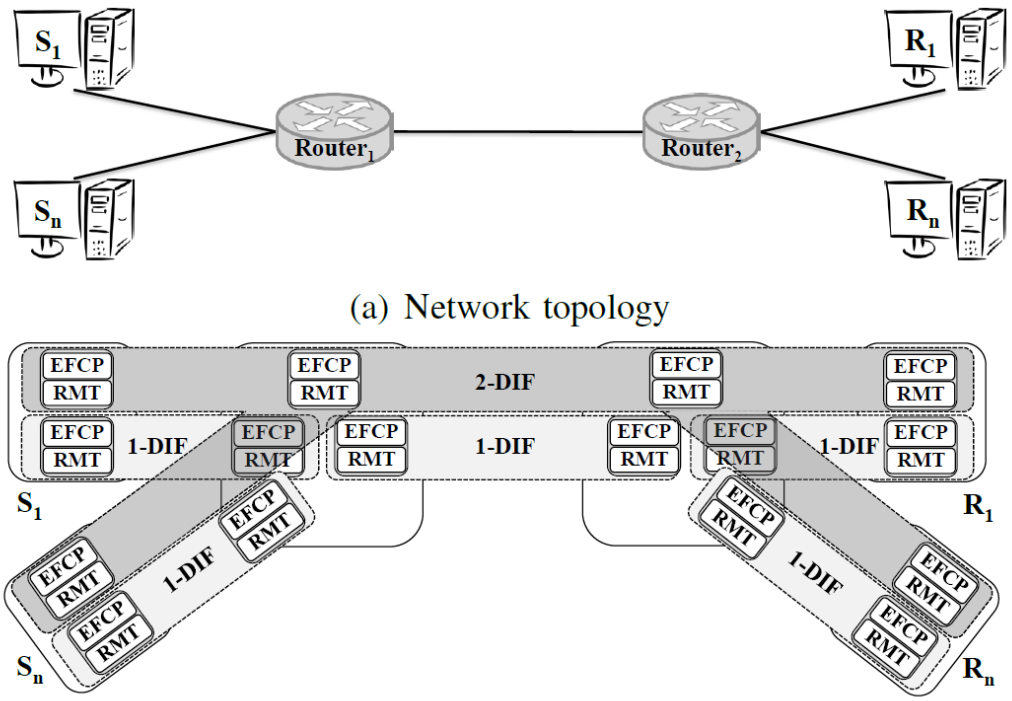
- Horizontal: Consecutive DIFs (i.e., side by side, not above each other). This is very similar to a PEP function called Connection Splitting [SplitTCP]. TCP splitters divide TCP connections by "lying" to the end systems, acting as the receiver towards the sender and as the sender towards the receiver.
- 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.



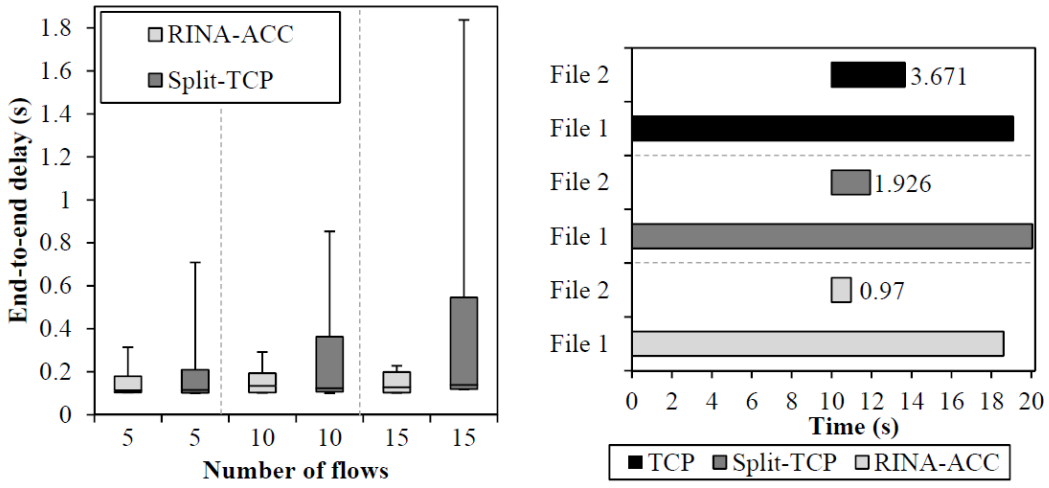
(a) Network topology

(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. $\mathbf{A} = (a_{ij})$ is called the community matrix where the value of a_{ij} determines the competitive effect of species $_i$ on species $_j$. We define $a_{ii} = (2\ S\text{-}1)/S$ and $a_{ij} = (S\text{-}1)/S$.

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, l^ is equal to the percentage of ECN signals a source gets during the n th interval. We show that l^ 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.

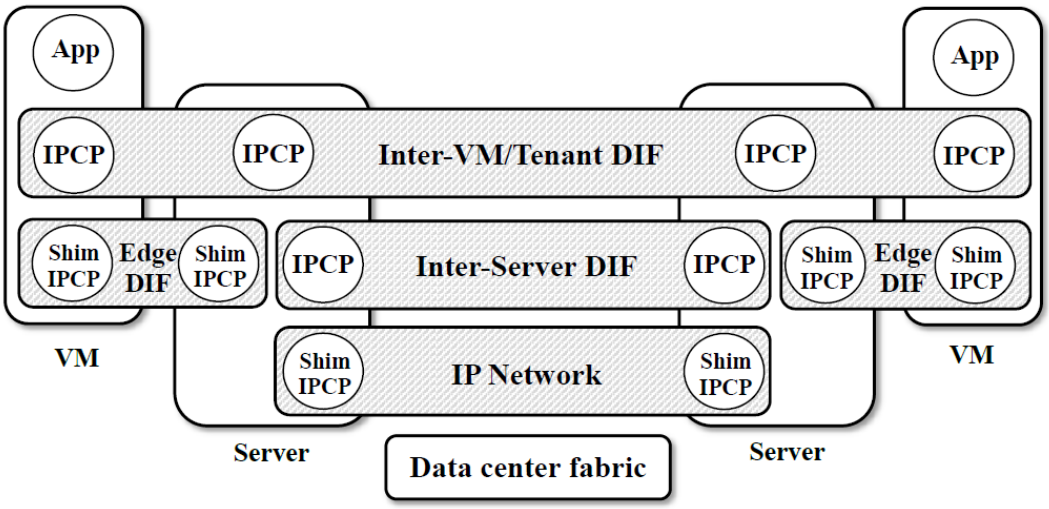


Figure 5. RINA as an overlay in a datacenter

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]).

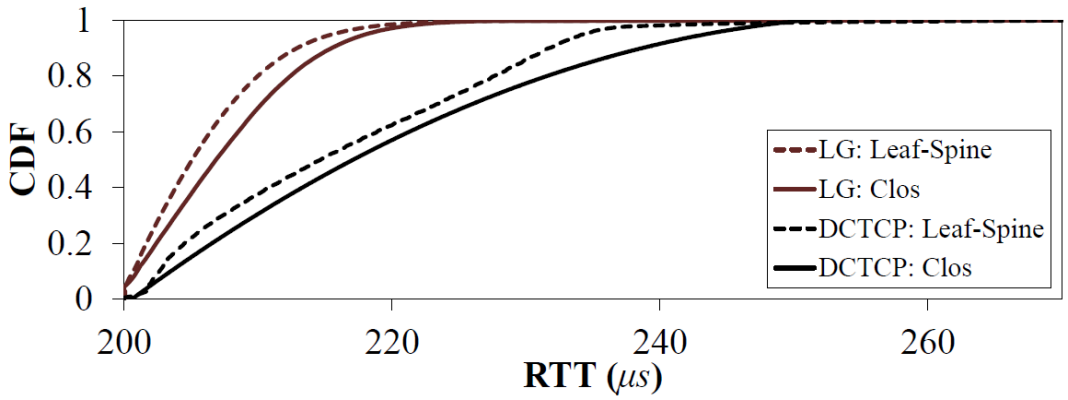


Figure 6. Cumulative distribution function of the RTT distribution under Clos and leaf-spine topologies

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].

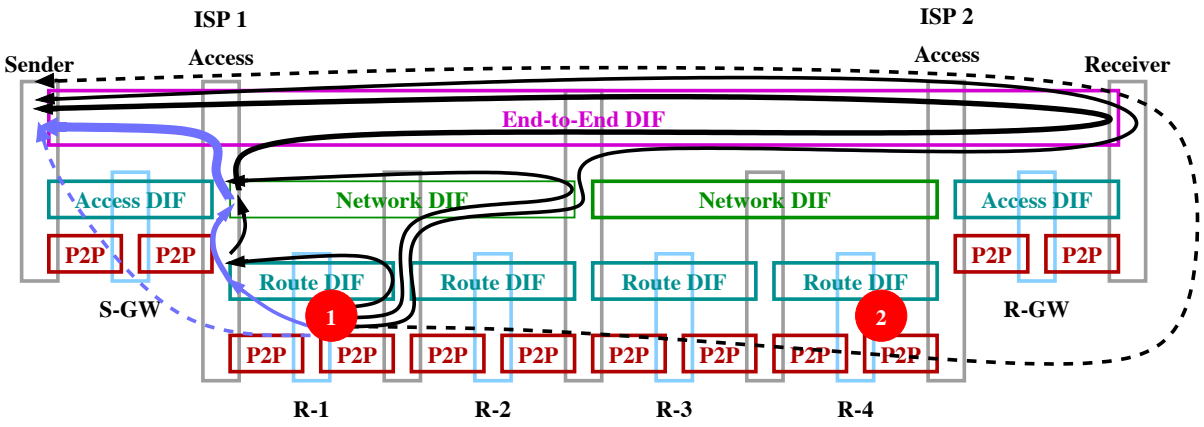


Figure 8. Feedback mechanisms being investigated

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

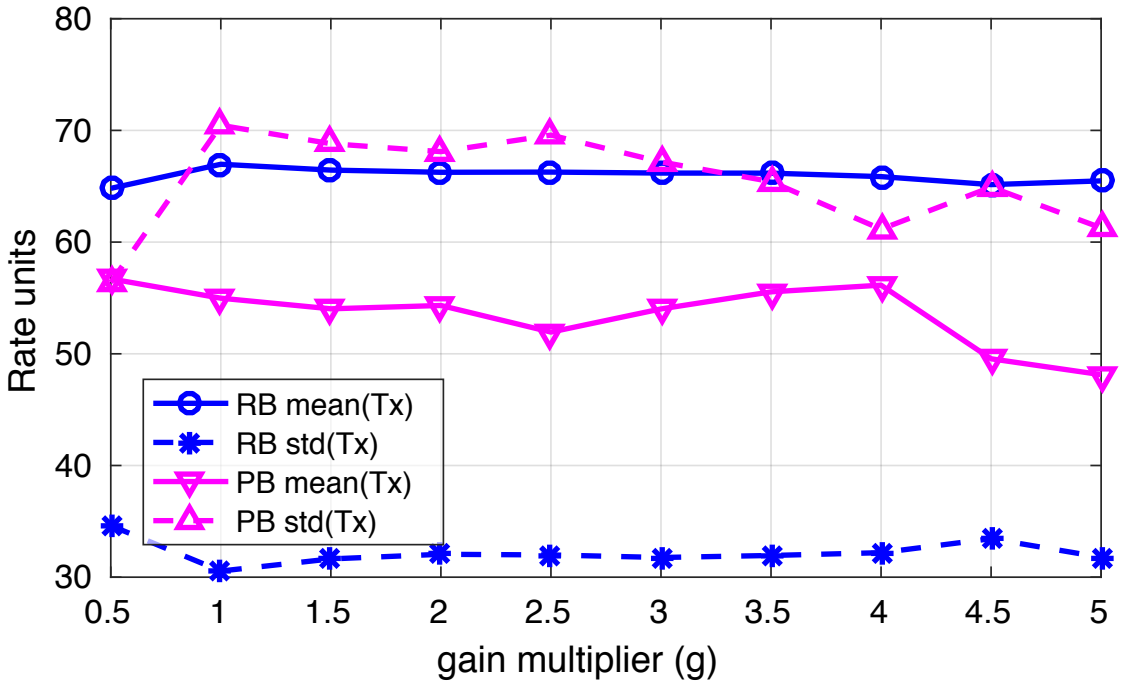


Figure 9. Comparison of the stability of the recursive backward feedback mechanism (RB) and the pure push-back mechanism (PB) when the control gains of all DIFs except the sender are varied

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.

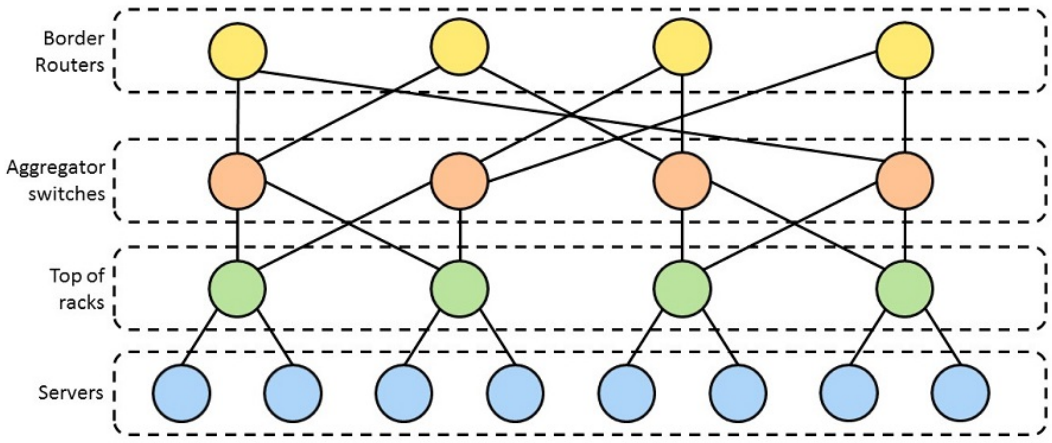


Figure 10. DC Organization

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).

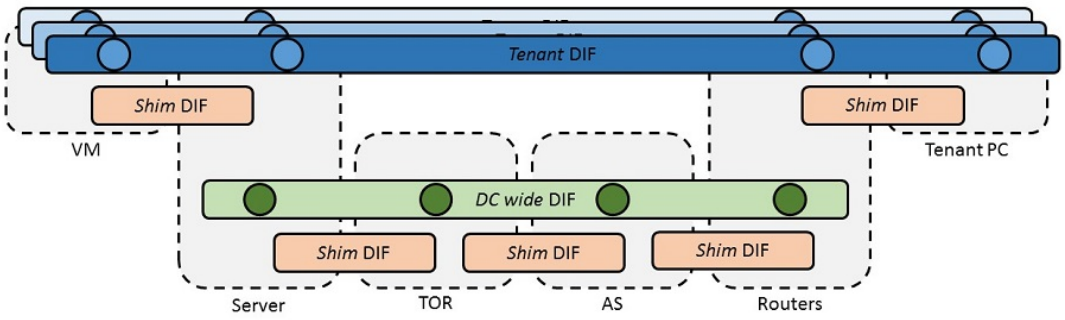


Figure 11. RINA layer organization within the DC.

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.

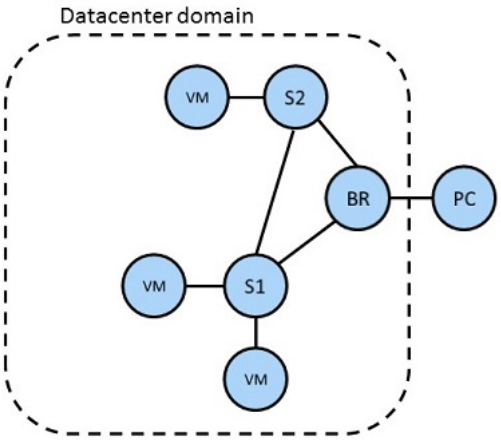


Figure 12. Network perceived by the tenant.

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.

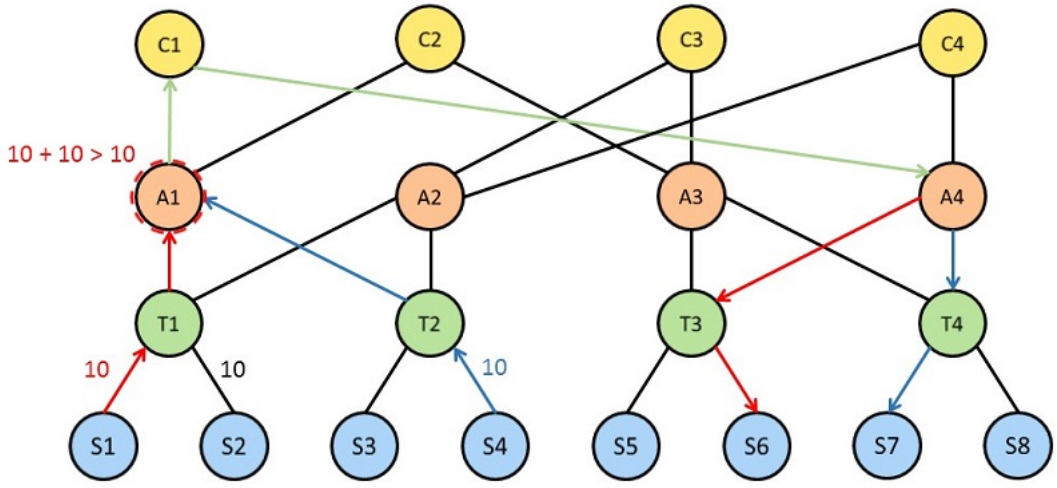


Figure 13. Two flows (red and blue) routed on the same path will cause congestion at A1. From that point on the PDU proceeding towards their destination will carry the ECN mark, which will cause the flow to react to the congestion in order to mitigate it. The green arrow indicates where the flows share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

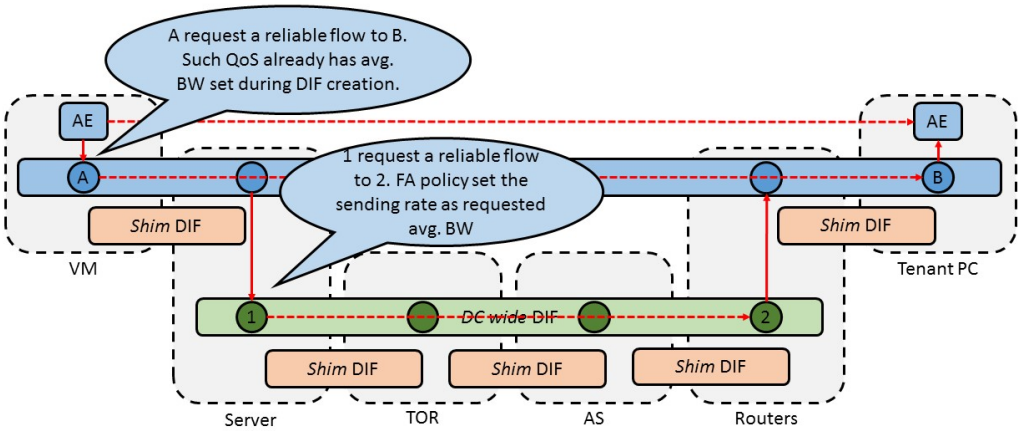


Figure 14. How the MGB is ensured to tenants at DC-DIF level flows.

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.

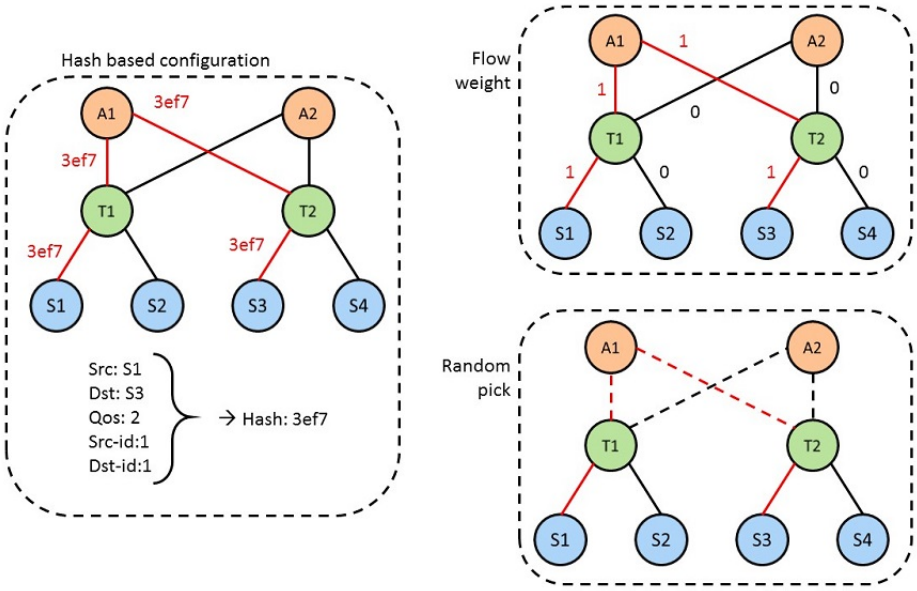


Figure 15. How different forwarding strategies handle the same flow request: hash based strategy will always route on the same port; flow weight will always choose the least loaded port; random pick will just select a random port selected from the valid ones.

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.

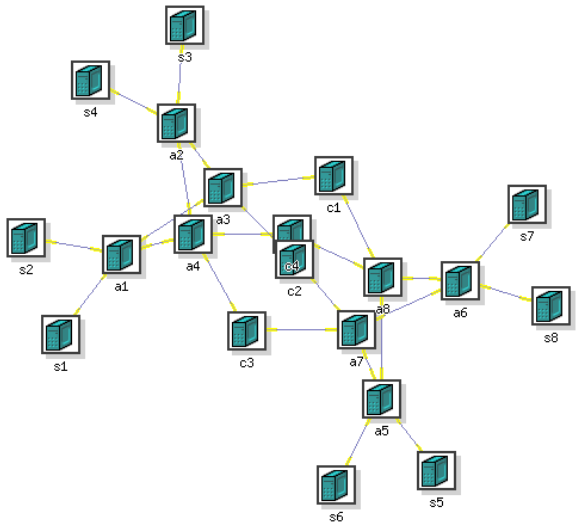


Figure 16. Configuration of nodes used in the Virtual Wall testbed.

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.

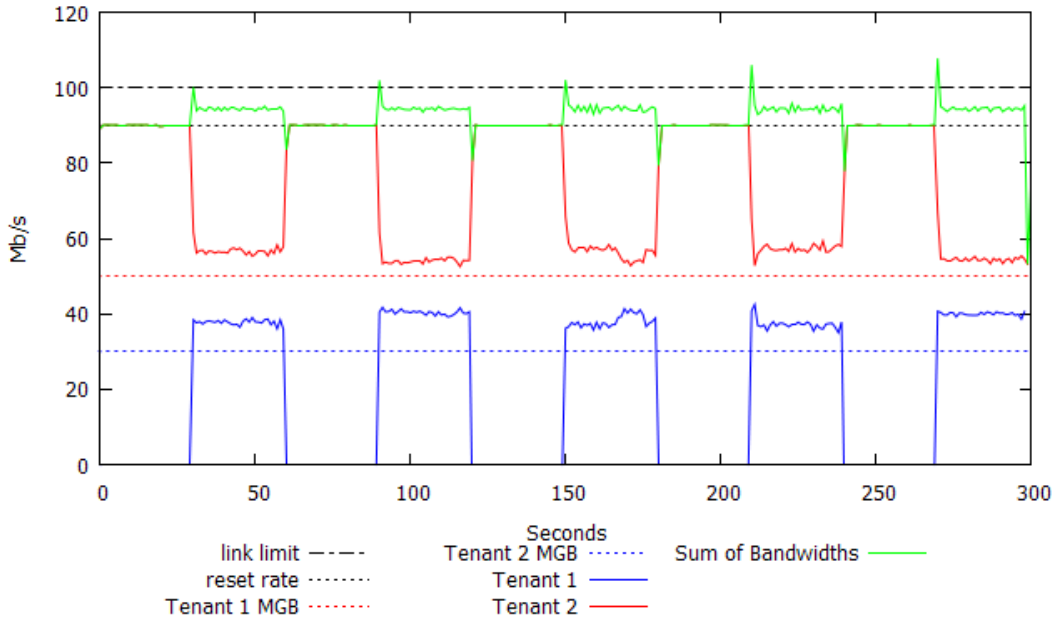


Figure 17. Experimental results. The continuous red and blue lines are tenants' instantaneous used bandwidths in Mb/s over time (seconds).

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.

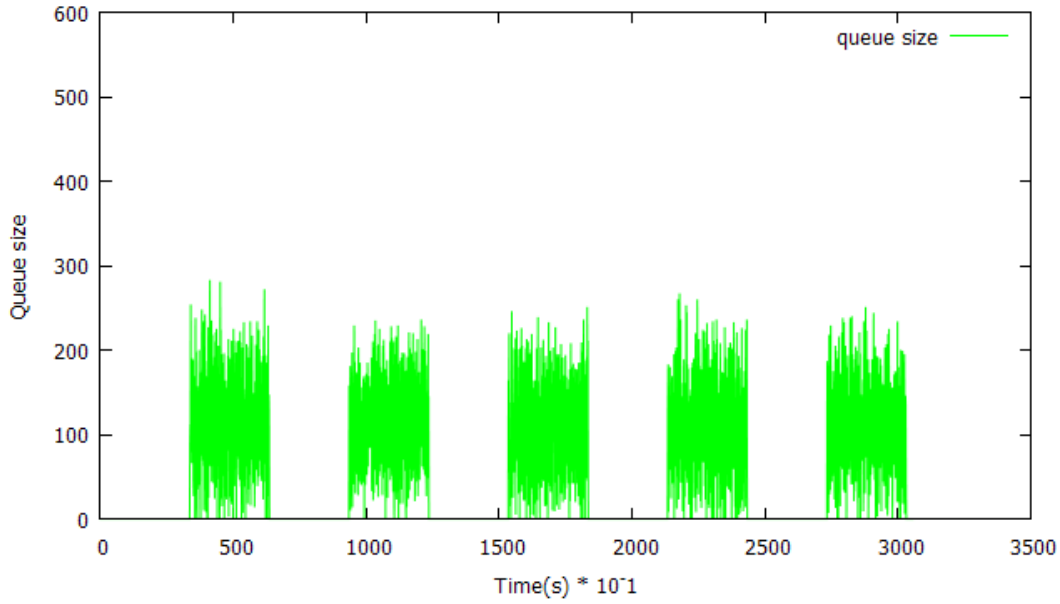


Figure 18. The state of the queue during the experiment is shown in green. The X axis shows the time of the experiment (in deciseconds) while the Y axis shows the number of PDUs in the queue.

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.

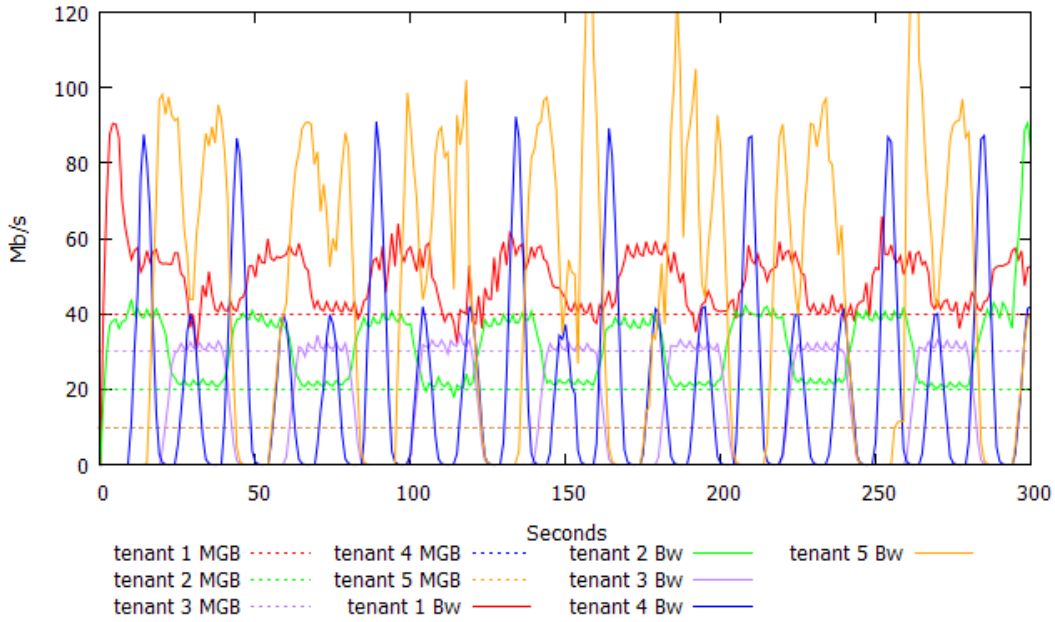


Figure 19. Congestion control mechanism when multiple tenants are active..

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.

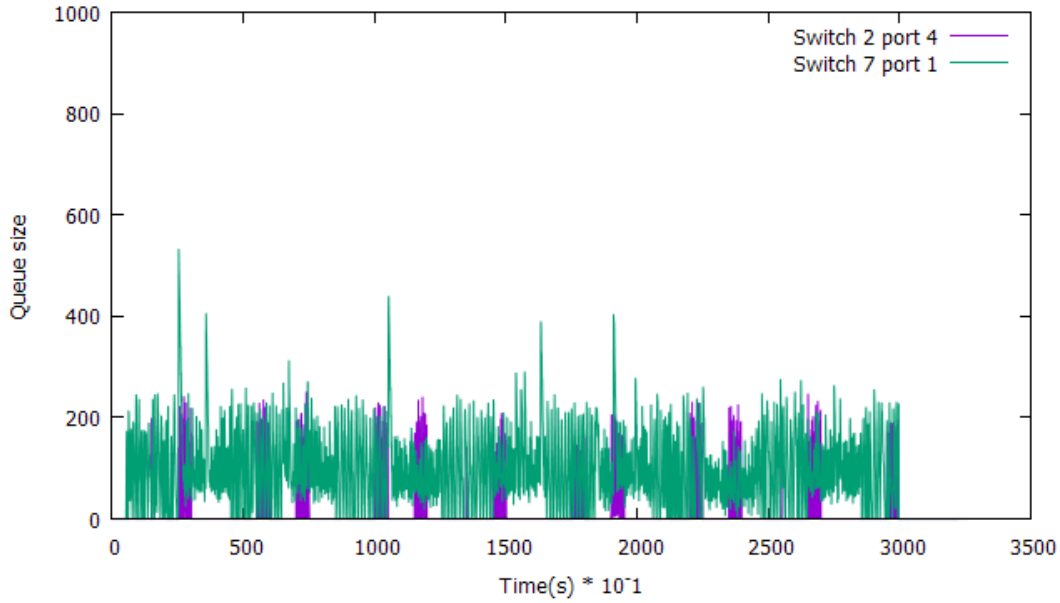


Figure 20. Status of the queues in the two congested switches during the experiment. The X axis shows the time of the experiment (in deciseconds), while the Y-axis shows the number of PDUs present in the queue.

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.

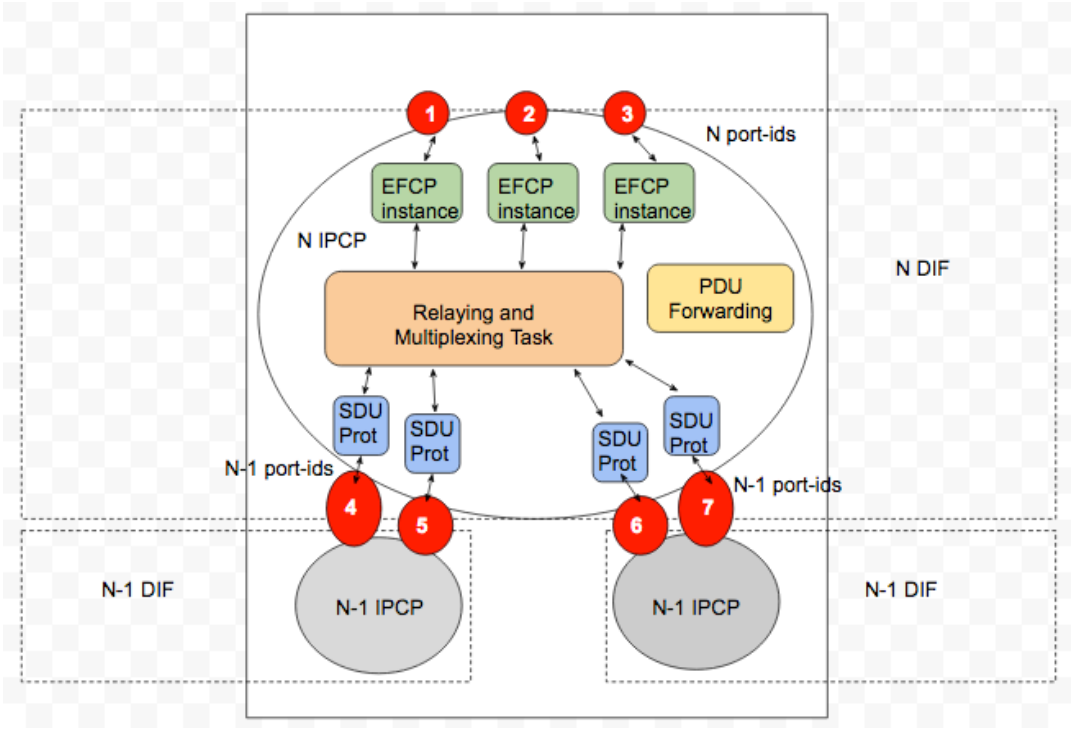


Figure 21. General structure of the data transfer parts of an IPC Process

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.

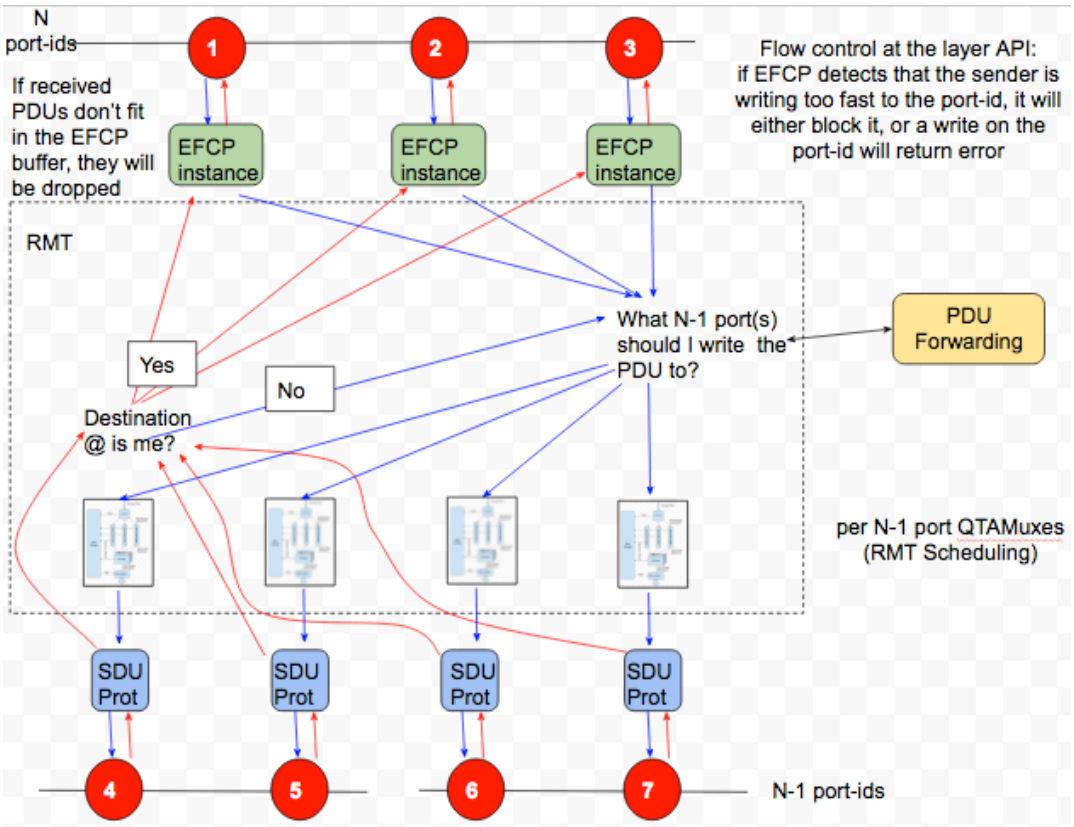


Figure 22. Group of QTA Muxes within the IPC Process

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.

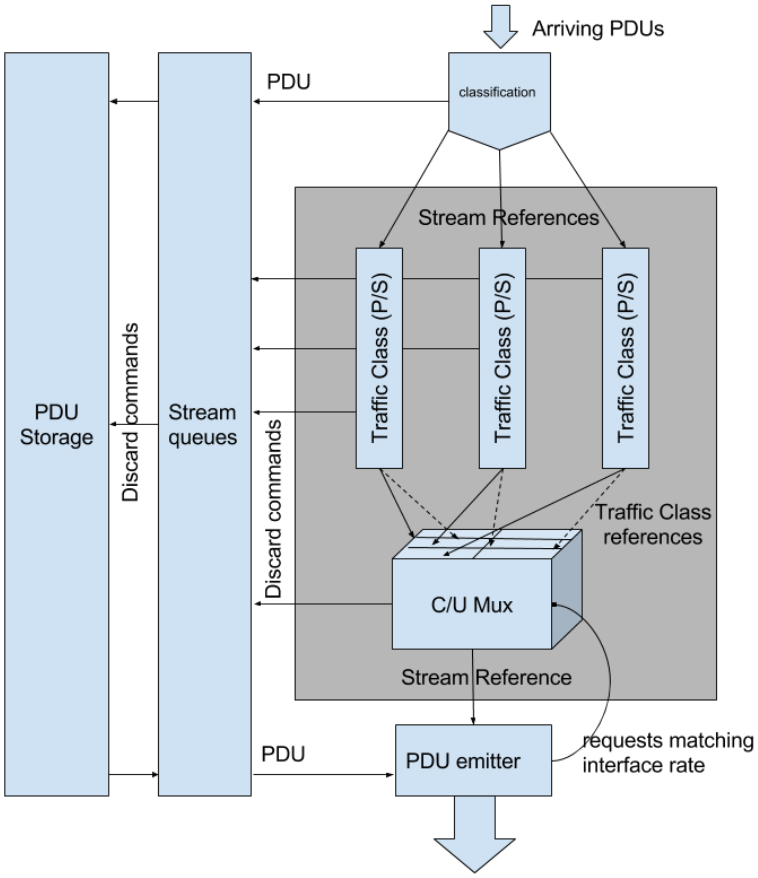


Figure 23. QTAMux block diagram

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.

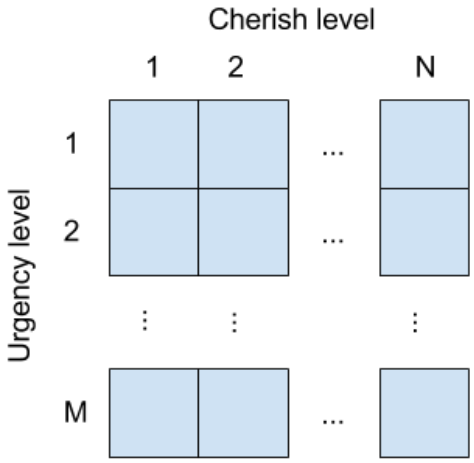


Figure 24. Two-dimensional cherish/urgency classification

[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:

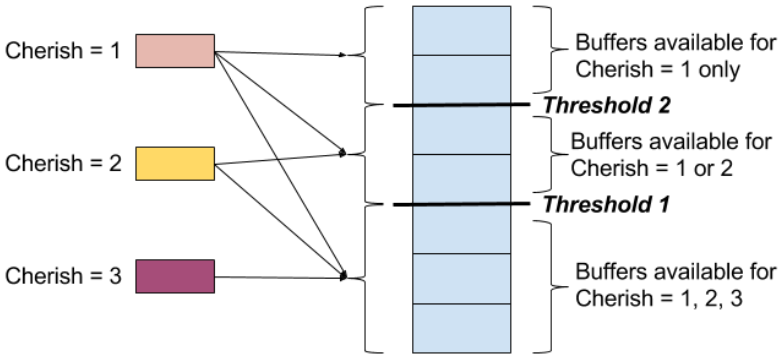


Figure 25. Partitioning of buffer space by cherish levels

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.

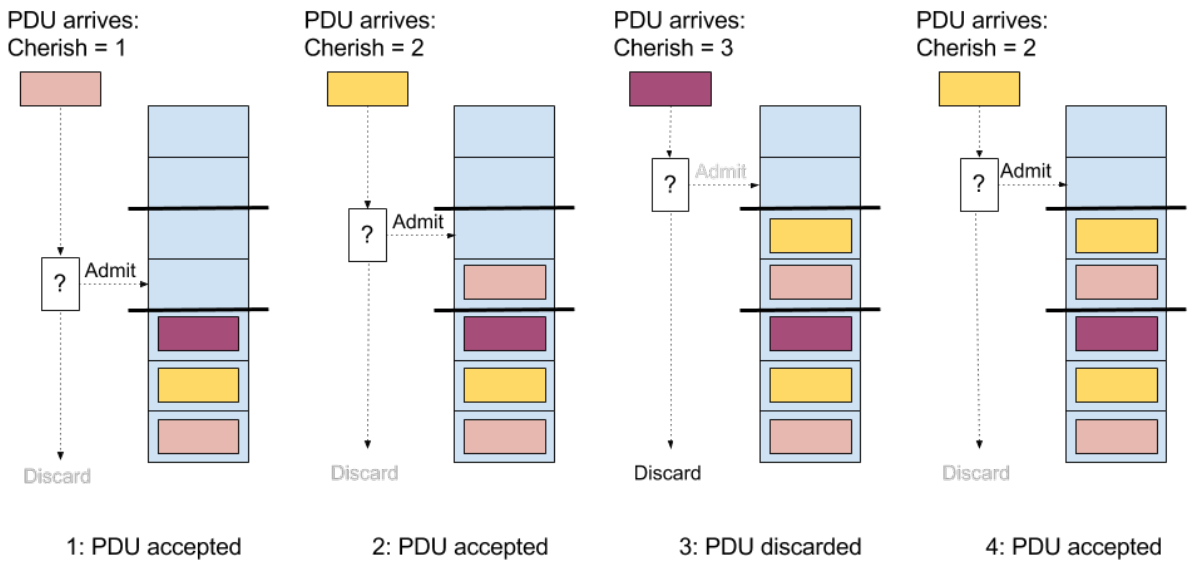


Figure 26. Admission and discarding of packets by cherish levels

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.

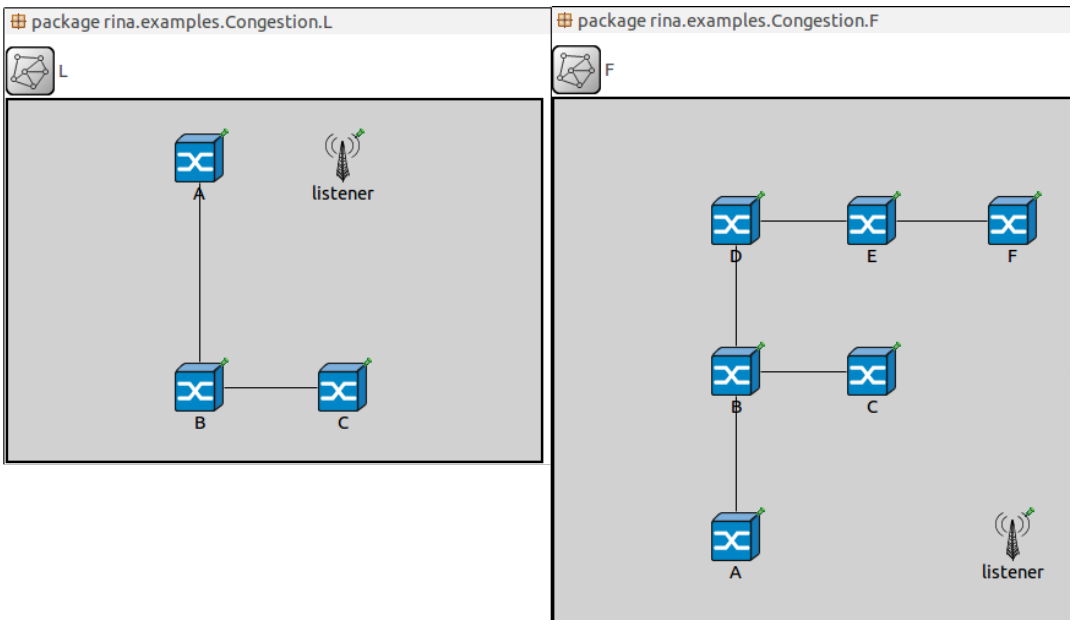


Figure 27. LF - L (left) and F (right) scenario network (omnet++)

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.

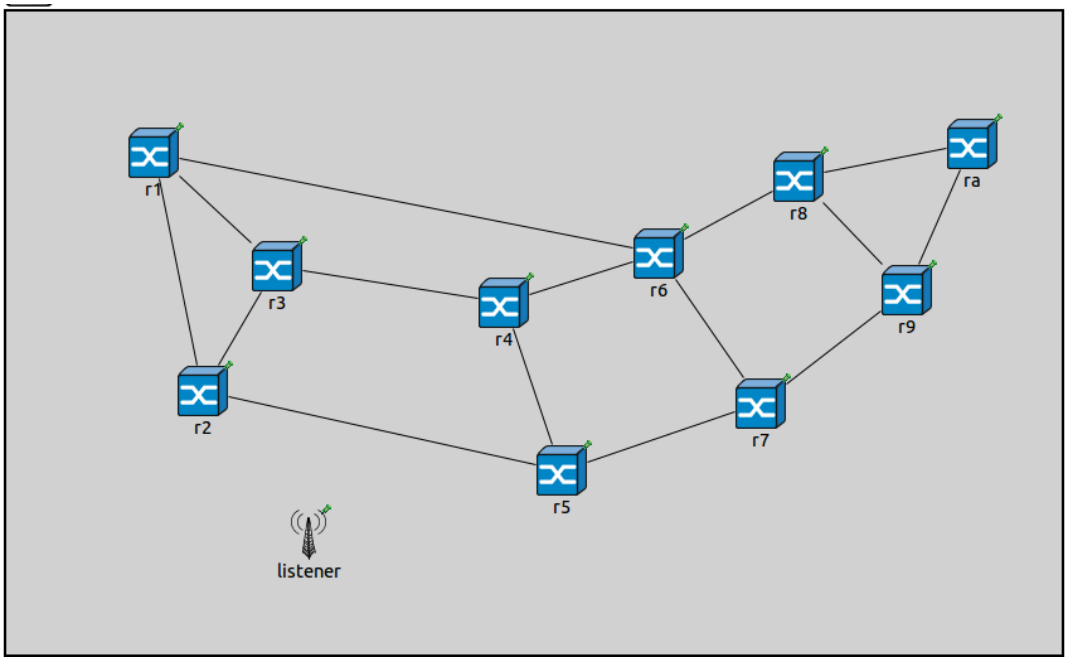


Figure 28. Backbone Scenario network (omnet++)

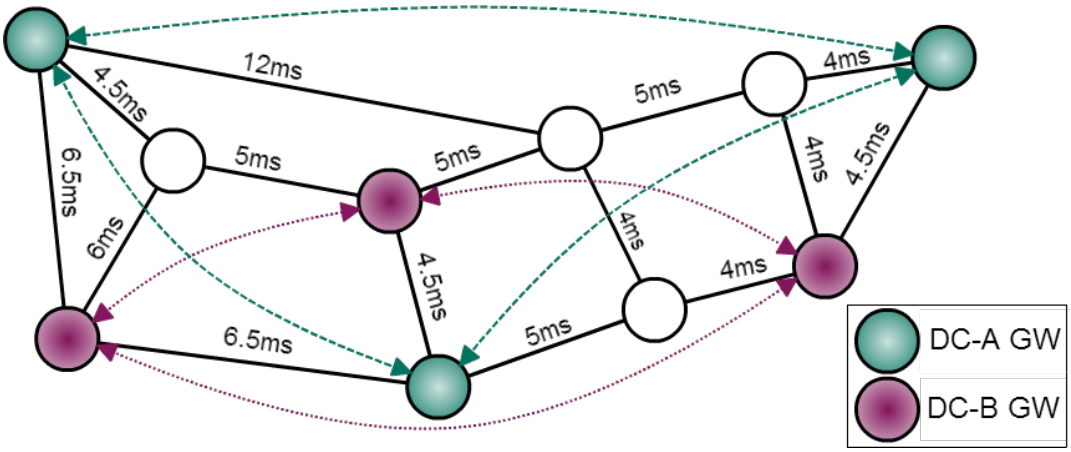


Figure 29. Backbone Scenario network with DC-GW placement

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.

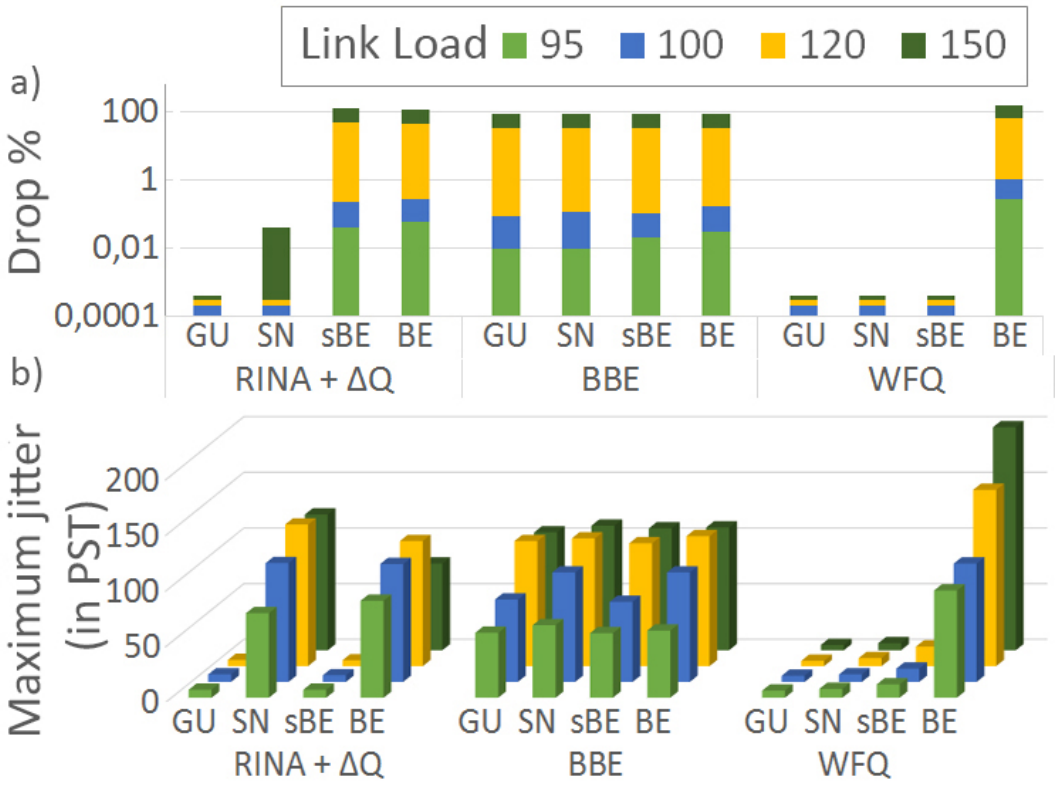


Figure 30. Average drop (a) and maximum jitter in PST (b) for GU, SN, sBE and BE flows depending on the scheduling policy used in the network

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.

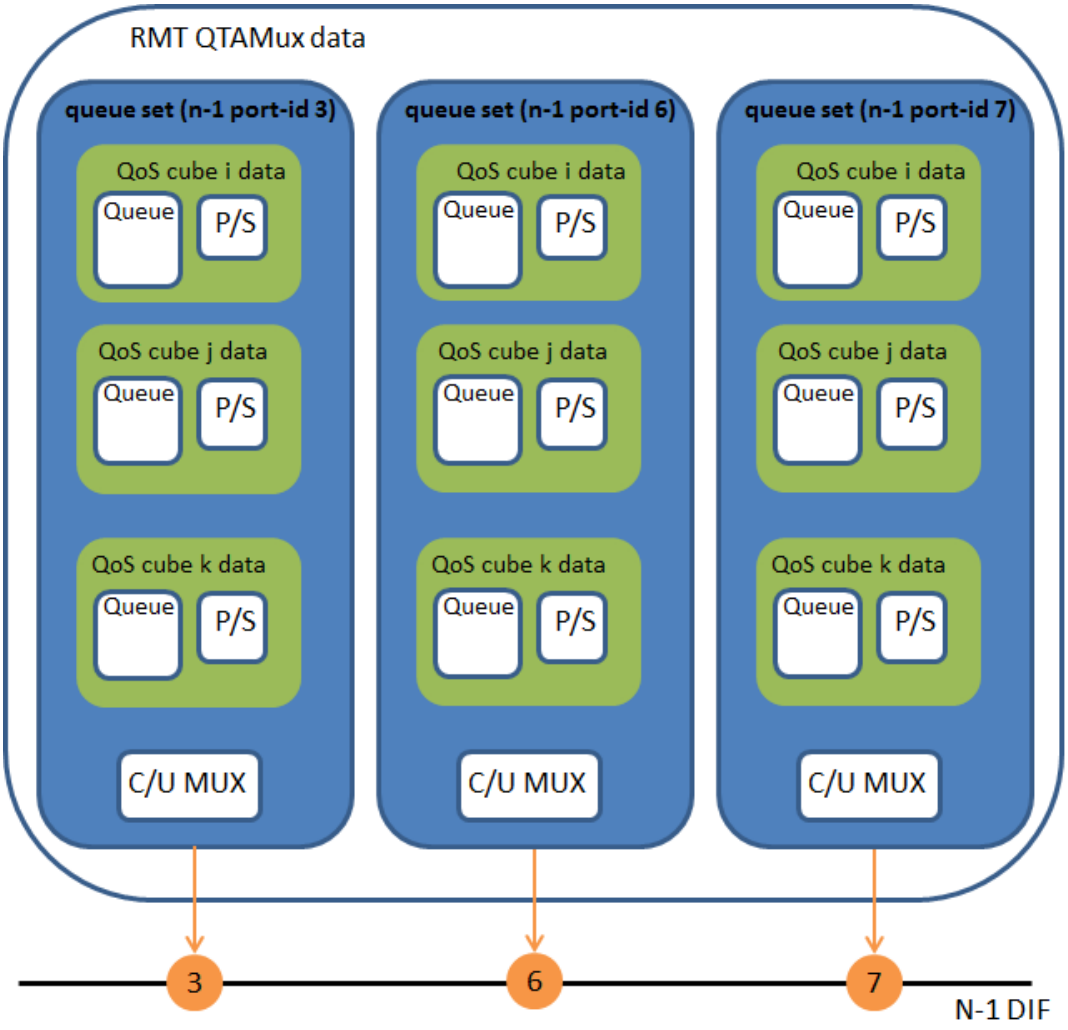


Figure 31. QTAMux block diagram as implemented in the IRATI stack

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).

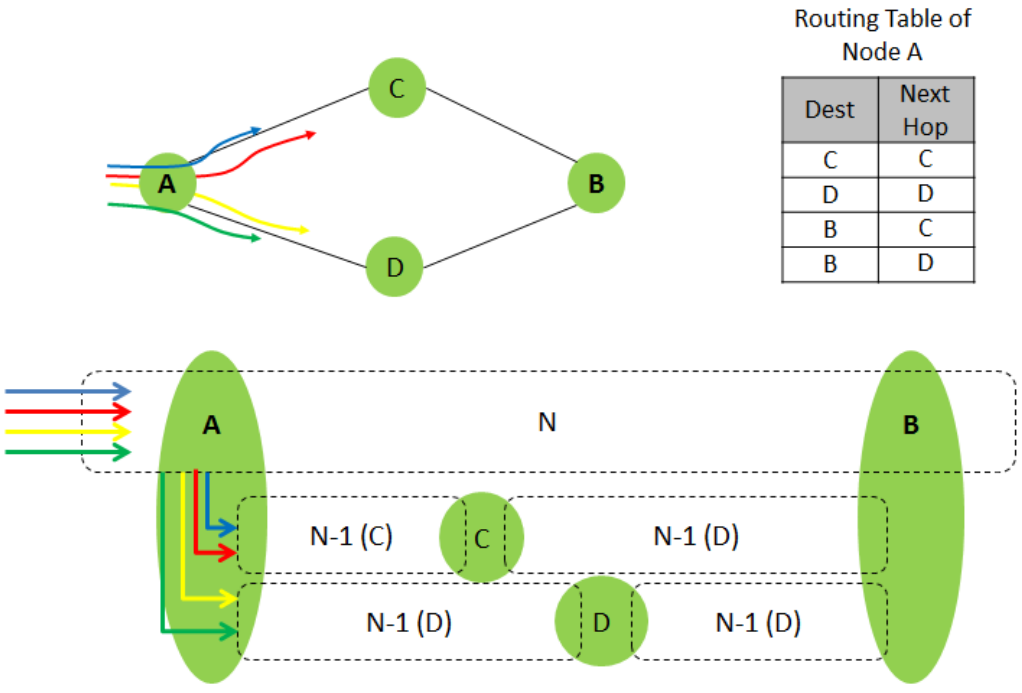


Figure 32. Simple multipath routing

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:

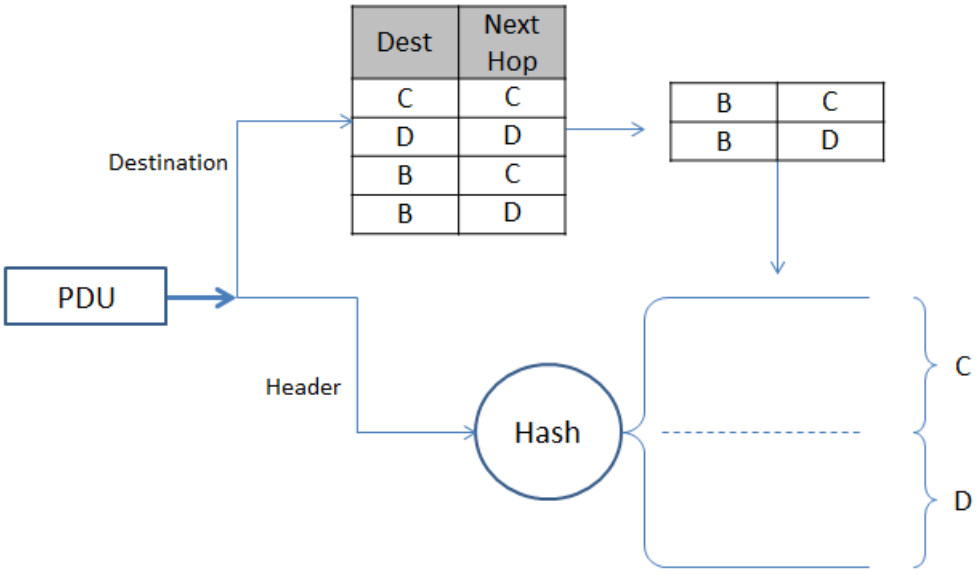


Figure 33. Simple multipath routing solution

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.

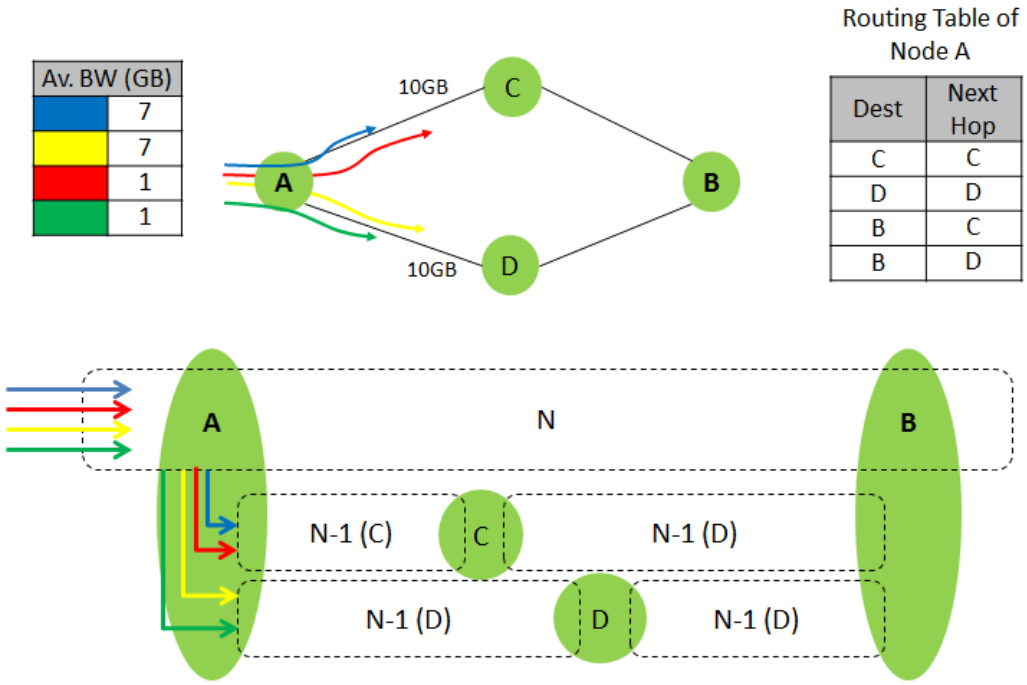


Figure 34. Static QoS-aware multipath routing

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.

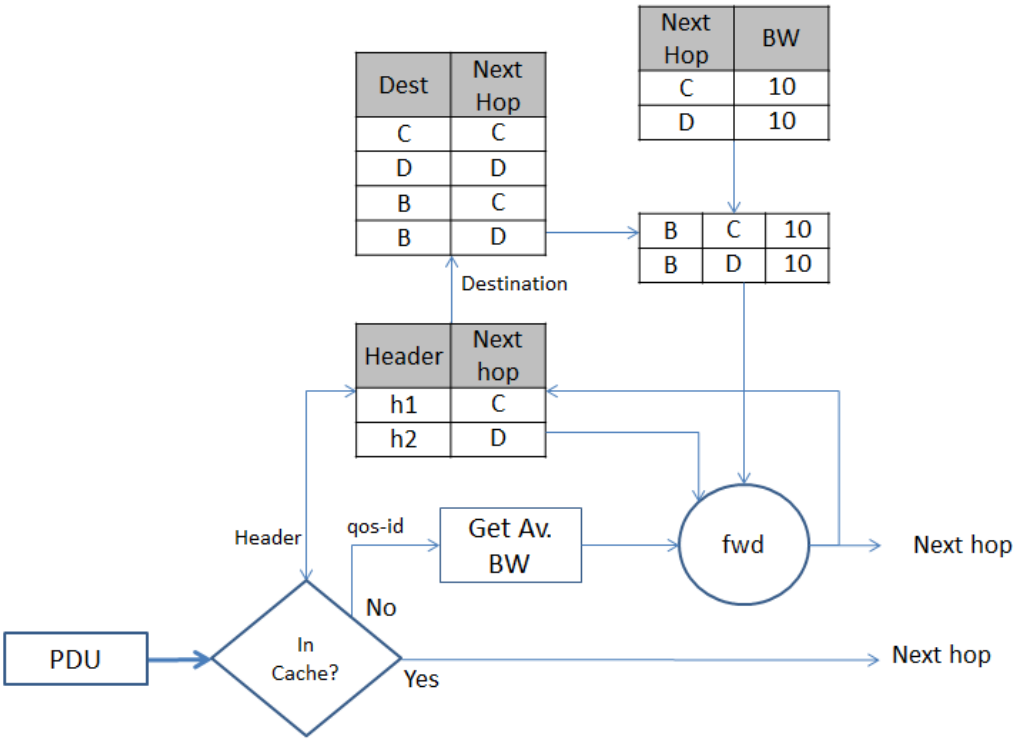


Figure 35. Static QoS-aware multipath routing solution

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.

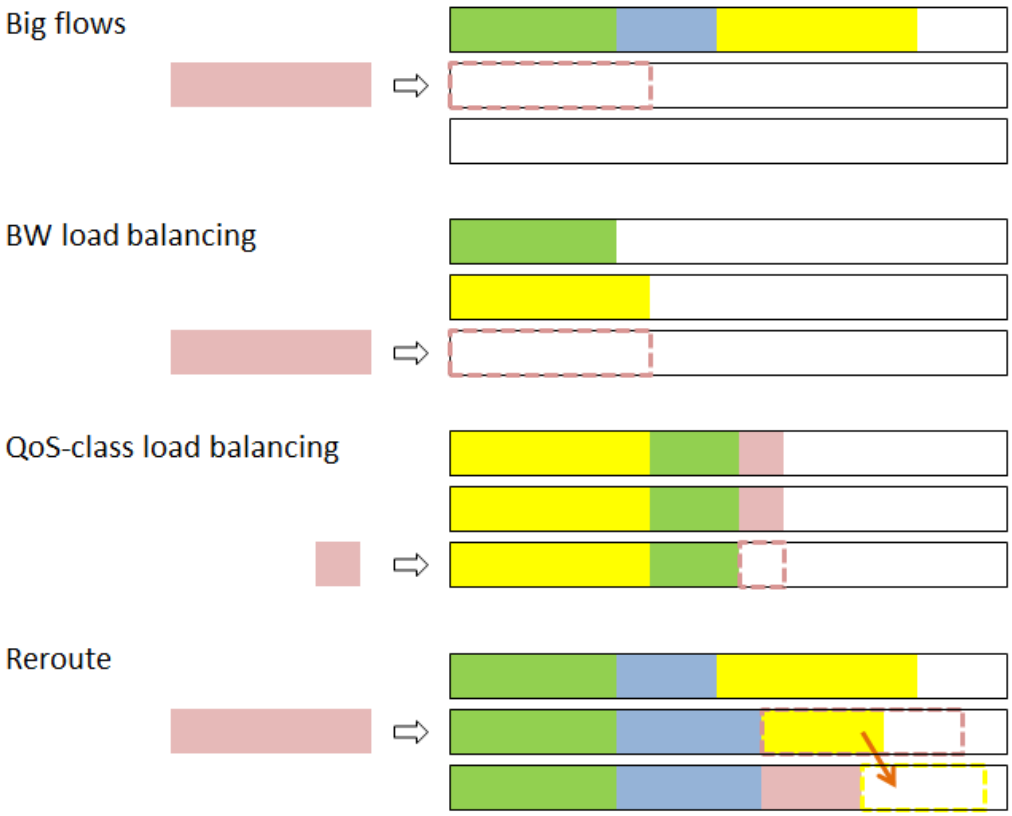


Figure 36. Static QoS-aware multipath routing algorithms

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.

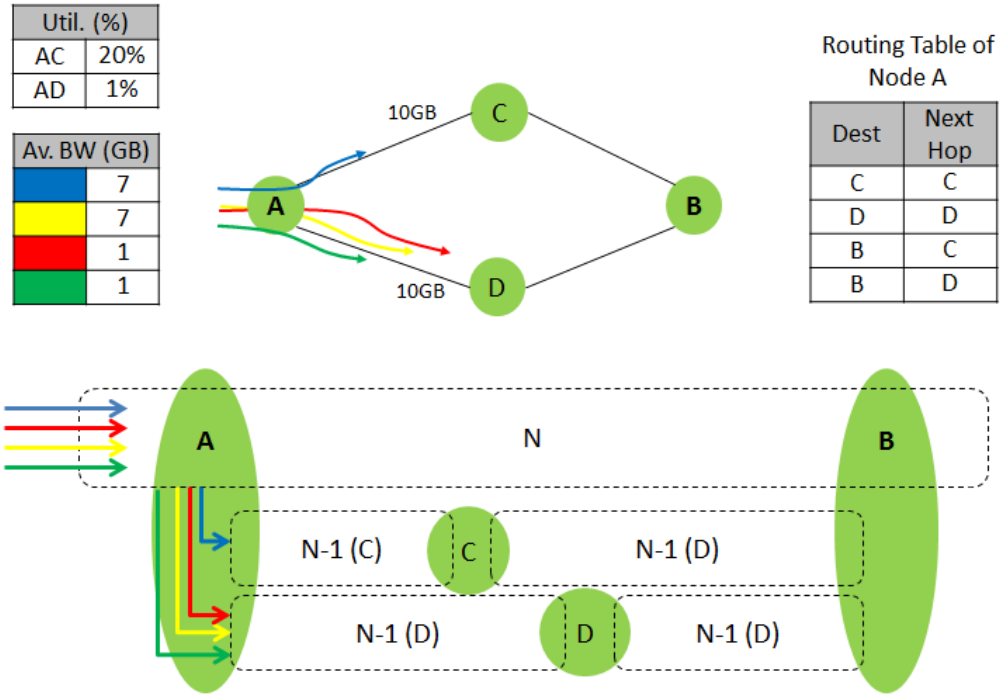


Figure 37. Dynamic QoS-aware multipath routing

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.

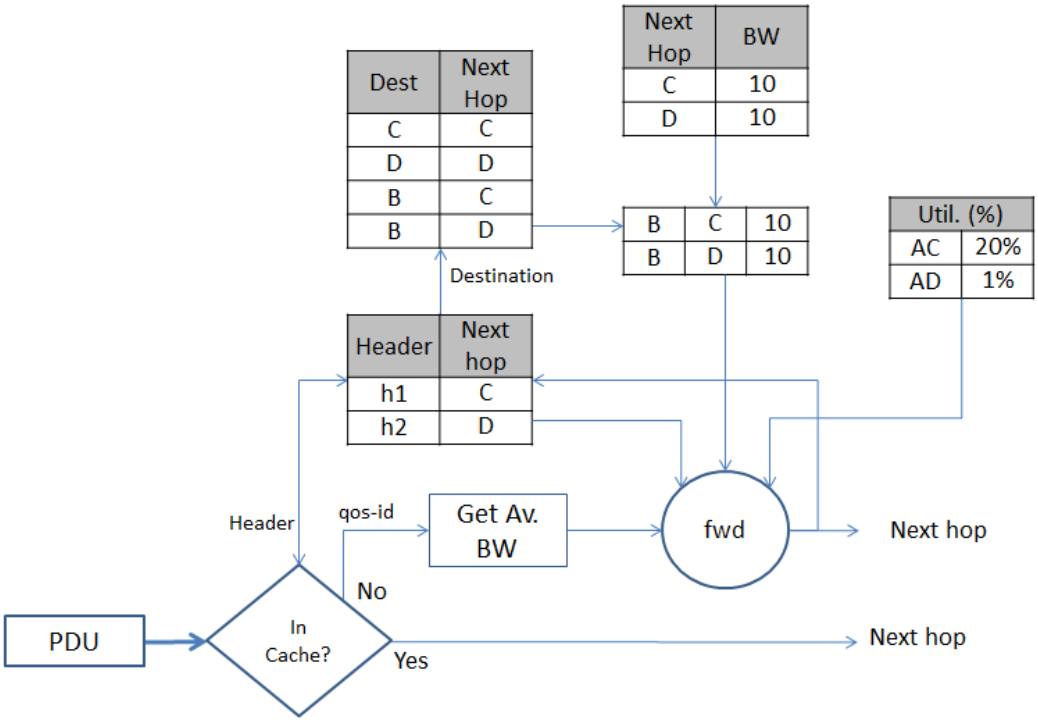


Figure 38. Dynamic QoS-aware multipath routing solution

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].

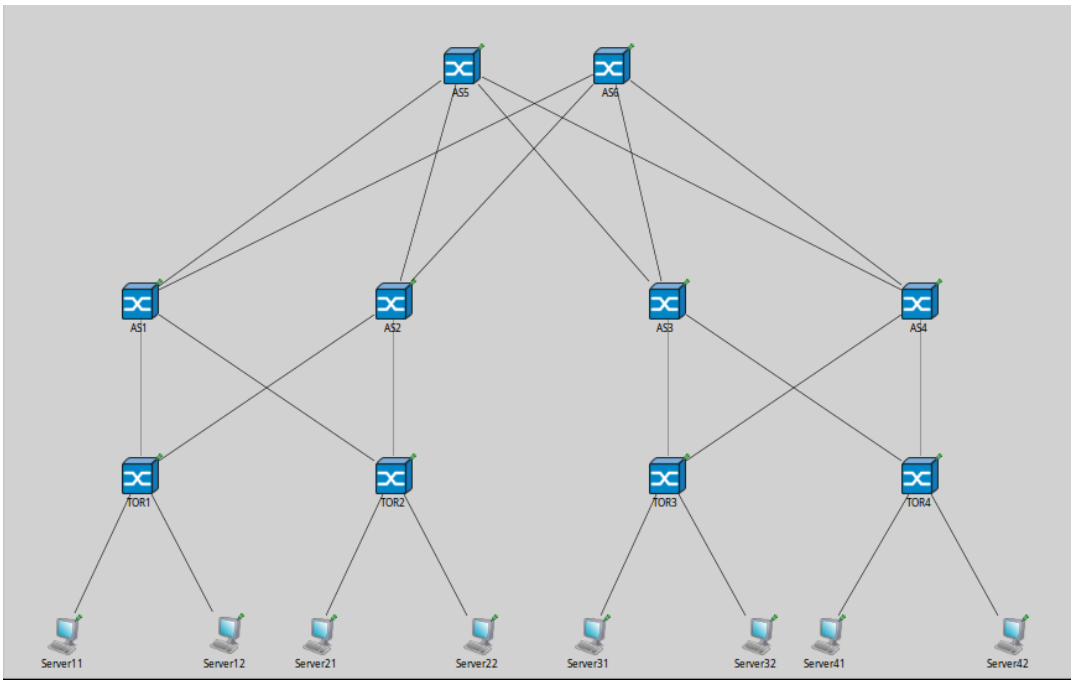


Figure 39. Multipath experiment scenario

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.

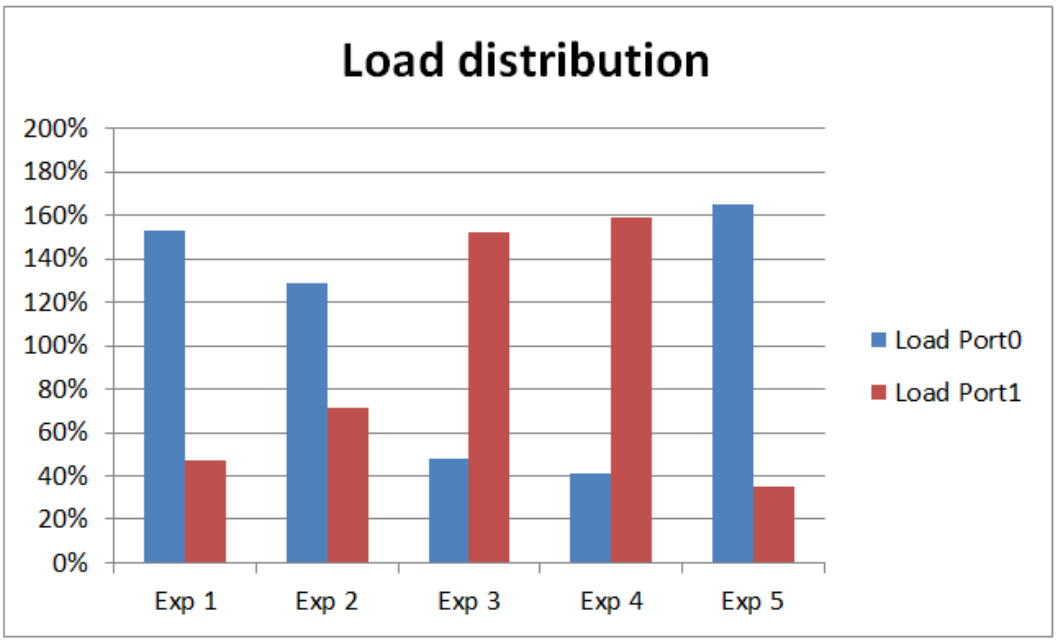


Figure 40. ECMP routing load distribution

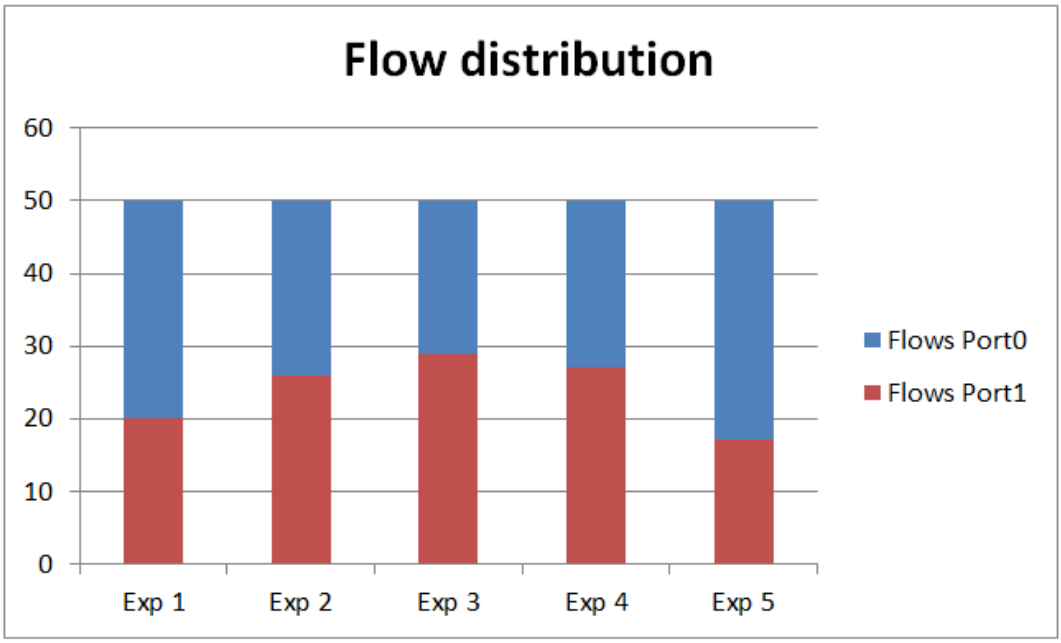


Figure 41. ECMP routing flow distribution

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.

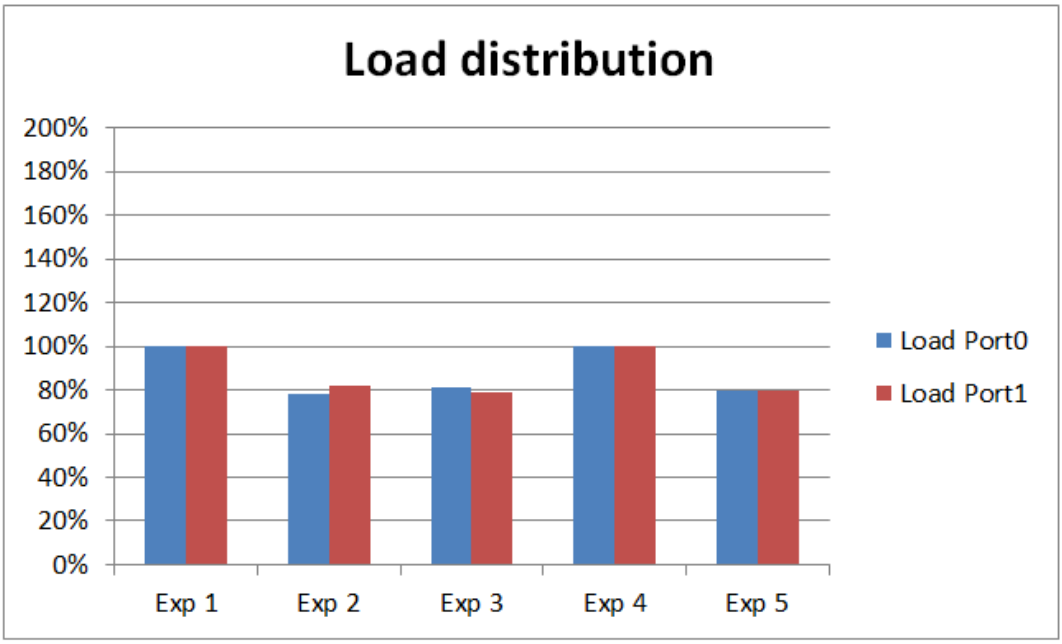


Figure 42. Static QoS-aware load distribution

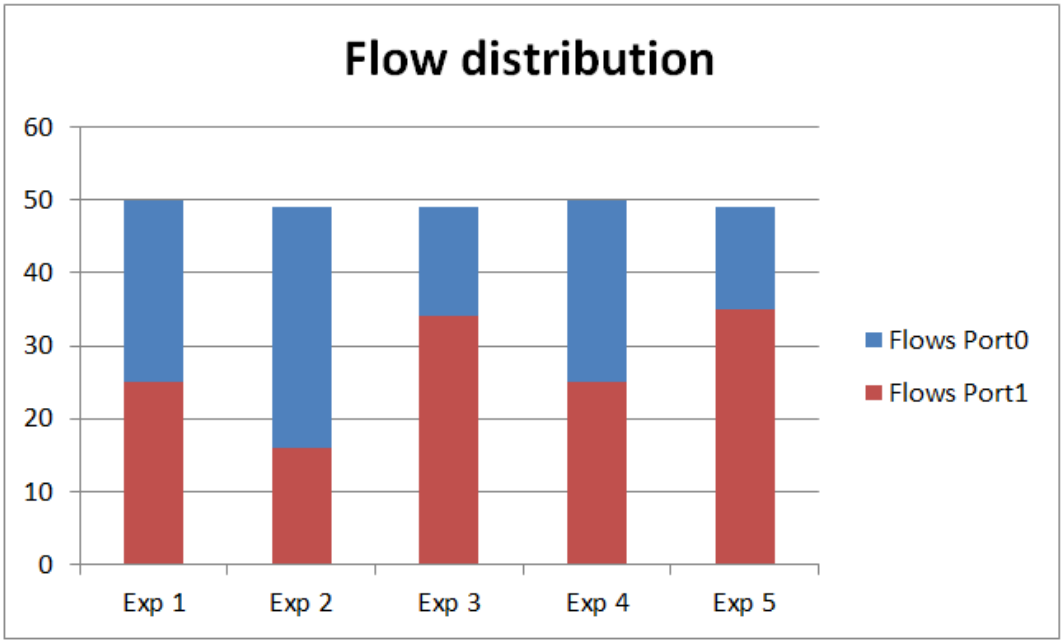


Figure 43. Static QoS-aware flow distribution

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.

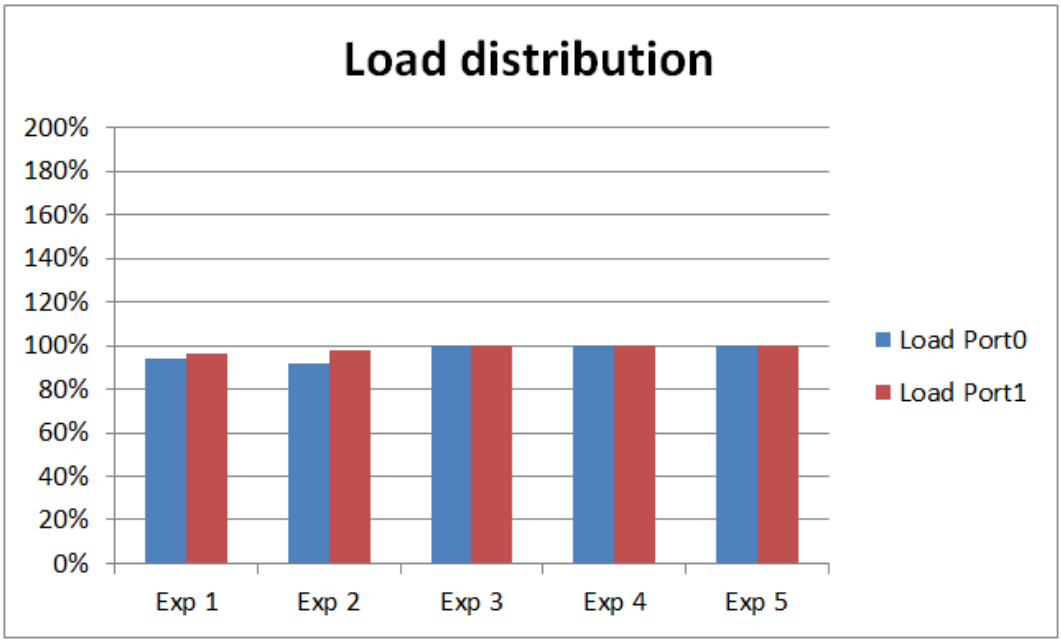


Figure 44. Dynamic QoS-aware load distribution

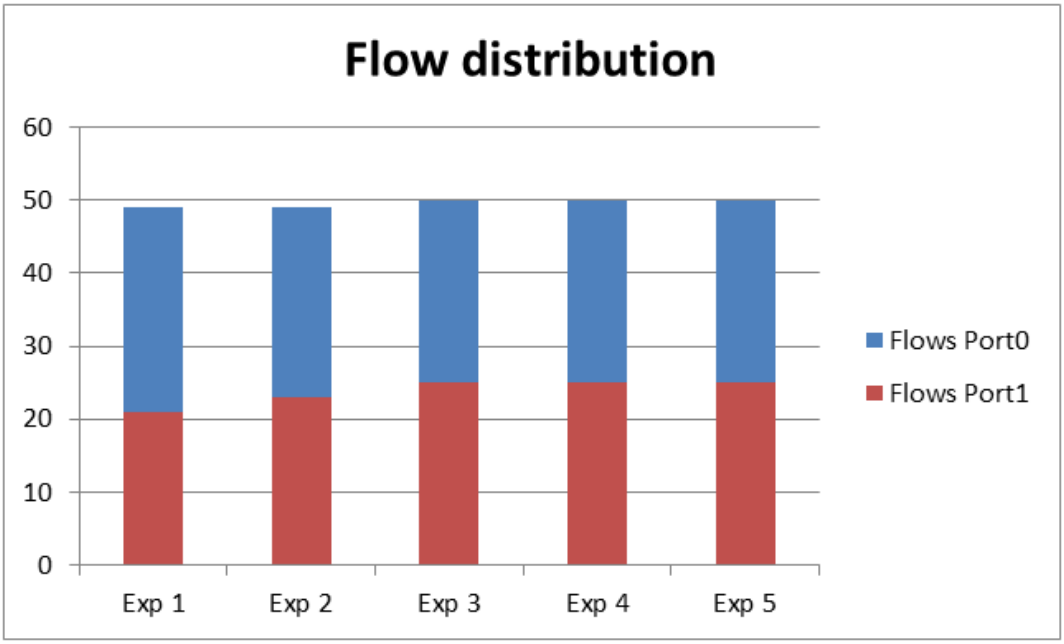


Figure 45. Dynamic QoS-aware flow distribution

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.

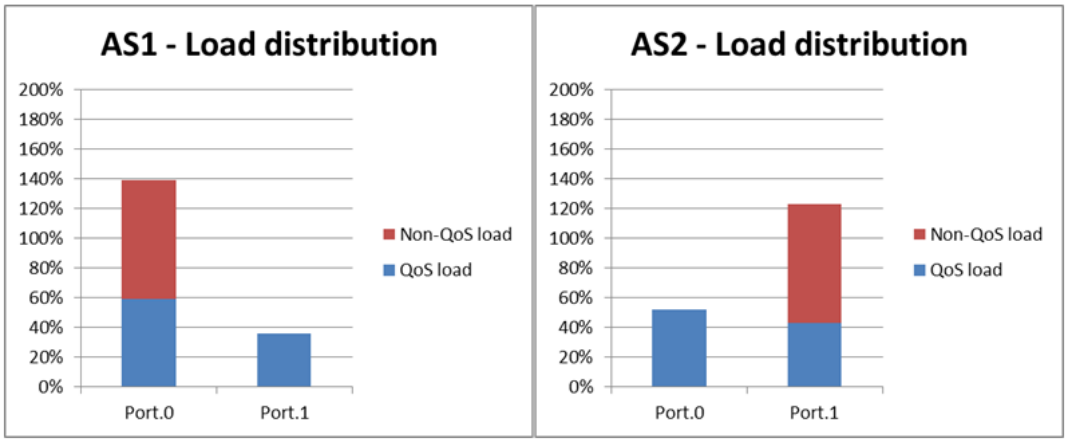


Figure 46. Static QoS-aware routing load distribution with best effort traffic

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.

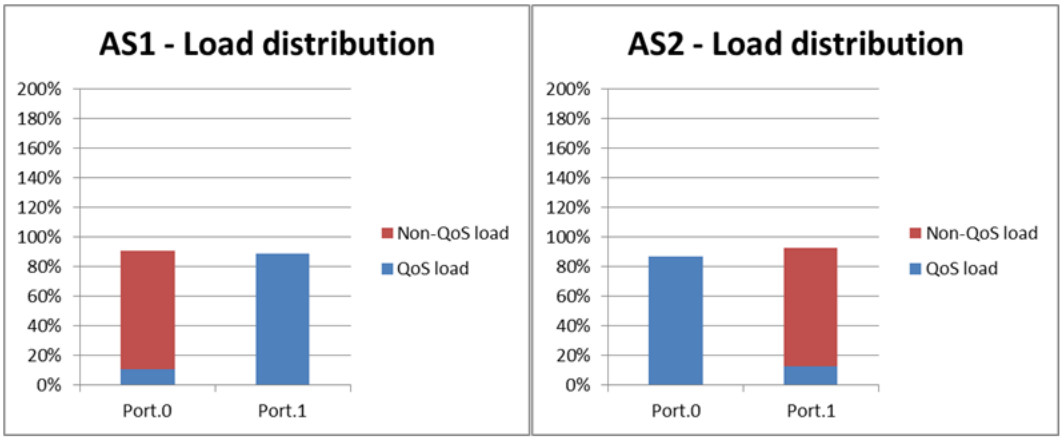


Figure 47. Dynamic QoS-aware routing load distribution with best effort traffic

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.

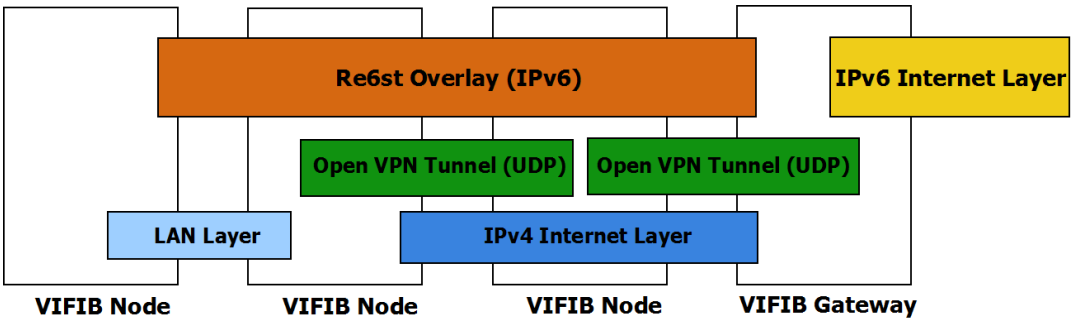


Figure 48. Architecture of the re6st overlay.

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.

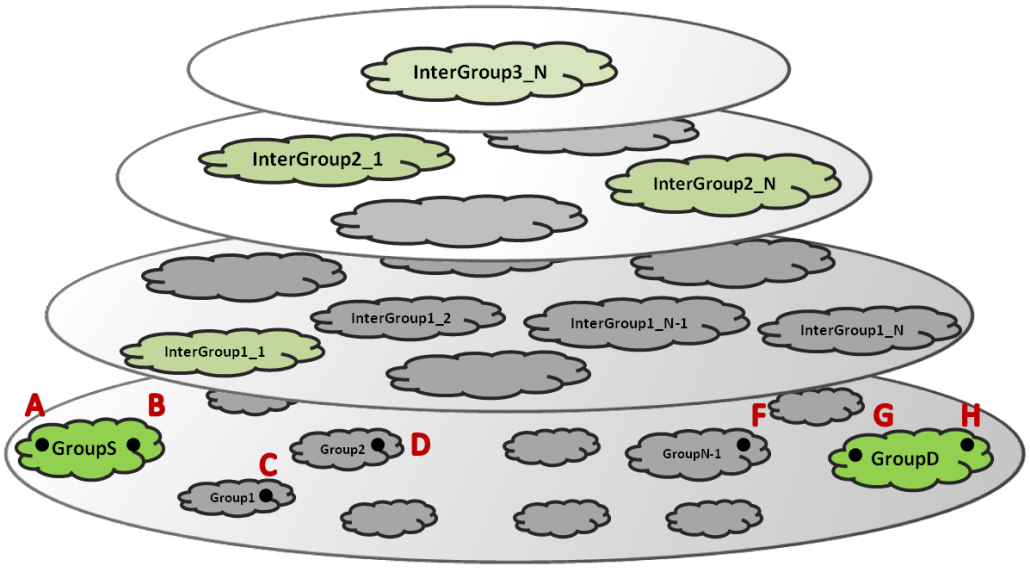


Figure 49. Example of SFR hierarchy with three levels.

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.

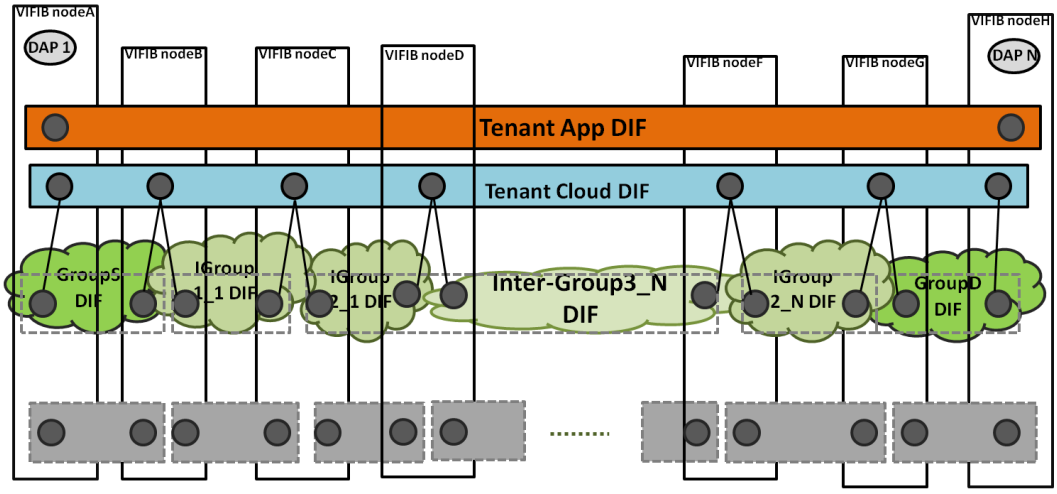


Figure 50. DIF Architecture.

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.

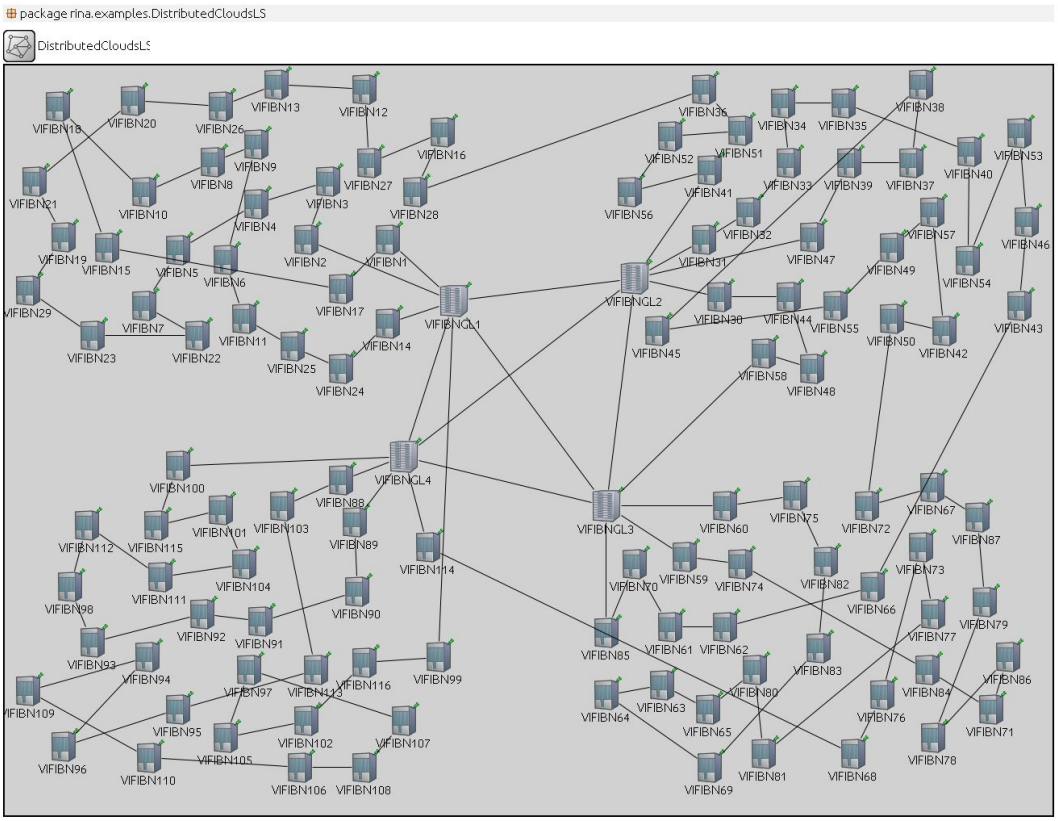


Figure 51. Distributed Clouds simulation scenario.

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.

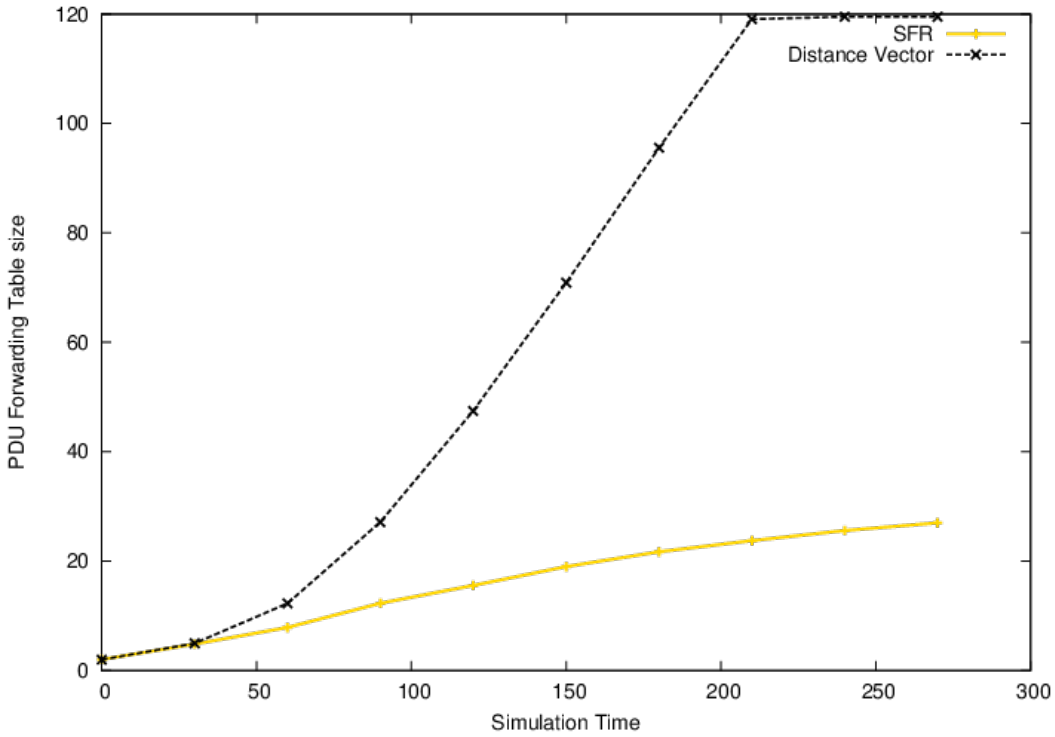


Figure 52. The variation of the PDU forwarding table size over time.

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.

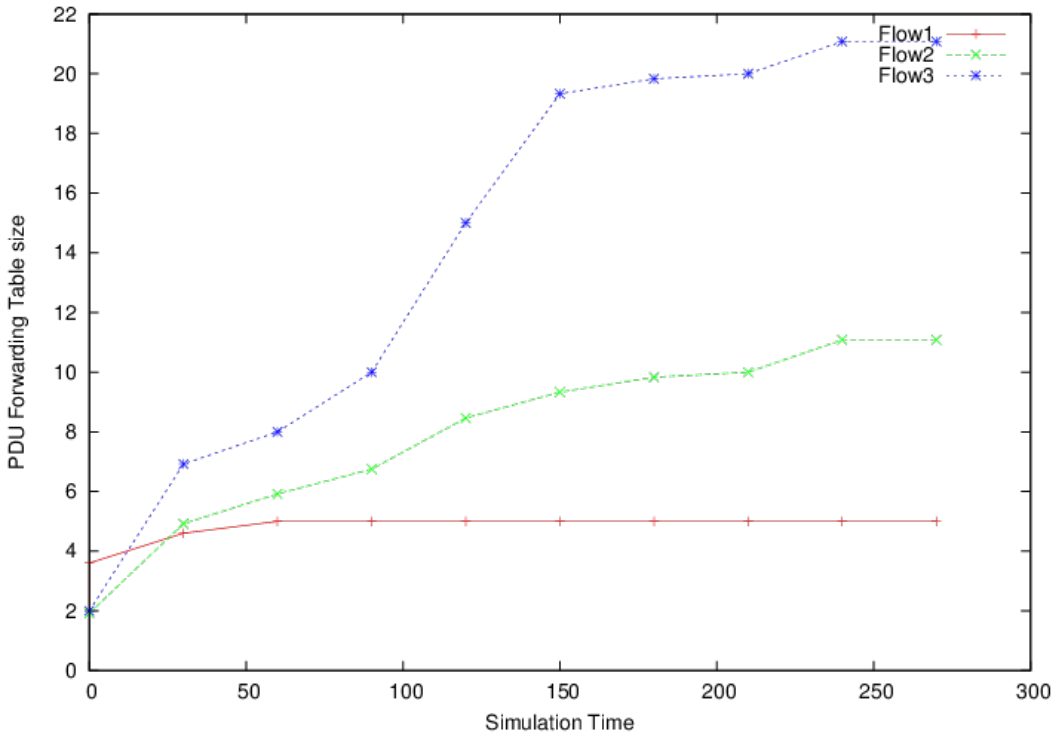


Figure 53. The variation of the PDU forwarding table size over time.

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.

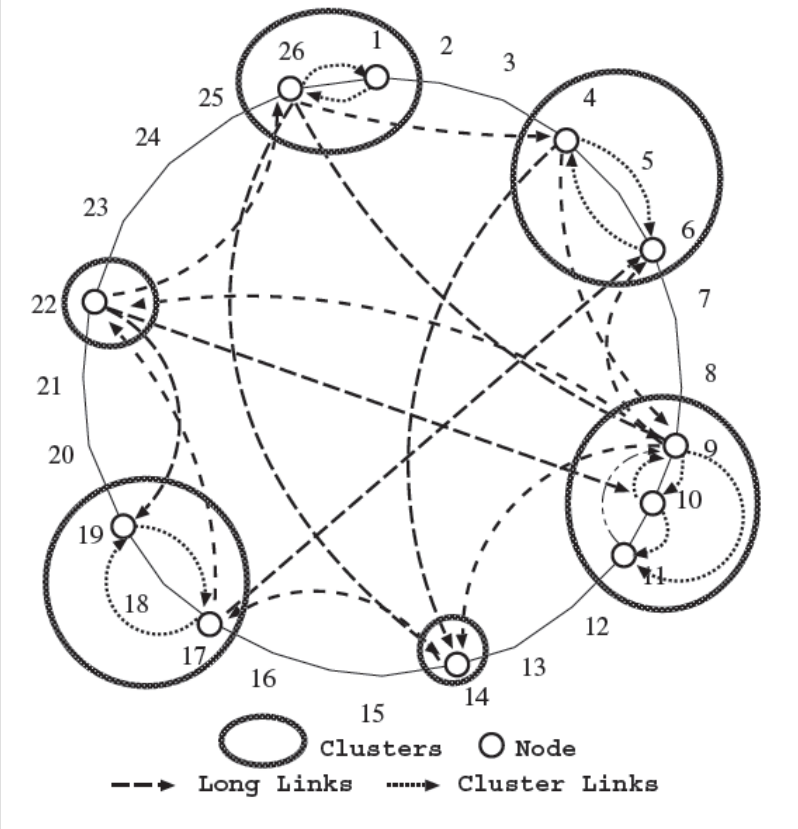


Figure 54. A small-world topology.

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.

1: Build first random connections between nodes (Apply re6snet strategy).
2: **repeat**
3: Partition the graph using Chinese whispers algorithm into multiple sub-graphs (groups) (see Algorithm 2). ▷ Define the small world short links
4: Select in each group a GroupLeader using the cost function: $C := w1 * Bandwidth + w2 * Links$
5: Build connections between GroupLeaders following Algorithm 3. ▷ Define the small world long links
6: **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.

```
1: if Node i has connection with Group G then
2:     Node i is a new member of Group G
3: else
4:     Build Short links within the requested Group
5:     Compute the Cost function of the node i:
6: Ci := w1 * Bandwidth + w2 * Links
7:     Determine the role of the node i within the requested
   Group:
8:     if Ci > Cgh then
9:         Node i is a the new Group Header
10:    else
11:        Node i is a new member of the Group
12:    end if
13: 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.

1: **if** Node is a Group Leader **then**
2: Select the substitutor with greatest cost function:
3: $Ci := w1 * Bandwidth + w2 * Links$
4: Rebuild short links within the group
5: **else**
6: Node i and its connections to other nodes are removed from the network
7: **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.

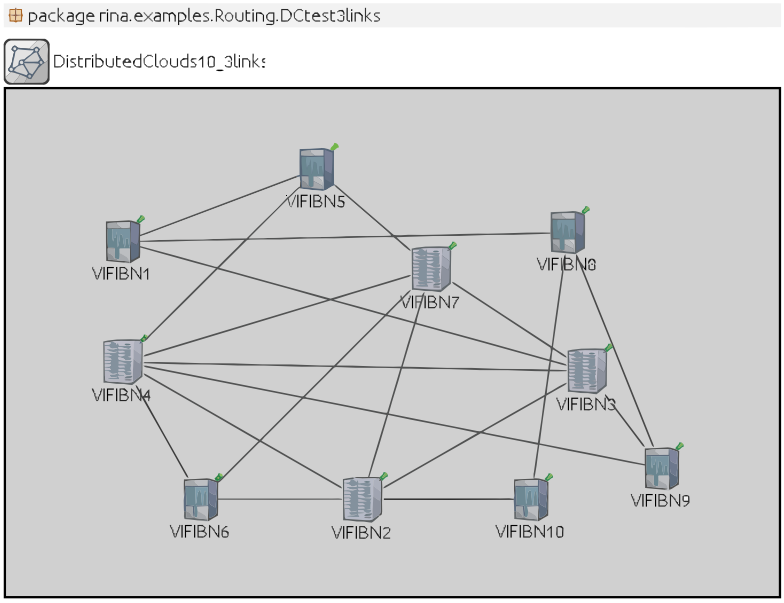


Figure 59. Simulation Scenario: small network.

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.

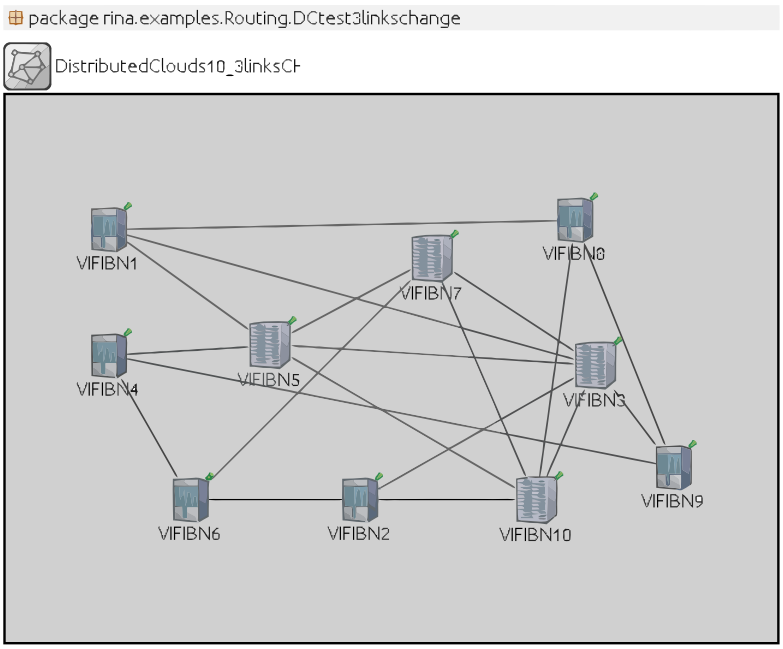


Figure 60. Simulation Scenario after change.

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.

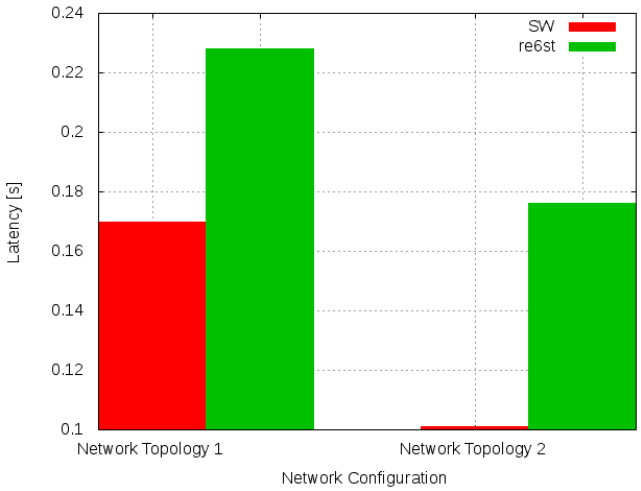


Figure 61. Average latency with respect to network topology.

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.

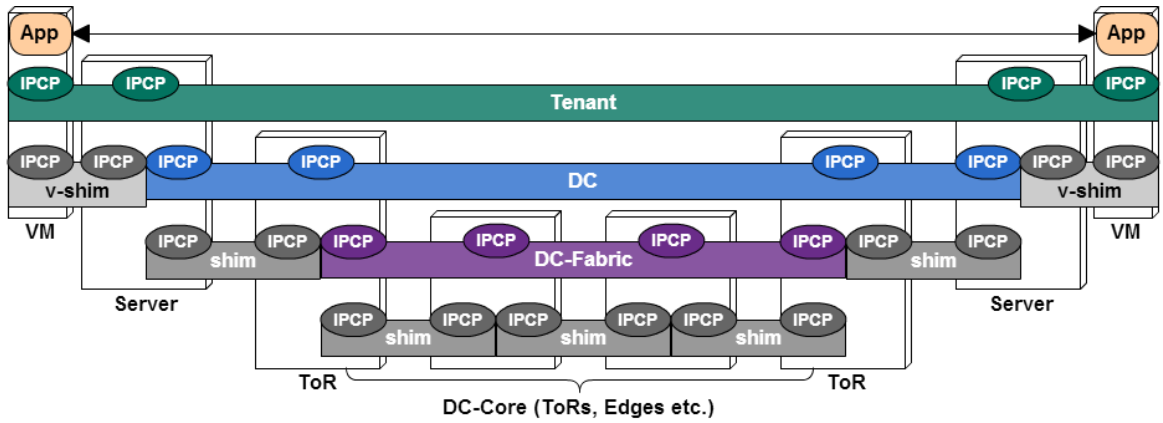


Figure 62. DIF setup inside a DC between Virtual Machines (VMs) running in DC servers. The DC-Fabric DIF (violet color) is the focus of this work.

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]

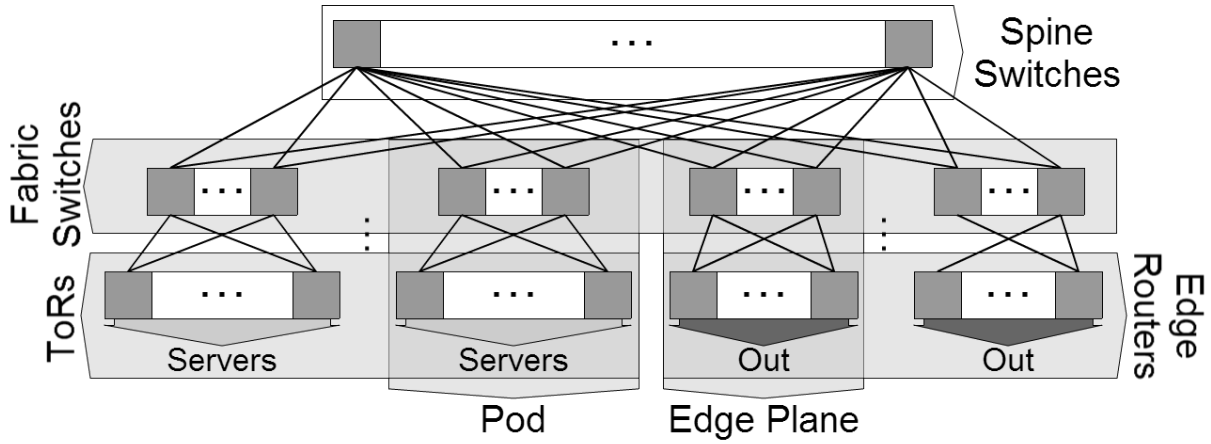


Figure 63. Google's DCN topology

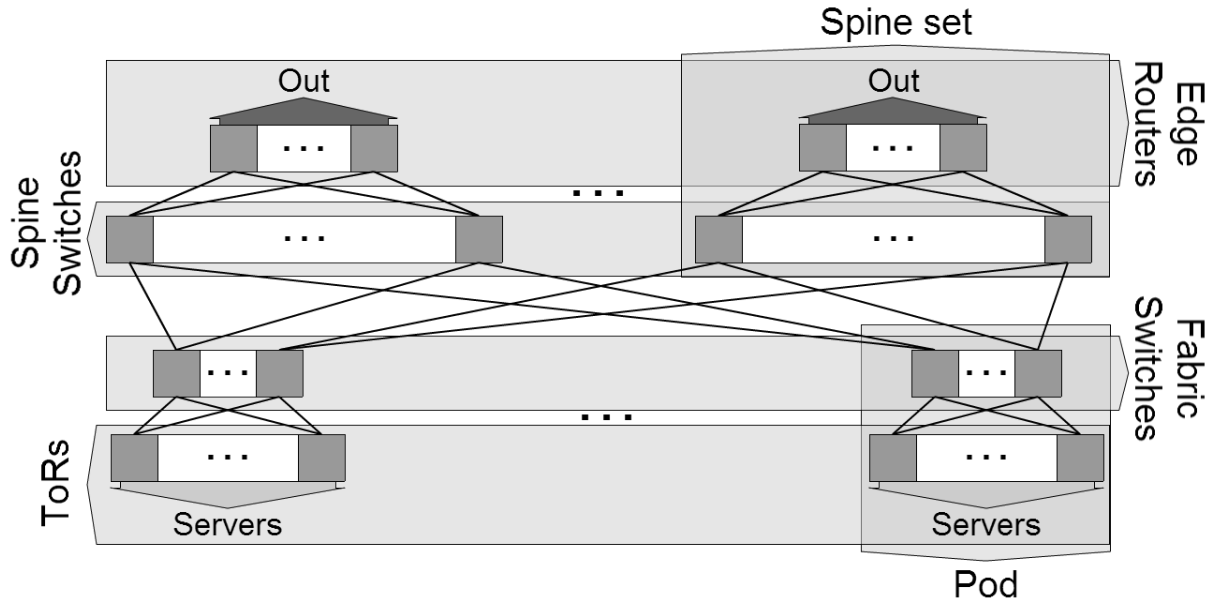


Figure 64. Facebook's DCN topology,

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.

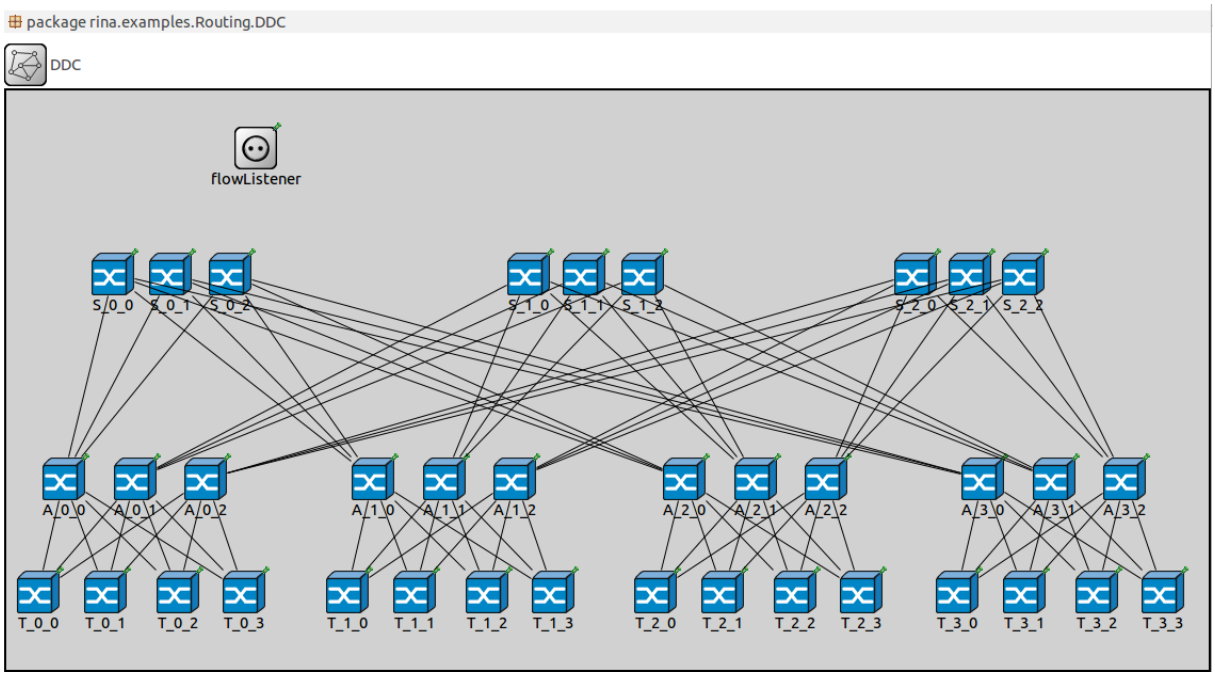


Figure 65. Network of Routing/DDC scenario

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-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.

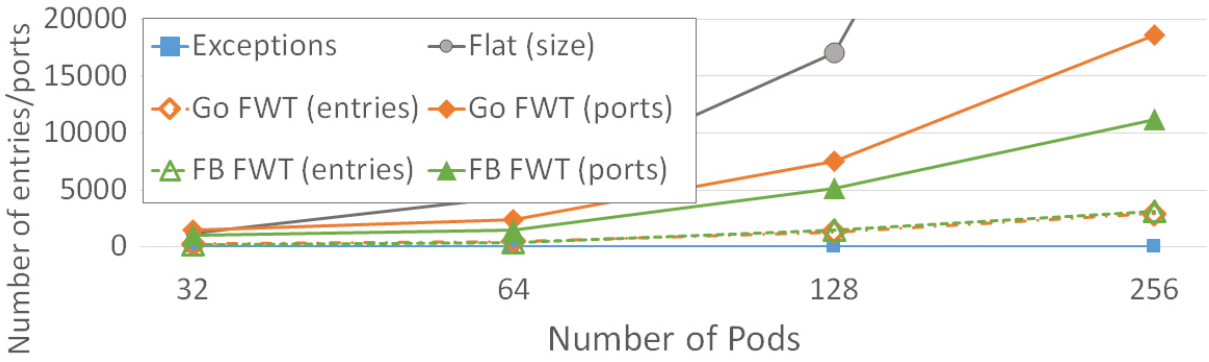


Figure 66. Avg. number of entries and stored ports given the number of pods

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)

Congestion Control in the Recursive InterNetworking Architecture (RINA)

Peyman Teymoori*, Michael Welzl†, Stein Gjessing‡, Eduard Grasa§,
Roberto Riggio¶, Kewin Rausch‖, Domenico Siracusa**
*†‡Department of Informatics, University of Oslo, Norway. Email:{peymant, michawe, steing}@ifi.uio.no
§i2CAT, Barcelona, Spain. Email: eduard.grasa@i2cat.net
¶‖**CREATE-NET, Trento, Italy. Email: {roberto.riggio, kewin.rausch, domenico.siracusa}@create-net.org

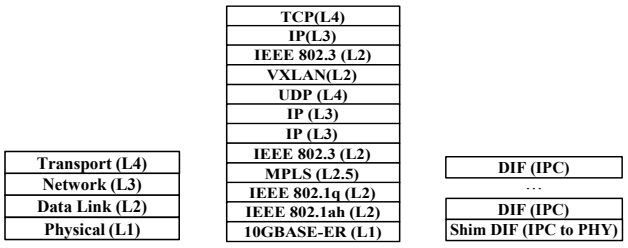
***Abstract*—RINA, the Recursive InterNetwork Architecture, is a novel "back to basics" type approach to networking. The recursive nature of RINA calls for radically different approaches to how networking is performed. It shows great potential in many aspects, e.g. by simplifying management and providing better security. However, RINA has not been explored for congestion control yet. In this paper, we take first steps to investigate how congestion control can be performed in RINA, and demonstrate that it can be very efficient because it is applied close to where the problem happens, and through its recursive architecture, interesting effects can be achieved. We also show how easily congestion control can be combined with routing, enabling a straightforward implementation of in-network resource pooling.**

I. Introduction

In the Internet, congestion control is embedded in a protocol at the transport layer or above – most commonly in TCP (which we will refer to in the following), but also in SCTP, DCCP and in certain RTP-based applications. The TCP protocol operates "end-to-end", where "end" is the host the application is running on. Although this might ensure scalability since interior network elements do not have to worry about congestion control,[1] the control is potentially executed far from where congestion appears in the network.

The Recursive Network Architecture (RINA) [2], [3], is a back to basics approach learning from the experience with TCP/IP [4] and other technologies in the past. In RINA, every layer (called a "Distributed InterProcess Communication (IPC) Facility" (DIF)) has the same set of mechanisms and goal: providing and managing the communication among its entities (called the "IPC Processes" (IPCPs)). All DIFs have the same internal structure. For example, each DIF has one transport protocol (the "Error and Flow Control Protocol" (EFCP)); this is a mechanism that cannot be changed. However, how this mechanism performs flow control, retransmission control or congestion control is defined as "policies" and can be programmed differently in various DIFs.

Many cases can be found in which there are layers in the network stack that do not follow the traditional "transport/network/data link/physical" architectural model (Fig. 1(a)). For example, in Fig. 1(b) we can see L2 over L2 (MAC-in-MAC), L2 over L3 (Ethernet over MPLS), L2 over L4 (VXLAN, L2TP), L3 over L3 (IP in IP, GRE, LISP), and so on. RINA allows theoretically an indefinite number of stacked DIFs to provide IPC services to each other (Fig. 1(c)). This is how it shows the ability to solve long-standing network problems of the Internet architecture (complexity, scalability, security, mobility, QoS or management, see [2], [5]–[7]). However, previous work on RINA has just focused on the above issues and did not investigate it on congestion control research.



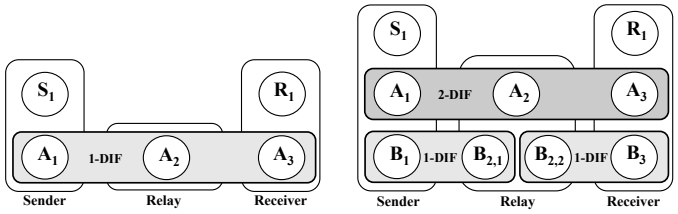
(a) The TCP/IP stack in theory (b) The TCP/IP stack in practice (c) RINA Stack

Fig. 1. Protocol stacks. (In RINA, shim DIF is the bottom DIF designed to operate on the physical layer, link layer, or any other specific technology.)

In this paper, we take a first look at how a congestion control method can work in this new architecture; by implementing a TCP-like congestion control policy, we compare it with similar approaches in the Internet, and discuss its benefits. We also elaborate how a range of different optimizations in congestion control can fit within a single framework. In RINA, *recursion* arises from the ability to arbitrarily arrange structurally-equivalent DIFs. Through simple topologies, we show that improvements that have been done to TCP such as Split-TCP on the internet "naturally appear" with RINA without their side effects; we present some DIF layouts to show how they solve some congestion control problems in the Internet. 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 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*.

We will elaborate on the design of RINA congestion control

[1]It has been said that this design of TCP matches the end-to-end argument. This is true, but the end-to-end argument really only prohibits implementing *application-specific* functions inside the network [1]. Much like routing, congestion control addresses a problem that occurs inside the network (the end-to-end argument does also not forbid complex routing algorithms).



(a) End-to-end (b) Split

Fig. 2. Two possible RINA stack configurations by different organizations of "Distributed InterProcessCommunication Facilities" (DIFs).

in Section II. In Section III, we will take a look at how a TCP-like mechanism would play out in RINA, focusing on some simple "toy" scenarios. Section IV discusses RINA congestion control in the context of prior work, and Section V concludes.

II. Congestion Control in RINA

In RINA, every function is bound to one DIF, and DIFs can be of different sizes, e.g. two nodes connected to each other can form a DIF, or a public DIF can contain all the nodes of a network. Every DIF can have a different type of congestion control (or none at all). DIFs can also be stacked; the lower DIF serves the upper one based on a list of requirements that the upper DIF needs for its flows. This provides the flexibility of mapping several upper flows with the same requirements (e.g. QoS) to just one lower flow. Here, we only discuss congestion control-related modules of RINA; however, for further information on RINA, see [2].

As a simple starting point, consider a path with one intermediate hop, given by a physical setup of 3 nodes: a sender, a relay/router node in the middle, and a receiver, as shown in Fig. 2. S_1 and R_1 are the application instances sending and receiving packets, respectively. A_i (or B_i) is called IPC Process (IPCP); it is an application process that is a member of a DIF and locally implements the functionality to support and manage IPC using multiple sub-tasks. In Fig. 2(a), there is one DIF for the network that performs congestion control with a control loop from A_1 to A_3. The module controlling congestion in A_1 and A_3 is called EFCP. If congestion appears in the relay IPCP (in the RMT module of A_2 which is responsible for relaying PDUs), the RMT can, for example, set the Explicit Congestion Notification (ECN) field of PDUs. The sender's window or rate is a function reacting to congestion in this network. This scenario has its advantages in simplicity and scalability, but it has some obvious disadvantages too: A_1 cannot benefit from the knowledge specific to the two links below A_1–A_2 and A_2–A_3 (just like TCP can usually not benefit from link-layer knowledge), and if congestion appears in one of these links, EFCP needs a full round-trip-time (from A_1 to A_3 and back) to react.

The diagram in Fig. 2(b) shows two separate DIFs at layer 1 and hence two separate congestion controllers. Although the two DIFs have the same set of mechanisms, each one can have different *policies* for flow or congestion control that are tailored to the underlying link layer technology or physical link characteristics. The programmable functions inside DIFs are called *policies*. For example, EFCP can be seen as a family

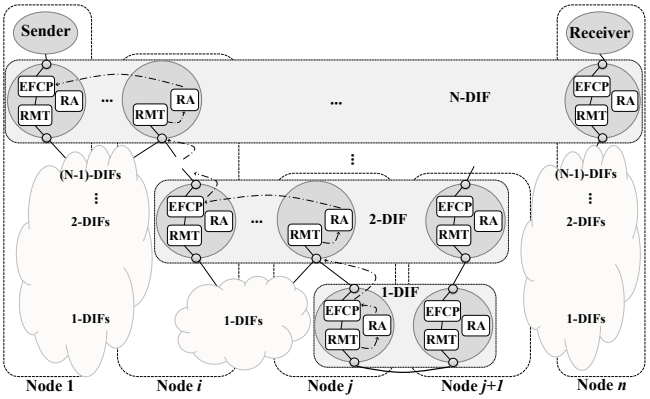
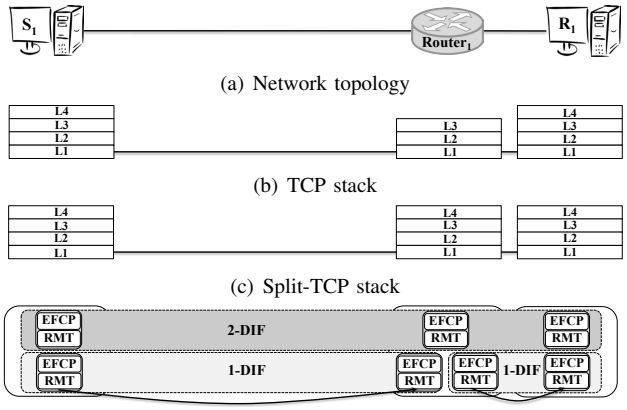


Fig. 3. Recursive congestion control in RINA: the recursive interactition of IPCP modules "Error and Flow Control Protocol" (EFCP), "Relaying and Multiplexing Task" (RMT), and "Resource Allocation" (RA).

of congestion/flow control protocols, and policy is a specific implementation. In the figure, we have another DIF on top. The top DIF's EFCP connection includes no congestion control in our example. EFCP connections inside the layer 1 DIFs include IPC processes $B_{2,1}$ and $B_{2,2}$, which are not directly visible to the layer 2 IPCP A_2. However, A_2 is connected to each one through a logical interface similar to a buffer, called *port*. If A_2 does not receive PDUs from $B_{2,1}$ as quickly as $B_{2,2}$ drains it, A_2 is able to tell $B_{2,1}$ that it can speed up. $B_{2,1}$ can then allow B_1 to send faster via EFCP's flow control. If, on the other hand, $B_{2,2}$ is sending data slower than it arrives at $B_{2,1}$, $B_{2,2}$'s buffer will become full. A_2 is then not able to forward PDUs anymore and, therefore, will have to stop taking them from $B_{2,1}$. Thus, $B_{2,1}$ will again obtain the information it needs to make B_1 slow down via EFCP's flow control.

We see that the way RINA controls congestion is a generalization of how it is done in the Internet: if there is only one DIF doing congestion control in the network, it operates in an end-to-end fashion. If two or more congestion controlled DIFs are concatenated, the end-to-end control loop is broken into shorter loops. As another interesting capability, RINA allows DIFs to be stacked, and upper DIFs can have their own congestion control policies. If, in Fig. 2(b), there are several flows from S_1 to R_1 through several EFCP connections from A_1 to A_3, packets of all of them are mapped to only one EFCP connection in the DIFs below; this means that at the lower DIFs, there is only one *aggregated* flow, and congestion control in these DIFs operates on aggregates. As we consider it an important feature, we generally refer to RINA's congestion control as "Aggregate Congestion Control" (ACC).

A general architecture of N layers is illustrated in Fig. 3. The dashed, curved arrows in the figure represent notification transmission between modules, and IPCPs are shown as circles. In case of congestion in the 1-DIF between nodes j and $j+1$, the Resource Allocation entity (RA) of the transmitting IPC Process sends a notification to the IPCP of the EFCP instance sending the PDU. This EFCP instance is located in the same IPCP. Congestion causes queue growth in this EFCP instance; when the queue reaches its maximum size limit, the EFCP instance shuts down its incoming port, and PDUs are



(a) Network topology

(b) TCP stack

(c) Split-TCP stack

(d) RINA stack. Only EFCP and RMT modules are shown in IPCPs.

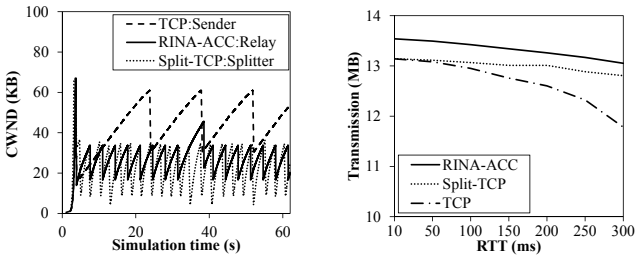
Fig. 4. The topology used for comparison and its correspoding stacks in end-to-end TCP, Split-TCP, and RINA.

backlogged in the upper RMT output port (the port will be unblocked again when the queue length decreases). Next, the output queue of the RMT in 2-DIF will reach its maximum threshold, and this continues; it sends a slow-down signal to the Resource Allocator (RA) module, and the RA sends a notification to the sending EFCP instance. This continues until the signal reaches the source of congestion, generating a "pushback" behavior that *emerges due to recursion.*

A full implementation of the scenario in Fig. 3 requires upstream notifications within a DIF, which can have implications on efficiency (probably positive) and scalability (probably negative) that are beyond the scope of the preliminary investigation presented here. We note, however, that scalability constraints are not as prohibitive here as they are in the Internet: because the Internet architecture essentially limits congestion control to the setup in Fig. 2(a), upstream notifications become impossible to use [8], whereas they merely constrain the maximum size of a DIF in RINA.

III. Simulations With TCP-like Congestion Control

To better understand the implications of placing congestion controlled DIFs next to or above each other, we have carried out simulations of a few "toy" scenarios using the OMNeT++ RINA module [9]. The simulator and scenarios can be found in the provided URL ([9]). We implemented a simple TCP Tahoe-like congestion control policy in EFCP (and, for the simulations in Sections III-A and III-B, nothing else: the effects were entirely achieved by configuring how DIFs are stacked). As we will see, TCP-like congestion control in RINA plays out in ways that are similar to certain functions that are getting deployed as "hacks" to the Internet infrastructure today (e.g. in performance-enhancing proxies (PEPs) [10]) as well as mechanisms that are proposed in the academic literature. In RINA, this behavior naturally appears as a result of layering, whereas the very design of PEPs illustrates the difficulty of overhauling the Internet's base architecture. In the next subsections, we will continue our discussions on three different arrangement types of DIFs which RINA allows us to do: consecutive DIFs, stacked DIFs, and "side-by-side" DIFs.



(a) CWND size of a run with RTT $= 200ms$ (in Kilo Bytes)

(b) Transmitted volume of data (in Mega Bytes)

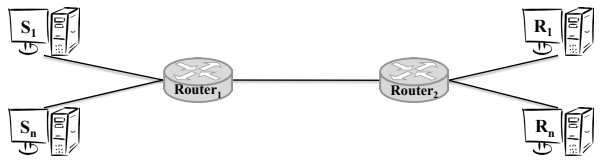
Fig. 5. Comparison results of a single flow in TCP, Split-TCP, and RINA.

A. Horizontal: Consecutive DIFs

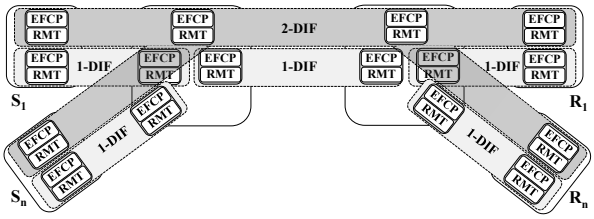
If we consider the description of Fig. 2(b) in the previous section and assume that the congestion control in the Layer 1 DIFs is TCP-like, what we have described is very similar to a PEP function called Connection Splitting [10]. TCP splitters divide TCP connections by "lying" to the end systems, acting as the receiver towards the sender and as the sender towards the receiver. As one significant difference to RINA-ACC, a TCP splitter must take care of end-to-end reliability, thereby breaking the end-to-end reliability semantics of TCP. While retransmitting packets from within the network can also have benefits and would easily be possible with RINA, it is not automatically tied to dividing the control loops as it is with TCP, and we do not consider it further here.

To compare RINA-ACC with Split-TCP and see the performance gain over end-to-end TCP, we simulated data transfers in the topology shown in Fig. 4; Router_1 acts as a splitter while using Split-TCP, which we have added to the OMNeT++ INET framework as implemented in [11]. The protocol stacks of the three approaches are also shown for clarity. The traffic sent by S_1 was a single large file, so there was only one flow from S_1 to R_1 in the network. The simulation was run for one minute, and after that we collected the total volume of data transmitted. The capacity of the link between the sender and the router was 10 Mbps, and for the link between Router_1 and R_1 it was 2 Mbps. Congestion appeared in the interface on the right-hand side of Router_1. We limited the output queue of this interface to the bandwidth-delay product (BDP) in the end-to-end TCP case. In the Split-TCP and RINA-ACC cases, we used the BDP of the Router_1–R_1 link for this buffer as well as the RMT's output buffer at the layer above.

The performance of end-to-end TCP, Split-TCP, and RINA with TCP per DIF is illustrated in Fig. 5. This diagram shows two separate and expected things: 1) Fig. 5(a) shows congestion window (CWND) sizes of the sender in end-to-end TCP ("TCP:Sender") and the second connections of Split-TCP and RINA ("Split-TCP:Splitter" and "RINA-ACC:Relay") between Router_1 and R_1; this implies that our implementation of TCP in RINA indeed performs very similar to Split-TCP (with a small throughput improvement because its pushback mechanism prevents packet drops and timeout), and 2) referring to Fig. 5(b), Split-TCP performs better than end-to-end TCP at large RTTs. The latter fact is known from earlier literature but it shows that our implementation of Split-TCP is correct.



(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.

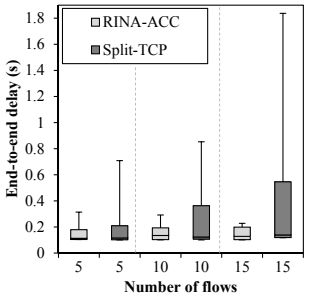
Fig. 6. Network topology for multiple flows and its RINA stack.

B. Vertical: Stacked DIFs

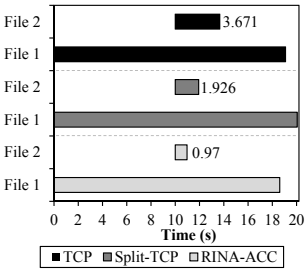
In the simplistic example above, the traffic carried by RINA came from one single end-to-end flow in the network. In a real RINA network where DIFs are stacked above each other, an N-DIF would carry an aggregate of flows from the (N+1)-DIF sitting above it. Edge router pairs would then only keep the congestion state of active flow aggregates between them. Here, RINA-ACC automatically avoids the competition between multiple end-to-end flows that occurs in the Internet today: when many end-to-end or Split-TCP flows individually push up the queue at the bottleneck, they harm each other via increased delay and loss, and it can be better to combine them [12], [13]. We simulated multiple flows competing in the network using the network topology shown in Fig. 6. Senders S_1 through S_n sent a large file to receivers R_1 through R_n, respectively. All the links had the same bandwidth; the $Router_1$–$Router_2$ link was the bottleneck. For RINA, the DIF structure is also indicated in Fig. 6: there were some consecutive lower DIFs and one upper-layer DIF on top. We compared RINA-ACC against the better Internet case from our previous simulations – Split-TCP, but with no aggregation.

In Fig. 7(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 $Router_1$–$Router_2$ 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 does not resolve the high jitter problem arisen by 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.



(a) End-to-end delay



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

Fig. 7. Benefits of RINA ACC.

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.

The previous simulation scenario showed the advantage of aggregating flows in the core of the network. A similar argument holds when a number of flows originate from the same sender to the same receiver. Following an example from [14], a user might be surfing a web site while downloading a file from it – in this case, it can be more efficient to use only one congestion controller between the two nodes.

We simulated this scenario and show the results in Fig. 7(b). The network topology was the same as in Fig. 6 with $n = 1$. The sender sent two files to the receiver. The transmission of File 1 with the size of 20 MB 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 into 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. Moreover, compared to TCP and Split-TCP, there is less competition in this case which results in a shorter transmission time for the first file. In Split-TCP, the two flows were split into six pieces, and every two connections competed for one of the three links. Compared with the TCP scenario in which there was only one bottleneck link, here there were three bottleneck links because sometimes, for example, the total congestion window size in the splitter of $Router_1$ became smaller than the total size in the sender. In our simulation, this caused one timeout in the second connection of File 1; this is why it took around 1 second longer than with TCP.

The effect shown here with ACC is comparable to a Congestion Manager (CM) [15] or multi-streaming as in e.g. SCTP [14] – but with these mechanisms, the benefit shown in Fig. 7(b) only appears when multiple flows are used as described between the same sender and receiver. With RINA-ACC however, the two files could just as well be transferred between different sender-receiver pairs, and the same holds if there are hundreds or thousands of senders: File 2 can be transmitted faster if *any* sender is transmitting data at any given time. In-network aggregate congestion control is, therefore, a much more powerful mechanism than previously proposed methods for congestion control coupling.

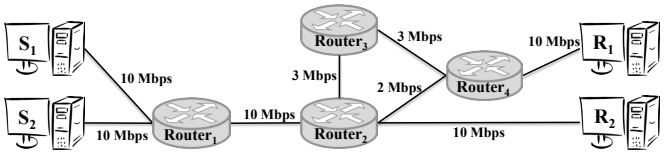


Fig. 8. A toy topology for evaluating INRP (adopted from [19]).

C. Around: In-Network Resource Pooling

Resource pooling [16] has recently gained significant attention from the routing and congestion control point of view. There exist some methods at the transport layer like Multi-Path TCP (MPTCP) [17] or at the routing layer like Equal-Cost Multi-Path Routing (ECMP) [18] with the aim of load-balancing. However, they operate independently. A recent approach to enable a combination of these two types of mechanisms was presented by the name of In-Network Resouce Pooling (INRP) in [19]. This work claims that dynamic routing/detouring compared with the Internet's static routing, as a way of reacting to congestion by finding alternative lower-load routes, combined with a hop-by-hop congestion control mechanism in case there is no other low-load path towards the destination, is beneficial. However, this approach might face Internet-specific problems such as head-of-line blocking in case of detouring TCP traffic [20].

RINA enables us to cope with these problems with its layered architecture; it offers various options for reliable out-of-order delivery between any two EFCP instances, and it is up to each layer how to serve upper layers (DIFs). For example, as proposed by [19], [21], DIFs are automatically able to serve upper DIFs with multiple flows like *flowlets* through multiple paths. These paths are provided through multiple side-by-side lower DIFs operating on different paths. Detouring can also be bound to some lower DIF in the core of a network, and that DIF might even provide in-order-delivery to upper DIFs. At the edge of a DIF, delay from head-of-line blocking is bounded whereas in TCP it is accumulated along the path.

To show how easily RINA can provide resource pooling, we implemented a load-based routing policy in the RINA simulator in OMNeT++. This policy is coupled with another short-term queue size monitoring policy to react to local congestion in an output queue. In case of congestion, first another low-load path to the destination is looked up. If no other path is available, the congestion control module reacts.

Fig. 8 shows the simple topology that was used to explain in-network resource pooling in [19]. In case of proper load-balancing, both S_1–R_1 and S_2–R_2 connections should be allocated a 5 Mbps share. The main advantage of resource pooling here is that it provides global fairness and local stability [19]. The corresponding RINA stack is similar to the one in Fig. 6 meaning that there is one 1-DIF per link, and a single 2-DIF connecting all the nodes on top. We ran this scenario in RINASim for one minute, and measured the volume of data received by R_1 and R_2. Jain's fairness index [22] with three digits precision was 0.999, which shows global fairness while local stability was provided through RINA-ACC as shown before.

IV. Discussion and Implications

Because it operates at the transport layer, it is impossible for TCP to always do "the right thing" for every network segment. By measuring only the round-trip time (RTT) and packet loss, it is very difficult to optimally adapt the transmission rate of the sender when the path to the receiver is a chain of technologies (e.g. WLAN, Ethernet, satellite, 3G). TCP cannot make any form of link technology-specific decision. This problem is compounded by well-known TCP issues like e.g. its inability to distinguish between congestion losses and losses that are due to link impairments. It should not be surprising that there is a huge amount of research work on cross-layering with a focus on TCP-over-X, where X is any link layer technology and X is assumed to be the only problematic link. Similarly, given that TCP must work everywhere, it should not be surprising that such research does not typically influence the TCP standard. TCP's end-to-end congestion control also does not scale well in several dimensions:

1) *The diameter of the network*. The effectiveness of any congestion control scheme will deteriorate with increasing network diameter. TCP maximizes this effect because it operates in "rounds" based on the round-trip time. This is particularly problematic in high bandwidth networks where the capacity does not become a limit, especially with short flows that are common in web traffic. Such flows typically terminate in TCP's slow-start phase, making the completion time strictly a function of the round-trip time. Recently, there has therefore been increased interest in reducing the number of round-trips for web traffic [23].

2) *The number of flows*. When multiple flows traverse a path, they compete for the available bandwidth, pushing up the queues and creating delay and loss. As shown in [12], jointly controlling them as a group can lead to much better behavior and enable precise prioritization between flows. This functionality is currently being proposed for WebRTC in the IETF RMCAT Working Group [24].

3) *The bottleneck link capacity*. TCP often poorly saturates high-capacity links due to its linear increase in standard TCP's congestion avoidance phase. This has been addressed approximately a decade ago; now, the most prominent solution is the CUBIC congestion control mechanism that is used by default in Linux hosts [25]. When CUBIC and other related mechanisms were proposed, explicit feedback-based schemes were found to operate best (e.g. XCP [26], RCP [27], MaxNet [28], and CADPC/PTP [29]).

RINA can solve all of these problems 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.

As we have seen in the previous section, with TCP, the behavior emerging with RINA can resemble Performance Enhancing Proxies (PEPs) [30]. PEPs usually break end-to-end connections into several closed-loop connections; in each connection, a congestion control scheme which considers local link characteristics can be exploited [31]. Also, hop-by-

hop congestion control has gained much attention in wireless networks due to serious problems of end-to-end methods [32].

Internet PEPs have several disadvantages. In addition to the already mentioned reliability problem (see Section III-A), complexities in using IPsec and SSL [33] arise because security is an end-to-end function in these cases. This is why works such as [33] are motivated by designing PEP-less solutions. PEPs also do not scale well with the number of flows [32], [34]. A mapping of incoming-outgoing connection pairs to interfaces must be maintained to be able to e.g. slow down the correct incoming connections when congestion is experienced on an outgoing interface [35]. TCP's fast retransmission/recovery is triggered by triple duplicate ACKs which needs a window size of at least 4 packets, meaning that a splitter's buffer size (advertised window) must be at least 4 times the number of flows [34]; otherwise, performance degrades. In addition, in terms of the processing delay, running a separate TCP instance for every flow is expensive [19].

The layered architecture of RINA does not have these problems of Internet PEPs because 1) security is a per-DIF function: PDUs are protected as they cross DIF boundaries [7], and because each DIF has its own congestion control, splitting encrypted connections is not problematic, and 2) flows towards each next hop are aggregated at the lower DIFs, which means that there is much less state to maintain.

In TCP splitters, buffer sizes have also been shown to affect the performance, but, as shown by [36], the required splitter's buffer size is fairly modest, and it is independent of the number of flows. The splitter's buffer size scales linearly with BDP of the bottleneck link [34]. In our evaluations, we used a buffer of the size of its link's BDP in RINA for RMT output queues at upper DIFs. Running the same simulation with different buffer sizes revealed that in our scenarios, buffers of at least this size result in a high performance, close to the upper limit, while imposing a reasonable queuing delay.

A. Stability and Scalability

We have discussed benefits of breaking up control loops; at the extreme end of this approach we have hop-by-hop congestion control. There has been a concern regarding stability of hop-by-hop congestion control methods since many years ago because some unstable behavior was observed in some networks [37], [38]. However, as [38] argues, "the fear of instability" might be a result of the unsuccessful use of non-discriminatory on-off type controls.

Many works have investigated the stability of hop-by-hop congestion control, e.g. [32], [35], [37]–[39]. They showed that using per-hop controllers accelerates the response to bandwidth changes in the path, and that series of control loops are stable under certain conditions. These methods usually use a rate-based mechanism with proven stability properties. Split-TCP consists of several consecutive TCP connections, and each TCP connection has been shown to have stability properties [35], [40] especially for shorter RTTs. In [41], it is also proven that even in case of not using TCP receiver-window based backpressure in Split-TCP, the system is stable under certain conditions.

In RINA, consecutive DIFs operate similarly to Split-TCP, and due to our TCP-like controller implementation, we inherit the same stability properties. If DIFs are layered, as shown in Fig. 3, it can be observed that we have a concatenation of TCP controllers; however, these controllers, from the 1-DIF, cross the layers up to the EFCP instance at the sender node in the N-DIF. In other words, there is one active controller between every two consecutive EFCP senders shown in the figure down to the bottleneck link. This implies that in the recursive form we have the same stability properties. Hop-by-hop rate-based congestion controllers with proven stability which were mentioned above can also be used in RINA. Due to the inherent flow aggregation in RINA, using this kind of controllers imposes limited overhead on intermediate nodes in RINA; this overhead has, however, been a major obstacle in deploying these approaches previously [35].

An important issue regarding using stable hop-by-hop congestion controllers is how well they can scale [32] because a larger sequence of controllers might affect stability. However, works such as [40] show that it is possible to achieve a scalable and stable method for arbitrarily large network topologies and arbitrary delays.

In general, any form of hop-by-hop congestion control or connection splitting can be viewed as a concatenation of several controllers. There is a large number of works in the control theory literature on the stability of interconnected systems, cf. [42]–[44]; each of them considers different assumptions in the system such as parameter dependencies between subsystems or time-varying properties. These works show that even under arbitrary interconnection of systems, stability can be achieved. These results can be helpful in the design of stable congestion controllers for RINA under the same assumptions.

RINA theoretically allows an infinite number of DIFs to be stacked; each DIF has its own buffers, and hence, the total end-to-end buffer size in RINA is another matter of further investigation. We do however not expect this to become a significant issue, as hop-by-hop rate-based congestion control schemes such as [32], [39] have already been shown to be more efficient than end-to-end approaches in terms of buffer usage (e.g. peak buffer size) at intermediate nodes.

V. Concluding Remarks

The Internet's end-to-end'ness of congestion control and its static routing are inevitable by-products of its architectural design. With RINA, the natural way of applying such algorithms is very different, with control executed closer to where the problem is. The main achievement of RINA is that the series of hacks and patches found in the Internet, with its problems that we have discussed in the case of congestion control, are not required. RINA is therefore an ideal vehicle for investigating drastic changes to how congestion control, and in-network resource pooling as another example, could be done, and it provides a suitable framework with many promising dimensions for future research.

In this paper, through some simple evaluation topologies, we have shown that congestion control in RINA "naturally" exhibits properties of various improvements that have been made to (or at least proposed for) the Internet, without inheriting the problems that come from imposing these mechanisms on an architecture that was not made for them (all the problems that PEPs have). However, it is just as "natural" that such a drastic departure from common methods requires answering many new questions, such as: 1) how to effectively manage buffers between DIFs, 2) which hop-by-hop congestion controllers are best to use, given their stability and scalability properties, 3) how to best apply in-network resource pooling, 4) effects of different congestion control policies at lower DIFs on congestion control policies of upper DIFs, and 5) how large a DIF can be without performance degradation, and how its scalability can be improved. We believe that these are exciting issues for future work, and hope that this paper inspires others to also consider RINA for congestion control research.

REFERENCES

[1] J. H. Saltzer, D. P. Reed, and D. D. Clark, "End-to-end arguments in system design," *ACM Trans. Comput. Syst.*, vol. 2, no. 4, pp. 277–288, Nov 1984.

[2] J. Day, I. Matta, and K. Mattar, "Networking is IPC: a guiding principle to a better internet," in *Proc. ACM CoNEXT*, 2008, p. 67.

[3] J. Day, *Patterns in Network Architecture: A Return to Fundamentals*. Prentice Hall, 2007.

[4] ——, "How in the heck do you lose a layer!?" in *Network of the Future (NOF), 2011 International Conference on the*, Nov 2011, pp. 135–143.

[5] G. Gursun, I. Matta, and K. Mattar, "On the performance and robustness of managing reliable transport connections," CS Department, Boston University, Tech. Rep., 2009, bUCS-TR-2009-014.

[6] V. Ishakian, J. Akinwumi, F. Esposito, and I. Matta, "On supporting mobility and multihoming in recursive internet architectures," *Computer Communications*, vol. 35, no. 13, pp. 1561–1573, 2012.

[7] J. Small, "Patterns in network security: An analysis of architectural complexity in securing recursive inter-network architecture networks," Master's thesis, Boston University Metropolitan College, 2012.

[8] F. Gont, "Deprecation of ICMP Source Quench Messages," RFC 6633 (Proposed Standard), Internet Engineering Task Force, May 2012. [Online]. Available: http://www.ietf.org/rfc/rfc6633.txt

[9] RINASim, "Rina simulator," https://github.com/kvetak/RINA/tree/UiO-ACC/examples/SmallNetwork2, 2015.

[10] 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

[11] A. Bakre and B. Badrinath, "I-tcp: indirect tcp for mobile hosts," in *Proc. IEEE ICDCS*, May 1995, pp. 136–143.

[12] S. Islam, M. Welzl, S. Gjessing, and N. Khademi, "Coupled congestion control for RTP media," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 4, p. 101, Aug 2014.

[13] F.-C. Kuo and X. Fu, "Probe-Aided MulTCP: An aggregate congestion control mechanism," *SIGCOMM CCR*, vol. 38, no. 1, Jan 2008.

[14] M. Welzl, F. Niederbacher, and S. Gjessing, "Beneficial transparent deployment of sctp: the missing pieces," in *Proc. IEEE GLOBECOM*, 2011, pp. 1–5.

[15] H. Balakrishnan, H. S. Rahul, and S. Seshan, "An Integrated Congestion Management Architecture for Internet Hosts," in *SIGCOMM*, 1999, pp. 175–187.

[16] D. Wischik, M. Handley, and M. B. Braun, "The resource pooling principle," *ACM SIGCOMM CCR*, vol. 38, no. 5, pp. 47–52, 2008.

[17] D. Wischik, C. Raiciu, A. Greenhalgh, and M. Handley, "Design, Implementation and Evaluation of Congestion Control for Multipath TCP," in *NSDI*, vol. 11, 2011, pp. 8–8.

[18] C. Hopps, "Analysis of an Equal-Cost Multi-Path Algorithm," RFC 2992 (Informational), Internet Engineering Task Force, Nov 2000. [Online]. Available: http://www.ietf.org/rfc/rfc2992.txt

[19] I. Psaras, L. Saino, and G. Pavlou, "Revisiting resource pooling: The case for in-network resource sharing," in *Proceedings of the 13th ACM Workshop on Hot Topics in Networks*, ser. HotNets-XIII. New York, NY, USA: ACM, 2014, pp. 24:1–24:7.

[20] B. Briscoe, A. Brunstrom, A. Petlund, D. Hayes, D. Ros, I.-J. Tsang, S. Gjessing, G. Fairhurst, C. Griwodz, and M. Welzl, "Reducing internet latency: A survey of techniques and their merits," *Communications Surveys Tutorials, IEEE*, vol. PP, no. 99, pp. 1–1, 2014.

[21] S. Sinha, S. Kandula, and D. Katabi, "Harnessing TCPs Burstiness using Flowlet Switching," in *3rd ACM SIGCOMM Workshop on Hot Topics in Networks (HotNets)*, San Diego, CA, November 2004.

[22] R. Jain, D. Chiu, and W. Hawe, "A Quantitative Measure of Fairness and Discrimination for Resource Allocation in Shared Computer Systems," DEC, Tech. Rep. TR-301, 1984.

[23] T. Flach, N. Dukkipati, A. Terzis, B. Raghavan, N. Cardwell, Y. Cheng, A. Jain, S. Hao, E. Katz-Bassett, and R. Govindan, "Reducing web latency: the virtue of gentle aggression," in *Proc. ACM SIGCOMM*, 2013.

[24] M. Welzl, S. Islam, and S. Gjessing, "Coupled congestion control for RTP media," Internet-draft (work in progress) draft-welzl-rmcat-coupled-cc-05.txt, 2015.

[25] S. Ha, I. Rhee, and L. Xu, "CUBIC: A New TCP-friendly High-speed TCP Variant," *SIGOPS Oper. Syst. Rev.*, vol. 42, no. 5, pp. 64–74, Jul 2008.

[26] D. Katabi, M. Handley, and C. Rohrs, "Congestion control for high bandwidth-delay product networks," *ACM SIGCOMM Computer Communication Review*, vol. 32, no. 4, pp. 89–102, 2002.

[27] N. Dukkipati, N. McKeown, and A. G. Fraser, "RCP-AC: Congestion control to make flows complete quickly in any environment," in *Proceedings of IEEE INFOCOM*, 2006, pp. 1–5.

[28] B. P. Wydrowski, L. L. Andrew, and I. M. Mareels, "Maxnet: Faster flow control convergence," in *Proc. Networking*. Springer, 2004.

[29] M. Welzl, *Scalable performance signalling and congestion avoidance*. Springer Science & Business Media, 2012.

[30] C. Caini, R. Firrincieli, and D. Lacamera, "PEPsal: a Performance Enhancing Proxy for TCP satellite connections," *IEEE Aerospace and Electronic Systems Magazine*, vol. 22, no. 8, pp. 7–16, 2007.

[31] A. Kapoor, A. Falk, T. Faber, and Y. Pryadkin, "Achieving faster access to satellite link bandwidth," in *Proc. IEEE INFOCOM*, 2006.

[32] Y. Yi and S. Shakkottai, "Hop-by-hop congestion control over a wireless multi-hop network," *IEEE/ACM ToN*, vol. 15, no. 1, pp. 133–144, 2007.

[33] T. T. Thai, D. M. L. Pacheco, E. Lochin, and F. Arnal, "SatERN: a PEP-less solution for satellite communications," in *Proc. IEEE ICC*, 2011, pp. 1–5.

[34] J. Zhu, S. Roy, and J. H. Kim, "Performance modelling of TCP enhancements in terrestrial-satellite hybrid networks," *IEEE/ACM Transactions on Networking (TON)*, vol. 14, no. 4, pp. 753–766, 2006.

[35] S. Bohacek, "Stability of hop-by-hop congestion control," in *Proc. IEEE Decision and Control*, vol. 1, 2000, pp. 67–72.

[36] M. Luglio, M. Y. Sanadidi, M. Gerla, and J. Stepanek, "On-board satellite "split tcp" proxy," *Selected Areas in Communications, IEEE Journal on*, vol. 22, no. 2, pp. 362–370, 2004.

[37] V. Kulkarni, S. Bohacek, and M. Safonov, "Stability issues in hop-by-hop rate based congestion control," in *Proc. Annual Allerton Conference on Communication Control and Computing*, vol. 36. University of Illinois, 1998, pp. 79–88.

[38] P. P. Mishra and H. Kanakia, "A hop by hop rate-based congestion control scheme," in *ACM SIGCOMM CCR 22(4)*, 1992.

[39] P. P. Mishra, H. Kanakia, and S. K. Tripathi, "On hop-by-hop rate-based congestion control," *IEEE/ACM TON*, vol. 4, no. 2, pp. 224–239, 1996.

[40] F. Paganini, J. Doyle, and S. Low, "Scalable laws for stable network congestion control," in *Proc. IEEE Decision and Control*, 2001.

[41] F. Baccelli, G. Carofiglio, and S. Foss, "Proxy caching in split TCP: Dynamics, stability and tail asymptotics," in *INFOCOM*, 2008.

[42] C. Langbort, R. S. Chandra, and R. D'Andrea, "Distributed control design for systems interconnected over an arbitrary graph," *IEEE Trans. Automatic Control*, vol. 49, no. 9, pp. 1502–1519, 2004.

[43] C. Maffezzoni, N. Schiavoni, and G. Ferretti, "Robust design of cascade control," *IEEE Control Systems*, vol. 10, no. 1, pp. 21–25, 1990.

[44] N. Motee and A. Jadbabaie, "Optimal control of spatially distributed systems," *IEEE Trans. Automatic Control*, vol. 53, no. 7, pp. 1616–1629, 2008.

Even Lower Latency, Even Better Fairness: Logistic Growth Congestion Control in Datacenters

Peyman Teymoori, David Hayes, Michael Welzl, Stein Gjessing
Department of Informatics
University of Oslo
Email: {peymant|davihay|michawe|steing}@ifi.uio.no

***Abstract*—Datacenter transport has attracted much recent interest, however, most proposed improvements require changing the datacenter fabric, which hinders their applicability and deployability over commodity hardware. In this paper, we present a novel congestion controller, Logistic Growth Control (LGC), for datacenters which does not require changes to the datacenter fabric. LGC uses a similar ECN marking as in DCTCP, but adapts to congestion using the logistic growth function. This function has been proven to have nice characteristics including stability, convergence, fairness, and scalability, which are very appealing for congestion control. As a result, our LGC mechanism operates in the datacenter network in a more stable and fair manner, leading to less queuing and latency. LGC also 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.**

I. Introduction

Standard transport protocols, such as the Transmission Control Protocol (TCP), do not perform well in datacenters [2]. Optimizing both the end systems and network fabric to the datacenter environment (e.g. [4]) has many benefits, but often proves difficult or impossible to deploy (see [9]), and the network fabric needs special approaches such as [23] to be able to run the new algorithm. In addition, in spite of the performance improvements achieved by custom-built datacenter fabrics, the price differences of commodity versus non-commodity networking hardware, especially at a large scale, has been a strong motivation towards using commodity hardware [1], [22].

Another approach in datacenters has been to focus on easily deployable solutions. These include purely end-system modifications, such as Data Center TCP (DCTCP) [2]) as well as scheduling transmissions [20], [11], though the later has scalability concerns due to their complexity or the use of a centralized arbiter [17]. Hence, like DCTCP [2], we too focus on an easily deployable mechanism. Such a mechanism operates purely at the transport layer of end systems and does not require changes to network equipments such as switches and routers.

Our work is inspired by logistic growth in nature [28] (see section II-A). We build upon earlier work that has found logistic growth to be a generally useful function for congestion control [29], [30], [10], and present the design and simulation-based evaluation of a new congestion controller for datacenters that is based on logistic growth. Cornerstones of our design are:

- Similar to DCTCP, we let packets be ECN-marked 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, and hence needs no hardware changes). However, different from DCTCP where this threshold is a function of the Bandwidth×Delay Product (BDP) [2], which can be very large in modern datacenters [13], 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 stability properties when used in congestion control (see Section IV-C), and lets us attain fairness among flows irrespective of the RTT, which is not the case for TCP and DCTCP.

Through presenting an analytical model, we discuss LGC properties, and through simulation in OMNeT++ [19], we show that it works under various scenarios. Simulation results show that it can reduce the queue length and latency, and it attains fairness among flows with heterogeneous RTTs.

Section II presents a congestion controller model based on the logistic growth function. In Section III we develop our congestion controller based on this model and then analytically evaluate it in Section IV. We show how it can be further improved in Section V before evaluating the performance of our logistic growth based congestion controller through simulations in Section VI. We finish with an overview of related work in Section VII and our conclusions in Section VIII.

II. Congestion Controller Model

A. Logistic Growth

The Logistic Growth (LG) function is often used to specify the growth of a population/species over time. It is described by the differential equation

$$\dot{N} = \frac{rN(t)(K - N(t))}{K} \tag{1}$$

TABLE I
NOTATIONS

Symbol	Description
S	the number of competing flows/species
x_i	the normalized rate of flow i
$x_i(0)$	the initial rate/population of flow/species i
r_i	the growth rate of flow i
a_{ij}	the competitive effect of species i on species j
$l(t)$ or $l[n]$	$\frac{S-1}{S}$ times the total load on the bottleneck link
$\hat{l}_i(t)$ or $\hat{l}_i[n]$	the approximation of $l(t)$ and $l[n]$ respectively
$c(t)$ or $c[n]$	the ECN-marking process
$\hat{x}_i(t)$ or $\hat{x}_i[n]$	an approximation of the fair sending rate of source i
K	the carrying capacity: the source's maximum sending rate (the normalization factor)
k	the ratio of the capacity of the bottleneck link to K
var(t)	is used to represent a continuous time variable
var$[n]$	is used to represent discrete variable at epoch n

where 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. We use the dot notation for time differentiation, i.e. $\dot{N} = \frac{dN}{dt}$. Table I summarizes the notation we use throughout the paper.

The idea of growth constrained by a capacity limit has direct parallels with network congestion control. It allows us to draw upon the large body of analytic research work on this model and its enhancements (e.g. [16]) which has proven nice characteristics including stability, convergence, fairness, and scalability. These properties make LG very appealing for congestion control [10].

We normalize (1) by defining $x(t) = N(t)/K$, which yields

$$\dot{x} = rx(t)\left(1 - x(t)\right) \tag{2}$$

with the solution:

$$x(t) = \frac{1}{1 + (\frac{1}{x(0)} - 1)e^{-rt}} \tag{3}$$

where $x(0)$ denotes the initial population. Clearly, $\lim_{t\to\infty} x(t) = 1$. In the rest of the paper, we use LG in this normalized form.

Fig. 1 plots (3) for different values of r, which affects the speed of convergence. In this diagram, there is one species with the normalized initial population of 0.1. We see that larger values of r result in a faster convergence to the carrying capacity 1. The LG function (3), however, models the population of only one species. When there is a competition among a number of species for a common resource, the competition is modeled using the Lotka-Volterra model, which can also parallel different network sources competing for shared network capacity.

B. Lotka-Volterra Competition Model

We consider the general Lotka-Volterra model of competition [27], where S species compete for a common limited

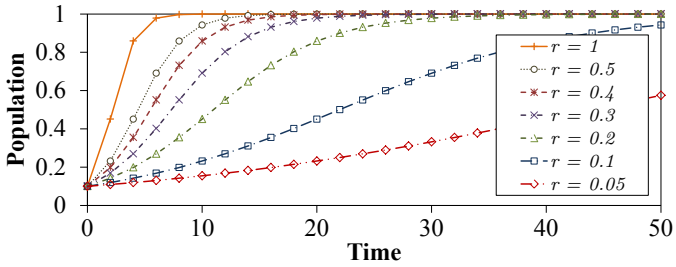


Fig. 1. Growth rate effects on the population growth of LG. $x(0) = 0.1$.

resource according to the Logistic Growth equation

$$\dot{x_i} = x_i r_i \left(1 - \sum_{j=1}^{S} a_{ij} x_j\right) \tag{4}$$

where $r_i > 0$ denotes the growth rate of species i. $\mathbf{A} = (a_{ij}) \in \mathbb{R}^{S\times S}$ is called the community matrix [18], where the value of a_{ij} determines the competitive effect of species i on species j.

We define matrix $\mathbf{A}$ as

$$a_{ii} = \frac{2S-1}{S}, \qquad a_{ij} = \frac{S-1}{S} \forall i \neq j. \tag{5}$$

When $a_{ij} > 0\ \forall i \neq j$, all species i have a competitive effect on all species j. This parallels network flows competing with one another. However if a_{ij} and a_{ji} have different signs, the interactions are more complex with predation between species. This can be used for more complex network analysis, but is beyond the scope of this paper.

The definition of $\mathbf{A}$ directly affects the stability of the system represented by (4) [14]; in the following, we elaborate more on why we define $\mathbf{A}$ as (5).

1) Equilibrium: The equilibria of (4), denoted by $\mathbf{x}^*$ in the vector form with elements x_i^*, are determined by solving the system of equations $\dot{x_i} = 0\ \forall i$. This implies that either $x_i = 0$ or $1 - \sum_{j=1}^{S} a_{ij}x_j = 0$. If no species dies out, $x_i > 0$ (or for network all flows are sending), and using (5), the only equilibrium point we get is

$$x_i^* = \frac{1}{S}. \tag{6}$$

Thus all species, or flows in our case, converge to an equal share of the capacity, jointly converging to 1.

2) Stability: It can easily be shown that because in our case matrix $\mathbf{A}$ is symmetric with positive real elements, its eigenvalues are real and positive. According to (5), $a_{ii} > a_{ij}\ \forall i \neq j$, which makes all the pivots of $\mathbf{A}$ positive. This accordingly implies that all eigenvalues of $\mathbf{A}$ are positive. Thus the system represented by (4) is stable around the equilibrium (6) [15]. The symmetric property of $\mathbf{A}$ with $a_{ij} > 0$ results in another nice property of the model: the above equilibrium becomes globally stable because there can be found a positive definite Lyapunov function minimized at the equilibrium with positive values at other points [15], [16]. In the networking context, this means that all network flows competing for the bottleneck link capacity using (4) will

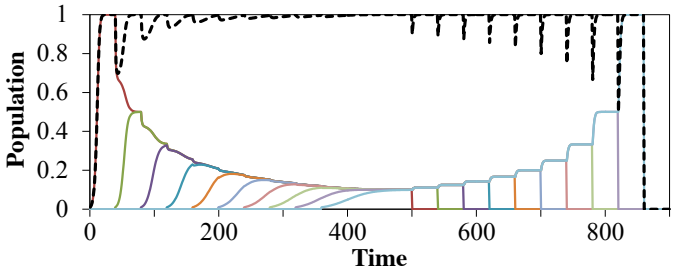


Fig. 2. Competition of 10 species (the solid lines) with $r_i = 0.5$ based on the LV system. The dashed line is the total population.

eventually converge to a stable equilibrium (6), irrespective of their initial sending rate.

3) Numerical evaluation: To illustrate how the system of equations presented in (4) behaves, a sample trajectory of 10 species is plotted in Fig. 2. Each species starts with an initial population of 0.01, but at different time instants. All the numbers are normalized with the carrying capacity equal to 1. It is clearly seen in Fig. 2 how each species' population grows until it receives its fair share of the total capacity, which is 0.1 once all 10 species have joined at around 400. As species *leave* the competition one by one, meaning that they are removed from the system, we can observe how the system converges to the new equilibrium. The stable adaption for varying numbers of species parallels varying number of network flows and how capacity is shared between them in the networking context.

III. Logistic Growth Control (LGC)

A. Congestion Controller

Applying (4) to the context of a distributed congestion controller that operates at discrete time intervals, we write (4) with (5) as follows:

$$x_i[n+1] = x_i[n] r_i \left(1 - \frac{2S-1}{S} x_i[n] - \frac{S-1}{S} \sum_{j,j\neq i} x_j[n]\right) + x_i[n]. \tag{7}$$

After simplifying the right-hand side of (7), we get

$$x_i[n+1] = x_i[n] r_i \left(1 - x_i[n] - l[n]\right) + x_i[n] \tag{8}$$

where $l[n] = \frac{S-1}{S} \sum_j x_j[n]$. At equilibrium $x_i[n+1] = x_i[n]$ in (7), yielding $x_i^* = \frac{1}{S} \ \forall i$ at that fixed point. This shows that the discrete equation has the same equilibrium point as (4).

Key to implementing LGC in a datacenter is being able to estimate $l[n]$. This requires that each source, i, is able to estimate the number of competing flows, S, as well as the combined sources transmission rate, $\sum_j x_j[n]$ (Remember these rates are normalized with respect to the source link's capacity). In the next section, we will explain how each source can have a good approximation of $l[n]$ without requiring to know the exact value of S.

B. Estimating $l[n]$

In an unmodified IP-based datacenter fabric, exactly calculating $l[n]$ is not possible. The only signal from the network is the ECN flag in the packet header; a set flag indicates the queue length has exceeded a particular threshold, and a clear flag means it hasn't. However, we show that it is possible to accurately estimate $l[n]$ using ECN in the datacenter environment when for source i, $\hat{l}_i[n] = c_i[n]$, where $c_i[n]$ is the proportion of congestion indicating packets (ECN marked) in epoch n. Thus equation (8) is implemented as:

$$x_i[n+1] = x_i[n] r_i \big(1 - x_i[n] - \hat{l}_i[n]\big) + x_i[n] \ . \tag{9}$$

When a source starts transmission, it does not have any information to calculate $\hat{l}[0]$ so we start with $\hat{l}[0] = 0$. Section IV-D discusses the accuracy of this estimate in detail.

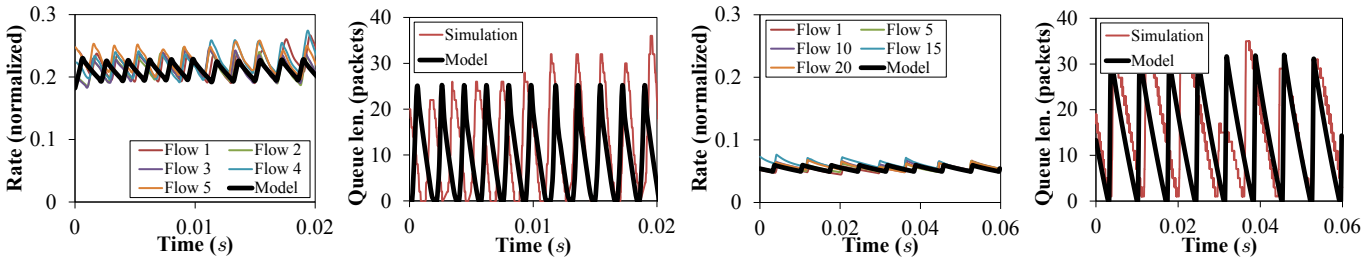
C. The Link Capacity

In (8) the capacity is normalized. Therefore, to send at a specific rate, every source should send at $x_i K$ where K is the maximum send rate of the source. In a general network topology, the link capacities in a path might be different or unknown. However, all the link capacities and the topology are known in datacenters, and as a general rule, we assume that K represents the smallest link capacity in a path. For example, in a datacenter with 10Gbps links between servers and TOR switches, and 40Gbps links between other switches, we use 10Gbps as the carrying capacity, K, in the calculations. We will show that if the 40Gbps link is congested, LCG is also able to react appropriately to the congestion, i.e. congestion in the backbone. This suggests that LGC may also be able to be applied outside datacenters to networks where the bottleneck capacity is not known.

D. Rate Update Time Intervals

The rate is regularly updated by each source using (9). As we see in (9), the fair rate only depends on $\hat{l}_i$, and as long as all sources *see* the same $\hat{l}$ (see Section IV-D), they converge to the same rate. This means that the steady state behavior of the controller is not sensitive to the update interval: updating it more or less often will naturally make the control converge faster or slower, but the point of convergence is not affected (but updating it extremely fast could lead to oscillations). There are several options for the update frequency such as using the smallest RTT value seen over a certain time interval (an estimate of the RTT without queuing delay). However, using a flow's smallest RTT can still be a long time interval for a large-RTT flow, making it sluggish.

In LGC, we use equal-sized intervals. Based on our observation in simulations, performance is robust if a flow can receive at least 4 acks per interval. Although there are many active flows in a datacenter, the number of competing flows for the same link is small [8], [17]. This means a rate update interval as short as the minimum RTT in a datacenter will still allow flows around 4 acks per interval (or the equivalent information using DCTCP's delayed ack mechanism).



(a) Rate of 5 flows

(b) Queue length of the bottleneck link when 5 flows are competing

(c) Rate of 20 flows. 5 sample flows are only shown.

(d) Queue length of the bottleneck link when 20 flows are competing

Fig. 3. Validation of the fluid model against simulation. The bottleneck link capacity is 10 Gbps, and the RTT is 200 μs

IV. Analytical Evaluation of LGC

We use a fluid model of (9) to investigate its stability and the accuracy of $\hat{l}$.

A. The Fluid Model

We model (9) as a continuous time fluid process:

$$\dot{x_i} = x_i(t) r_i \left(1 - x_i(t) - \hat{l}_i(t)\right) \tag{10}$$

$$\dot{\hat{l}}_i = c\left(t - \tau_i\right) - c\left(t - 2\tau_i\right) \tag{11}$$

$$c(t) = 1_{\{q(t)>0\}} \tag{12}$$

$$q(t) = -k + \sum_{i=1}^{S} x_i(t) \tag{13}$$

where $c(t)$ is the ECN-marking process, τ_i is the time that takes for sender i to receive the ECN signal from the bottleneck link, and k denotes the ratio of the bottleneck link capacity to K. Unless otherwise specified we use $k = 1$ in calculations.

B. Model Validation

We validate the fluid model using simulations of the LGC mechanism performed in OMNeT++. Two scenarios are considered: 5 flows and 20 flows. Fig. 3 shows the comparative results for the source send rates and bottleneck queue size. We had two scenarios with 5 and 20 flows. In both scenarios the bottleneck link capacity is 10 Gbps with a source RTT of 200 μs. We see that in both cases, the rate of sources and the queue length match the model.

C. Stability and Equilibrium

The stability of LGC cannot be directly investigated by (9) because there is a delay in getting ECN signals from the bottleneck link router, i.e. $\hat{l}(t) = c(t - \tau)$. As it is illustrated in Fig. 3(a) and Fig. 3(c), this causes some periodic behavior. However, assuming that $c(t - \tau) \approx c(t - 2\tau)$ and considering the system model (10)-(13), we are able to 1) directly prove the stability of (9) based on the discussion in II-B2, and 2) calculate the equilibrium of LCG. Setting the right-hand side (RHS) of (10), (11), and (13) to zero and trying to solve the equations yields

$$1 - x^* - \hat{l}^* = 0 \,, \tag{14}$$

$$c^* = \hat{l}^* \,, \tag{15}$$

$$S(x^* + x^* r(1 - x^* - \hat{l}^*)) = k \,. \tag{16}$$

Assuming $k = 1$, the above system has a solution at equilibrium of $x^* = \frac{1}{S}$ with $\hat{l}^* = \frac{S-1}{S}$. This also reveals that process $c(t)$ approximates $\frac{S-1}{S}$ at equilibrium, i.e. $\frac{S-1}{S}$ percent of packets are ECN-marked. Since the number of competing flows is usually small in datacenters [8], [17], we do not expect $\frac{S-1}{S}$ becomes very close to 1. In Section V-A, we will discuss how we can have $c(t - \tau) \approx c(t - 2\tau)$, i.e. how to get approximately the same ECN-marked percentage in consecutive rate update intervals.

D. Accuracy of $\hat{l}_i(t)$

We discuss in more detail why $\hat{l}_i(t)$ is a good approximation of $l_i(t)$ especially when $\sum_j x_j(t)$ is close to 1, i.e. the sum of source rates is close to the bottleneck link capacity. Formally speaking, we claim that

$$\lim_{t_{\max}\to\infty} \left(\frac{1}{t_{\max}} \int_0^{t_{\max}} c(t) dt \right) = \frac{S-1}{S}. \tag{17}$$

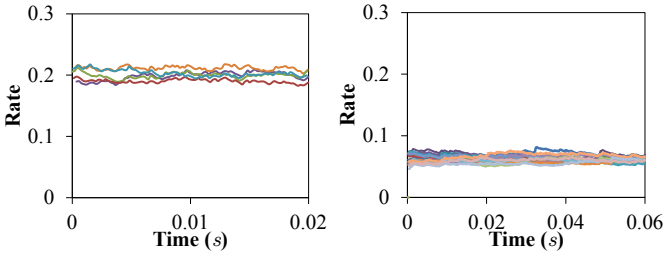
However, without requiring to solve (17), we can look at the RHS of (17) as how c behaves around the equilibrium. We solve (10) to obtain the relationship between system variables; this yields

$$\begin{aligned} x_1^* + \hat{l}_1^* &= 1 \,, \\ x_2^* + \hat{l}_2^* &= 1 \,, \\ &\vdots \\ x_S^* + \hat{l}_S^* &= 1 \,. \end{aligned} \tag{18}$$

Let us assume that sources experience the same proportion of ECN-marked packets (or very close to the same). We will discuss how this can be assured in Section V-A. This yields $\hat{l}_i^* = \hat{l}^* \; \forall i$. From (18) we obtain

$$x_1^* = x_2^* = ... = x_S^* = 1 - \hat{l}^* \,. \tag{19}$$

Since sources keep increasing their sending rate until the bottleneck link is 100% utilized, we have at equilibrium



(a) Rate of 5 flows (b) Rate of 20 flows

Fig. 4. Source send rates with exponentially-distributed packet pacing.

$\sum_{i=1}^{S} x_i^* = 1$ and since all sources transmit at the same rate, $x_i^* = \frac{1}{S}$. Thus (19) yields $\hat{l}^* = \frac{S-1}{S}$ which confirms that $l^* = \hat{l}^*$. This implies that the ECN marking process $c(t)$ also approximates $\frac{S-1}{S}$.

1) Validation Results: We validated the above argument using the simulation scenario of Section IV-B; the average value of $\hat{l}$ in case of 5 flows with three digit precision is 0.796 with a standard deviation of 0.124, and in case of 20 flows, it is 0.937 with a standard deviation of 0.113. We see that in both cases, the measured values by sources are very close to $\frac{S-1}{S}$, i.e. 0.8 in the first case, and 0.95 in the second one.

E. Different Bottleneck Link Capacity

Here we discuss what happens if the bottleneck link capacity is not equal to the maximum sending rate of the sources. Referring to (14)-(16), in general at equilibrium the normalized source send rate and load indicator are:

$$x^* = \begin{cases} \frac{k}{S} & k < S \\ 1 & k \geq S \end{cases} \quad \text{and} \quad \hat{l}^* = \begin{cases} \frac{S-k}{S} & k < S \\ 0 & k \geq S \end{cases}$$

If $k > S$, the sources combined maximum send rate is less than the bottleneck link's capacity, giving $x_i^* = 1$ and $\hat{l}^* = 0$. We ran a simulation experiment where there are 10 competing nodes connected to a 10 Gbps link (K=10 Gbps), and the bottleneck link capacity took the values $\{10, 20, 30, 40, 50, 60\}$ Gbps resulting in respective values of $k = \{1, 2, 3, 4, 5, 6\}$. We also added 40% of the link capacity Poisson traffic as the background traffic. In all six cases each nodes' normalized send rate was very close to $x_i^* = \frac{k}{S}$; confirming the above argument.

V. Further Improvements

A. Exponential Delay

As we discussed in Section IV-C, having $c(t-\tau) \approx c(t-2\tau)$ helps improve the stable behavior of LGC. Although a moving average with a large-enough window over ECN signals can reflect the same value, we are able to improve it further by pacing packets with an exponentially-distributed delay between them. The advantage of rate-based transmission with exponentially-distributed inter-packet delay is that, due to the PASTA property (Poisson Arrivals See Time Averages) [32], sources in the limit observe the same average congestion marking. When measured over the epoch n we have $\hat{l}_i(n) \approx \hat{l}_j(n) \approx \frac{S-1}{S}\ \forall i, j$.

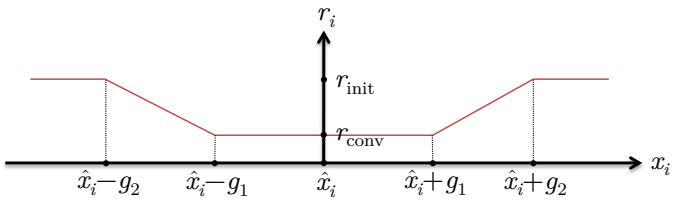


Fig. 5. Adapting r depending on the expected fair share and current rate.

Fig. 4 shows rate changes in LGC sources when packets have exponentially-distributed inter-packet times. Compared with the rates in Fig. 3(a) and 3(c), we see that rates are more stable with few large abrupt changes. In Section VI, we show how adding exponentially-distributed delays affects queue length.

B. Adapting the growth rate

The performance of LGC can be improved by tuning the growth rate parameter, r. Here, our goal is *to be more aggressive in changing the rate of a source that is getting a smaller or larger amount of resource than the fair share*.

Since $\hat{l}_i \approx \frac{S-1}{S}$, and $1 - \hat{l}_i \approx \frac{1}{S}$, we estimate the fair share as $\hat{x}_i = 1 - \hat{l}_i$. However, to avoid fluctuations we estimate this over a larger time interval:

$$\hat{x}_i[n] \triangleq 1 - \frac{1}{W} \sum_{w=1}^{W} c_i[n-w]$$

where W denotes the moving average window size.

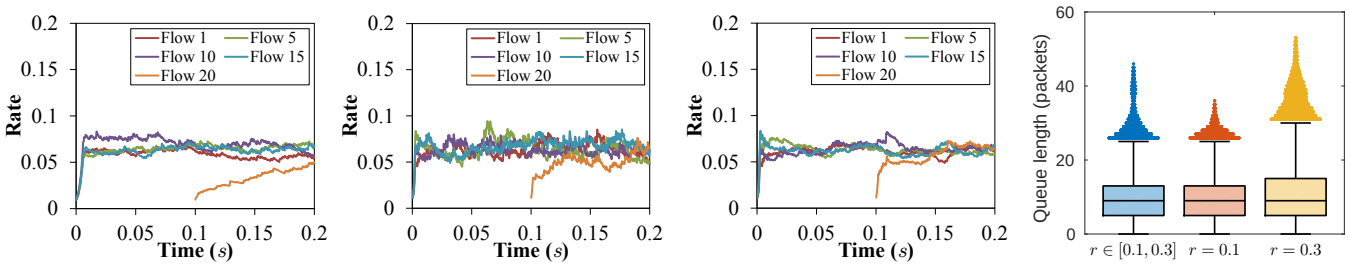
In Fig. 1, we see how r affects aggressiveness. We would like to dynamically change r to decrease the convergence time of sources that are far from their fair share and stably maintain source rates when sources are close to their fair share. To achieve this we define a range of values where $r \in [r_{\text{conv}}, r_{\text{init}}]$. r_{init} denotes the initial value of r. Sources start by setting $r = r_{\text{init}}$, and they use r_{conv} when $x_i \approx \hat{x}_i$. Fig. 5 shows how LGC chooses the value for r_i for source i. Depending on the value of $\hat{x}_i$, we use a simple linear function which is bounded by r_{conv} and r_{init}. g_1 and g_2 denote the rate range used for changing between r_{conv} and r_{init}.

Fig. 6 illustrates three different scenarios: $r = 0.1$, $r = 0.3$, and adaptive r. $g_1 = 0.015$ and $g_2 = 0.45$. We see that using an adaptive r, as shown in Fig. 6(c), LGC takes advantage of the stability seen in Fig. 6(a) and faster convergence of Fig. 6(b) without any significant effects on the queue length distributions illustrated in Fig. 6(d) using box-and-whisker[1] plots.

VI. Comparative Performance Results

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++ [19]. Our

[1]Boxes span the middle 50% of data, with whiskers extending up to 1.5 times the interquartile range. Outliers are offset so the density of points is clear.



(a) Rate of 5 sample flows out of 20 where $r = 0.1$.
(b) Rate of 5 sample flows out of 20 where $r = 0.3$.
(c) Rate of 5 sample flows out of 20 where $r \in [0.1, 0.3]$
(d) Queue length distribution for LGC.

Fig. 6. The effect of the growth parameter, r, on source aggressiveness. 19 sources start sending at $t = 0$ s, with the 20^{th} source joining at $t = 0.1$ s.

OMNeT++ DCTCP implementation was validated against its reference *NS*-2 implementation.

Results in this section are collected from a series of 10 simulation runs, each with unique random seeds. Sources randomly start transmission in an interval shorter than one RTT. Unless otherwise mentioned, we use the following parameters for LGC: $g_1 = 0.015$, $g_2 = 0.045$, $r_{\text{init}} = 0.3$, $r_{\text{conv}} = 0.1$, $w_l = 40$, and a rate update interval of 200 μs. We use the recommended configurations of DCTCP in [2] for optimum operation.

A. Small-Scale Evaluation

We simulated an incast traffic pattern for evaluation. Incast is an important scenario to evaluate congestion control in datacenters [17]. This traffic pattern occurs when client sends a request to several servers in the same rack connected to the same switch, and the servers all respond at almost the same time exceeding the capacity of the link to the client. In our scenario, the link capacity between the rack devices and the Top Of Rack (TOR) switch is 10Gbps.

1) Queue Length: To evaluate the effect the different mechanisms have on the bottleneck queue length distribution we test 5, 20, and then 40 sources (clients) sending data to the same destination (server). Fig. 7 shows a box-and-whisker plot[2] for DCTCP, LGC without exponentially distributed packet pacing, and LGC. We observe that LGC reduces the queue length compared to DCTCP for a comparable average throughput (DCTCP 9.64, LGC-no exp 9.52, and LGC 9.50 Gbps). The exponentially distributed packet pacing further reduces the queue length, though the benefit diminishes as the number of flows increase because flows get fewer acks per rate update intervals resulting in their load estimate $\hat{l}[n]$ becoming coarser. If LGC is to be used in scenarios where more than 20 competing flows is common, we suggest increasing the rate update interval to improve the precision of $\hat{l}[n]$. In the "LGC - no exp. delay" case the sources become synchronized when there are large numbers of competing flows, resulting in load estimates of $\hat{l}[n] = 1$ for most rate update intervals, and $\hat{l}[n] = 0$ for a small number of intervals. The source

[2]Boxes span the middle 50% of data, with whiskers extending up to 1.5 times the interquartile range. Outliers are offset so the density of points is clear.

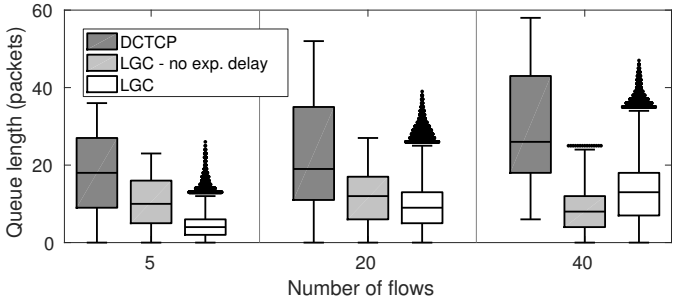


Fig. 7. Queue length distribution of the three methods.

synchronization, LG response to $\hat{l}$, and more deterministic packet arrival pattern result in lower queuing delays.

2) Fairness: We compare the short-term flow fairness of DCTCP and LGC using Jain's fairness index [12]. 5 sources send data to the same destination, with Jain's fairness index calculated over 1 ms intervals. Both LGC and DCTCP are tested in two scenarios: (i) all sources have homogeneous RTTs of 200 μs, (ii) the 5 sources have RTTs of 140, 170, 200, 230, and 260 μs respectively. Fig. 8 illustrates four simulation cases plotting the average of the 10 different runs. Error bars spanning the range of calculated fairness are plotted every 10 ms to indicate the variation in simulation runs.

First considering the case of homogeneous RTTs, we observe that LGC's fairness begins at close to 1, and remains very close to 1 throughout the simulation. DCTCP sources start using the same congestion window, which makes it fair at first. However, during first RTTs, sources do not have an accurate approximation of congestion (parameter α in [2]), and their start time is slightly different, which leads to a drop in fairness. As sources receive more acks, fairness improves, but it fluctuates in the range of 0.97 to 1. The fairness drop does not happen in LGC during first RTTs because sources pace packets with exponentially-distributed delays, which compensates for the initial inaccurate estimate and slight differences in start times.

In the heterogeneous RTT case both LGC and DCTCP sources start using the same congestion window size, but with the RTT difference this yields a fairness of around 0.96 at first. The LGC sources converge to their fair share (≈ 1.0) in just a few milliseconds, however, the DCTCP sources–after a drop in fairness due to not having an accurate approximation of

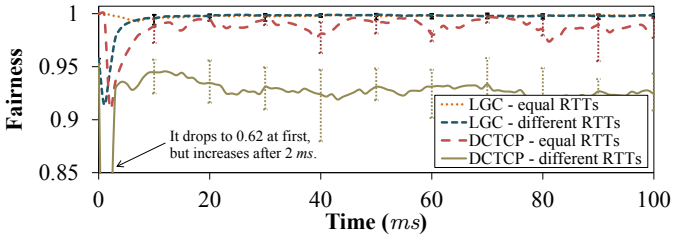


Fig. 8. Short-term fairness of 5 competing flows using LGC and DCTCP under equal and different RTTs using Jain's fairness index. Error bars span the range of values at 10 ms intervals.

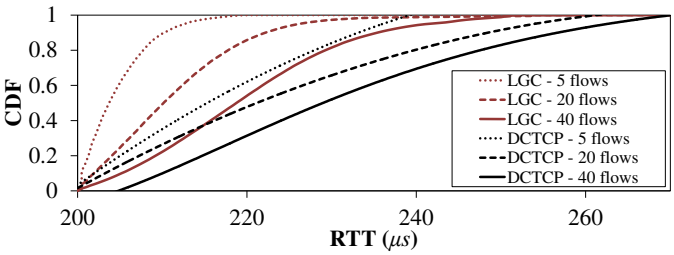


Fig. 9. Cumulative distribution function of the RTT distribution in an Incast scenario.

congestion–attain only a fairness of about 0.93. This illustrates RTT independence of LGC's mechanism.

3) Round-Trip Time (RTT): One of the direct consequences of reducing the queue length is reducing the RTT. In the next scenario, we calculate the Cumulative Distribution Function (CDF) of the RTT distribution for both LGC and DCTCP with tests of 5, 20, and 40 competing flows using a base RTT of 200 μs. In all cases both LGC and DCTCP were configured optimally. Fig. 9 shows the results. LGC shows a significant reduction in RTT over DCTP in all three tests.

B. Large-Scale Evaluation

Here we investigate the comparative performance of LGC with respect to DCTCP under two large scale simulation scenarios:

(i) a classic Clos topology (8-ary fat-tree, 64 nodes in total) [22], [1], and
(ii) a similar leaf-spine topology to the one in [4], [9] (144 nodes, 9 leaf switches, 4 spine switches).

In the leaf-spine topology, all the links between hosts and 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.

We use a *longest path uniform random* traffic pattern [17] in which a source sends data to a randomly-chosen destination with the longest path in the network. However, the set of destinations is smaller than the set of sources to ensure that congestion happens. Fig. 10 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 for comparable average throughput (Clos: DCTCP 9.56 and LGC 9.45 Gbps, and leaf-spine: DCTCP 9.51 and LGC 9.41 Gbps). The average number of competing flows in these scenarios was around 8, yielding

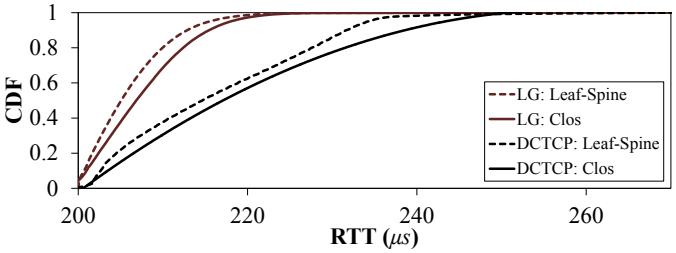


Fig. 10. Cumulative distribution function of the RTT distribution under Clos and leaf-spine topologies.

a comparatively shorter RTT for both LGC and DCTCP than was achieved with the incast results shown in Fig. 9.

VII. Related Work

Datacenters are a challenging environment for efficient timely transmission of data with features including: multiple paths, small propagation delays and mixtures of long and short flows (see Zhang et al. [33] for a survey of transport control in datacenters). A key objective is to minimize flow latency, particularly flow completion times. This is a multifaceted issue [6], [21], of which congestion control is a significant component. Many proposed solutions require changes to multiple components within the datacenter, affecting the ease with which the different schemes can be deployed [21], [9].

Our work focuses on easily deployable end-system based mechanisms. The closest datacenter specific work in this area to ours is Data Center TCP (DCTCP) [2], [5]. Apart from the end point transport mechanism, it requires only a low queue threshold based ECN packet marking from the bottleneck queue. A number of enhancements to DCTCP have been proposed in the literature that improve particular aspects [33]. HULL [3] adds packet pacing and virtual (phantom) queues at switches to reduce queuing and allow for earlier congestion notification at the expense of slightly lower throughput. D2TCP [26] enhances DCTCP to include the deadline aware advantages of D3 [31]. In general, these enhancements improve the mechanism but make it more difficult to deploy [9]. Our mechanism is able to improve many aspects of DCTCP while still remaining easily deployable.

Logistic growth as basis for congestion control was first proposed in [30] and later in [10]. It has been shown to be stable and reduce queue latency. We build on these ideas adapting the theory to the datacenter environment.

VIII. Conclusions and Further Work

In this paper, we have developed and evaluated a new congestion control algorithm for datacenters called Logistic Growth Control (LGC). The algorithm is inspired by logistic population growth and uses the logistic growth function to control packet transmissions over a congested path. We first analyzed LGC analytically using a fluid model to investigate its stability and accuracy. Based on this we were able to improve the algorithm and have implemented it in the INET framework of OMNeT++ [19]. Using this simulation framework, we have run extensive evaluations of LGC against another important datacenter transport protocol, DataCenter TCP

(DCTCP) [2]. We simulated both a small-scale network with the important datacenter incast traffic pattern, and large-scale Clos and leaf-spine topologies. Our evaluations consistently show that communication latency in a datacenter is improved greatly by LGC, and also that LGC achieves much better fairness between flows than DCTCP.

LGC is relatively easy to model and understand, and has good convergence, RTT-independent fairness and stability properties. This makes it a promising candidate for more complex control scenarios where multiple congestion controls may be nested (e.g. to control flow aggregates between hypervisors in a Grid rather than changing the OS in the VMs) or chained together (as e.g. with middleboxes that act as TCP splitters). Recursive architectures such as RINA [7] and RNA [25] (see [24] for more details) generalize this approach and challenge traditional concepts of congestion control. We plan to further apply LGC in the datacenter environment and extend our work to more general recursive architectures.

ACKNOWLEDGMENT

This work has received funding from the European Union's FP7 research and innovation programme under grant agreement No. 619305 (PRISTINE). The views expressed are solely those of the author(s).

REFERENCES

[1] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 4, pp. 63–74, 2008.

[2] 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.

[3] M. Alizadeh, A. Kabbani, T. Edsall, B. Prabhakar, A. Vahdat, and M. Yasuda, "Less is more: Trading a little bandwidth for ultra-low latency in the data center," in *Presented as part of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)*. San Jose, CA: USENIX, 2012, pp. 253–266.

[4] M. Alizadeh, S. Yang, M. Sharif, S. Katti, N. McKeown, B. Prabhakar, and S. Shenker, "pfabric: Minimal near-optimal datacenter transport," *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 4, pp. 435–446, 2013.

[5] S. Bensley, L. Eggert, and D. Thaler, "Datacenter TCP (DCTCP): TCP Congestion Control for Datacenters," Working Draft, IETF, Internet-Draft draft-bensley-tcpm-dctcp, Feb 2014.

[6] B. Briscoe, A. Brunstrom, A. Petlund, D. Hayes, D. Ros, I.-J. Tsang, S. Gjessing, G. Fairhurst, C. Griwodz, and M. Welzl, "Reducing internet latency: A survey of techniques and their merits," *Communications Surveys Tutorials, IEEE*, vol. PP, no. 99, pp. 1–1, 2014.

[7] J. Day, *Patterns in Network Architecture: A Return to Fundamentals*. Prentice Hall, 2007.

[8] P. X. Gao, A. Narayan, G. Kumar, R. Agarwal, S. Ratnasamy, and S. Shenker, "phost: Distributed near-optimal datacenter transport over commodity network fabric," in *Proceedings of the CoNEXT*, 2015.

[9] M. P. Grosvenor, M. Schwarzkopf, I. Gog, R. N. M. Watson, A. W. Moore, S. Hand, and J. Crowcroft, "Queues don't matter when you can jump them!" in *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*. Oakland, CA: USENIX Association, May 2015, pp. 1–14.

[10] G. Hasegawa and M. Murata, "TCP symbiosis: Congestion control mechanisms of tcp based on lotka-volterra competition model," in *Proceedings from the 2006 Workshop on Interdisciplinary Systems Approach in Performance Evaluation and Design of Computer & Communications Sytems*, ser. Interperf '06. New York, NY, USA: ACM, 2006.

[11] C.-Y. Hong, M. Caesar, and P. B. Godfrey, "Finishing flows quickly with preemptive scheduling," *SIGCOMM Comput. Commun. Rev.*, vol. 42, no. 4, pp. 127–138, Aug 2012.

[12] R. Jain, D. Chiu, and W. Hawe, "A Quantitative Measure of Fairness and Discrimination for Resource Allocation in Shared Computer Systems," DEC Research, Technical report TR-301, Sep 1984.

[13] L. Jose, L. Yan, M. Alizadeh, G. Varghese, N. McKeown, and S. Katti, "High speed networks need proactive congestion control," in *Proceedings of the 14th ACM Workshop on Hot Topics in Networks*, ser. HotNets-XIV. New York, NY, USA: ACM, 2015, pp. 14:1–14:7.

[14] D. Luenberger, *Introduction to dynamic systems: theory, models, and applications*. Wiley, 1979.

[15] R. MacArthur, "Species packing and competitive equilibrium for many species," *Theoretical population biology*, vol. 1, no. 1, pp. 1–11, 1970.

[16] R. M. May, *Stability and complexity in model ecosystems*. Princeton University Press, 2001, vol. 6.

[17] R. Mittal, V. T. Lam, N. Dukkipati, E. Blem, H. Wassel, M. Ghobadi, A. Vahdat, Y. Wang, D. Wetherall, and D. Zats, "Timely: Rtt-based congestion control for the datacenter," in *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, ser. SIGCOMM '15. New York, NY, USA: ACM, 2015, pp. 537–550.

[18] J. D. Murray, *Mathematical Biology I: An Introduction*. Springer, New York, NY, USA, 2002, ISBN: 978-0-387-95223-9.

[19] OMNeT++, "The inet framework," https://omnetpp.org/, 2016.

[20] J. Perry, A. Ousterhout, H. Balakrishnan, D. Shah, and H. Fugal, "Fastpass: A centralized zero-queue datacenter network," in *Proceedings of the 2014 ACM conference on SIGCOMM*. ACM, 2014, pp. 307–318.

[21] R. Rojas-Cessa, Y. Kaymak, and Z. Dong, "Schemes for fast transmission of flows in data center networks," *IEEE Communications Surveys Tutorials*, vol. 17, no. 3, pp. 1391–1422, 2015.

[22] A. Singh, J. Ong, A. Agarwal, G. Anderson, A. Armistead, R. Bannon, S. Boving, G. Desai, B. Felderman, P. Germano *et al.*, "Jupiter rising: A decade of clos topologies and centralized control in google's datacenter network," in *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*. ACM, 2015, pp. 183–197.

[23] A. Sivaraman, M. Budiu, A. Cheung, C. Kim, S. Licking, G. Varghese+, H. Balakrishnan, M. Alizadeh, and N. McKeown, "Packet transactions: High-level programming for line-rate switches," in *Proceedings of the 2016 ACM conference on SIGCOMM*. ACM, 2016.

[24] P. Teymoori, M. Welzl, G. Stein, E. Grasa, R. Riggio, K. Rausch, and D. Siracuss, "Congestion control in the Recursive Internetwork Architecture (RINA)," in *IEEE International Conference on Communications (ICC), Next Generation Networking and Internet Symposium*, May 2016.

[25] J. D. Touch and V. K. Pingali, "The RNA metaprotocol," in *Computer Communications and Networks, 2008. ICCCN'08. Proceedings of 17th International Conference on*. IEEE, 2008, pp. 1–6.

[26] B. Vamanan, J. Hasan, and T. Vijaykumar, "Deadline-aware datacenter TCP (D2TCP)," in *Proceedings of the ACM SIGCOMM 2012 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, ser. SIGCOMM '12. New York, NY, USA: ACM, 2012, pp. 115–126.

[27] J. Vano, J. Wildenberg, M. Anderson, J. Noel, and J. Sprott, "Chaos in low-dimensional lotka–volterra models of competition," *Nonlinearity*, vol. 19, no. 10, p. 2391, 2006.

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

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

[30] ——, "Traceable congestion control," in *Proceedings of the 3rd International Conference on Quality of Future Internet Services and Internet Charging and QoS Technologies 2Nd International Conference on From QoS Provisioning to QoS Charging*, ser. QofIS'02/ICQT'02. Berlin, Heidelberg: Springer-Verlag, 2002, pp. 273–282.

[31] C. Wilson, H. Ballani, T. Karagiannis, and A. Rowtron, "Better never than late: Meeting deadlines in datacenter networks," in *Proceedings of the ACM SIGCOMM 2011 Conference*, ser. SIGCOMM '11. New York, NY, USA: ACM, 2011, pp. 50–61.

[32] R. W. Wolff, "Poisson Arrivals See Time Averages," *Operations Research*, vol. 30, no. 2, pp. 223–231, 1982.

[33] J. Zhang, F. Ren, and C. Lin, "Survey on transport control in data center networks," *IEEE Network*, vol. 27, no. 4, pp. 22–26, Jul 2013.

Feedback in Recursive Congestion Control

David A. Hayes, Peyman Teymoori, and Michael Welzl

University of Oslo, Norway {davihay,peymant,michawe}@ifi.uio.no

Abstract. In recursive network architectures such as RINA or RNA, it is natural for multiple layers to carry out congestion control. These layers 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. In this paper, we report on a first analysis of feedback methods in recursive networks that we carried out using a fluid model with a packet queue approximation. We find that the strict pushback feedback based on queue size can have stability issues, but robust control can be achieved when each congestion controller receives feedback from all sources of congestion within and below its layer.

1 Introduction

Since the beginning of research on congestion control, the use of many different forms of feedback has been investigated. It can be implicit (e.g., using packet loss or growing delay as an indication of congestion) or explicit (e.g., using an Explicit Congestion Notification (ECN) bit or multiple bits). From the location where congestion appears, it can propagate forward towards the receiver (from where it is in some way reflected to the sender) or directly backward towards the sender. There have been proposals for hop-by-hop congestion control and *backpressure* mechanisms – some of them are discussed in [6], a survey that was published as early as 1980. We mention ATM as a technology that is notable for supporting many of the explicit feedback methods mentioned above, via Resource Management (RM) cells that allowed explicit rate feedback signalling as well as ECN marking, both in the forward and backward direction.

Despite the multitude of feedback methods covered by the literature on congestion control over the last decades, to the best of our knowledge, how to best provide congestion control feedback in a *recursive* network architecture has never been studied in detail before. Recursion has only recently been proposed as an architectural approach to networking that should remove some problems that have been identified with traditional network layering; notable examples are the Recursive InterNetwork Architecture (RINA) [3] and the Recursive Network Architecture (RNA) [18]. One of the concerns about such architectures is the stability of the whole recursive network where there are several congestion controllers operating at different layers over various ranges. RNA proposes a simple approach to manage this problem for some cases [17]. However, a thorough evaluation is missing in both of RNA and RINA.

We have begun to investigate congestion control for recursive networks with [16]. Using some simple scenarios, this work shows that concatenated/stacked layers ("DIFs")

of RINA can improve performance. However, only one type of feedback was considered in these scenarios (every DIF ran a simple TCP-like congestion control with ECN).

In this work, we take the next step, by carrying out a first investigation of the possibilities for feedback in recursive networks. We do this by means of a fluid-type model using Simulink [14], using a sender behavior that is suitable for the experimentation and representative of a modern congestion control mechanism, which makes our findings broadly applicable. Our contribution consists of both new, and perhaps unexpected, findings about congestion control feedback as well as the model itself, which seeks to strike a balance between being simple enough to manipulate and complete enough to allow for meaningful investigations of control loop interactions in recursive networks.

2 Related Work

Before and since the aforementioned survey [6] was published, a huge number of congestion control mechanisms have been devised, in many contexts, using all kinds of feedback along the categories that we have explained. The diversity is such that we point at two textbooks ([8,20]) instead of even trying to provide a reasonably comprehensive set of examples here. In the following, we briefly introduce the two recursive network architectures RINA and RNA instead.

RINA was firstly presented in [3] as a way of solving the problems of the Internet architecture (e.g. complexity, scalability, security, mobility, quality of service or management to name a few, see [4,7,9,15]). It is known that the traditional "transport/network/data link/physical" network stack does not match all the usage scenarios (e.g. MPLS, tunneling) and may implement similar functions (e.g. flow control) redundantly at different layers. However, RINA comprises a number of functionally-equivalent layers, which might be customized depending on the environment.

In RINA, every layer, called a "Distributed InterProcess Communication (IPC) Facility" (DIF), has the same structure and set of "mechanisms"; a DIF provides IPC services to its processes over a certain scope. The mechanisms of a DIF can be programmed via "policies"; this optimizes the DIF for its operating environment (e.g. over wireless links). As two examples of mechanisms, every DIF has a transport-like protocol called "Error and Flow Control Protocol" (EFCP), and a "Relaying and Multiplexing Task" (RMT) module which represents routing and multiplexing functions. Both EFCP and RMT can be customized with appropriate policies. For example, in case of a wireless link, EFCP can be empowered by a congestion controller policy for wireless environments, and RMT can also use routing policies with better performance under that condition. Therefore, each DIF in the network can detect and manage the congestion on its own resources, pushing back to/signaling higher layer DIFs when resources are overloaded.

RNA [18] aims at addressing the challenges the current Internet protocol architecture is dealing with regarding layering; it presents a single "reusable" protocol (it is also called "metaprotocol") for all the layers of the protocol stack. This protocol presents a number of generic services to the other horizontal or vertical layers/protocols. In other words, RNA indicates that a single protocol can be tuned at different layers to dynamically support new services and unify all the conventional and overlay layers [17]. This

metaprotocol is, of course, capable of adopting a variety of communication functions including congestion control.

3 A model based investigation of feedback for recursive congestion control

Recursive architectures allow congestion control mechanisms to operate on flow aggregates at lower layers as well as end-to-end at the uppermost layer. This potentially allows for more versatile control, however, it also creates complex control interactions between the various loops. We investigate these interactions using a simple, but not trivial, model. Too simple a model will lack the complex control loop interactions that we wish to investigate, too complex a model will not be solvable. We adopt the RINA terminology in our description and discussion of our model, but the principles are applicable to other recursive models sush as RNA. Figure 1 shows the RINA topology we model, representing how hosts might be connected via Internet Service Providers (ISPs) [13]. All DIFs in the model implement congestion control except the lowest peer-to-peer (P2P) DIFs.

In contrast with two well-known non-recursive feedback mechanisms that we use as a baseline — (i) End-to-end forward Explicit Congestion Notification[1] (ECN) like feedback, and (ii) backward ECN (BECN)[1] like feedback — we investigate the comparative stability of three different possible recursive congestion feedback mechanisms: (i) Recursive forward feedback, (ii) Recursive backward feedback, and (iii) Recursive pure pushback feedback. We explain these mechanisms with an example in Sec. 3.1.

3.1 Experimental setup

In our experiments the router nodes (x-GW and R-n in Fig. 1) are the bottlenecks in the end to end path, each with the capacity to carry 100 rate units. Packet queue models are limited to a maximum of 100 packets in these tests. Test traffic flows from the sender to the receiver with a propagation delay between each node in the network of 10 ms. Delay between DIFs at the same node is modelled as 0 ms.

Apart from the traffic flowing end-to-end, cross traffic traverses each of the router nodes. The amount of cross traffic changes every second, drawn from a uniform random distribution between 0 and 10% of the capacity (average 5% of capacity).

To investigate the effect sudden disturbances have on the stability of the system, we introduce a cross traffic of 40% of the bottleneck capacity, first at location ① ($t = [50, 100]$ s) and then later at ② ($t = [150, 200]$ s). ① was chosen to be relatively close to the sender and ② close to the receiver to highlight the difference between the backward and forward feedback mechanisms.

Figure 1 shows the feedback message paths for the mechanisms being investigated using disturbance ① as an example. In our experiments we feedback a congestion signal based on queue size. This contains more information than the binary ECN signal

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

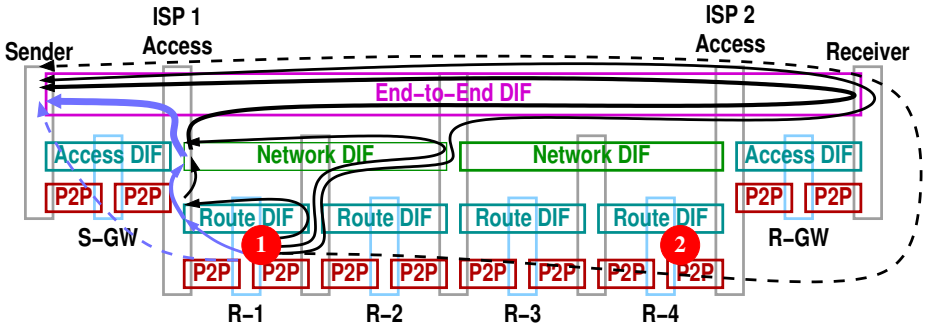


Fig. 1. Feedback mechanisms being investigated: Non-recursive congestion control with (i) Forward ECN like feedback (dark dashed arrow), (ii) Congestion control with Backward ECN like feedback (light blue dashed arrow); Recursive congestion control with: (iii) Backward feedback (light blue arrows), (iv) Forward feedback (dark arrows). (v) Pure pushback 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.

used in the Internet and the pure pushback signal currently used in RINA, however, the aim in this initial work is to compare each feedback mechanism at its best.

Non-recursive feedback. These mechanisms are used as a base to compare the recursive RINA feedback mechanisms with. In these experiments the end points are connected by routers with no intermediate layers. Only end-to-end congestion control operates between the sender and the receiver. Figure 1 shows the Internet like non-recursive feedback paths as dashed lines with arrows. The b dashed line depicts the path of standard forward ECN signals, while the lighter dashed lines with arrows depicts the path of the backward ECN signals. The model uses queue size as the congestion signal, a richer signal than generally used in the Internet, but better for comparison.

Recursive feedback. The solid lines depict two possible recursive RINA congestion notification paths. The black lines with arrows depict forward end reflection type notifications. These are similar to forward ECN, except that they operate on each affected DIF in the layered architecture. The lighter lined arrows depict back propagating notifications, similar in some sense to backward ECN, but operating at each layer. As the congestion signals move up the layers they carry congestion information from all of the layers below them (indicated in the diagram by thicker lines). Both the recursive forward and backward mechanisms use pushback congestion notifications in addition to their explicit notifications.

This pure pushback feedback mechanism allows a very neat demarcation between the layers, however, it prevents overarching feedback loops common in system control applications [11,12]. Pure pushback congestion notification relies on the lowest DIF closest to the congestion reacting and reducing its send rate. This in turn causes the DIF above to react to queue growth and reduce its send rate. The process repeats, pushing the congestion notification back all the way to the sender (if necessary). This process may involve signalling within a DIF if congestion is detected at some place other than

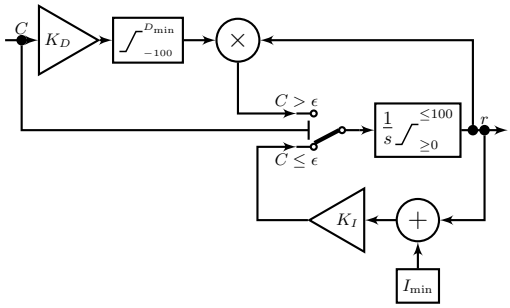


Fig. 2. Sender model. Input: congestion signal C. Output: send rate r.

the start (in terms of traffic direction) of the DIF. In this case a backward signalling mechanism is used to relay the signal. Currently only the pure pushback feedback like mechanism is defined in RINA [16].

3.2 Description of the model

We model Fig. 1 as a fluid type model using Simulink [14]. This allows us to observe the stability dynamics of the system at all levels, especially the uppermost level which is presented here. We use a common generalised congestion control mechanism for the DIFs. We proceed in our description by looking at the key building blocks in the model: sender, DIFs, and bottlenecks. Receivers are modelled only in how they relay signals to the congestion controls on the transmission side.

Sender. Figure 2 shows the model used for the sender based on the following differential equation:

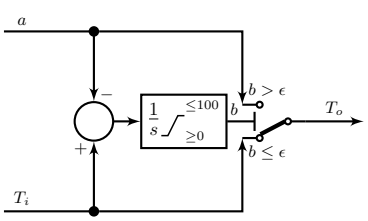
$$\dot{r} = \begin{cases} rCK_D & C > 0 \\ (r + I_{\min})\, K_I & C = 0 \end{cases} \tag{1}$$

where r is the send rate, C the collective measure of congestion ($C = \sum c$, for all c relevant to the particular feedback scenario), K_I is the increase gain, K_D is the negative decrease gain, and $I_{\min}$ is the minimum increase factor. The sender is greedy, modelling an aggregate of senders, increasing their send rate until they are notified of congestion. The sender's send rate increase is in proportion to its current send rate. This is not how the probably best-known Internet based congestion control mechanism — Additive-Increase Multiplicative-Decrease (AIMD) in Standard TCP — increases its congestion window; it more closely resembles the increase mechanism in Scalable TCP [10]. A scalable mechanism makes model more versatile and easier to configure and experiment with, and is also a shortcoming modern TCP enhancements target [2,20].

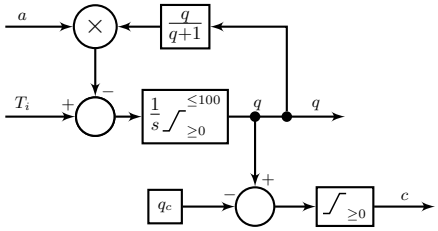
The response to congestion is proportional to the current send rate and the level of congestion.

When there is no congestion ($C \leq \epsilon$)[2], the rate increases in proportion to the current rate (r) at gain K_I. When there is congestion ($C > \epsilon$) the rate decreases in proportion to the current rate (r) and the congestion signal ($C \times K_D$).

[2] $\epsilon = 0.01$ is used instead of 0 to aid the numerical solvers used in Simulink



(a) Traffic out Queue (ToutQ). Inputs: proved available rate a and input traffic rate T_i. Output: output traffic rate T_o.



(b) Packet Queue (PktQ). Inputs: link probed available rate a and input traffic rate T_i. Outputs: packet queue size q and local congestion signal c.

Fig. 3. DIF queueing models

Congestion controled DIFs. The behavior of lower congestion controled DIFs are only different to the sender in that they do not greedily probe for available capacity. Their job is to track the capacity that is available relative to the traffic they must relay. In this way DIFs buffer temporary excesses in traffic, and probe for more when these excesses are sustained. They also reduce their send rate limit when the traffic demand does not require such a high limit. This congestion control behavior has parallels with recent enhancements of TCP to support application limited traffic [5].

The dynamics of lower DIFs are governed by two queueing models: an instantaneous fluid send queue (ToutQ, see Fig. 3a) that models how traffic is sent through the node, and a fluid approximation of packet queue dynamics (see Fig. 3b) to model congestion.

Traffic out queue. ToutQ (see Fig. 3a) integrates (minimum limited to 0) the difference between the traffic arrival rate (T_i) and the probed available link rate (a). If the queue size is above ϵ, traffic is sent at a rate equal to a. Otherwise traffic is sent at the traffic arrival rate (i.e. $T_o = T_i$).

Packet queue. The packet queue model (Fig. 3b) is based on an approximation of packet queue dynamics proposed in [1]:

$$\dot{q} = \lambda - \frac{\mu_0 q}{q+1} \tag{2}$$

where q is the queue length in packets, λ is the traffic arrival rate (T_i in our model), and μ_0 is the maximum service rate (DIF probed available rate, a, in our model). The congestion threshold is set to equilibrium load of about 90%, which gives rise to an equilibrium packet queue size of 9 packets. The queue size above this threshold is fed back as the congestion signal using the various feedback mechanisms being tested.

The queueing model in these experiments is based on an M/M/1 queue, which is sufficient for the dynamics we wish to analyse. However the same pointwise stationary fluid approximation technique can be applied to more complex arrival and service models as shown in [19], and may be useful in further studies of more complex traffic scenarios.

The congestion signal, c, is the amount of queueing above the 90% load queue level (q_c).

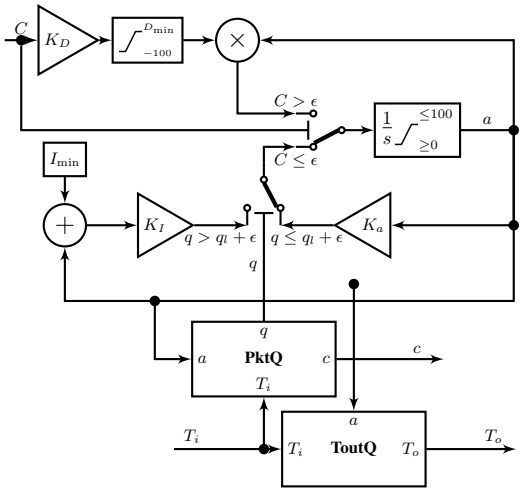


Fig. 4. Congestion control model for DIFs. Inputs: input traffic rate T_i and congestion signal C. Outputs: output traffic rate T_o and local congestion signal c.

Complete DIF congestion control model. Figure 4 shows the model used for the DIFs based on the following differential equation:

$$\dot{a} = \begin{cases} aCK_D & C > 0 \\ aK_a & C = 0 \wedge q < q_l \\ (a + I_{\min})\, K_I & C = 0 \wedge q \geq q_l \end{cases} \tag{3}$$

were a is the DIF probed available rate, $C = \sum c$ for all c relevant to the particular feedback scenario, K_I is the increase gain, K_D is the negative decrease gain, $I_{\min}$ is the minimum increase factor, q is the packet queue size, and q_l is a lower packet queue limit. Similar to the sender model (see Fig. 2), when there is congestion ($C > \epsilon$, where $C = \sum c$ for all c relevant to the particular feedback scenario) the send rate limit decreases in proportion to the current rate and the congestion ($C \times K_D$). When there is no congestion and the DIF queue (q) is large, the send rate limit increases in proportion to the current rate limit at gain K_I. When there is no congestion and the DIF queue is small, the send rate limit decreases in proportion to the current probed available link rate (a), at gain K_a. ToutQ and PktQ are described earlier and shown in Fig. 3.

Also, when this queue is above zero, and there is no congestion, the DIF probed available rate, a, is increased in proportion to its current send rate. When this queue is zero, the maximum DIF send rate slowly decreases since the traffic is being limited by some other element in the network. The DIF send rate decreases in proportion to congestion and its current send rate. The aggressiveness of the different DIF send rate adjustments are tuned to be proportional to the number of hops they control.

Bottleneck router and cross traffic Bottleneck router nodes do not have any congestion control function, but do produce a congestion signal c. The capacity of the link is $Z = 100$ in these experiments.

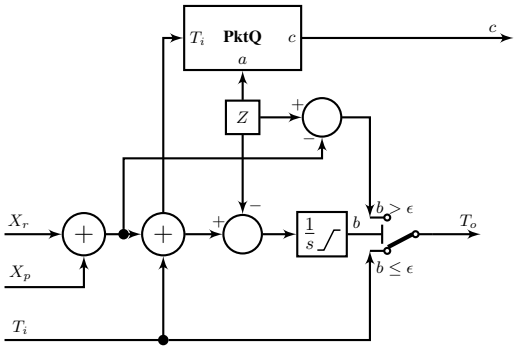


Fig. 5. Bottleneck nodes (x-GW and R-n in Fig. 1). Inputs: random 5% cross traffic X_r, pulse 40% cross traffic X_p (only at *R-1* and *R-4*), input traffic rate T_i. Outputs: output traffic rate T_o and local congestion signal c.

The Bottleneck router nodes have cross traffic X_r and X_p traversing them, restricting the capacity available for T_i to $Z - (X_r + X_p)$. X_r supplies small (average 5% of Z) random perturbations to the available link capacity the constantly changing nature of a packet switched network. X_p is used to assess stability in the presence of a large sudden perturbation.

T_o is fed to the next node, with a traffic out queue similar to Fig. 3a used to decide whether T_i can be sent out on the link, or the available capacity or$Z - (X_r + X_p)$. PktQ is used to measure congestion, see Fig. 3b.

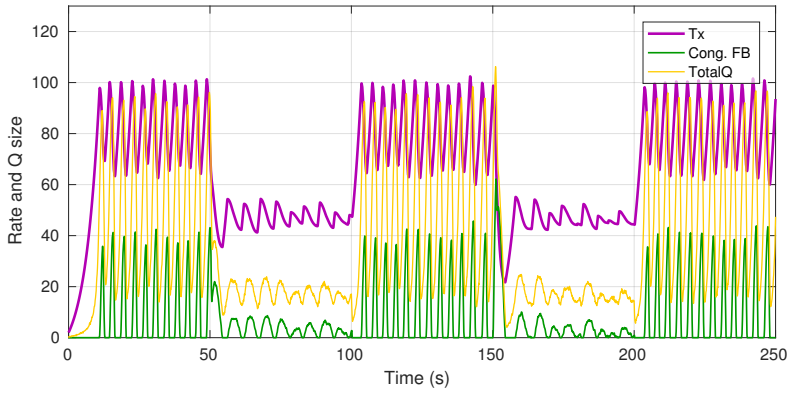
3.3 Notes on solving the model

To aid solving the ordinary differential equations (ODEs) that result from the model, signals are limited in the rate they can change to a gradient of plus and minus twice the Capacity (i.e. 200 in these tests).

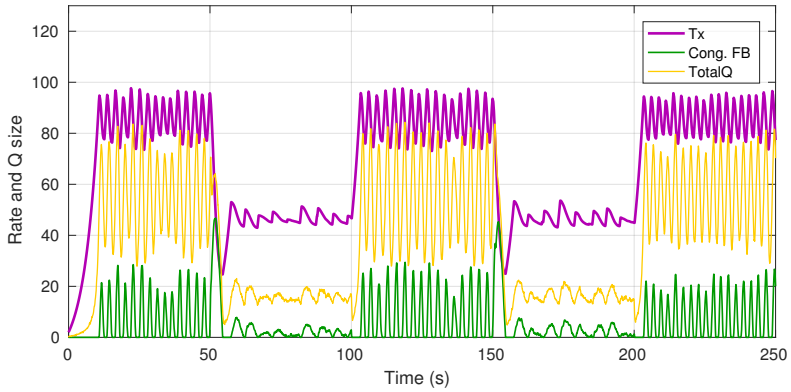
4 Simulation Results

To understand system dynamics we look first at how the five different feedback scenarios affect the following key parameters: 1. Sender's transmission rate (Tx), 2. Congestion signal (queueing in excess of 90% load queueing) fed back to the Sender (Cong. FB), 3. The cumulative total of all the queues in the system (TotalQ). We then examine the effect that the gain parameters, K_I and K_D have on the overall stability of the system by looking at the average sender's transmission rate (mean(Tx)) and standard deviation of the sender rate (std(Tx)) for the recursive pushback and recursive backward feedback scenarios.

A large disturbance X_p is introduced at *R-1* and *R-4* to help assess the stability of the system (see Fig. 1). At $t = [50, 100]$ for *R-1* and $t = [150, 200]$ for *R-4* the available capacity on the bottleneck is suddenly restricted by 40% of Z.



(a) Non-recursive forward feedback



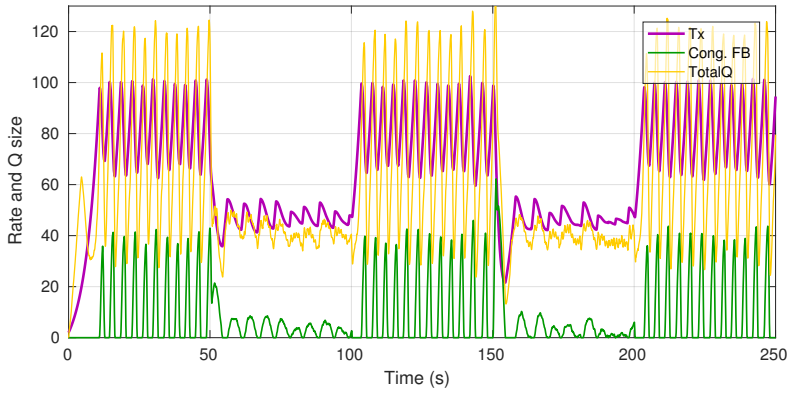
(b) Non-recursive backward feedback

Fig. 6. Non-recursive rich signal feedback. (Tx is the sender transmission rate, Cong. FB is the congestion signal, and TotalQ is the cumulative total of all queues in the system.)

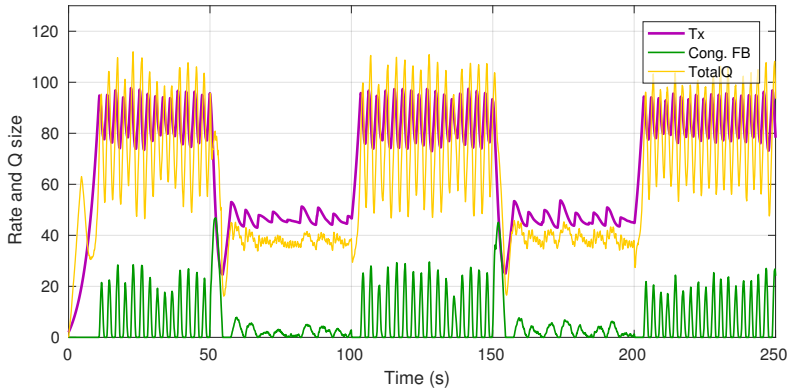
4.1 Dynamics of send rate, congestion signal, and total queueing

In these experiments the Sender and all DIFs have identical model gains: $K_D = -0.01$ and $K_I = 0.2$. Figure 6 shows the results for the non-recursive feedback mechanisms as a reference for the recursive feedback mechanisms with both forward and backward congestion signalling. When congestion is closer to the sender (see especially $t = [50, 100]$ s), the benefits of backward congestion notification are most apparent. Note that the end-to-end model has no intermediate DIFs.

We observe that the forward (see Fig. 7a) and backward (see Fig. 7b) feedback congestion control mechanisms perform with similarly stability to the non-recursive congestion control. Looking at $t = [0, 10]$ s, notice that the buffers, indicated by TotalQ, begin to fill sooner than they do in the corresponding scenarios in Fig. 6. This is because each DIF tries to keep its load, with respect to the probed available capacity on the link (a), within an acceptable range as measured by the DIF packet queue. Recursive congestion control does have a higher total buffer usage. This is not completely unexpected since the recursive architecture has buffers at every DIF. When designed appropriately this buffering can help the network to maintain a high performance in



(a) Recursive forward feedback



(b) Recursive backward feedback

Fig. 7. Recursive rich signal feedback. (Tx is the sender transmission rate, Cong. FB is the congestion signal, and TotalQ is the cumulative total of all queues in the system.)

normal fluctuating traffic demands. However, if not designed appropriately they can unnecessarily delay a packet's end-to-end transit.

Using backward feedback (see Figs. 6b and 7b) provides improvements for both the recursive and non recursive feedback mechanisms.

In choosing between the forward and backward congestion notification mechanisms one must consider that the backward mechanism is more difficult to implement and deploy in a layered architecture. Because lower layers, working on different aggregates of end-to-end flows, do not have the necessary information to signal the correct sender (in RINA, lower DIFs are not meant to inspect packets for headers from upper DIFs), each node must make a local decision about which of the sources of incoming traffic should be informed of congestion, and to what extent. This decision will influence the fairness of rate allocations. Forward signalling is easier to implement as it can follow the path of the packet up the layers.

The pushback mechanism has the nice architectural property of keeping the congestion control at each DIF layer independent. Unfortunately, this results in poor performance (see Fig. 8) with continuous large oscillations in send and receive rates. To

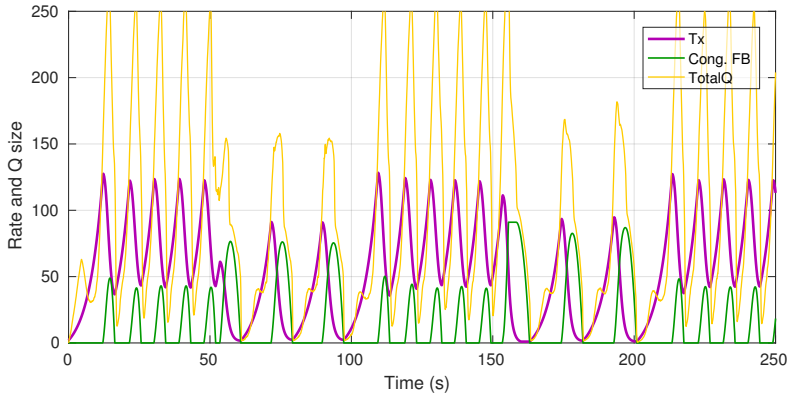
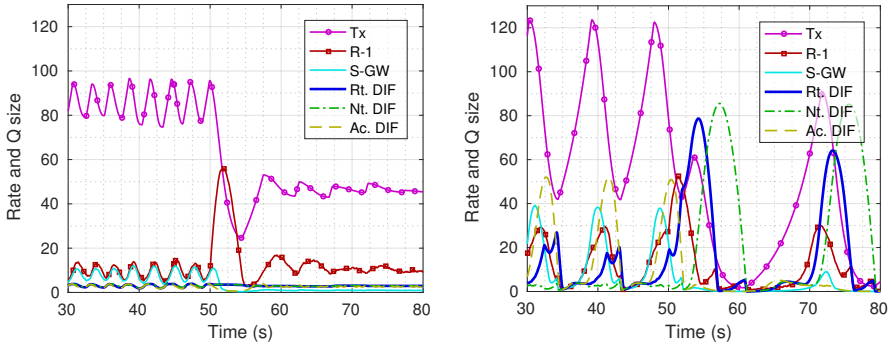


Fig. 8. Recursive pushback feedback. (Tx is the sender transmission rate, Cong. FB is the congestion signal, and TotalQ is the cumulative total of all queues in the system.)



(a) Recursive backward feedback queues (b) Pushback queues

Fig. 9. Comparing recursive pushback and backward feedback queue growth. Referring to Fig. 1: Tx→send rate; queue sizes at: *R-1*→router where congestion is induced (Router DIF), *S-GW*→gateway router closest to the sender, *Rt. DIF*→Route DIF at ISP 1 access, *Nt. DIF*→Network DIF at ISP1 Access, *Ac. DIF*→Access DIF at sender.

investigate the reasons for this we look at the DIF buffer dynamics of the recursive backward feedback and pure pushback mechanisms at the $t = 50$ s cross traffic perturbation at ① in Fig. 1.

Looking first at the recursive backward feedback queues shown in Fig. 9a, when the sudden increase in cross traffic starts at $t = 50$ s, the queue at the affected router begins to grow (*R-1*). This causes *Rt. DIF* and *Nt. DIF* operating across that path to receive a congestion signal and reduce its allowable sending rate (a in Fig. 4). There is no delay between the reaction of *Rt. DIF* and *Nt. DIF* as they are in the same network element. Due to the start of congestion just before $t = 50$ s, the sender had already started reducing its rate so the queues at *Rt. DIF* do not increase. The reduction in traffic alleviates congestion at *R-1*, though it remains the bottleneck. In this experiment all DIF congestion signal gains (K_D) and increase gains (K_I) are the same. Some performance improvement may be able to be made by tuning K_D and K_I by taking into account the collection of congestion signals that collectively make up the C input to each DIF, and

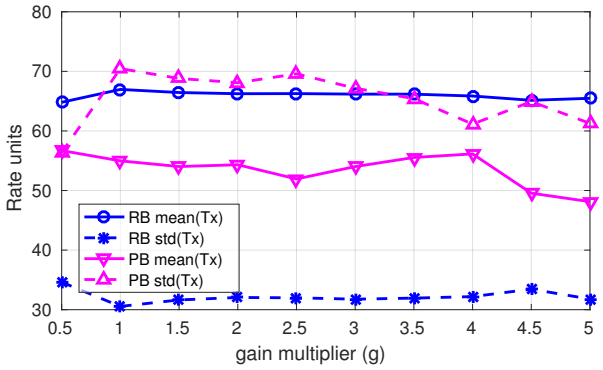


Fig. 10. Effect of increase and decrease gain ($g \times K_I$ and $g \times K_D$ for all lower DIFs) on mean and standard deviation of the resulting sender transmit rate (Tx).

round trip time a particular DIF operates over. However, this is beyond the scope of this paper and an area of future work.

The interactions in the pure pushback control scenario shown in Fig. 9b are more complex. There is general instability caused by the inter-DIF interactions. However, looking just at the disturbance at $t = 50\,\text{s}$, we see that this occurs when the sender is at one of the lower points in its large send rate (*Tx*) oscillations. The queue at *R-1* begins to grow, which causes *Rt. DIF* to reduce its rate, causing *Rt. DIF's* queue to grow and eventually *R-1*'s queue to shrink. *Nt. DIF* also reduces its a, but *Nt. DIF's* queue does not grow immediately as might be expected since the traffic arriving at *Nt. DIF* was initially less than a due to the previous rate cycling (note the capacity of the bottleneck is between 50 and 55). As *Tx* increases and *Nt. DIF's* a decreases, *Nt. DIF's* queue rapidly grows; *Rt. DIF's* queue shrinks as *Nt. DIF* restricts traffic flow. *Nt. DIF's* queue growth signals the sender to reduce its rate (*Tx*).

Notice the push-back queue behaviour caused by the DIF interaction from $t = 65\,\text{s}$, *R-1* push-back to *Rt. DIF*, push-back to *Nt. DIF*. This is repeated until congestion the large cross traffic disturbance stops at $t = 100\,\text{s}$ with the sender rate going from 0 to a peak and back to 0 again. The system is not able to obtain an efficient stable state. There are two key reasons for the instability in the pure pushback mechanism: 1. it takes time for the queueing signal to be pushed back, 2. the sender is not aware of the total queueing along the path—congestion at lower DIFs fedback to the sender.

4.2 Effect of DIF gains on stability

Since DIFs under the end-to-end Sender DIF operate over a smaller topology, there may be some advantage in their gains being more aggressive than the Sender's. Leaving K_I and K_D the same for the Sender, we adjust the lower DIF gains, multiplying them by the factor $g = 0.5, 1, \ldots, 5$. Figure 10 illustrates the effect that varying K_I and K_D in the lower DIFs has on the overall system stability for the recursive backward feedback (RB) and the pushback (PB) scenarios. The simulation results have been re-sampled to avoid bias in points calculated through Simulink's variable step solver. Mean(Tx) is the average over the entire 250s, while std(Tx) is calculated using the variance about

the mean of each 50 s interval, matching the step changes in available rate in the simulations. The recursive pure pushback mechanism suffers from poorer throughput and a much higher variance than the recursive backward feedback mechanism. The performance of both mechanisms generally diminishes as the lower DIF congestion controls become more aggressive than the Sender ($g \geq 1$), though to a lesser extent with the recursive backward feedback mechanism. When the lower DIFs are less aggressive than the sender the recursive backward feedback mechanisms has a small drop in performance as the senders rate adjustments are limited by the DIFs. This dampening helps reduce the standard deviation of Tx a little in the pushback scenario.

4.3 Observations from conducting experiments

Constructing the model and conducting these experiments gave insight into the dynamics of congestion control in recursive architectures. In this section we outline some of these insights.

Congestion control with recursive feedback may require more buffering between end points than end-to-end congestion control. This is particularly apparent in these experiments where all physical links have the same capacity, with the available capacity only being slightly different due the cross traffic. Whether this significantly affects end-to-end latency needs further investigation.

For the recursive congestion controls to work well each DIF needs to accurately track the available capacity, which happens in these experiments. This is especially critical for pushback. A DIF only probes for more available capacity when its buffers begin to fill, and reduces its maximum send rate when its buffers reduce or it is notified of congestion. In the topology we model where all links had a similar available capacity, this can result in cascading delays as each DIF probes for more capacity to meet demand. In the pure pushback scenario this probing and corresponding reactions to congestion result in wild oscillations.

These experiments suggest that the pushback mechanism is the least stable of the five feedback mechanisms tested. We explored manually tuning the pushback mechanism's parameters to yield better results, but with little success. In experiments with a simpler topology, half the size and with one less layer, we were able to manually tune the pure pushback to produce some improvement. Even so, in the smaller simpler topology, it was still the least stable and worst performing of the five feedback mechanisms.

Since the sender is the ultimate source of traffic, it is important that it receives a complete picture of congestion along the end-to-end path in order to react appropriately to it. The other two recursive congestion control feedback mechanisms give DIFs and the sender a complete picture of congestion along their respective end-to-end controlled paths, and performed equivalently well as a non-recursive mechanism.They therefore seem to be the best way forward for a stable recursive congestion control that will not diminish the advantages of using a recursive network architecture such as RINA and RNA.

These experiments use identical parameters for all congestion control DIFs in Sec. 4.1. This makes it easier to compare mechanisms, but is not an optimal solution for a recursive architecture where each DIF may be able to adapt to the topology beneath it and use their observation of feedback delays and optimise their responses accordingly. Sec. 4.2

shows that simply making the congestion control in lower DIFs more aggressive does not give performance benefits. The experiments do not show how well a individually tailored and optimised recursive congestion control for each DIF may be able to work; an area for further investigation. To date such a congestion control mechanism does not exist, and the interactions demonstrated in this paper indicate that a stable efficient design may be difficult. Backstepping control theory [12] has similarities with recursive architecture congestion control and may hold promise in designing such a mechanism.

5 Conclusions and future work

Our findings demonstrate that the layer independent pure pushback mechanism performs badly in a realistic topology. Delays in the congestion signal being pushed back and up through the layers are observed to be part of the problem. Also we observe that due to the strict layer independence, congestion controllers at each layer—and ultimately the sending sources—are unable to respond to the full extent of congestion as that is hidden from them.

We observe that a feedback mechanism which allows all congestion controllers to receive signals from all sources of congestion along the path they control allows for a better performing and more stable control. Backward feedback provides the greatest benefits, especially for sources of congestion that are closer to the sending sources than the destinations. However, we acknowledge that this type of feedback can be difficult to implement in a real recursive architecture. Forward end reflected feedback does not perform quite as well, but does still provide a workable mechanism.

Work on congestion control in recursive architectures is in its infancy. The area itself is very large, and this work takes the first steps at developing an understanding for the way feedback mechanisms affect the recursive congestion control system in a simple but non-trivial network. We plan to continue work in this area investigating the use of multiple loop control theory, such as backstepping control, as a means of developing a stable efficient mechanism for congestion control in recursive network architectures such as RINA and RNA.

Acknowledgment

This work has received funding from the European Union's FP7 research and innovation programme under grant agreement No. 619305 (PRISTINE). The views expressed are solely those of the authors.

References

1. Agnew, C.E.: Dynamic modeling and control of congestion-prone systems. Operations Research 24(3), 400–419 (1976)
2. Bensley, S., Eggert, L., Thaler, D., Balasubramanian, P., Judd, G.: Datacenter TCP (DCTCP): TCP Congestion Control for Datacenters. Internet-Draft draft-ietf-tcpm-dctcp-01, Internet Engineering Task Force (May 2016), https://tools.ietf.org/html/draft-ietf-tcpm-dctcp-01, work in Progress

3. Day, J.: Patterns in Network Architecture: A Return to Fundamentals. Prentice Hall (2007)
4. Day, J., Matta, I., Mattar, K.: Networking is IPC: a guiding principle to a better internet. In: Proc. ACM CoNEXT. p. 67 (2008)
5. Fairhurst, G., Sathiaseelan, A., Secchi, R.: Updating tcp to support rate-limited traffic. RFC 7661, RFC Editor (October 2015)
6. Gerla, M., Kleinrock, L.: Flow control: A comparative survey. IEEE Transactions on Communications COM-28(4), 553–574 (April 1980), (Also published in Computer Network Architectures and Protocols, P. Greed, Ed., Plenum Press, pp. 361-412, 1982.)
7. Gursun, G., Matta, I., Mattar, K.: On the performance and robustness of managing reliable transport connections. Tech. rep., CS Department, Boston University (2009), bUCS-TR-2009-014
8. Hassan, M., Jain, R.: High performance TCP / IP networking - concepts, issues, and solutions. Pearson Education (2004)
9. Ishakian, V., Akinwumi, J., Esposito, F., Matta, I.: On supporting mobility and multihoming in recursive internet architectures. Computer Communications 35(13), 1561–1573 (2012)
10. Kelly, T.: Scalable TCP: Improving performance in highspeed wide area networks. ACM SIGCOMM Computer Communications Review 33(2), 83–91 (Apr 2003)
11. Khalil, H.K.: Nonlinear Systems. Prentice Hall (2001)
12. Krstic, M., Kokotovic, P.V., Kanellakopoulos, I.: Nonlinear and Adaptive Control Design. John Wiley & Sons, Inc., New York, NY, USA, 1st edn. (1995)
13. Lopez, D. (ed.): Use cases description and requirements analysis report. PRISTINE project (May 2014), http://ict-pristine.eu/wp-content/uploads/2013/12/pristine-d21-usecases_and_requirements_v1_0.pdf
14. The Mathworks, Inc., Natick, Massachusetts: MATLAB SIMULINK version 8.7 (R2016a) (2016)
15. Small, J.: Patterns In Network Security: An Analysis Of Architectural Complexity In Securing Recursive Inter-Network Architecture Networks. Master's thesis, Boston University Metropolitan College (2012)
16. Teymoori, P., Welzl, M., Stein, G., Grasa, E., Riggio, R., Rausch, K., Siracuss, D.: Congestion control in the Recursive Internetwork Architecture (RINA). In: IEEE International Conference on Communications (ICC), Next Generation Networking and Internet Symposium (May 2016)
17. Touch, J., Baldine, I., Dutta, R., Finn, G.G., Ford, B., Jordan, S., Massey, D., Matta, A., Papadopoulos, C., Reiher, P., Rouskas, G.: A dynamic recursive unified internet design (druid). Computer Networks 55(4), 919 – 935 (2011), http://www.sciencedirect.com/science/article/pii/S138912861000383X, special Issue on Architectures and Protocols for the Future Internet
18. Touch, J.D., Pingali, V.K.: The rna metaprotocol. In: Computer Communications and Networks, 2008. ICCCN'08. Proceedings of 17th International Conference on. pp. 1–6. IEEE (2008)
19. Wang, W.P., Tipper, D., Banerjee, S.: A simple approximation for modeling nonstationary queues. In: Proc. of the IEEE International Conference on Computer Communications (INFOCOM). vol. 1, pp. 255–262 (Mar 1996)
20. Welzl, M.: Network Congestion Control: Managing Internet Traffic (Wiley Series on Communications Networking & Distributed Systems). John Wiley & Sons (2005)

Assuring QoS Guarantees for Heterogeneous Services in RINA Networks with ΔQ

Sergio Leon Gaixas, Jordi Perelló, and Davide Careglio
Universitat Politècnica de Catalunya
Barcelona, Spain
email: slgaixas@ac.upc.edu

Eduard Grasa, Miquel Tarzan
Fundació Privada i2CAT
Barcelona, Spain

Neil Davies and Peter Thompson
Predictable Network Solutions
Glastonbury, United Kingdom

***Abstract*— With the increasing usage of cloud computing and dependence on a diverse set of distributed applications, users are reliant on consistent outcomes from a shared infrastructure. This drives the need for improved QoS guarantees for heterogeneous communication requirements over shared networks. The Recursive Inter-Network Architecture (RINA) is a fundamental programmable network architecture that provides a consistent model for supporting QoS across multiple layers. In this work we evaluate the performance outcomes provided by such polyservice RINA networks in conjunction with per-layer ΔQ-based resource allocation policies. ΔQ provides a resource allocation model able to enforce strict statistical limits on the maximum experienced losses and delays through the smart utilization of traffic policing and shaping strategies, together with an analytical pre-dimensioning of buffer thresholds. Our target scenario is a backbone network that prioritizes communications among geographically distributed datacentres using resources shared with best-effort background traffic. Results obtained with the RINASim simulation software show that a ΔQ-enabled RINA network can yield the desired absolute QoS guarantees to the assured traffic classes without negatively impacting the rest, unlike current MPLS-based VPN solutions.**

Keywords*— *RINA; QoS; deltaQ; inter-datacentre

I. INTRODUCTION

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

In this context, the Recursive InterNetwork Architecture (RINA) [1,2] allows applications to request communication services with statistically bounded metrics such as loss or delay. RINA is a recursive, multi-layer architecture that models computer networking as distributed Inter-Process Communications (IPC). All layers in RINA perform the same set of functions, but configured with specific policies that best deliver the requested outcomes. In RINA, each layer knows the expectations of the applications using it, and the level of service that it can expect from the layers below. Given this recursive, programmable architecture and the knowledge of requirements, RINA becomes an interesting Internet model capable of supporting a myriad of distributed applications with heterogeneous requirements.

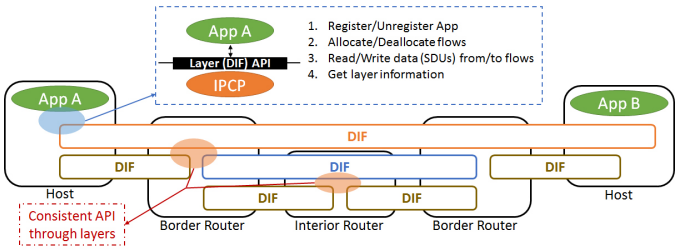


Fig. 1 Structure of the Recursive InterNetwork Architecture (RINA)

In this work we focus on evaluating the service outcomes provided by such polyservice RINA networks exploiting the concept of ΔQ [3,4,5], which leads to a resource allocation framework able to guarantee predictable outcomes to data flows with heterogeneous quality requirements. The rest of the paper is organized as follows. Section II introduces RINA and some of its benefits compared to the current Internet model. Section III introduces the concept of ΔQ and how can it be incorporated in RINA networks. Section IV details our use case scenario and the specific assumptions made in our analysis. Section V provides illustrative numerical results. Finally, section VI concludes the paper.

II. RECURSIVE INTERNETWORK ARCHITECTURE

RINA is an architecture for computer networking based on the idea that networking is distributed IPC and only IPC [6]. As illustrated in Fig. 1, RINA presents a single type of layer, a distributed application called a Distributed IPC Facility (DIF), that repeats as many times as needed by the network design. While in the Internet architecture each layer contains a different set of functionalities and offers different Application Programming Interfaces (APIs) to upper layers, RINA defines a single programmable layer that contains all the functions that are needed to provide IPC services to applications or higher level DIFs. RINA specifies the different functions and components contained in the DIF, and a set of variable

This work is partly funded by the European Commission through the PRISTINE project (Grant 619305), part of the Future Networks objective of the Seventh Framework Programme (FP7). Moreover, it has been partially funded by the Spanish National project SUNSET (TEC2014-59583).

behaviours (policies) for each function. This allows the network administrator to properly select policies for each DIF depending on its scope, operating environment, and level of service provided by its lower level DIFs.

Even though all DIFs provide the same type of service, the characteristics of the offered service will vary between DIFs. Each DIF defines a set of supported QoS Cubes, i.e., QoS classes for flows providing statistical bounds on metrics like data rate, latency, losses, etc. Given these QoS Cubes, applications and upper DIFs can request flows with specific requirements, and the DIF then provides a flow with the QoS Cube that matches these requirements, using the policies (scheduling, routing, etc.) best suited for its purpose.

The tasks of creating and managing DIFs are accomplished by the Network Management System (NMS), a distributed application composed of one or more Manager(s), and a Management Agent (MA) residing in each device. The NMS is responsible for creating RINA nodes, triggering their enrolment into DIFs, monitoring them, updating their configuration, etc. As each DIF knows the requirements of all its upper flows, RINA can make more informed and optimal decisions within each layer, including the requests to lower layers, e.g., to allocate new flows with certain QoS levels.

Given the layering structure of the DIFs and the control of the NMS, changes in the connectivity between nodes in one layer can be completely hidden from upper layers. This, together with its complete naming and addressing schemes and configurable policies, makes RINA a smart substitute for the current IP model. Despite RINA's clear differences from the current IP model, it remains compatible with it, allowing a progressive migration at little expense, either by replacing lower layers with RINA while keeping IP networks and services on top, or by using any network protocol (e.g. Ethernet, WDN, IP, UDP, etc.) as a bearer for RINA.

III. Concept of Quality Degradation

In a computer network, quality is something that is never gained, but can only be lost. An ideal network would copy information with zero data loss and zero latency. However, real networks introduce latency and can lose data. This reduction in quality is called attenuation or degradation, and how this is shared between different flows competing for shared resources is the core of any QoS assurance strategy. In particular, three properties are important in terms of quality: bandwidth, latency and losses.

We use 'ΔQ' to designate the quality attenuation between two points along a network path. The performance of IPC depends on different details of the packet delivery depending on the protocol/application; hence, average measures do not contain sufficient information to guarantee the performance of every IPC. Thus, the measure of ΔQ is based on distributions instead of averages or moments. For example, saying that in traversing a network segment "at most 0.01% of packets will experience more than 5ms of latency" provides more useful bounds that "the average latency is 3ms".

With respect to latency and loss, there are two distinct causes of degradation in any network. Firstly, any bearer introduces some degradation, given by metrics like packet service time and distance, which is independent of the current usage of it. Secondly, there is variable degradation due to contention for shared resources such as buffers. This variable degradation involves three parameters: offered load, latency and loss, connected by a relation with two degrees of freedom. Specifically:

- For a fixed load, reducing latency means increasing losses.
- For a fixed latency, as the offered load increases, so does the probability of dropping packets.
- For a fixed loss probability, increasing load increases latency.

Within these constraints, it is possible, for a given load, to adjust the sharing of latency and loss degradation to correlate with the needs of the flows.

As some applications are more sensitive to losses than others, and the same can be said for latency, we can say that some flows are more cherished (require lower losses) or more urgent (require less latency). Hence, their requirements can be mapped into a Cherish/Urgency (C/U) matrix, that is, an *N*x*M* matrix with relative latency and losses at each edge (a 3x2 C/U matrix is shown in Table I). This has a straightforward implementation called a Cherish/Urgency multiplexor (C/U mux) [3]. A C/U mux provides differential loss probability using a shared buffer with higher thresholds for packets of more cherished flows, and differential urgency by giving higher precedence service for packets of more urgent flows. Just as total delay is conserved under scheduling [7], ΔQ is conserved in C/U multiplexing.

TABLE I Example of a 3x2 cherish/Urgency matrix

C/U	**+ Cherished**	**….**	**- Cherished**
+ Urgent	A1	A2	A3
- Urgent	B1	B2	B3

In addition to the C/U multiplexor, a Policer/Shaper (P/S) can be used to re-shape, limit the flow rate or even change the C/U class of flows, to better adapt to the different QoS requirements and provide protection versus breaches on contracts or the maturing of overbooking hazards. While not always required, this P/S allows a better assurance of QoS and a way to relate the distinct services in different congestion scenarios, and gives considerable flexibility in controlling the ΔQ experienced by assured flows. The C/U mux by itself implements a strict partial order between the ΔQ experienced by the different classes; combining this with bounds on the offered load delivers absolute statistical guarantees.

ΔQ can be composed and decomposed along a path [5], corresponding to multiplexing points at DIF boundaries. ΔQ at an N-level DIF can also be related to the ΔQ provided by its underlying (N-1)-level DIF(s) and its own operation. Thus, the compositional properties of ΔQ and the DIF structure of RINA are complementary. Furthermore, flows sharing a C/U class experience similar ΔQ, allowing for complete flexibility in aggregating and disaggregating flows, enabling a scalable approach to ensuring quality.

IV. Scenario Description

Given the importance of QoS-aware networks and cloud computing for future network scenarios like 5G, remote applications for smart devices, sensor networks, etc., providing and assuring differentiated and bounded levels of

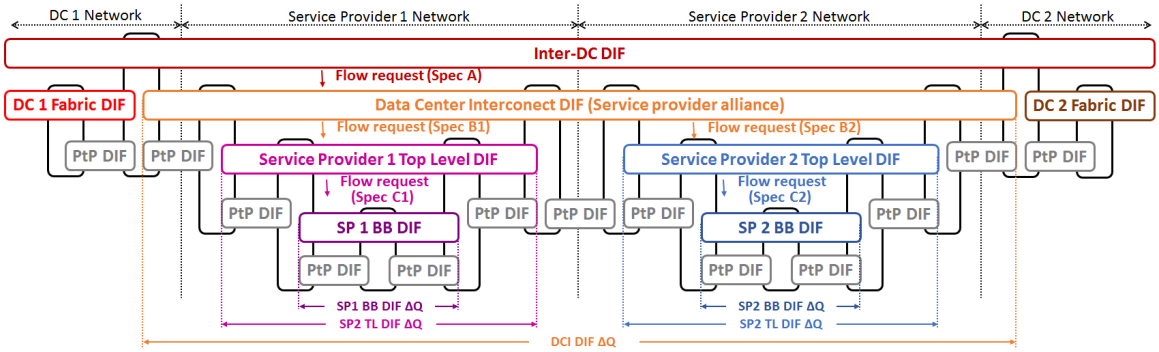


Fig. 2 DIF configuration in the assumed scenario

QoS on backbone networks is a must, so we took these networks as an interesting study case.

In the scenario under consideration, a network provider offers different types of IPC services. Some are targeted to providing connectivity between distributed datacentres, as illustrated by Fig. 2, while others are for general best-effort traffic. The provider network consists of two main layers: the Top Level DIF, which is exposed to customers and allows them to access the provider's IPC services, and the backbone (BB) DIF, which is an internal layer where most of the routing and resource allocation policies are enforced. We will analyse the backbone DIF, in charge of providing adequate differential traffic treatment for the different traffic classes.

To analyse the benefits of a RINA backbone DIF with ΔQ-based policies, we consider a DIF with the same internal structure as the 10-node IP/MPLS layer of the Internet-2 backbone network [8] as shown in Fig. 3, where link latencies are derived from the real physical distances between node locations. Circles represent nodes in the DIF (called IPC Processes or IPCPs) and solid lines flows provided by the multiple N-1 level point-to-point DIFs shown in Fig. 2 (called N-1 flows). Moreover, we assume that all N-1 flows have a capacity of 10 Gbps and impose a Maximum Transmission Unit (MTU) of 5KB. Note, however, that the achieved results would also be representative of scenarios with higher N-1 flow capacities (40 or 100 Gbps), provided that the offered loads are scaled accordingly.

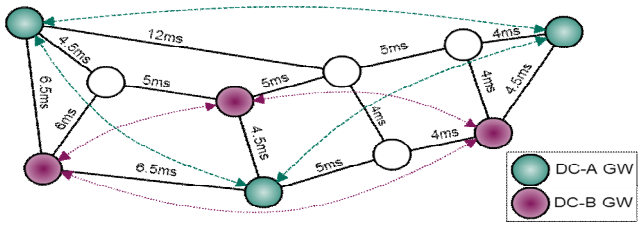


Fig. 3 ISP Backbone DIF describing a network of 10 nodes and 15 bidirectional links used for the experiments, where the locations of the datacentres are highlighted. Dashed and dotted arrows illustrate inter-datacentre communications.

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

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

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

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

Each pair of RINA IPC processes (referred to as "nodes" from now on) exchange sBE and BE flows (representing a full-mesh background Internet traffic) with a 3:7 sBE:BE traffic ratio, so that all DIF links are filled with a similar load level. Our motivation behind this is to assume a worst-case scenario (in terms of congestion) in which all links are heavily loaded. Furthermore, we use a simple Equal Cost Multi-Path (ECMP) forwarding with memory per flow [9,10] in order to perform load balancing without introducing jitter or the need for reordering caused by disjoint path usage by the same flow.

A. Configuring RINA ΔQ-POLICIES

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

We can observe that, while we have requirements for low latency, the stationary latency caused by the distance between nodes may be higher than any delay introduced by nodes due to medium or moderately high congestion. This has two important implications when configuring the DIF. Firstly, it is clear that following the shortest paths (in terms of hops) may not provide optimal latency for delay-sensitive flows. Instead, we can use the total stationary latency as a metric for routing the more urgent flows, resulting in an overall improvement of

their experienced performance. Secondly, although the stationary factors may dominate the average latency, the congestion in network nodes is the source of jitter. In order to control this jitter, we are going to put strict requirements on the degradation of latency in each hop, in terms of packet service times (PST).

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

To reify these goals, we have set the following end-to-end requirements (w/ high probability) in our scheduling policies:

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 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.

B. QoS to Cherish/Urgency classes

We can map the requirements of the GU, SN, sBE and BE traffic classes, into the 4x2 Cherish/Urgency matrix shown below in Table II. In order to simplify the configuration, we consider the same worst-case scenario at each node, and consider ΔQ in a way that requirements are met in the longest paths. To achieve this, we set some requirements per hop for the different levels of cherish and urgency depending on the current aggregated load on the link, subsequently illustrated in Tables III and IV.

TABLE II QOS TO CHERISH/URGENCY CLASSES

C/U	+ Cherished	….		- Cherished
+ Urgent	GU		sBE	
- Urgent		SN		BE

TABLE III MAXIMUM AVG. LOSS (%) PER LOAD PER HOP

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

TABLE IV MAXIMUM PST REQ. PER LOAD PER HOP

Load/QoS	0.90	0.95	1.00	1.20	1.50
GU, sBE	<10				<15
SN, BE	<75	<125	<200	<225	<250

Being in a RINA scenario where the load of the flows can be controlled at source and destination IPCPs in a trusted way, the requirement for flow policing is minimized, so we omitted it from the analysis. Instead, we used a modified version of the C/U mux based on: i) a first heuristic threshold; ii) a second absolute threshold of total buffer occupancy. When the total occupancy reaches the heuristic threshold, i.e., congestion starts to occur, a proportion of the incoming packets will be independently randomly dropped. With this simple heuristic threshold, we can largely reduce the probability of consecutive losses when reaching the absolute threshold while, at the same time, allowing the different cherish classes to better share losses.

C. ΔQ analysis

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

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

TABLE VI PARAMETRISED BUFFER PER QOS

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

After analysing the possible configuration ranges, we settled on the one in Table VI, which fulfils the imposed requirements while allowing for a small margin of error in case of having non-Poisson arrival patterns.

V. Numerical Results

A. ΔQ configuration validation

In order to validate the configuration for our scenario, we perform simulations with the RINASim simulation software developed in the FP7 PRISTINE Project [11,12]. As we have configured our DIF in a way that ΔQ requirements are ensured per hop in a worst-case scenario, our first tests aim to check whether the expected behaviour computed via analysis of loads and requirements is delivered in a simple single-hop scenario. This scenario is simply composed of a sequence of three nodes inter-connected by two 10Gbps links. For each offered load value and ratio between inter-datacenter flows and best-effort ones (1:9, 2:8 and 3:7), a 10s simulation is performed. In each run, 1000 flows are established, served following an exponential distribution and with average data rate distributed between 7 and 13 Mbps (At 100% load) and spread between the distinct QoS given the stated rates between QoS. To add more randomness, packet sizes were uniformly distributed between 2KB and 4KB.

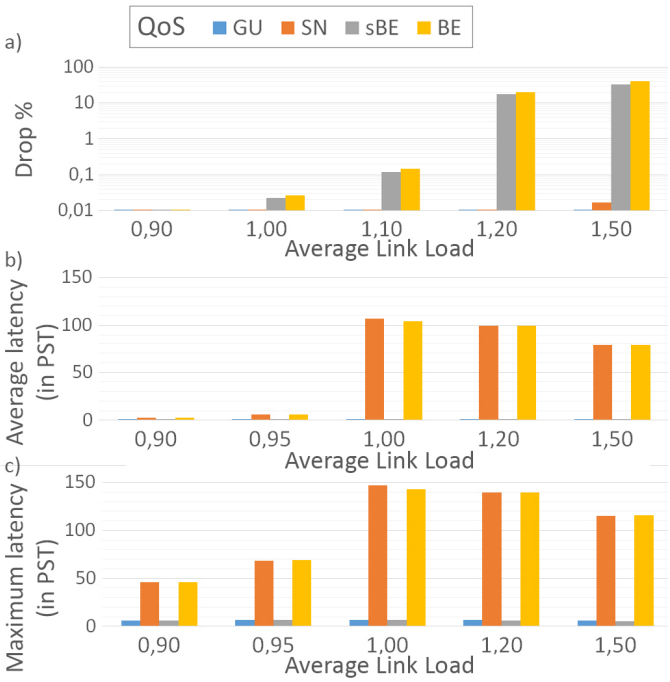


Fig. 6. For single hop flows at distinct loads: a) Average drop (%); b) Average delay; c) Max delay.

The results from these first experiments confirm those from the analysis. As shown in Fig. 6-a, inter-DC flows (GU and SN) remain practically lossless in all the experiments, successfully fulfilling their loss requirements. In addition, the losses experienced by sBE and BE flows are within the allowed limits and their losses match pretty much the over-load in the network. In terms of latency, as shown in Fig. 6-c, urgent flows encountered less than 10 preceding packets, having to wait on average for 0 or 1 packets to be served, as shown in Fig. 6-b. Less urgent flows also remained within acceptable delays, correctly shared between both classes. As a note, is interesting to see how the increment in losses for sBE and BE flows resulted in smaller latencies for non-urgent flows given the changes on accepted loads for each QoS.

B. Simulating the full backbone DIF

Having validated the configuration of isolated nodes, we proceed to perform backbone DIF-wide experiments. For these experiments, we offer a load to the DIF so that all links are equally loaded, as mentioned in the previous section. We consider the previously stated distribution of bandwidth for a full load, allocating the same type of flows (7 to 13Mbps at 100% load, 2-4KB PDUs, etc.) between the distinct pairs of nodes, and then scaling the flow data rate to the desired DIF load. For inter-DC traffic 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 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 ΔQ increases along their paths, whereas 'dumb' flows used to fill links are point-to-point. This means that their ΔQ does not affect the behaviour at other nodes, that is, the losses at one congested node do not reduce the incoming rate of PDUs at downstream nodes, which would happen if the network had been filled with multi-hop flows.

In this work, we do not limit ourselves to only establishing that end-to-end requirements are met using the RINA + ΔQ-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 [13,14]. In both cases, we used the same routing for fair comparison purposes, where urgent flows use latency as the metric to optimize and the rest use number of hops. For the Best-Effort scenario, 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 favour of RINA + ΔQ-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. 7-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 + ΔQ 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. 7-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.

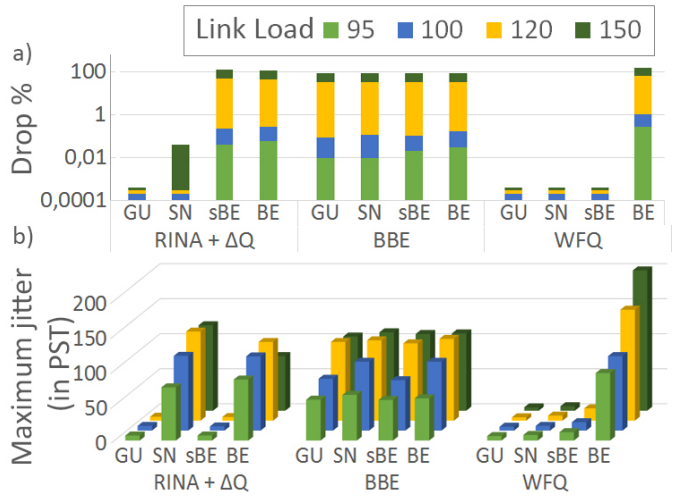


Fig. 7. Average drop (a) and maximum jitter in PST (b) for GU, SN, sBE and BE flows depending on the scheduling policy used in the network

Conversely, in addition to providing better assurance on delay and losses, the RINA + ΔQ-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 thought there are necessarily high losses in average, the soft requirement of having at most 3 consecutive losses in sBE flows was pretty much fulfilled. Under higher loads, it becomes nearly impossible to avoid consecutive losses given the multiple points of congestion. However, those are still limited to, for example, less than 1% situations of more than 3 consecutive packets for an offered load of 120%.

VI. Conclusions

Future networks must become demand-attentive in order to support the requirements of multiple distributed applications over the same infrastructure. Assuring absolute QoS guarantees is a must, so that each application using the network gets the quality it requires, no more and no less. In this paper, we have presented RINA, a multi-layer, recursive, programmable network model based on the distributed IPC paradigm that provides scalability and QoS assurance guarantees. In conjunction with the ΔQ 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.

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. 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, and providing a wide range of QoS classes to ensure any type of ΔQ requirement.

References

[1] E. Trouva, E. Grasa, J. Day, I. Matta, L. T. Chitkushev, P. Phelan, M. P. d. Leon and S. Bunch, “IS THE INTERNET AN UNFINISHED DEMO? MEET RINA!”, October 2010.

[2] J. Day, I. Matta, and K. Mattar. “Networking is IPC: A Guiding Principleto a Better Internet”. In CoNEXT’08: Proceedings of the 2008 ACMCoNEXT Conference, pages 1-6, New York, NY, USA, 2008. ACM.

[3] Predictable network Solutions Limited, “A Study of Traffic Management Detection Methods & Tools”. OfCom, United Kingdom, August 2015.

[4] N. Davies, “Delivering predictable quality in saturated networks,” Technical Report, September 2003, available online at http://www.pnsol.com/public/TP-PNS-2003-09.pdf

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

[6] J. Day, Patterns in network architecture: a return to fundamentals, Prentice Hall, January 2008.

[7] L. Kleinrock “Queuing System, Vol 1: Theory” Wiley, 1975, p.117

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

[9] Y. Ganjali and A. Keshavarzian "Load balancing in ad hoc networks: Single-path routing vs. multi-path routing", Proceedings of IEEE INFOCOM

[10] Srikanth Kandula, Dina Katabi, Shantanu Sinha, and Arthur Berger. 2007. Dynamic load balancing without packet reordering. SIGCOMM Comput. Commun. Rev. 37, 2, pp 51-62, March 2007.

[11] PRISTINE FP7-619305, Programmability in RINA for European supremacy of virtualized networks, http://ict-pristine.eu

[12] RINASim, available at https://github.com/kvetak/RINA

[13] Haeryong Lee; Jeongyeon Hwang; Byungryong Kang; Kyo "End-To-End QoS Architecture for VPNs: MPLS VPN Deployment in a Backbone Network". Parallel Processing, 2000. Proceedings 2000 International Workshops, pp 479-483, 2000.

[14] V. Fineberg, "A practical architecture for implementing end-to-end QoS in an IP network", IEEE Comm. Mag., vol. 40, no. 1, pp.122 -130, January 2002

SFR: Scalable Forwarding with RINA for Distributed Clouds

Fatma Hrizi, Anis Laouiti, Hakima Chaouchi
Telecom SudParis, CNRS Samovar, UMR 5117, France
Email: fatma.hrizi,anis.laouiti,hakima.chaouchi@telecom-sudparis.eu

***Abstract*—Distributed clouds have been regarded as the key enabling technology to provide new trends of services. The contribution of this paper is twofold. First, we discuss current distributed clouds architecture and highlight the issues involved, specifically scalability. Second, we investigate the benefits of the use of the Recursive InterNetwork Architecture (RINA) as a networking solution for the distributed clouds. Our proposal, called *Scalable Forwarding with RINA (SFR)*, has been evaluated via simulations and showed to reduce the routing table size by around 75%. The paper concludes by highlighting the advantages of the RINA-based approach over current distributed clouds networking solutions in terms of scalability, simplicity and manageability.**

***Keywords*—Distributed Clouds, RINA, network management, routing, topological addressing, performance evaluation, simulation.**

I. Introduction

In the last few years, cloud computing has gained considerable interest from researchers, industries and standardization bodies [1], [2], [3]. 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. Basically, distributed clouds rely on resources where dedicated facilities are deployed in traditional datacenters. Nevertheless, another kind of distributed clouds has emerged. It consists mainly of resources scattered in offices, costumers' homes and/or data centres participating to the cloud services in a volunteer fashion. This new concept of 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 therefore needed to cope with the evolution of the cloud systems. VIFIB [4], [5] 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 copying each associated database in at least three different distant sites, the probability of mass destruction 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 master and slave design. The master controls the different computers running slaves. In terms of networking, the master and the slaves at different locations are interconnected through multiple IPv6 providers. In order to guarantee a high reliability, VIFIB uses an overlay called re6st [6], [7], which creates a mesh network of OpenVPN tunnels on top of several IPv6 providers and uses the Babel protocol [8] for computing the routes between nodes. Despite its effectiveness, the re6st has certain limitations related mainly to scalability and security. The re6st creates a flat topology that does not scale in case of large networks, which becomes an important concern for the future of the distributed clouds.

In this paper, we study mainly the issues related to scalability in distributed clouds, specifically, we consider 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 a solution based on a new promising architecture, namely, the Recursive InterNetwork Architecture (RINA) [9]. RINA, by its design, is better suited to handle large scale networks and provide interesting benefits towards the current over IP solutions, like enhanced security or extended programmability. The objective of this paper is to demonstrate how our RINA-based architecture, namely Scalable Forwarding with RINA (SFR), outperforms the re6st overlay and gives better results. Using the RINASim [10] platform, we illustrate how SFR showed to be significantly more adapted to distributed clouds, improving the forwarding table size by around 75%. This paper is organized as follows. Section II gives some background on the VIFIB distributed clouds. In Section III the recursive RINA architecture is described. Section IV addresses the possible application of RINA to efficiently manage the VIFIB system. Section V details some evaluation studies that have been conducted in the scope of PRISTINE project [10] and using RINASim simulator. Finally, section VI reports the conclusions and provides directions for further research.

II. Distributed Clouds: the re6st (Resilient Overlay Networking System)

In this Section, we give some background on the VIFIB system. Furthermore, we discuss the issues and challenges related to its networking system: the re6st. VIFIB is a decentralized cloud system, also known as resilient computing [4]. Resources are scattered in computers that are located in customers' homes and in offices. The VIFIB system is based on

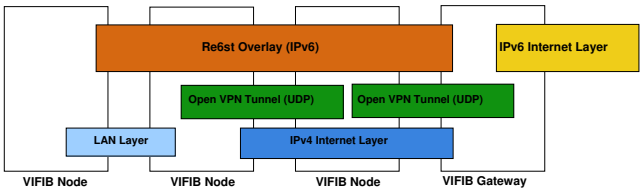


Fig. 1. Architecture of the re6st overlay.

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. Two services running in different locations, compatible or not with IPv6, can be interconnected through a secure link. Tunnels between computers change every 5 minutes. To minimize the latency, VIFIB implements a strategy that uses the best possible tunnels according to a heuristic metric. The less used tunnels are the ones replaced first but always maintaining the compromise between resilience and low latency. This secure, resilient and low latency overlay network is called re6st [6](see Figure 1). The re6st creates a mesh network of OpenVPN tunnels on top of several IPv6 providers and uses the Babel protocol to calculate the best routes. Babel protocol [8] 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 [7]. The system should ensure higher privacy and resilience in order to avoid losing connectivity.

Despite its robustness, the re6st still presents some weakness related mainly to scalability and security. 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 significantly 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 as they are created randomly 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 issues in the VIFIB system and propose to apply RINA to enhance its performance. In the next section, we give an overview of the architecture RINA.

III. RINA: Recursive InterNetwork Architecture

In this section, we introduce RINA and highlight some interesting features related to networking, routing and addressing in this architecture. RINA [9] is a clean slate architecture that introduced a new science of networking based on the "Inter-Process Communication" (IPC) model [11]. The theory of "networking is IPC" is not new as it was first introduced in the ARPANET and XNS/Ethernet [9]. In RINA, the idea is to leverage this approach and develop it with the main aim to revisit the current Internet architecture and provide a more structured model. Networking is no more a layered set of different tasks (as in TCP/IP model) but it is considered instead as a single layer of distributed IPC that could be recursively repeated in different scopes. Each layer of distributed IPC implements the same mechanisms, as illustrated in Figure 2 but policies are tuned to operate over different scopes of performance (bandwidth, range and scale). This provides a clean separation between mechanism and policy which gives considerable freedom in expressing a variety of policies. Another feature of RINA is the fact that it adopts and extends the Saltzer's model [12] in the context of a recursive and scalable model. As depicted in Figure 2, data transfer, control and management are separated in order to isolate short, medium and long time-scales. Reiterating the IPC model over multiple levels provides a comprehensive architecture to support well-organized and well-scoped services. This *"divide and conquer"* concept enables scalability over large networks and avoids issues related to growing routing tables due to the limited scope of each IPC layer.

RINA with its IPC model captures the common elements of distributed applications, called DAFs (Distributed Application Facilities) [9]. A DAF is defined as the collection of "Distributed Application Processes" (DAPs), which collaborate to perform a given task. They communicate using a specific application protocol which enables DAPs to exchange structured data in the form of objects. These DAPs uses an underlying facility in order to communicate which is called "Distributed IPC Facility" (DIF). A DIF is a collection of DAPs cooperating to provide Inter-process communication (IPC). The DAPs that are members of a DIF are called IPC Processes or IPCPs. Basically, the DIF facilitates flow allocation between DAPs over different scopes. Figure 2 shows an example of two levels of DIFs (each at a given scope) used by three different DAPs. RINA by its recursive model gives more flexibility to define scalable design. DIFs are independent and isolated which assists scalability in terms of both network operation and management.

Forwarding and Addressing in RINA

Forwarding in RINA is addressed basically from two perspectives, i.e. routing inside the DIF and "routing" or mapping between DIFs. The former concerns "local" routing in the scope of the DIF itself. It is a task in the relaying and multiplexing mechanism of RINA architecture. In a given DIF, the choice of a particular forwarding algorithm is a matter of policy 2. The latter has a wider scope and it is better called mapping. It provides the mapping between the DIFs. It is managed by the DIF allocator [13] component of RINA. The DIF allocator is a function of the IPC Management that determines the set of DIFs to cross in order to reach a given destination. If the desired application is not on one of the available DIFs, then a new DIF can be created to enable the communication. In order to take full advantage of RINA environment, an intelligent mapping should be implemented in the DIF allocator to manage the DIFs creation and the inter-DIFs routing. Especially, in dynamic large-scale scenarios, dynamic creation and suppression of DIFs should be considered. The idea is to use the DIF allocator to create the needed DIFs ensuring the connectivity between communicating IPCPs. The creation of the DIFs will be done dynamically and on demand following RINA logic. This will support scalability as we optimize the use as well as the management of the DIFs.

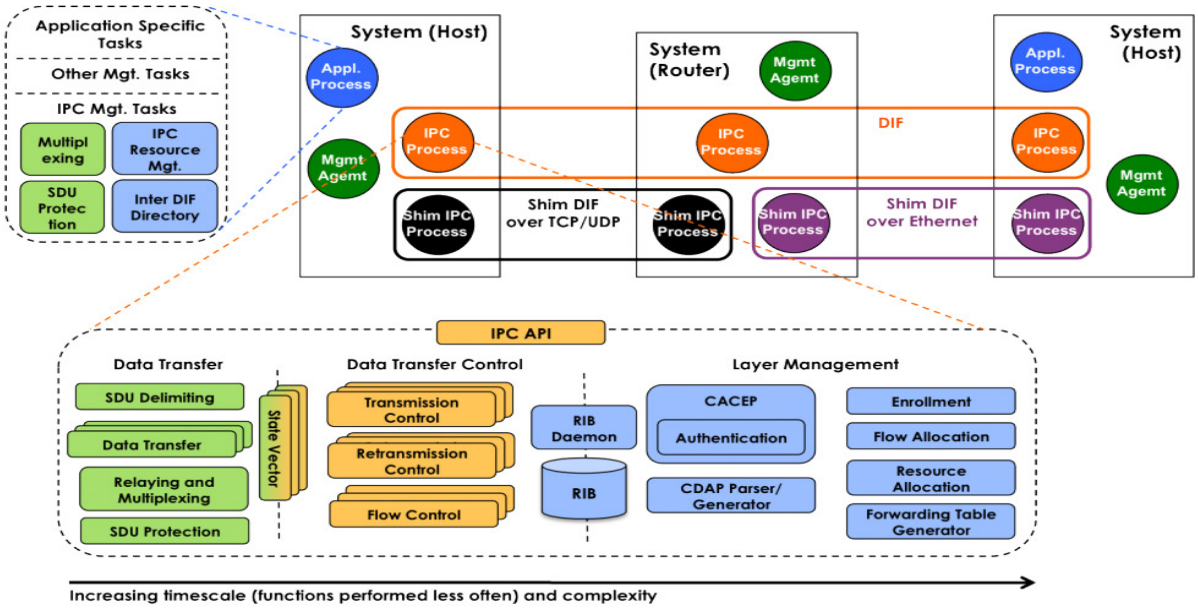


Fig. 2. RINA Reference Model [9].

As mentioned above, RINA adopts the saltzer's naming and addressing schema. It has a complete naming and addressing model, providing names for the basic entities of the network (applications, nodes and Point of Attachments). As a consequence RINA supports mobility and multi-homing inherently. Moreover, it is worth mentioning that in RINA there is no a global address space. Addresses are private and managed in the limited scope of the DIF which will solve the issues related to address scalability.

IV. Applying RINA to the VIFIB System

In this section, we investigate the application of RINA to the distributed clouds, and more specifically, the VIFIB system. In the scope of this paper, we will focus on the system architecture from 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 proposal for routing and addressing: Scalable Forwarding with RINA (SFR).

SFR: Scalable Forwarding with RINA

Naturally, in distributed cloud systems, 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 large scale environment. In this work, we propose to adopt the divide and conquer concept inspired from RINA in order to have a hierarchy of smaller clouds providing connectivity between the pairs of the system in an efficient way. The main idea of our proposal is to divide the clouds 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 *"Groups 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 3 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 where is the destination VIFIB node (F). To represent this scenario in RINA logic, we draw Figure 4. 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 overall VIFIB nodes that we called *"Group Leaders"* will have also 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.

In Figure 4, 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"*.

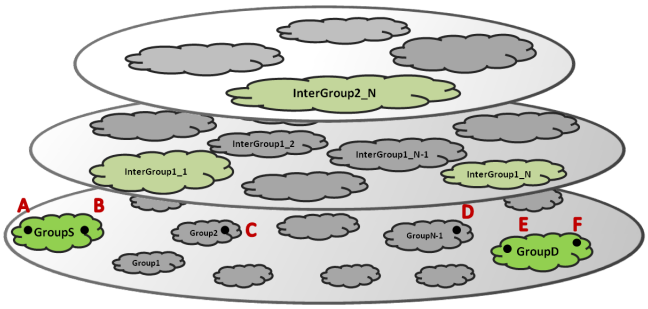


Fig. 3. Example of SFR hierarchy with three levels.

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. Tenant App-DIF is destined 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 node could act as Border routers and at the same time as customers application. The tenant Cloud DIF ensures scalability and flexibility as it maintains a global view of the network to manage dynamically 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 accordingly to manage connectivity inside the region.

To summarize, there are basically 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.

Dynamic Creation of Tenant Cloud DIFs

Some DIFs could be pre-configured to support the connectivity between customers that are frequently communicating. However, in order to support scalability, Tenant Cloud DIFs should be created on demand which means that when resources are requested to be allocated between source and destination, the DIF allocator will be in charge to build (if not existing already) the required Tenant cloud-DIFs to ensure the connectivity. If a customer's VIFIB node do not find a common DIF where it can see the destination, it should query peer DIFs which may have a DIF with the destination Application process. The path followed by the search request will be the sequence of the DIFs to use given by the DIF Allocator. In this way, we can efficiently manage the use of cloud DIFs as they are created/used on demand.

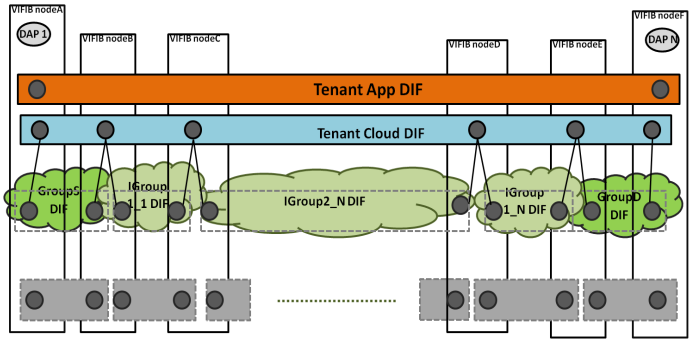


Fig. 4. DIF Architecture.

Routing Policies

The routing algorithm to be run in the DIFs will depend essentially on the built DIF hierarchy. In the Groups DIFs at the lowest level of the hierarchy, VIFIB nodes have to store routes leading to the *Group Leaders*. Consequently, traditional routing e.g. link state or distance vector could be used. Then, a Group Leader can determine the next hop based on topological addresses. Traditional routing might be used also inside the groups of the upper level layers of the hierarchy if needed. In the next section, we will investigate the address assignment aspect.

Topological Addressing

Figure 5 illustrates an example of topological address configuration that could be deployed for each layer. Basically, each layer has its own address space that is independent of the adjacent layer. However, the upper branches of the hierarchy may have a topological relation with lower layers which could simplify routing calculations. Addresses belonging to the same authorities or located in the same geographic place could be similar. Moreover, for each layer in the hierarchy more granularities should be provided. In the example in Figure 5, in the lowest level of the hierarchy the address is built from country prefix(FR, TN), ISP prefix(SFR, TNTEL) and number of nodes(150,160). As the scope becomes larger in upper levels, we lose more and more granularity in the address configuration i.e., the address space at the upper level in the example is built from only country prefix and node number and there we can find VIFIB nodes belonging to different countries where addresses start with FR, TN and so on. Following the whole address, the needed path could be found. Accordingly, the topological address defines the concept of *nearness*. This provides location dependence without route dependence [12].

V. Performance Evaluation

In this section, we evaluate the performance of our scheme for distributed clouds. We assess the benefit 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

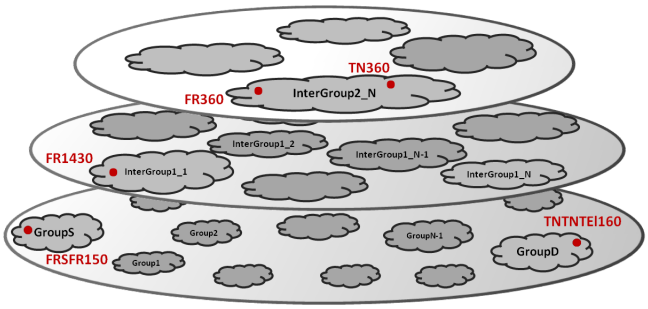


Fig. 5. Example of Topological Addressing Assignment.

System is using. In the following, we introduce the simulation setup and present the results of our experiments.

A. Simulation Scenario

We have conducted a set of experiments to analyse the performance of our proposal. We have used RINASim [10]. It is a simulation platform implementing RINA architecture in Omnet++ [14]. It is intended to enable the study of RINA architecture and also to perform simulation experiments with RINA applications. Figure 6 represents the scenario that we set up in RINASim. It consists in a medium size network of 120 nodes, divided into four regions, within each region 30 VIFIB nodes including the group leader. All the nodes inside the regions are interconnected randomly and connected to the group leader. All the *Group Leaders* are interconnected among each others. 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 that contains all the nodes that are communicating.

Figure 7 shows how the configuration of routing is performed for each DIF. In RINASim, several policies have been implemented in order to handle routing within RINA networks. For example, in Figure 7, "SimpleDV", a distance vector routing policy, is used in the scope of each DIF. Table I summarizes the different configuration parameters used for running the simulations. In this scenario, we consider that VIFIB nodes use a ping application to communicate where the maximum packets size is set to 1500 bytes.

Parameter	Value
Number of VIFIB nodes	120
Number of regions	4
Number of VIFIB nodes per region	30
Application	Ping
Packet size	1500 Bytes
Ping Starts at	140s
Ping rate	5
Simulation Time	300s for each run

TABLE I. SIMULATION PARAMETERS CONFIGURATION.

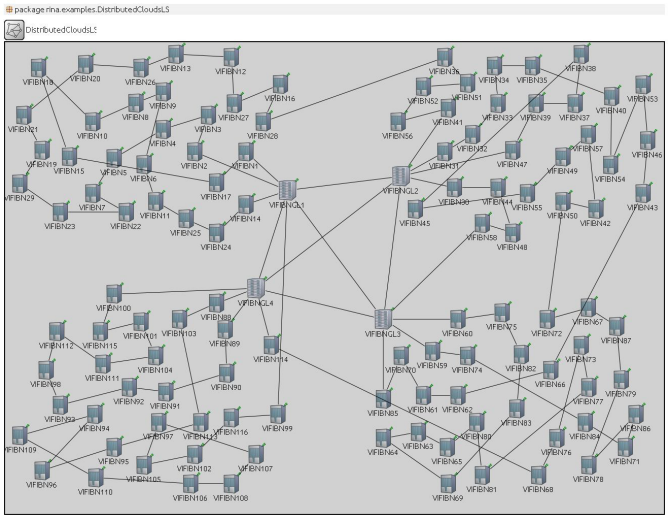


Fig. 6. Distributed Clouds simulation scenario.

```
# RMT Forwarding policies
**.VIFIBN*.TenantIPC.relayAndMux.ForwardingPolicyName = "SimpleTable"
**.VIFIBN*.GIPC.relayAndMux.ForwardingPolicyName = "SimpleTable"

**.VIFIBNGL*.IGIPC[*].relayAndMux.ForwardingPolicyName = "SimpleTable"

# forwarding generator policies
**.VIFIBN*.TenantIPC.resourceAllocator.pdufgPolicyName = "SimpleGenerator"
**.VIFIBN*.GIPC.resourceAllocator.pdufgPolicyName = "SimpleGenerator"

**.VIFIBNGL*.IGIPC[*].resourceAllocator.pdufgPolicyName = "SimpleGenerator"

# Routing policies
**.VIFIBN*.TenantIPC.routingPolicyName = "SimpleDV"
**.VIFIBN*.GIPC.routingPolicyName = "SimpleDV"

**.VIFIBNGL*.IGIPC[*].routingPolicyName = "SimpleDV"
```

Fig. 7. Routing configuration on RINASim for distributed clouds.

B. Simulation Results

In this section, we demonstrate the assets of applying RINA to distributed clouds and present the results obtained by simulations. Figure 8 illustrates the PDU forwarding table size with regards to the simulation time. Basically, 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 [8]. 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 intergroups DIFs. We can see that in case of Distance vector protocol, the PDU forwarding table size goes to around 120 entries corresponding to basically the whole network size. This is mainly due to the beneficial impact of the use of RINA in the forwarding scheme and specially its *"divide and conquer"* strategy that helped to bound the routing table size.

We have assessed also the dynamic creation of tenant cloud DIFs. Figure 9 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 the simulation time. We can see from this figure that

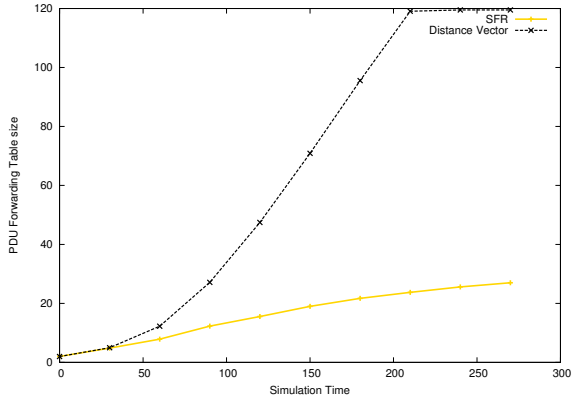


Fig. 8. The variation of the PDU forwarding table size with regards the simulation time. A comparison between SFR and simple distance vector routing protocol.

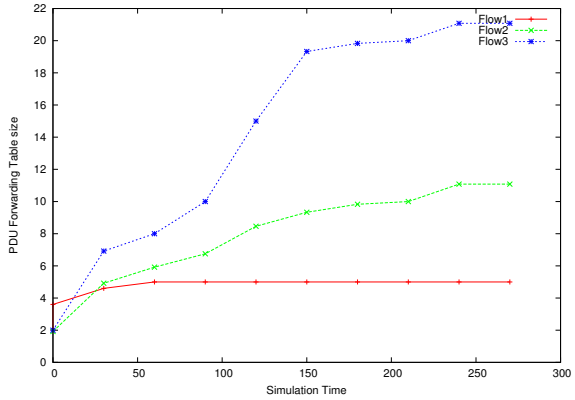


Fig. 9. The variation of the PDU forwarding table size with regards the simulation time. Dynamic Tenant Cloud DIF management.

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. Red line in Figure 9 represents the results for a flow between only five nodes, so we can observe that at the end of the simulation the entries of the forwarding table of each node is filled with only five entries. So, only nodes actively communicating 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 managing dynamically the tenant DIFs ensures more limitation to the forwarding table size and thus, more flexibility and scalability.

VI. Conclusion

In this paper, we presented a new and generic architecture for routing and addressing tailored to cope with the distributed clouds requirements most notably in terms of scalability, reliability and efficiency. We have identified the limitations and issues of the current implemented solutions for distributed clouds. We have investigated the case of a real distributed cloud use case (VIFIB). We then described SFR, our generic networking architecture which has been designed to benefit from RINAs recursive architecture features. SFR assumes that all nodes in the distributed clouds are located in an overlay. SFR then builds a hierarchical DIF architecture from the overlay to manage efficiently the forwarding in the network. The obtained simulation results showed that SFR achieves its design goal by limiting the routing table size compared to the simple distance vector routing protocol used currently by the VIFIB system. An aspect that we further investigate is the dynamic behaviour of SFR, we showed how it adapts to scalability as DIFs in the architecture are created on demand.

In future works, we plan to further study our SFR architecture and evaluate the assets of using RINA in larger distributed clouds scenarios. Moreover, we intend to consider other performance evaluation metrics (i.e. latency and throughput) to efficiently assess our proposal and demonstrate its applicability to distributed clouds services. Moreover, based on the evaluation results, we intend to implement SFR in the RINA Linux Software Development Kit being developed currently by the FP7 PRISTINE project[10] which is based on the prototype implemented within the FP7 IRATI project [15], [16]. We also plan to deploy our solution within the VIFIB infrastructure in order to compare it with the current re6st overlay.

Acknowledgment

This research is partly funded by the European Commission through the PRISTINE project (Grant 619305), part of the Future Networks objective of the Seventh Framework Programme (FP7).

References

[1] "Amazon elastic compute cloud (ec2)," http://www.amazon.com/gp/browse.html?node=201590011, accessed: 2015-07-10.

[2] "Google cloud comuting," http://www.googlecloud.com/, accessed: 2015-07-10.

[3] "Final version of nist cloud computing definition published," http://www.nist.gov/itl/csd/cloud-102511.cfm, accessed: 2015-07-10.

[4] "Vifib web page," http://www.vifib.com/, accessed: 2015-07-11.

[5] "Slapos community," http://community.slapos.org/wiki/osoe-Lecture.SlapOS.Extended, accessed: 2015-07-11.

[6] "Nexedi re6st resilient overlay mesh network," https://www.erp5.com/NXD-re6st.Two.Page, accessed: 2015-08-20.

[7] U. Beaugnon, "Building a resilient overlay network: Re6stnet," *Internship at Nexedi*, 2012.

[8] 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.

[9] J. Day, "Patterns in network architecture: A return to fundamentals," *Prentice Hall*, vol. ISBN 978-0-13-225242-3, 2007.

[10] "Fp7 pristine project website," http://ict-pristine.eu, accessed: 2015-08-20.

[11] I. M. J. Day and K. Mattar., "Networking is ipc: A guiding principle to a better internet," *CoNEXT'08: Proceedings of the 2008 ACM CoNEXT Conference*, vol. pages 16, New York, NY, USA, 2008.

[12] J. Saltzer, "On the naming and binding of network destinations," *RFC 1498 (Informational)*, 1993.

[13] J. D. E. Trouva, E. Grasa and S. Bunch, "Layer discovery in rina networks," *IEEE CAMAD*, 2012.

[14] "Omnet web page," http://www.omnetpp.org, accessed: 2015-07-11.

[15] "Fp7 irati project website," http://irati.eu, accessed: 2015-08-20.

[16] "Fp7 irati rina implementation," http://irati.github.io/stack, accessed: 2015-08-20.

Benefits of Programmable Topological Routing Policies in RINA-enabled Large-scale Datacenters

Sergio Leon, Jordi Perelló, Davide Careglio
Universitat Politècnica de Catalunya (UPC)
Barcelona (Spain)
Email: slgaixas@ac.upc.edu

Eduard Grasa
Fundació Privada i2CAT
Barcelona (Spain)
Email: eduard.grasa@i2cat.net

Diego R. López, Pedro A. Aranda
Telefónica I+D
Madrid (Spain)
Email: diego.r.lopez@telefonica.com

***Abstract*— With the proliferation of cloud computing and the expected requirements of future Internet of Things (IoT) and 5G network scenarios, more efficient and scalable Data Centers (DCs) will be required, offering very large pools of computational resources and storage capacity cost-effectively. Looking at todays' commercial DCs, they tend to rely on well-defined leaf-spine Data Center Network (DCN) topologies that not only offer low latency and high bisectional bandwidth, but also enhanced reliability against multiple failures. However, routing and forwarding solutions in such DCNs are typically based on IP, thus suffering from its limited routing scalability. In this work, we quantitatively evaluate the benefits that the Recursive InterNetwork Architecture (RINA) can bring into commercial DCNs. To this goal, we propose rule-based topological routing and forwarding policies tailored to the characteristics of publicly available Google's and Facebook's DCNs. These policies can be programmed in a RINA-enabled environment, enabling fast forwarding decisions in most scenarios with merely neighboring node information. Upon DCN failures, invalid forwarding rules are overwritten by exceptions. Numerical results show that the scalability of our proposal depends on the number of concurrent failures in the DCN rather than its size (e.g., number of nodes/links), dramatically reducing the total amount of routing and forwarding information to be stored at nodes. Furthermore, as routing information is only disseminated upon failures across the DCN, the associated communication cost of our proposals largely outperforms that of the traditional IP-based solutions.**

Keywords— Data center network; RINA; topological routing

I. INTRODUCTION

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 the Google's and Facebook's DCN topologies available in [1] and [2], respectively Moving toward 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, routing and forwarding solutions in DCNs, typically based on TCP/IP, do not scale well, resulting in large forwarding table (at least in the order of several tens of thousands of entries in highly-optimized configurations [3]), routing burden and communication cost (information exchanged to populate routing tables and re-converge upon failures). This problem was identified longtime ago, but the TCP/IP protocol suite, not designed for cloud networking, limited the improvements that achievable by the solutions proposed in the literature [4].

In contrast to the rigidness of the TCP/IP protocol stack, the clean-slate Recursive Internetwork Architecture (RINA) [5] brings a programmable environment [6] where Quality of Service (QoS), security, 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, outperforming 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.

This work aims to quantify the benefits that topological routing and forwarding policies can offer in a RINA-enabled large-scale DCN. Our policies make use of the DCN topology knowledge to forward packets to the closest neighboring device to their destination based on rules. In the non-failure scenario, this approach only requires the storage of forwarding information per adjacent neighbor (compared to traditional forwarding tables, which may contain up to one entry per network node). Upon failures in the DCN, some forwarding rules may not succeed to deliver packets to destination. In this case, few exceptions overriding those rules are stored at forwarding devices, the only time when additional forwarding information is required.

The remainder of this paper continues as follows. Routing solutions for DCNs available in the literature are reviewed in section II. Next, in section III we introduce the assumed RINA-enabled DC scenario based on Google's and Facebook's DCNs characteristics. The configuration of the routing and forwarding policies in both use cases are further elaborated in Sections VI and V. In Section VI, we provide numerical results comparing our forwarding and routing

This work is partly funded by the European Commission through the FP7 PRISTINE project (FP7-619305). Moreover, it has been partly funded by the Spanish project SUNSET (TEC2014-59583) that receives funding from FEDER.

policies against current solutions based on TCP/IP. Finally, Section VII draws up some conclusions.

II. Related Work

Given the regular and known topology of a DCN, deterministic routing [7] was the scheme initially deployed in many DCs. In such a scheme, the addresses of nodes are based on their topological properties, so that the route between any pair of nodes is known beforehand and does not change over time. The route is usually encoded in the packets in the form of a bit-stream or coordinates (e.g., see [8]). While scalable, this rigid scheme has two major drawbacks: the lack of automation in defining addresses, and thus the setup of routes, and no multipath support, preventing the recovery upon DCN failures. The valiant routing scheme [9] was proposed as a solution to overcome such deterministic routing shortcomings, bringing multipath support and load balancing. For a communication between any pair of nodes, a random intermediate address *i* is selected first and the path is composed by routing packets from source to *i* and then from *i* to destination. This way, multipath support is enabled, but at expenses of longer paths and still an automation process for the naming.

The adopted routing scheme in many large DCs is today based on IP because of the low-cost of IP-based commodity servers. To mitigate the inherent limitations of routing solutions initially designed for an Internet with arbitrary topology, modifications to link-state and path-vector routing have been introduced. For example, Facebook's DCN uses BGP-4 [10] to avoid the need for an address per interface (as required by IP), assigning an ASN per node, routing to the node instead of to the interface [11]. Nonetheless, BGP-4 suffers from many limitations, e.g., path exploration upon failures, manual configuration of timers, TCP connections between any pair of connected ASNs, etc. As a result, these schemes imply a high communication cost and require many entries in routing and forwarding tables to take optimal routing decisions and allow route recovery upon failures.

A new trend in intra-DC routing is a Software Defined Networking (SDN) approach centralizing all forwarding decisions, where only a few nodes know the state of the full DCN. For example, Google's DCN uses its SDN-based approach to control packet forwarding within the DCN [1]. Although this strategy allows taking efficient decisions at low communication cost, the complexity of centralized decisions increases with the network size, potentially imposing scalability issues as the network grows larger.

III. Scenario under study

RINA is a computer network architecture that unifies distributed computing and telecommunications [5]. RINA's fundamental principle is that computer networking is Inter-Process Communication (IPC). RINA reconstructs the overall structure of the Internet forming a model that comprises a single repeating layer, the DIF (Distributed IPC Facility). Each DIF instance implements the same functions and mechanisms, which are configured via policies in order to adapt to the specific scope (operating environment). In this paper we focus on a RINA deployment inside a DC following the DIF setup depicted in Fig. 1. Such a RINA-enabled DCN network is partitioned into three main types of DIFs of different scopes: 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. Note that in the figure the underlying point-to-point links are abstracted as "shim" DIFs, which allow the deployment of RINA over legacy technologies or physical media [12].

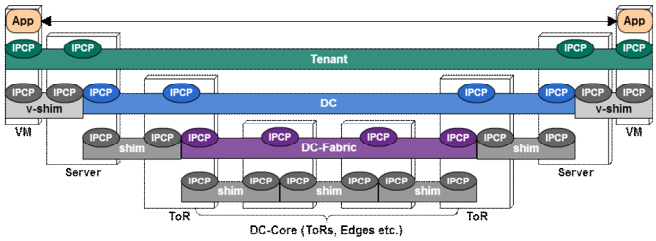


Fig. 1. DIF setup inside a DC between Virtual Machines (VMs) running in DC servers. The DC-Fabric DIF (violet color) is the focus of this work.

In this work, we focus on the DC-Fabric DIF, that is, the one providing connectivity between Top-of-the-Rack (ToR) switches and between edge routers and ToR switches. To provide outcomes applicable to realistic DC scenarios, we assume that the DC-Fabric DIF follows the topologies of the large Google's and Facebook's DCNs shown in Fig. 2. As for the Google's DCN topology (Fig. 2, top), a unique plane of spine switches interconnects all pods and edge planes in the DCN, offering multiple equal cost paths between each pair of ToRs and edges, even under multi-failure scenarios. Regarding the Facebook's DCN (Fig. 2, bottom), the fabric switches of a pod connect each one to a distinct spine set that provides connectivity to all other pods and to edge nodes. Again, high redundancy is introduced to survive multiple concurrent failures across the DCN.

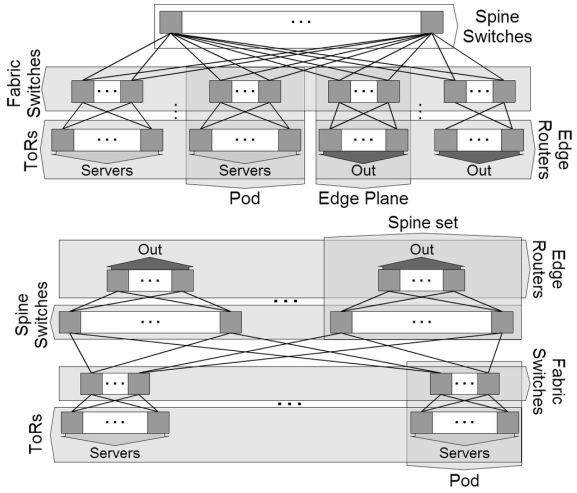


Fig. 2. Google's (top) and Facebook's (bottom) DCN topologies, extracted from references [1] and [2].

A key benefit of RINA is its programmable behavior using policies, as well as the automation of the enrollment and naming processes of a new node in a DIF [6]. This allows to get rid of the constraints imposed by both the conventional deterministic routing and the IP-based solutions, opening an opportunity for cheap customizable hardware. In the following section we elaborate on topological routing and forwarding

policies particularly designed for a potential DC-Fabric DIF in the Google's DC premises, exploiting its DCN characteristics for superior efficiency and scalability.

IV. RINA-ENABLED GOOGLE'S DCN USE CASE

Taking a look at Google's DCN, we can see that the DC-Fabric DIF can be described by 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). Moreover, given the regularity of the topology, we find that only a few types of nodes exist. This becomes particularly useful for performing topological forwarding, which depends on the relations between node locations. Specifically, we have spine switches (hereafter referred as R0 nodes), fabric switches (R1 nodes) and ToR switches/edge routers (R2 nodes).

From now on, we are going to refer to pods and planes (of spine or edge switches) indistinctly as groups. Taking advantage of R0, R1 and R2 node groups in the DCN, we propose a location-dependent but route-independent node-addressing scheme A.B, where A identifies a group and B is the identifier of the node within the group. Although simple, this scheme allows inserting the topological location of any node in the DCN in its address as follows:

A = 0, B ∈ [1, s] → Spine B
A ∈ [1, P+E], B ∈ [1, f] → Fabric B at pod/Edge Plane A
A ∈ [1, P], B ∈ (f, f+t] → ToR B at pod A
A ∈ (P, P+E], B ∈ (f, f+e] → Edge B at edge Plane A

A. Forwarding policy

A key requirement of any forwarding policy is the ability to quickly decide the neighboring node to which a packet must be forwarded to. Forwarding policies in RINA are not restricted to be a traditional table. Instead, any forwarding function capable to quickly perform accurate forwarding decisions can be used. To this avail, we leverage on the regular topology of Google's DCN (Fig. 2, top) to design a minimalistic forwarding function.

Being aware of the specific DCN topology (from the set of parameters listed at the beginning of this section) and the location of the node in it, only forwarding entries to adjacent neighbors need to be stored at DCN nodes and simple forwarding rules can be used (detailed in the next sub-section). When failures occur across the DCN, however, it may happen that primary forwarding rules fail in delivering a packet to its destination. Thus, we require exceptions to overwrite the erroneous decisions of primary rules. These exceptions are similar to traditional forwarding table entries, but are only required upon certain failure scenarios. Moreover, the total number of exceptions tends to be considerably smaller than the number of entries required in a traditional forwarding table (at most the same in the very worst case), as many communications across the DCN remain unaffected by specific link or node failures. Therefore, only storing exceptions to primary rules upon failures can yield a large reduction in terms of memory usage compared to a traditional forwarding table.

1) Forwarding rules

In order to quickly access neighbor information, we assign a locally unique identifier (Neighbor-Id) to every neighbor, abstracting its real address. By using Neighbor-Ids as index, we can store all neighbor nodes' information, including port address or status, in a direct access structure. These Neighbor-Ids are assigned as follows:

At **R0**: R1 → (A - 1) * f + B - 1
At **R1**: R0 → B - 1, R2 → s + B - f – 1
At **R2:** R1 → B - 1

Forwarding rules use Neighbor-Ids to easily define the set of valid neighbors 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 R2 nodes (ToR switches and edge routers) will be established. Therefore, forwarding rules only need to consider R2 nodes as possible destinations. These rules are depicted in Fig. 3.

At **R0: Rule (A.B)** → [(A – 1) * f, A * f)
At **R1: Rule (A.B)** → [0, s)
At **R2: Rule (A.B)** → [0, f)

Fig. 3. Pseudo-code of primary forwarding rules in the Google's DCN

Let us show how those rules work to reach any R2 node A.B. At R0 nodes, any neighbor R1 node of the group A can be used to reach the destination (an ECMP-like policy can be chosen to load-balance the traffic). In an R1 node, we have two possibilities: either we are in a different group than the destination, so that we can use any R0 node to reach that (again load-balancing can be used), or we are in the same group (A), never reaching the rule as it is a direct neighbor. Finally, at R2 nodes we can use any R1 neighbor to reach any other R2 in the network.

In all cases we have a range of valid neighbors. Then, upon having an unreachable neighbor, the rules would simply remove it from the valid ones. This allows keeping most of the rules still valid when failures affect the current node (otherwise multiple exceptions could be necessary).

2) Forwarding exceptions

Focusing on the exceptions to reach R2 nodes, we find 3 kinds of them: to a specific node A.B, to a specific group A and to all other groups. It should be noted that exceptions to other groups are used neither in R0 nodes nor in R1 or R2 nodes for destinations in the same group.

Given the high number of neighbors that some nodes have, an important point is how Neighbor-Ids are stored at the exception entries. When encoding exceptions, we use two different encoding modes, being the use of one or another specified as a flag in the exception header. With the default encoding, the stored Neighbor-Ids represent the valid neighbors to reach the destination. When the number of valid neighbors is high, an inverse encoding can alternatively be used, where the stored Neighbor-Ids represent the invalid neighbors to reach a destination. Jointly with inverse encoding at R1 nodes, a direction flag is used to specify if that list applies to only R0 nodes (UP), R2 nodes (DOWN) or both. This proposal yields significant memory optimization, as most exceptions can be described as "To reach X go UP/DOWN, without using Y".

3) Forwarding decision

Putting together direct routes to neighbors, exceptions and rules, the full forwarding decision can be described by the simple pseudo-code in Fig.4.

```
Forward (A.B)
If is Connected Neighbor (A.B) → Forward (A.B)
If A = 0 || B <= f  → Unreachable
If is Exception (A.B)  → Exception (A.B)
If is Exception (A)  → Exception (A)
If (My A != A & is Exception ())  → Exception ()
Else  → Rule (A.B)
```

Fig. 4. Forwarding pseudo-code with exceptions and primary rules

The forwarding function needs to be executed per packet. DCN performance requirement will most probably impose forwarding rules implementation to be on hardware. For this, we first have that neighbors can be stored in a direct access structure, making these last hops automatic. Then, taking profit from the small number of exceptions, we can have them ordered as R2 nodes, specific groups or other groups and simply iterate them until finding the first match. Being desirable for flows to maintain the same path during its lifetime (to prevent packet reordering), both rule and exception execution can use a fast hashing of the flow identifier of the packet to decide on the next hop, instead of deciding it randomly.

B. Routing policy

The previously described forwarding policy requires knowing the affected routes to destinations upon failures 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 obtain exceptions to the primary rules upon 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, but only that of those experiencing failures. To this end, we propose the link-failure routing policy, a variation of link-state routing based on failure propagation, where instead of having all nodes propagating their full neighbor table, only failed links are propagated while the rest is assumed to be working, resulting in a large reduction of 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, if some constraints on valid paths are considered. Constraints on valid paths are, in fact, required to reduce the complexity of the algorithms. Even so, those are thought taking into account the high number of available paths towards any R2 node, and that it is better to have unreachable destinations (with possible movements of VMs) than filling the network with traffic routed through sub-optimal paths. For example, we consider the following two constraints at R0 nodes: 1) a group is reachable if it has at least one R1 neighbor connected to at least one R2; 2) an R2 node is reachable if it has at least one R1 neighbor for which there exists a 1 or 3 hops path to reach it in the group.

C. Computing the forwarding exceptions

Given the list of possible failures in the DC-Fabric DIF, we parse and process them in order to compute the exceptions to problematic destinations. For example, for R0 nodes, the pseudo-code in Fig. 5 can be used.

```
Parsed data and functions used:
unreachableGroups ←groups with all R1 unreachable
unreachableNodes ←R2 disconnected from all R1
R2Fails ←R2 disconnected from some R1
R1notReachR2 ←R1 with problems reaching R2
Reachable (A.B) ← Check if neighbor A.B can be reached
reachableGroupR1(A) ← reachable A.* R1s
reachableR1At (A.B) ← reachable R1s from R2 (A.B)
reachableR2At(A.B) ← reachable R2s from R1 (A.B)
Algorithm:
Exceptions = Ø
if I am disconnected then return Exceptions
GroupsWithProblems = Ø
for each A.B in R1notReachR2 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 ∉ R1notReachR2 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 R2Fails do
  if A ∈ unreachableGroups then continue
  if (A.B) ∈ unreachableNodes then continue
  myReach = reachableGroupR1(A)
  dstReach = reachableGroupR1At(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(reachableR2At (A.B'))
  reachNei = Ø
  for each node (A.B') in myReach do
    reachNei .add(reachableR2At(A.B'))
  validPorts = Ø
  if reachNei ∩ reachDst != Ø then
    validPorts .add(A.B')
  Exceptions.add(A.B, validPorts )
return Exceptions
```

Fig. 5. Pseudo-code for computing exceptions

Algorithms to compute exceptions like the one described in Fig. 5 aim to direct the search toward failures that may

require an exception, while discarding the rest. An example can be seen when failures between R0 and R1 nodes are only considered for the current node, as other ones are not included in feasible paths given the imposed constraints. Another case is seen for failures between R1 and R2 nodes where the depth of the search depends on the specific failures within the group, avoiding for example a search in depth if there are some shared R1 nodes between the current node and the one affected by failures. 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 the traditional route computation, as we do not need to compute and store reachability information to all destinations in the network, but only to problematic ones.

V. RINA-ENABLED FACEBOOK'S DCN USE CASE

Looking at the Facebook's DCN in Fig. 2, we observe that it can be described by only 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). Although this DCN can be described by fewer parameters than the Google's one, here we have 4 types of distinct nodes: ToRs, fabric switches, spine switches and edge routers. Hence, we propose the following addressing scheme based on A.B.C addresses:

ToR: 0.Pod-Id.Tor-Id
Fabric: 1.Pod-Id.Spine-set
Spine: 2.Spine-set.Spine-Id
Edge: 3.Spine-set.Edge-Id

Like for the Google's DCN, we also propose a forwarding policy based on rules and exceptions. In this case, Neighbor-Ids are defined as follows:

At **ToR**: Fabric → C
At **Fabric switch**: ToR → s + C, Spine → C
At **Spine switch:** Fabric → A, Edge → P + C
At **Edge router:** Spine → C

ToR: Rule (A.B.C)
If A = 3 → {B}, Else → [0, f)
Fabric: Rule (A.B.C)
If A = 3 & B != My B → [s, s+t), Else → [0, s)
Spine: Rule (A.B.C)
If A = 0 → {P + C}, Else → [0, P)
Edge: Rule (A.B.C) → [0, s)

Fig. 6. Pseudo-code of primary forwarding rules in the Facebook's DCN

Fig. 6 details the pseudo-code of the primary forwarding rules proposed for the Facebook's DCN. With information of the current failures, exceptions to overwrite those rules can be computed in a similar way as in the Google's DCN (not detailed here due to the lack of space). Note in this case that losing a fabric switch automatically multiplies the path length to reach its pod from some edge routers. Therefore, when computing the respective exceptions, we require either to have less restrictive constraints in these cases, thus incrementing the complexity of the algorithms to avoid unreachable areas across the DCN, or more restrictive ones, reducing complexity but at expenses of having unreachable areas.

VI. COMPARISON WITH CURRENT SOLUTIONS

Current routing and forwarding solutions for IP impose multiple limitations that the RINA architecture already solves. For example, for addressing DCN devices, we are not forced to use 4 or 16-byte addresses as imposed by IPv4 or IPv6, but can use scenario-specific addresses. Besides, public addresses of servers/VMs are not propagated with routing updates, only focusing on the smaller set of node addresses in the DCN. While the benefits of RINA are enough to contemplate its usage inside DCs, we also want to quantitatively evaluate the performance of the proposed routing and forwarding policies against that of currently available solutions for the same purposes.

Aiming to analyze the number and size of traditional Forwarding Table entries vs. Rules plus exceptions in our policies, we have considered two different DC-Fabric DIFs. The first one, named DIF-Go, reproduces the Google's DCN topology, whereas the second one, named DIF-FB, reproduces that in the Facebook's DCs. Table I depict the parametrization of both DIFs, taking the number of pods (P) as 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 I. DETAILS OF THE DCN-FABRIC DIFS

Pods (P)		P
ToRs per pod (t)		P/2
Fabric switches per pod/edge plane (f)		$Log_3(P)$
DIF-Go	Edge planes (E)	P/4
	Edge routers per edge plane (e)	P/2
	Spine switches (s)	P
DIF-FB	Edge routers per spine set (e)	$P^2/8f$
	Spine switches per spine set (s)	P/2
Total number of servers		**$P^2/2$**
Total number of edges		**$P^2/8$**

As a first objective, we compare the number of entries in a forwarding table against the number of neighbor entries plus exceptions for large-scale DCNs. For this, we fix in our first tests P=100, resulting in DCNs with 5000 ToR switches. Being our approach dependent of the number of concurrent failures across the DCN, we perform our tests for 0, 1, 2, 5 and 10 concurrent ones, being those either link or node failures (randomly chosen). We perform 50000 tests for each DIF and number of failures, averaging the obtained results.

TABLE II. 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-Go FWT	11.6	11.9	12.3	13.2	14.8
DIF-FB FWT	11.4	12.1	12.8	14.8	18.1

In Table II 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 being most of the stored entries 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. With this, the number of entries can be lowered by 11 to 18% compared to the trivial FWT solution where one entry per destination node is stored. While

this represents a noticeable reduction, it cannot approach by far the scalability of our proposals.

In addition to the number of entries, we are also interested in comparing the amount of forwarding data stored. For this purpose, we focus on the amount of stored ports in the forwarding entries and exceptions rather than on their encoding, so as to become independent from specific data structures. Fig. 7 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 scenarios from 0 to 10 concurrent link/node failures. With forwarding tables, we also limit the number of stored ports per destination to 16, a common limit in ECMP implementations. In the figure, 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 neighbors' information plus exceptions, remaining the number of forwarding entries almost constant as the size of the DIFs grows up.

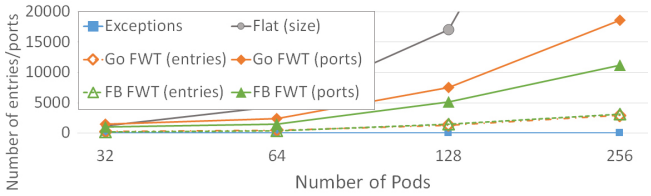


Fig. 7. Avg. number of entries and stored ports given the number of pods

Although the benefits of our proposals against the traditional forwarding tables are good enough to justify topology dependent policies, the benefits on the computational and communication cost of searching exceptions, given a specific number of failures, should also be investigated. Regarding the latter, we have a clear improvement in the sense that, as we know the topology, we can take some knowledge as granted. Given this knowledge of the topology, we can avoid the initial routing information flooding that any common on link-state or distance-vector routing protocol needs for populating nodes' routing and forwarding tables. Besides, most IP solutions require some type of refresh of routing information to ensure that the knowledge is updated. In our case, RINA DIFs can provide reliable communications between nodes, which can be configured in a DC-Fabric DIF as well. This makes routing information refreshes unnecessary.

Finally, in terms of computation cost, in order to validate our proposed approach to compute forwarding exceptions, we compare its complexity against a traditional solution based on link-state routing and Dijkstra's routing algorithm. For this purpose, we take the pseudo-code proposed for R0 nodes in the Google DCN in Fig. 5. Moreover, to simplify the results we consider the same parametrization described in Table I, and use the number of pods (P) and failures R as the two only parameters. With the traditional link-state solution, computing either exceptions or a forwarding table has a complexity lower bound of $o(P^2*Log(P))$. Conversely, with our approach we find a complexity upper bound of $O(R*P)$. Note that this upper bound will never be reached, as it would require the same failures to affect R2 nodes in all groups. Since the number of concurrent failures in this type of networks is small by design, we found a lower upper bound, ensured for the cases where R < P/2. In these cases, if we use the known failures to check reachability between R1 nodes instead of a brute force approach, our complexity can be bounded to $O(R*Log(R)*Log(P))$, representing a big improvement in performance, still without considering that in the non-failure scenario we only have the constant cost of checking that there are no failures in the network.

VII. Conclusions

In this paper, we proposed rule-based topological routing and forwarding policies for RINA-enabled large-scale DCNs, based on those recently made publicly available by Google and Facebook. These policies use the knowledge of the DCN topological characteristics for superior efficiency and scalability, achieving fast and 100% successful forwarding decisions in the non-failure scenario with merely neighboring node information. Upon DCN link or node failures, forwarding exceptions are computed and stored at DCN nodes to override the possible invalid forwarding decisions of primary rules. To minimize the size of the stored exceptions, only the invalid neighbors (instead of all valid ones) are contemplated, something that gives a great improvement, given the large number of redundant paths to any destination. Regarding the proposed routing policy, only failed link information is disseminated, reducing the communication cost to a large extent. The obtained results in large-scale DCNs illustrate the perfect scalability of our routing and forwarding policies.

References

[1] 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.

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

[3] 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.

[4] 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".

[5] J. Day, I. Matta, and K. Mattar. "Networking is IPC: A Guiding Principleto a Better Internet". In CoNEXT'08: Proceedings of the 2008 ACMCoNEXT Conference, pages 1 6, New York, NY, USA, 2008. ACM.

[6] V. Maffione, F. Salvestrini, E. Grasa, et al. "A Software Development Kit to exploit RINA programmability". In IEEE ICC 2016, Kuala Lumpur, May 2016.

[7] M.E. Gómez, P. López, J. Duato. "A Memory-Effective Routing Strategy for Regular Interconnection Networks". In IPDPS'05 conference, 2005.

[8] S. Habib, F. S. Bokhari, and S. U. Khan, "Routing Techniques in Data Center Networks," in Handbook on Data Centers, S. U. Khan and A. Y. Zomaya, Eds., Springer-Verlag, New York, USA, 2015, ISBN 978-1-4939-2091-4, Chapter 16.

[9] K. Chen, et al. "Survey on routing in data centers: insights and future directions", IEEE Network, vol. 25, no. 4, pp. 6-10, July 2011.

[10] Y. Rekhter, T. Li, S. Hares. "A Border Gateway Protocol 4 (BGP-4)". RFC 4271, January 2006.

[11] P. Lapukhov, A. Premji, J. Mitchell. "Use of BGP for routing in large-scale data centers". IETF Network Working Group, draft-lapukhov-bgp-routing-large-dc-07, February 2014.

[12] S. Vrijders, E. Trouva, J. Day, E. Grasa, et al. "Unreliable IPC in Ethernet: migrating to RINA with the shim DIF". ICUMT 2013, Almaty.