Telecommunication Systems
https://doi.org/10.1007/s11235-018-0489-2

CrossMark

End-user traffic policing for QoS assurance in polyservice RINA networks

Sergio Leon Gaixas[1] · Jordi Perelló[1] · Davide Careglio[1] · Eduard Grasa[2] · Miquel Tarzán[2]

© Springer Science+Business Media, LLC, part of Springer Nature 2018

Author Proof

Abstract

Looking at the ever-increasing amount of heterogeneous distributed applications supported on current data transport networks, it seems evident that best-effort packet delivery falls short to supply their actual needs. Multiple approaches to Quality of Service (QoS) differentiation have been proposed over the years, but their usage has always been hindered by the rigidness of the TCP/IP-based Internet model, which does not even allow for applications to express their QoS needs to the underlying network. In this context, the Recursive InterNetwork Architecture (RINA) has appeared as a clean-slate network architecture aiming to replace the current Internet based on TCP/IP. RINA provides a well-defined QoS support across layers, with standard means for layers to inform of the different QoS guarantees that they can support. Besides, applications and other processes can express their flow requirements, including different QoS-related measures, like delay and jitter, drop probability or average traffic usage. Greedy end-users, however, tend to request the highest quality for their flows, forcing providers to apply intelligent data rate limitation procedures at the edge of their networks. In this work, we propose a new rate limiting policy that, instead of enforcing limits on a per QoS class basis, imposes limits on several independent QoS dimensions. This offers a flexible traffic control to RINA network providers, while enabling end-users freely managing their leased resources. The performance of the proposed policy is assessed in an experimental RINA network test-bed and its performance compared against other policies, either RINA-specific or adopted from TCP/IP. Results show that the proposed policy achieves an effective traffic control for high QoS traffic classes, while also letting lower QoS classes to take profit of the capacity initially reserved for the former ones when available.

Keywords Internet · RINA · QoS · ΔQ · Traffic policing · Network management · End-user

1 Introduction

As networking environments evolve, the inherent limitations of the current TCP/IP protocol stack to cope with the increasing variety of communication requirements of heterogeneous distributed applications are clearer than ever [1]. TCP/IP not only lacks true Quality of Service (QoS) support, but also misses any standard way for applications to express their service requirements or expectancies. Thus, with unknown application requirements, network providers cannot differentiate flows traversing their networks effectively, being limited to guess application requirements based on manual inputs, port numbers and past information. In this regard, the Recursive InterNetwork Architecture (RINA) [2, 3] provides an enhanced medium for QoS-based solutions. Unlike most common solutions aiming to enhance the current TCP/IP model, RINA is a clean-slate recursive Internet architecture based on the idea of distributed Inter-Process Communication (IPC), which aims to progressively replace the current TCP/IP Internet model.

In contrast to the well-known TCP/IP and OSI stacks, RINA provides a recursive stack of layers, called Distributed IPC Facilities (DIFs), where each layer is defined by a networking domain, rather than a subset of networking functions (e.g., there is no network or transport layer as in the OSI stack, for example). In fact, all DIFs provide a complete set of networking functions (forwarding, scheduling, security, etc.), but each one's operation can be configured via programmable policies, allowing to deliver the best outcomes in

✉ Sergio Leon Gaixas
slgaixas@ac.upc.edu

1 Advanced Broadband Communications Center (CCABA), Universitat Politècnica de Catalunya (UPC), Jordi Girona 1-3, 08034 Barcelona, Spain

2 Fundació Privada i2CAT, Barcelona, Spain

Journal: 11235 MS: 0489 ☐ TYPESET ☐ DISK ☐ LE ☐ CP Disp.:2018/7/7 Pages: 13 Layout: Large

Uncorrected proof

Author Proof

its particular scope. In addition, by having the same type of layer at each level, RINA provides a consistent Application Programming Interface (API) across the stack.

In this work, we focus on the RINA's QoS model. Specifically, RINA bases all QoS-related functionalities on the definition of QoS Cubes, namely, quality assurances that a DIF can provide under normal operation. Applications are thus capable of requesting flows with specific QoS requirements that will then be mapped to the best suited QoS Cube. Moreover, thanks to the recursive structure and the consistent API, such QoS requirements can easily be shared among DIFs, while each DIF is responsible for ensuring that those are met for all flows or, at least, inform when they are unfeasible for a certain one.

In order to provide the best service to a diverse set of distributed applications, a key point to consider is how resources are shared between flows. In this regard, the information that QoS Cubes give on flow requirements facilitate the configuration of different scheduling policies in a RINA DIF. Specifically, QTAMux [4] is a scheduling policy based on the ΔQ framework [5–7] that takes great advantage of the QoS Cube information in RINA. It is a scheduling policy that provides differentiated flow treatment without excessively degrading those flows with the lowest QoS requirements (as opposed to what happens with the well-known weighted-fair queuing strategy, for example).

However, when applications are free to inform the network about their QoS needs, greedy users can hamper the sustainability of the solution [8]. Indeed, the scenario can end either as a best-effort scenario (all applications request the highest QoS to the network) or, even worse, as a scenario where respectful users receive poor network service because greedy ones are exceeding reasonable QoS demands. While RINA allows for a more dynamic and accurate service assurance than TCP/IP, it cannot deal with such greedy users by itself, unfortunately. So, limitations have to be imposed on their usage. In this work, we focus on the evaluation of the effects of outgoing traffic policing with QoS guarantees in overbooked networks, focusing on a typical Internet home-user scenario, and the limitations that an Internet Service Provider (ISP) can impose to its clients. We propose a new RINA scheduling policy based on the ΔQ framework that, instead of enforcing rigid per-flow or per-QoS class rate limitation, it offers enhanced flexibility by limiting the outgoing traffic simultaneously in various independent dimensions (e.g., urgency and cherish). As a result, the proposed policy offers explicit traffic control to RINA network providers, while allowing users to freely manage their leased resources.

The rest of the paper is organized as follows. Section 2 introduces the RINA Software Development Kit (SDK). Section 3 introduces the ΔQ framework and the existing QTAMux policy. Section 4 introduces the proposed rate limitation policy and its implementation within the RINA SDK. Section 5 provides experimental results in a home-user scenario. Finally, Sect. 6 concludes the paper.

2 Rina SDK

RINA is a clean-slate architecture for computer networking based on the idea that networking is distributed IPC and only IPC [9]. RINA presents a single type of layer, called DIF, which repeats as many times and levels as needed by the network designer (see Fig. 1). This contrasts with the TCP/IP model, where the different layers are designed to perform different functions (transport, networking, security, etc.). RINA defines the DIF as a programmable layer, capable of performing any of the functions needed to provide IPC services to applications or higher level DIFs, offering also a common API at each level. DIFs are composed of IPC Processes (IPCPs) running at each node. Those execute layer functions and communicate among them using the same two protocols: a data transfer protocol called Error and Flow Control Protocol (EFCP) and an object-oriented application protocol called Common Distributed Application Protocol (CDAP) that carries all the information exchanged by the DIF management tasks (usually known as control plane, in TCP/IP terms). Both EFCP and CDAP protocols can be adapted to the different requirements of each DIF via policies [10], namely, a set of variable behaviours that can customise the different mechanisms available in the two protocols. This programmability allows network administrators to configure each DIF with the policies that best adapt to its scope, operating environment and offered levels of service.

The implementation of the RINA architecture is in constant development and multiple projects have been progressively make it a reality (FP7 IRATI [11], FP7 PRISTINE [12], H2020 ACRFIRE [13], etc.). In this regard, the experimental tests conducted in this paper have employed the publicly available RINA implementation reported in references [14, 15], that is, a free software implementation of the full RINA stack for Linux systems. An overview of the software architecture of this implementation is presented in Fig. 2. As seen, it spans both kernel and user spaces, performing those tasks with the highest speed requirements (i.e., data transfer and part of the data transfer control tasks) in the kernel modules, while running all management and configuration tasks in user space, thus freeing resources to other tasks and facilitating their programmability.

As stated before, one of the key points of RINA is the configurability of DIF tasks via programmable policies. In this regard, the approach taken by the current RINA implementation is to treat policies as independent modules, compiled and loaded separately from the fixed part of the stack. The differ-

uncorrected proof

Journal: 11235 MS: 0489 TYPESET DISK LE CP Disp.:2018/7/7 Pages: 13 Layout: Large

Author Proof

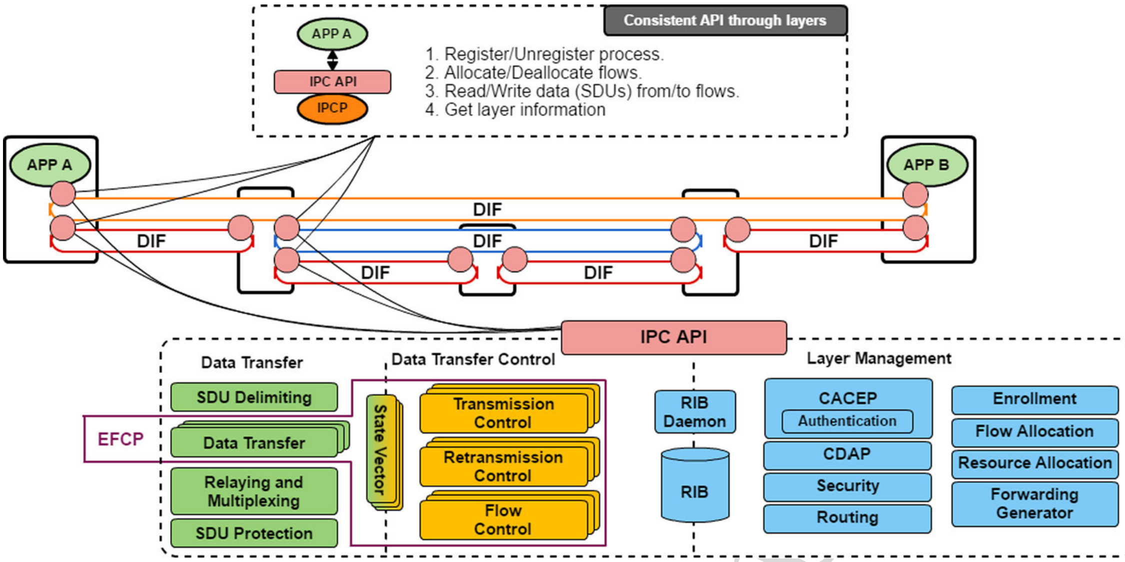

Fig. 1 RINA architecture overview

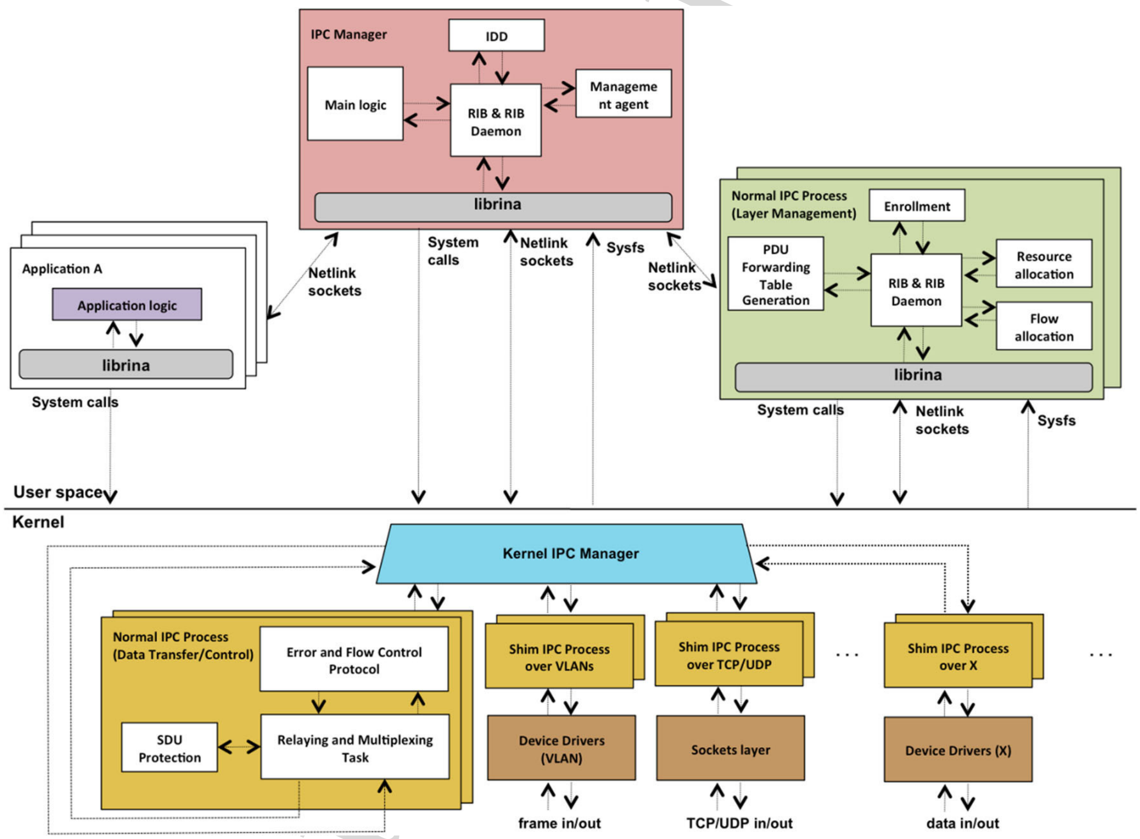

Fig. 2 Software architecture of IRATI's RINA implementation

ent tasks of the RINA stack and policies communicate among them by means of specific policy-hooks (i.e., for requesting specific actions from the policy). In addition, the Resource Information Base (RIB) is a distributed database used to store and share information between different modules and IPCPs in a DIF. Policies can also use private information and share it with compatible policies (e.g., scheduling queues are private data-structures to the scheduling related policies, but are seen as raw pointers to the task itself).

Journal: 11235 MS: 0489 TYPESET DISK LE CP Disp.:2018/7/7 Pages: 13 Layout: Large

Author Proof

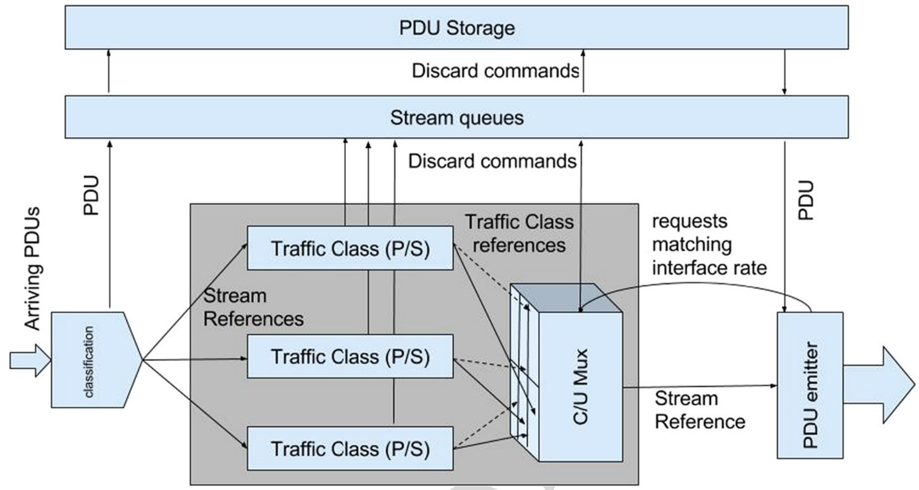

Fig. 3 QTAMux modules and workflow

Table 1 Example of 4×3 Cherish/Urgency Matrix

Cherish\urgency	Lossless	Max Cherish	Mid Cherish	Min Cherish
Max urgency	A1	A2	A3	A4
Mid urgency	B1	B2	B3	B4
Min urgency	C1	C2	C3	C4

3 ΔQ and QTAMux

Applications depend on information to complete computations, and distributed computation necessarily involves the translocation of information generated by one computational process to another located elsewhere, which essentially is IPC. Instantaneous and completely loss-less translocation is physically impossible, even within the same machine, since any translocation experiences some relative impairment to this ideal. The Degradation of Quality (referred to as ΔQ) is a measure of this impairment, having several sources, including the time for signals to travel between distant points ($\Delta Q|G$) and the time taken to serialise/de-serialise information ($\Delta Q|S$). In packet-based networks, statistical multiplexing is an additional source of impairment ($\Delta Q|V$), in which quality impairment (loss and delay, as measured by ΔQ) is conserved. Thus, in packet networks, ΔQ is an inherently statistical measure of the statistical properties of independent packets and streams of such packets, capturing both the effects of the network's structure and extent and the impairment due to statistical multiplexing.

Whether an application delivers fit-for-purpose outcomes depends entirely on the magnitude of ΔQ and the application's sensitivity to it. What applications require is for the network to translocate the amount of information that they need to exchange with an impairment no greater than what they can tolerate. A formal representation of such a requirement is called a "Quantitative Timeliness Agreement" (QTA), providing a way for an application and a network to negotiate performance. In RINA, this translates from the QoS Cubes into a contract, in which the application agrees to limit its load in return for a promise from the network to transport it with suitably ΔQ. This idea is embodied in the design of QTAMux (Fig. 3).

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 NxM matrix with relative latency and losses at each edge. An example of 4×3 C/U Matrix is shown in Table 1. This has a straightforward implementation, called a Cherish/Urgency multiplexor (C/U Mux) [16], included within the QTAMux design. Just as the total delay is conserved under scheduling [17], ΔQ is conserved in C/U multiplexing. 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. In order to do that, the C/U Mux maintains a priority queue of queue references (P/S queues), instead of moving PDUs from one queue to another.

While the C/U Mux provides an effective inter-flow contention of resources, this is not enough to ensure a suitable ΔQ for any flow, but only a 2-dimensional priority order between flows. In this regard, in order to provide more precise QoS assurances, the QTAMux employs multiple Policer/Shaper (P/S) sub-modules to manage intra-flow contention, as well as to decide the specific cherish and urgency level of each queue reference processed in the C/U Mux. With the use of P/Ss, the QTAMux is not limited to place

uncorrected proof

Journal: **11235** MS: **0489** ☐ TYPESET ☐ DISK ☐ LE ☐ CP Disp.:**2018/7/7** Pages: **13** Layout: **Large**

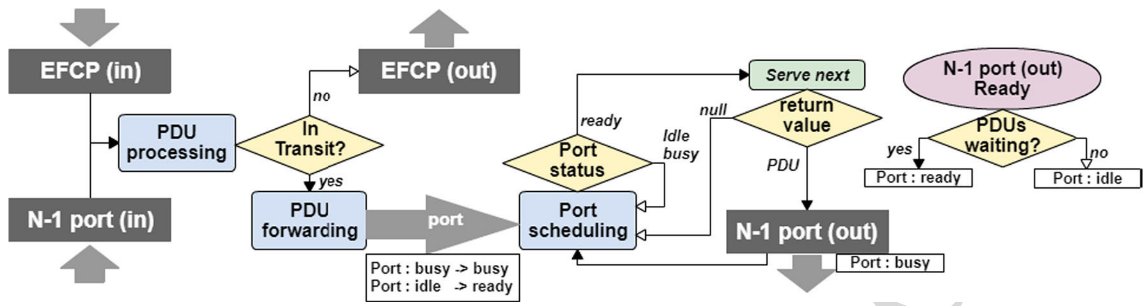

Fig. 4 Workflow of RINA RMT task

Author Proof

each QoS Cube within a fixed cell of the C/U Matrix, but may vary the placement of the queue references based on the current network usage or QoS requirements (e.g., two similar QoS Cubes may share the same cell in the C/U Matrix under low usage, but the less requiring one may see the urgency of half of its references decremented under high usage). In addition, the P/S sub-modules perform multiple functions, like rate limitation, over-use notification or spacing of packets, enforcing in these ways end-users' QTA.

The QTAMux implementation is part of the Relay and Multiplexing Task (RMT) in the RINA IPC process. This task, whose workflow is described in Fig. 4, is responsible for relaying packets between the different nodes in a DIF. With respect to the RMT task policies, we find 4 different policy-hooks in there:

rmt_q_create_policy and rmt_q_destroy_policy These policies create and destroy all internal RMT queues and private data within the RMT N − 1 port (being N the level of the current DIF in the recursive stack, i.e., an N − 1 port allows injecting traffic to a DIF directly below this one).

rmt_enqueue_policy In charge of checking against queue overrun, monitoring queues and inserting in queue. This policy is called whenever a packet arrives and needs to be forwarded through an N − 1 RMT port.

rmt_next_scheduled_policy_tx In charge of selecting the next packet to be relayed from queues, if any. It is called whenever the N − 1 port is ready and there are packets waiting in the queues. If needed, it can delay the serving of PDUs returning a null PDU pointer.

Given these 4 policy-hooks, the QTAMux implementation tightly follows the workflow presented in Fig. 4. It maintains multiple queues, one per QoS Cube, which store pointers to the stored packets. Upon arrival of a packet (rmt_enqueue_policy), this one is stored in its designated queue and, then, one of the P/S sub-modules is informed, depending on the QoS Cube of the packet. This one will decide whether to drop one of the packets of the queue (not necessarily the last one) or, otherwise, record the arrival and packet length. Then, when a departure is expected (rmt_next_scheduled_policy_tx), QTAMux iterates through the different P/S sub-modules and forward some queue references to the C/U Mux module, accompanied by a cherish and urgency level. The C/U Mux will then decide if it denies the incoming queue reference based on its cherish level, deleting a packet from the queue or, otherwise, placing it into a priority queue based on its urgency level if accepted. Finally, if any, it will serve the first packet from the first queue stored in the C/U Mux priority queue.

4 Home-user rate-limiting policy

As said before, in order to provide QoS assurances in networks susceptible to congestion, it is required to have a well-distributed traffic, where the amount of priority traffic does not represent the majority of traffic. In controlled environments, an administrator can know or have some control over the traffic from the different users (e.g., as in a data-center network). In this way, long-term solutions can be used together with short-term solutions like the QTAMux policy to ensure the good behaviour of the network. In contrast, when dealing with wilder scenarios like those closer to home-users, we have little control (or not at all) over what they can do with their allocated resources.

Focusing on the specific case of an ISP providing services to home-users, QoS requirements must be met within the network. To achieve this, as already discussed before, the amount of priority traffic leaving end-users' networks must be limited to only a small fraction of the total. In this regard, QoS Cubes allow imposing a maximum data rate and burst size. Therefore, RINA can provide by itself a fine control of the networking resources used per flow. In addition, the use of policies like QTAMux can improve that with extended control over the aggregated flows, as it allows limiting the rate of groups of flows that use certain QoS Cubes. With these ones, or other similar policies, it is easier for a provider to limit the amount of high priority traffic that their users insert into the network.

Straightforward solutions dividing the capacity leased by users, thus ensuring that the amount of incoming priority

uncorrected proof

Journal: 11235 MS: 0489 TYPESET DISK LE CP Disp.:2018/7/7 Pages: 13 Layout: Large

Author Proof

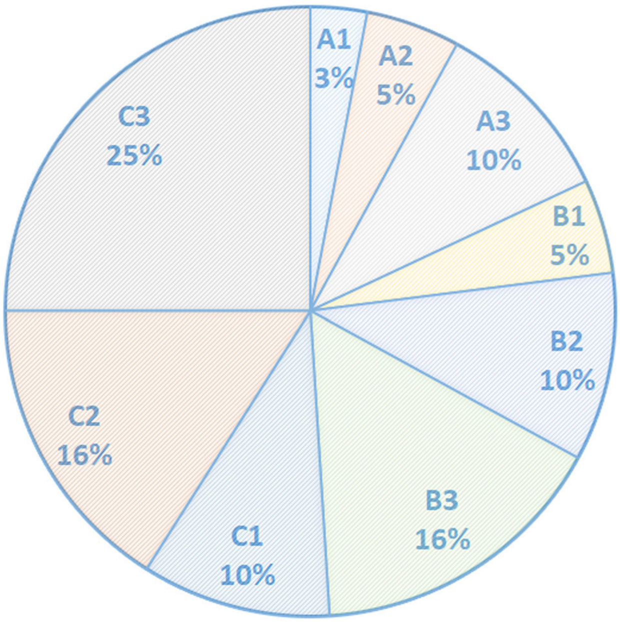

Fig. 5 Example of QoS rate limitation in a 3 × 3 C/U Matrix

traffic never exceeds predefined limits, can be acceptable for network service providers. Nevertheless, such solutions are disadvantageous to end-users that see their link capacity divided into multiple smaller flows. In Fig. 5, we can see an example of how an ISP could limit incoming traffic from their users using a strict per-QoS rate limiting policy, with QoS Cubes distributed within a 3 × 3 C/U Matrix. Note that this example is only one possible configuration, as QoS requirements are fully scenario-dependent. However, it serves to illustrate that, as the range of supported QoS classes increases, the effective capacity offered to end-users becomes more dispersed.

Strict per-QoS class rate limitation forces end-users to underuse their leased capacity eventually, as QoS classes cannot employ the capacity dedicated to others. Still, end-users not so naïve can start attempting to inject multiple flows with different QoS requirements to fill their entire leased capacity. As a result, even requiring a low priority flow initially, they turn into injecting higher priority traffic in the network. For example, with the limits in Fig. 5, a flow assigned to QoS Cube B2 with a rate of 20% the capacity is unfeasible. Hence, 3 flows filling the rates of QoS Cubes A2, B1 and B2 might be used instead.

In order to avoid that, it is important to seek an appropriate solution for both parties. In this regard, we propose a new rate-limiting scheduling policy that provides the same upper limits for priority traffic injected into the provider network, while giving end-users the freedom to decide on how they use their leased resources. With this in mind, we propose a 2-dimensional rate limiting policy, based on the ΔQ framework and the C/U Matrix (although easily extendible to other dimensions if required), that limits the amount of outgoing traffic depending on its urgency and cherish level independently.

With this policy, instead of imposing limits on a per-QoS Cube basis, providers are able to impose limits on the aggregated traffic, up to some priority level for each QoS dimension. For example, in Fig. 6 we can see how the previous per-QoS class limits in Fig. 5 translate into such 2-dimensional limits. Recall that, with the strict per-QoS class limitation, we could only transmit up to 3% of A1 traffic (see Fig. 5). In contrast, 18% A1 can be transmitted with these new limits (i.e., the sum of capacities initially assigned to A1, B1 and C1 traffic flows). Moreover, these limits allow us to use more traffic of high priority classes if needed, avoiding to use more traffic of both A* and *1 QoS Cubes, maintaining in this way the same limits in both the amount of maximum urgent and maximum cherished traffic as in the strict solution. In a similar way, the previous problem appearing when having to allocate 20% B2 traffic would be inexistent here, as end-users could directly request those resources for that QoS Cube (in this case, even after using 18% for A1 traffic), avoiding the need for borrowing additional capacity from higher priority QoS Cubes, something useful from both end-user and provider perspective.

It has to be remarked, though, that it is a policy designed for border routers on the end-user side. This policy does not consider QoS assurance on an end-to-end basis, but focuses

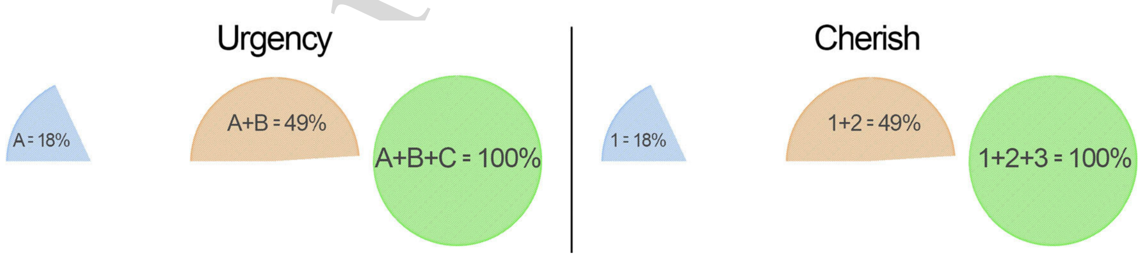

Fig. 6 Example of imposed urgency and cherish rate-limits with traffic in a 3 × 3 C/U Matrix

Journal: **11235** MS: **0489** ☐ TYPESET ☐ DISK ☐ LE ☐ CP Disp.:**2018/7/7** Pages: **13** Layout: **Large**

Author Proof

on enforcing rate limitation based on future flow requirements along its path. Of course, a proper path selection between each source–destination pair will also be crucial to effectively provide the QoS assurances specified by QoS cubes across a DIF. It is noteworthy that, although we focused on two specific dimensions (urgency and cherish) in line with ΔQ, providers could define their own QoS dimensions (e.g., cherish, urgency and packet size), requiring then to provide an appropriate mapping between upper flows and QoS Cubes on flow allocation.

With respect to the implementation of the policy, it shows similar complexity as that of QTAMux. It uses one queue per cell in the defined C/U Matrix (similarly as using one queue per P/S). When a packet arrives, it is stored in the queue matching its QoS Cube's cherish and urgency levels. When the policy is called, it serves the oldest packet from the most urgent non-empty queue with available rate for both urgency and cherish levels. In order to know which queues have available rate, we maintain a counter that records the amount of sending credit for each urgency and cherish level. These counters are increased each time the policy is called, in accordance to the available rate for that level, as well as the time elapsed from the last call, similarly as with a leaky bucket approach (e.g., with a 100 Mb link and the limits in Fig. 6, in 1 ms urgency A will gain 18 Kb, B 31 Kb and C 51 Kb of credit). When selecting the available queues, we limit those up to the highest urgency and cherish levels with a positive amount of credits (e.g. if we have [0, − 5, 20, 50] credits for cherish levels from 0 to 3 respectively, we can serve only queues with cherish level 2 or 3). Finally, when serving a packet, we remove the used credits. If this case, if there are not enough credits in the current level, we take them from upper levels, only leaving the original level in negative if not enough credits are available between all upper levels. Given that lower levels can use credits from upper ones, and that we allow negative credits, in order to avoid complications our credit-based system presents some peculiarities that makes it different from a typical leaky bucket. Credits are assigned from lower to higher levels. When encountering a level with negative credits, it will get all new gains until reaching 0 credits, at which point the gains will be given either to the next level with negative credits or the level that originally owned them. After all new credits are assigned, then, from higher to lower, each level with credits exceeding their maximum backlog will pass its surplus to the next level. Pseudo-codes 1 and 2 describe the process of credit consumption and gain for both Cherish and Urgency, respectively.

```
Consume (int Credits [N], int level, int credit)
for i = level .. 0 do
    if credit <= Credits[i] then
        Credits[i] -= credit;
        return;
    else
        credit -= Credits[i];
        Credits[i] = 0;
Credits[level] -= credit;
```

Pseudo-code 1. Credit consumption given current credits, current level and spent credits

```
Gain (int Credits [N], int Added [N],
int MaxCredit [N])
int j = N-1;
int t = 0;
for i = N-1 .. 0
    t = Added[i]
    while j > i and t > 0 do
        if Credits[j] < 0 then
            if t >= Credits[j] then
                t += Credits[j];
                Credits[j] = 0;
            else
                Credits[j] += t;
                t = 0;
                break;
        j++;
         Credits[i] += t
for i = 0 .. N-1
    Credits[i] += t
    if Credits[i] > MaxCredits[i] then
        t = Credits[i] - MaxCredits[i]
                Credits[i] = MaxCredits[i]
```

Pseudo-code 2. Credit gain given current credits, added credits and maximum credits

Uncorrected

Journal: 11235 MS: 0489 TYPESET DISK LE CP Disp.:2018/7/7 Pages: 13 Layout: Large

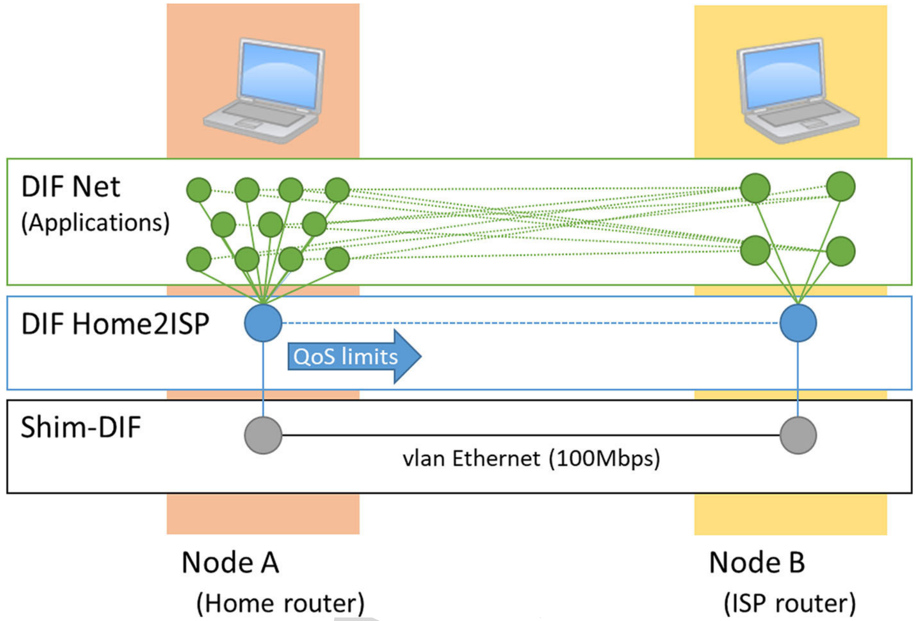

Fig. 7 RINA test-bed for rate-limiting policy evaluation

Author Proof

Uncorrected

5 Numerical results

5.1 Experimental scenario

To assess the proposed rate limiting policy, we have conducted an experimental evaluation using the RINA SDK delivered by the FP7 PRISTINE project [12]. To this goal, we have deployed the point-to-point RINA network test-bed depicted in Fig. 7. In this scenario, two nodes, A and B, emulate a home router and its ISP gateway. For this, we have used two laptops using the latest version of the RINA/IRATI stack [14] over a Debian 8 system with kernel 4.9. These two nodes are connected using a 1 Gbps Ethernet link on which a VLAN, with its rate limited to 100Mbps is configured to connect them in the RINA environment (using a VLAN Ethernet Shim-DIF). Over the shim-DIF, we set a normal DIF (Home2ISP) providing QoS support. In that DIF, one aggregated flow for each available QoS Cube is allocated, and node A is required to ensure that the different rate limits are achieved. Finally, we set a conventional DIF (DIF Net) on top that mimics an Internet-wide DIF providing communication between applications in both sides of the network.

For the different experiments, we define the seven generic QoS Cubes depicted in Table 2, based on a 3×3 C/U Matrix. Moreover, we define and impose the rate-limits depicted in Table 3, limiting the amount of traffic that node A can inject into the network up to each cherish/urgency level. As mentioned before, the mapping between upper flows and QoS Cubes should consider how these flows are routed across the DIF to effectively ensure the QoS requirements end-to-end. However, for these tests, we considered a straightforward mapping between application requirements to QoS Cube

Table 2 Defined QoS cubes for tests

Cherish\urgency	Max Cherish	Mid Cherish	Min Cherish
Max urgency	A1	A2	–
Mid urgency	B1	B2	B3
Min urgency	–	C2	C3

Table 3 Imposed rate-limits for tests

Parameter\levels	Max (Mbps)	Max + Mid (Mbps)	All (Mbps)
Urgency	15	60 (+45)	100 (+40)
Cherish	15	60 (+45)	100 (+40)

(i.e., C/U Matrix cell), leaving the end-to-end QoS assurance consideration out of the scope of this paper.

5.2 Validating the policy

Once the scenario, QoS Cubes and rate limitations are decided, our first goal is to assess the behaviour of the proposed rate-limiting policy within the specified environment. To avoid stationary scheduling states, for these tests we use an application that sends packets at constant intervals, but with their size varying between a minimum and maximum size, always maintaining an average rate of 1Mbps (including headers). We run multiple experiments were traffic matrices are configured to reach 100% of the acceptable rates in average with the goal to assess that rate limits are enforced correctly by the rate-limiting policy.

To check that the policy behaves as expected, we captured the packets received at node B in a post-execution run with

Journal: **11235** MS: **0489** TYPESET DISK LE CP Disp.:**2018/7/7** Pages: **13** Layout: **Large**

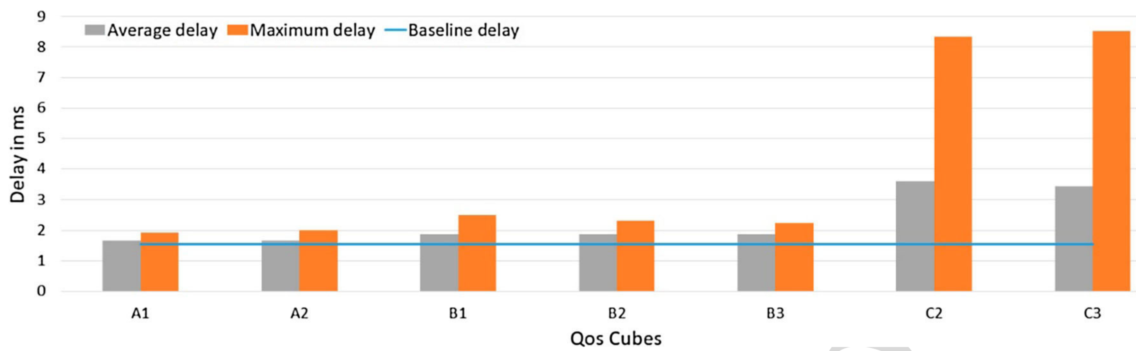

Fig. 8 Comparison of average and maximum delay for the different QoS cubes

Author Proof

a tcpdump on the incoming port. Then, we used a similar approach as for the rate limiting policy. Using the recorded arrival time and size of packets, we computed the gain and expenditure of credits in a discrete way. As expected, the results validated that the policy maintained the outgoing traffic under the required limits, not reaching negative credits along the entire test. Even so, it has to be noted that the maximum backlog of credits considered in the validation process was set slightly higher, as to amount to the internal queues of Ethernet ports, managed independently to the CPU and different encoding times depending on packet size.

In addition, while the policy is not specifically designed to provide QoS guarantees, we aim to ensure that the priorities defined by the C/U Matrix in Table 2 are properly delivered. In this regard, Fig. 8 presents a comparison between the average and maximum delay suffered by the flows assigned to each QoS Cube. As can be seen, the urgency priority is maintained in both average and worst cases (i.e., maximum delay). In order to emphasize the effects of the scheduling policy, we also compare it to the average delay in an uncongested scenario (baseline delay), where we ensure that queues are always emptied between incoming packets. In comparison with this baseline scenario, we see that urgent QoS cubes (A1 and A2) incur almost no additional delay on average, with its maximum growing up mostly due to collisions of packets with the same priority or small bursts. A similar behaviour can be seen for mid-urgent flows (B1, B2 and B3), but with slightly higher delays given their lower priority. In contrast, non-urgent flows suffer from higher delays. This is expected, and works as a measure to avoid losses due to the small overbooking of the network (e.g., in the most extreme situations, we can experience bursts at up to 120% of the link rate). While such delays are high, it has to be noted that we are considering an overbooked low-rate link in these tests. If we consider the number of preceding packets in queue instead of the time spent there, non-urgent packets only wait for 25 preceding packets in average, 90 in the worst case. As the drop threshold was set to 100 packets for non-cherished flows, no losses were experienced in the tests).

Table 4 Rate limits per QoS cube employing the QTA Mux and DiffServ policies

Cherish\urgency	Max Cherish (Mbps)	Mid Cherish (Mbps)	Min Cherish
QTA:Max urgency	A1:5	A2:10	–
QTA:Mid urgency	B1:10	B2:15	B3:20 Mbps
QTA:Min urgency	–	C2:20	C3:20 Mbps
DS	1:15	2:45	3:no-limit

5.3 Comparison with other solutions

Once the behaviour of the policy has been validated, we also compare it against the main QoS scheduling policy in RINA, namely, QTAMux (QTA), configured with limits per C/U cell, as well as against a DiffServ-based policy (DS) [18, 19] with limits per cherish level. In order to do that, we set a scenario where limits per QoS and limits per quality can be compared, using the same test-bed described in Fig. 7 and QoS Cubes defined in Table 2. For the proposed rate-limiting policy (configuration R-lim), we consider the same limits for cherish and urgency levels described in Table 3, and for the QTAMux and DiffServ we consider the limits per QoS Cube described in Table 4. It has to be noted that those limits are only a possible configuration for this scenario (ISPs should freely decide or modify the limits they impose to their clients).

Besides, we consider three types of traffic:

- *Voice flows* Based on G.722 [20]. Constant interval between packets, but with their size varying between voice and silence periods. Urgent but admits some losses, minimum A2.
- *Video* Based on YouTube HD and fullHD qualities [21]. MTU size packets with varying bitrate. Mid urgent, but requires to avoid losses, minimum B1.
- *Data* P2P like flows. MTU size packets with a maximum rate of 5Mbps. Non-urgent and can withstand losses, minimum C3.

Uncorrected proof

Journal: 11235 MS: 0489 TYPESET DISK LE CP Disp.:2018/7/7 Pages: 13 Layout: Large

Author Proof

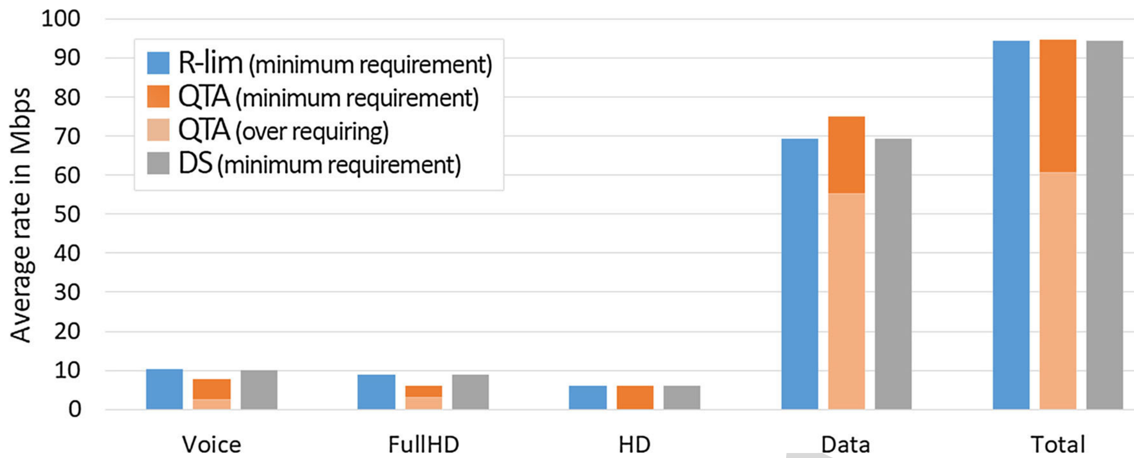

Fig. 9 Comparison of average rate per application (in Mbps)

Given these applications and flow constraints, we setup our scenarios in a way that the maximum number of voice, video and data flows can be supported without exceeding the imposed limits. With this into consideration, we setup our offered traffic matrices as follows:

R-lim and DS scenarios:

- 150 voice flows with QoS Cube A2
- 3 FullHD with QoS Cube B1
- 4 HD flows with QoS Cube B1
- 12 P2P flows with QoS Cube C3

QTA scenario:

- 25 voice flows with QoS Cube A1 and 95 with A2
- 1 FullHD flow with QoS Cube A1 and 1 with B1
- 4 HD flows with QoS Cube B1
- 3 P2P flows with QoS Cube B2, 4 with B3, 4 with C2 and 4 more with C3.

Before presenting the results in this scenario, it has to be noted that the same QoS Cube has to be kept across layers. This is important, as the DS policy does not degrade packets that exceed the rate-limit, but drop them (otherwise, they would regain their priority when reaching destination). With this in mind, we can realize from the construction of the scenario itself that requirements are better translated into QoS Cubes in the R-lim and QTA scenarios, as those can differentiate not only by cherish, but also by the urgency of flows. In addition, the fewer restrictions in R-lim removes the need for differentiating traffic with identical requirements, increasing the amount of flows that can successfully be accepted in the network.

Regarding the network utilization, we can see in Fig. 9 a comparison between the amounts of traffic successfully sent in the network per application, as well as the overall link occupation in each case. As observed, the amount of successfully sent data belonging to voice and video flows results slightly higher with R-lim and DS than with QTA. In contrast, data flows are boosted with QTA. This was expectable, as less voice and video flows can successfully be accepted with the requirements of the QTA scenario. In addition, we can see in Fig. 10 a comparison between the amounts of traffic assigned to each QoS Cube in each scenario. These results, mainly describing the assumed traffic matrices, also highlight the need for a fair rate-limiting policy. In summary, as traffic cannot use the QoS Cube that better adapts to its requirements in the QTA case, we end in a scenario where 60% of the outgoing traffic ends assigned to QoS Cube providing a better service than required (denoted as over requiring in the legends of Figs. 9 and 10).

Besides the problems that an unfair rate-limiting policy imposes to the ISP, a strict rate limitation also affects the service that applications eventually receive. Indeed, when imposing too strict rate limits, we enforce an artificial differentiation among flows with the same requirements. Figure 11 shows a comparison between the service received per application. In the QTA scenario, voice flows get more or less the same service (all have the same urgency). However, we see oscillations in video flows, where FullHD urgent flows experience a smaller delay than the rest, similarly to that experienced by voice flows. In contrast, mid-urgent flows get slightly higher average delay and an extra 0.5 ms of maximum delay in comparison. In a similar way, we can see how Data flows suffer large variations, near to 1 ms, between the maximum delay of those assigned to QoS Cubes B2/B3 and C2/C3. In comparison, in the R-lim scenario, we see all flows of each application receiving similar services (as expected), but more importantly, all suffering lower delays (both average and maximum) than flows sharing the same QoS Cube in the QTA scenario. In contrast to the two ΔQ-based policies, when using the DS policy, flows do not experiment any visi-

uncorrected proof

Author Proof

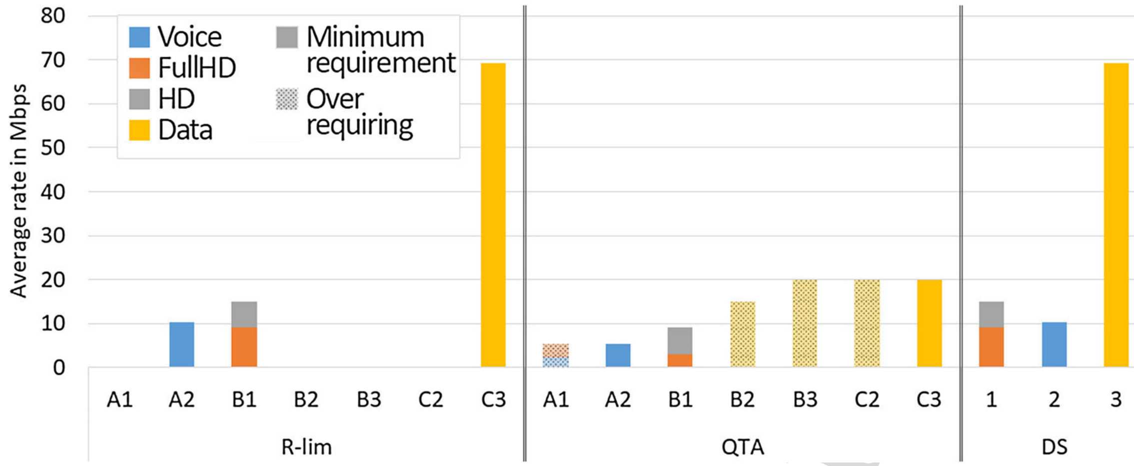

Fig. 10 Comparison of average rate per QoS cube (in Mbps)

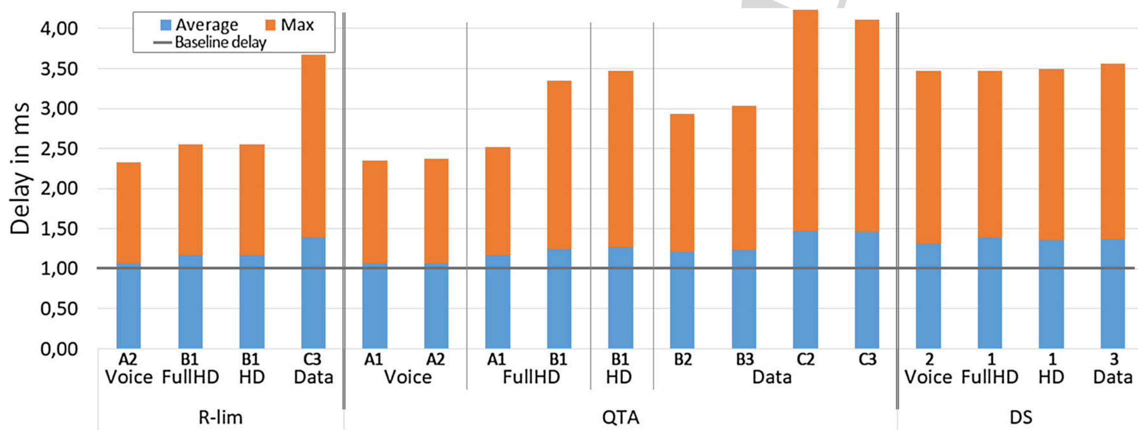

Fig. 11 Comparison of average and maximum delay per application and QoS cube using the proposed rate limiting policy, QTA Mux and DS

ble differentiation in terms of delay, resulting in a best-effort scenario.

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

Finally, with respect to this particular test-bed and the obtained results, some particularities have to be considered. Firstly, the test-bed used a 100 Mbps VLAN over a 1 Gbps Ethernet [22] link as shim-DIF. This has some peculiarities with respect to using the Ethernet link directly at its maximum rate. Firstly, we have to consider the slightly larger headers of the Ethernet frame due to use of a VLAN. Secondly, given that the VLAN works at 1/10th of the Ethernet link capacity, the inter-frame delay used to separate Ethernet frames does not affect us, as all packets are served with higher separations. Furthermore, while we are emulating routers, we are doing it using machines, not only offering networking functionality but also running the same applications that generate that communication, while at the same time having multiple active background processes. This affects negatively to all networking processes, as those have to compete for the CPU time with other non-related processes.

6 Conclusions

Given the increasing number of heterogeneous distributed applications populating the network, each one with specific QoS requirements, it is evident that future networks must provide a way to allow an effective QoS differentiation.

Journal: 11235 MS: 0489 TYPESET DISK LE CP Disp.:2018/7/7 Pages: 13 Layout: Large

Author Proof

RINA, with its default QoS support employing QoS Cubes, together with the incorporated ΔQ-based scheduling policies, can yield superior performance to this end compared to the current TCP/IP-based Internet. Even so, in order to allow home-users to request differentiated QoS treatment for their flows, it is imperative for Internet Service Providers to upper limit the amount of high priority traffic that these users can inject in the network. While RINA and the ΔQ-based QTAMux scheduling policy already provide ways to impose such limits, they are not end-user friendly and can lead to undesired end-user behaviours. To solve that, in this work, we have proposed and experimentally evaluated a RINA rate limiting policy based on the ideas of ΔQ, which limits urgent and cherished traffic independently. The proposed policy not only succeeds in avoiding end-users filling the network with high priority traffic, but also achieves it in an end-user friendly way, allowing them to use the available capacity in the way most suited for their needs. While gracefully solving the targeted issue of limiting end-user priority traffic to allow a global differentiated QoS treatment, there is room for improvement. Although not explained in depth, the proposed policy bases its internal multiplexing on that of simple ΔQ scheduling policies. In this regard, it is left for future work to check the benefits of other multiplexing options, which could not only consider the urgency of flows, but also if they are taking unused resources of higher priorities.

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

In addition, in this work we have limited to a scenario centred on the communication between home router and the ISP, without considering home devices or the interaction with other policies (congestion control, flow allocation, etc.). In this regard, future work in this area will aim to expand this scenario, taking into consideration fast congestion control and retransmission policies, as well as QoS Cube based flow allocation mechanisms. Furthermore, while the policy manages flow contention on the shim level, it is left for future work the mechanism for translating QoS requirements of upper level flows into QoS Cubes, something that would require considering the different sub-networks traversed by them.

Acknowledgements This work has been partially supported by the Spanish Ministry of Economy and Competitiveness under contract TEC2017-90034-C2-1-R (ALLIANCE Project) that receives funding from FEDER.

Compliance with ethical standards

Conflict of interest All the authors declare that they have no conflict of interest.

References

1. Chen, Y., Farley, T., & Ye, N. (2004). QoS requirements of network applications on the internet. *Information Knowledge Systems Management, 4*(1), 55–76.
2. Trouva, E., Grasa, E., Day, J., Matta, I., Chitkushev, L. T., Phelan, P., & Bunch, S. (2011). Is the internet an unfinished demo? Meet RINA!. In *TERENA networking conference* (TNC) *2011,* Prague, Czech Republic.
3. Day, J., Matta, I., & Mattar, K. (2008). Networking is IPC: A guiding principle to a better internet. In *4th international conference on emerging networking experiments and technologies (ACM CoNEXT) 2008,* New York, USA, (pp. 1–6). 4
4. FP7 PRISTINE (2013) Deliverable 3.2 Initial specification and proof of concept implementation of techniques to enhance performance and resource utilization in networks, (pp. 39–55). http://ict-pristine.eu/wp-content/uploads/2013/12/pristine_d33_draft.pdf.
5. Davies, N., (2003) Delivering predictable quality in saturated networks. Technical Report. http://www.pnsol.com/public/TP-PNS-2003-09.pdf.
6. Davies, N., Holyer, J., & Thompson, P. (1999). An operational model to control loss and delay of traffic in a network switch. In *3th IFIP workshop on traffic management and design of ATM networks*.
7. Davies, N., Holyer, J., & Thompson, P. (1999). *A queueing theory model that enables control of loss and delay at a network switch.* Technical Report, University of Bristol, Bristol, United Kingdom, 1999.
8. Kesselman, A., Leonardi, S., & Bonifaci, V. (2005). Game-theoretic analysis of internet switching with selfish users. In *Internet and network economics (WINE) 2005,* Hong Kong, China, (pp. 236–245).
9. Day, J. (2008). *Patterns in network architecture: A return to fundamentals*. Upper Saddle River: Prentice Hall.
10. Grasa, E., Gastn, B., van der Meer, S., Crotty, M., & Puente, M. A. (2016). Simplifying multi-layer network management with RINA. In *TERENA networking conference (TNC),* Prague, Czech Republic.
11. IRATI FP7-317814. Researching and Prototyping the recursive InterNetwork architecture to support distributed computing. http://irati.eu.
12. PRISTINE FP7-619305. Programmability in RINA for European supremacy of virtualized networks. http://ict-pristine.eu.
13. ARCFIRE H2020-687871. Experimenting RINA on FIRE+. http://ict-arcfire.eu.
14. Maffione, V., Salvestrini, F., Grasa, E., Bergesio, L., & Tarzan, M. (2016). A software development kit to exploit RINA programmability. In *Next generation networking and internet symposium (IEEE ICC) 2016,* Kuala Lumpur, Malaysia.

15. GitHub - IRATI/stack: RINA implementation for OS/Linux. https://github.com/IRATI/stack.
16. Leon, S., Perelló, J., Careglio, D., Grasa, E., Davies, N. J., Thompson, P., (2016) Assuring QoS guarantees for heterogeneous services in RINA networks with ΔQ. In *Workshop on network infrastructure services as part of cloud computing (NetCloud) 2016*, Luxembourg.
17. Predictable network Solutions Limited. (2015). *A study of traffic management detection methods and tools*. United Kingdom: OfCom.
18. Almquist, P., (1992). Type of service in the internet protocol suite. RFC 1349, https://doi.org/10.17487/rfc1349, https://www.rfc-editor.org/info/rfc1349.
19. Nadeem, R. M., Saleem, R. M., Bashir, R. N., & Habib, S. (2017). Analysis of impact of differentiated services (DiffServ) on the quality of services (QoS) Parameters of major services of internet. *Indian Journal of Science & Technology, 10*(31), 1–24.
20. Mermelstein, P. (1988). G.722: A new CCITT coding standard for digital transmission of wideband audio signals. *IEEE Communications Magazine, 26*(1), 8–15.
21. Youtube: Live encoder settings, bitrates, and resolutions. https://support.google.com/youtube/answer/2853702. 5
22. IEEE Computer Society. IEEE Standard for Ethernet. IEEE Std 802.3TM-2015.

uncorrected proof

Author Proof

Journal: 11235 MS: 0489 ☐ TYPESET ☐ DISK ☐ LE ☐ CP Disp.:2018/7/7 Pages: 13 Layout: Large

Journal: 11235
Article: 489

Springer
the language of science

Author Query Form

Please ensure you fill out your response to the queries raised below and return this form along with your corrections

Dear Author

During the process of typesetting your article, the following queries have arisen. Please check your typeset proof carefully against the queries listed below and mark the necessary changes either directly on the proof/online grid or in the 'Author's response' area provided below

Query	Details required	Author's response
1.	Kindly check and confirm whether the corresponding author is correctly identified and amend if necessary.	correct
2.	Kindly provide authors photo and biography for all the authors.	Atached in mail
3.	Please confirm if the author names are presented accurately and in the correct sequence (given name, middle name/initial, family name). Author 1 Given name: [Sergio Leon] Last name [Gaixas]. Also, kindly confirm the details in the metadata are correct.	Given name Sergio, family Leon Gaixas (I use both because there is another Sergio Leon).
4.	Kindly check inserted year is correct for reference [4].	correct
5.	Kindly provide complete details for the reference [22].	ISO/IEC/IEEE International Standard for Ethernet," in ISO/IEC/IEEE 8802-3:2014(E) , vol., no., pp.1-3754, April 1 2014 doi: 10.1109/IEEESTD.2014.6781545

Author Proof
uncorrected proof