Pristine

Deliverable-3.1

Draft specification of techniques to enhance performance and resource utilization in networks

Deliverable Editor: Michael Welzl,UiO

Publication date:	30-September-2014
Deliverable Nature:	Report
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:	RINA, congestion control, distributed resource allocation, delta-Q, topological routing, hierarchical routing
Synopsis:	Draft specification which covers techniques on congestion control, distributed resource allocation and topological addressing/routing in RINA networks. D3.1 describes the theoretical analysis of these 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.

Disclaimer

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

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

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

Executive Summary

Preceding the initial specification and proof of concept implementation of techniques to enhance performance and resource utilization in networks that is due in month 16, this document outlines a "draft specification" of these techniques (a technical description of the developments that are planned for / have just started in this work package). Following the three tasks in WP3, there is a chapter for each of the respective developments: i) programmable congestion control; ii) unification of connection-oriented and connectionless resource allocation in support of multiple levels of service; iii) topological addressing to bound routing table sizes.

Table of Contents

Acronyms

ACC	Aggregate Congestion Control
AE	Application Entity
AI	Application Instance
AP	Application Process
CACEP	Common Application Connection Establishment Protocol
CCP	Continuity Check Protocol
CDAP	Common Distributed Application Protocol
DA	Distributed Application
DAF	Distributed Application Facility
DIF	Distributed IPC Facility
DTCP	Data Transfer Control Protocol
DTP	Data Transfer Protocol
E2E	End to End
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
IRM	IPC Resource Manager

ISP	Internet Service Provider
LAN	Local Area Network
LIFO	Last In, First Out
MAC	Medium Access Control
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
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

TCP	Transmission Control Protocol
WLAN	Wireless LAN

1. Programmable congestion control

In the Internet, congestion control is done by the TCP protocol, which normally operates "end-to-end", where "end" is the host the application is running on. This ensures scalability – interior network elements do not have to worry about congestion control – but it also comes with an embedded ignorance regarding the underlying technology. A TCP connection does not know what this technology is; it could e.g. operate just within one Ethernet domain, or it could traverse a wireless LAN, followed by a satellite link, followed by a 3G link. Hence, it cannot make any form of link-technology-specific decision. Congestion is usually detected implicitly via packet loss, which may badly interact with lower-layer mechanisms; moreover, the lack of any form of backwards push-back from within the network makes the control loop sluggish, causing local buffers to grow. RINA, with its flexible layering, allows to attain scalability in a divide-and-conquer manner, which enables usage of more elaborate mechanisms inside the network wherever they do fit. This is also illustrated by Figure 11 of the PRISTINE DoW.

PRISTINE’s Aggregate Congestion Control ("ACC") is a function where an N-1 flow pushes back to the N-IPC Process that uses the flow (which is a sender or a relay of an N-EFCP-connection). Since the N-layer does not have direct access or knowledge of N-1 EFCP connections, just of the flows it is provided by the N-1 layer, control is only executed via the speed at which the N-IPC process drains or fills buffers of N-1 IPC processes with its read and write calls. It is also called "Programmable Congestion Control" because it allows to customize the congestion control function can be customized for segments of an end-to-end path. In what follows, aggregate congestion control will be described by means of a simple example.

As a starting point, consider a path with one intermediate hop, given by a physical setup of 3 nodes: a sender, a relay node in the middle, and a receiver. There are two ways to do congestion control with RINA when interconnecting these 3 nodes, and they are depicted in Figure 1 below.

Note that the Shim DIFs depicted in the diagrams are assumed to be associated with one already existing link layer technology each. This means that e.g. Shim DIF1 could run over a Wireless LAN and Shim DIF2 could run over a 3G connection. Also note that, even in a future "all-RINA" world, at the lowest layer at which it is implemented, flow control is most efficiently done by incorporating the characteristics of the physical transmission media. That is, 802.11 MAC is probably the most efficient way to control rates / windows across an 802.11 link.

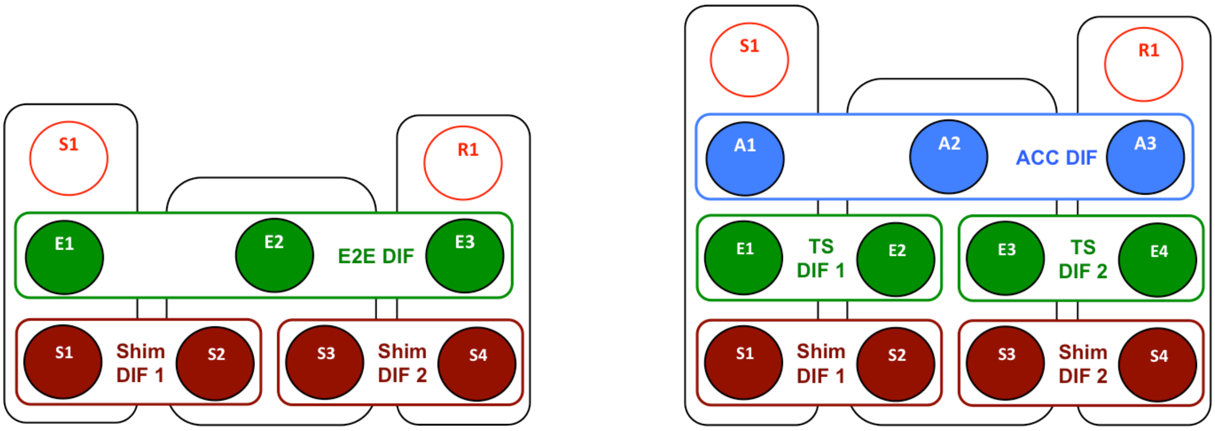


Figure 1. Two ways of doing congestion control with RINA. Left: the traditional ("Internet", "end-to-end") way; right: the ACC way. S1 and R1 are the application's sending and receiving IPC processes.

In the diagram on the left, the "E2E DIF" includes three IPC processes. EFCP sends data from E1 across E2 (which is a relay IPC process) to E3. If congestion appears in the relay, it can, for example, ECN-mark packets. The sender's window or rate is controlled by EFCP's flow control, and this is the function reacting to congestion in this network, i.e. EFCP policies here define congestion control along that path. This setup is pretty similar to the Internet when ECN is in use – here, the EFCP connection takes the role of a traditional Internet TCP connection, with only one flow control / congestion control function for the whole path. This scenario has its advantages in simplicity and scalability, but it does not make it possible for the function that controls the sender's rate to benefit from knowledge of WLAN-specific congestion control (MAC) over the WLAN link (Shim DIF 1) and 3G-specific congestion control (MAC) over the 3G link (Shim DIF 2): if congestion appears in one of the two Shim DIFs then EFCP in the E2E DIF needs a full round-trip-time (from E1 to E3 and back) to react.

The diagram on the right-hand side of Figure 1 shows a different approach, which is possible to do with RINA due to its flexible layering. Here, we have two separate technology-specific ("TS") DIFs, and hence two separate EFCP connections. Each of them can have different policies for flow (or congestion) control; they can be tailored to the underlying link layer technology or physical link characteristics. Additionally, we have an ACC DIF on top. This DIF's EFCP connection includes a relay IPC process (A2). Even though the EFCP connections inside the TS DIFs that it uses (via IPC processes E2 and E3) are not directly visible to A2, it must ensure that TS DIF 1 gives the correct upstream feedback via the interface that it uses to talk to E2 and E3.

The necessary interface here is about control of a buffer. If the N-IPC process A2 does not receive data from the N-1 IPC process E2 via E2's receive buffer as quickly as the other N-1 IPC process E3 drains it from its send buffer, the N-IPC process A2 should be able to somehow tell E2 that it can speed up. This should then cause the flow control in TS DIF 1 to make E1 increase its rate (or window). E2 automatically obtains the information it needs by monitoring the state of its receive buffer which is constantly emptied by A2. If, on the other hand, E3 is sending data slower than it arrives at E2, E3's send buffer will become full, which will cause E3 to push-back to A2. A2 is then not able to send anymore and therefore will have to stop taking data from E2's receive buffer, causing this buffer to get full. Thus, E2 will again obtain the information it needs in order to make E1 slow down via EFCP's flow control.

Indirectly learning about the correct sending rate or window because EFCP's flow control reacts upon the receiver buffer overflowing or getting emptied may entail some delay in the reaction, at least in the overflow case (first, E3's send buffer will overflow, then, E2's receive buffer will overflow). This could perhaps be improved upon by enriching the interface that IPC processes use as they read from a buffer or write into a buffer; it will be a matter of further investigation to see whether such enrichment is necessary.

Now we turn to slightly larger example of how ACC (right-hand side of Figure 1) congestion control would operate. Consider the network depicted in Figure 2. For clarity, Shim DIFs are not shown. S1 sends data to R1, S2 sends data to R2. If, in this topology, congestion appears (or disappears) between E4 and E5, it is important for S1 and/or S2 to reduce (or increase) the rate. Let us assume an overload situation between E4 and E5 and see what happens:

On the connection between IPC processes E4 and E5, EFCP's flow control will tell E4 to slow down. This will let the send buffer of E4 reach a threshold that will make E4 start pushing back to A3 when writing to that flow (i.e. the write_to_flow operation is blocked or a return error is given). A3, as it relays traffic on the two ACC-DIF EFCP connections A1-A5 and A2-A6, is the N-IPC process that fills this N-1 buffer, and it deals with two N-1 receive buffers: the one where E3 is receiving from E1 and the one where E3 is receiving from E2. It can now implement various policies to slow down one or both of the senders by deciding how quickly it takes data from these two receive buffers. Taking data from one of the buffers slower than they arrive will cause the E1-E3 or E2-E3 EFCP connection to react, which will in turn lead to a reduction of the rates of both senders.

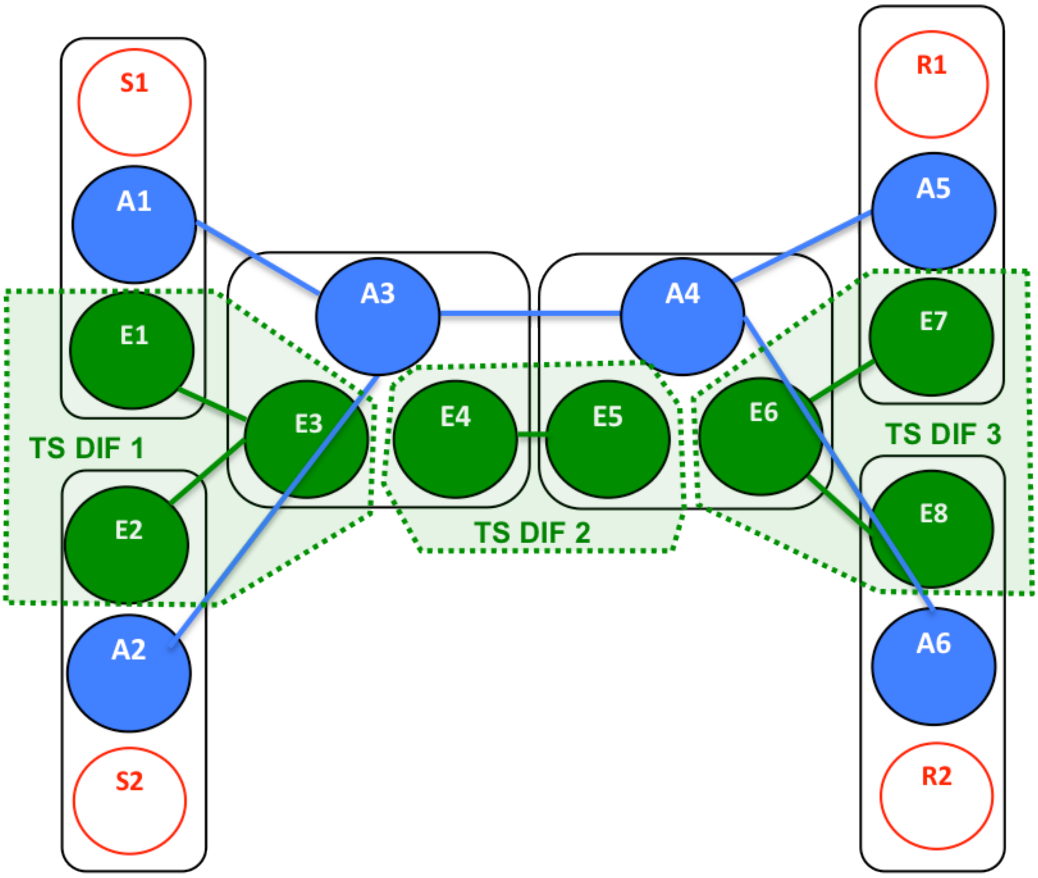


Figure 2. A network with two senders and receivers: S1 sends data to R1, S2 sends data to R2.

In this example, congestion control works without involving the receiver or in fact any IPC process beyond the point of congestion along the end-to-end path. It operates on traffic aggregates: congestion is detected on the single EFCP connection E4-E5, which carries traffic from two flows in the N-layer (S1-R1 and S2-R2). One could also imagine multiple parallel ACC DIFs which all would use TS DIF 2. Then, as soon as any of the senders in the ACC DIFs have reduced their sending rates enough, congestion disappears on the E4-E5 link.

Congestion happening at the E4-E5 link, and A3 reacting to it is a relatively easy case. If we consider congestion happening e.g. on the link E6-E8, the situation becomes more complicated: A4 sees it, and understands that it should slow down sender S2 in some way. Distinguishing by their roles, we can consider the following 3 types of relays, and combinations thereof (e.g., an IPC process can be both type "fan-in" and "fan-out" at the same time):

- *plain forwarding:* this is an N-IPC process that receives data from one buffer of one N-1 IPC process (here called the "incoming buffer") and sends it ahead via one buffer of another N-1 IPC process (here called the "outgoing buffer"). The congestion control function of such a relay is determined by the speed at which it takes data from the incoming buffer and forwards it to the outgoing buffer. If the outgoing buffer gets filled due to congestion somewhere further downstream, this IPC process cannot forward any more data from the incoming buffer and the incoming buffer also fills, causing upstream pushback in the receiver-side N-1 IPC process that the relay is talking to. Rather than such binary stop/go behavior, a "plain forwarding" relay could also work with filling levels of the buffer, to e.g. start draining the incoming buffer slower when the outgoing buffer exceeds a threshold. **Such variations can be implemented in the RMTQMonitorPolicy and MAXQPolicy of the Relaying and Multiplexing Task (RMT).**

- *fan-in:* this is an N-IPC process that receives data from the buffers of multiple N-1 IPC processes (here called "incoming buffers") and sends it ahead via one buffer of another N-1 IPC process (here called the "outgoing buffer"). The congestion control function of this type of relay is similar to the "plain forwarding" relay but additionally can include local decisions about the incoming buffers. For example, when it has to slow down, it can do this by stopping to take data from only one buffer and keep servicing others, or it can slow down equally across all incoming buffers. **This decision can be implemented in the RMT-SchedulingPolicy of the Relaying and Multiplexing Task (RMT).**
- *fan-out:* this is an N-IPC process that receives data from one buffer of one N-1 IPC process (here called the "incoming buffer") and sends it ahead via one (assuming unicast, else multiple) buffer of multiple N-1 IPC process (here called the "outgoing buffer"). The congestion control function of this type of relay is different because it may execute control over the wrong sender if it just operates on the incoming buffer. Thus, this process must either ECN-mark traffic such that the correct sender is notified via the receiver, or send a message (CDAP) to the correct sender to make it slow down. The latter function has to be regarded as experimental, as it may become a potential scalability hazard.

In the example in Figure 2, A3 is a fan-in type relay and A4 is a fan-out type relay; if congestion would not occur between E4 and E5 but between E6 and E8, simply slowing down the speed at which data is taken from the receive buffer of IPC process E5 would slow down both senders, when in fact only sender S2 should slow down. Thus, it would be preferable for A4 to ECN-mark traffic in this situation (or perhaps send a CDAP message to A2).

These examples have introduced the basic functioning of Aggregate (= Programmable) Congestion Control in RINA. The planned research will include investigating potential scalability limits of the idea, and considering contraints from routing (ACC as described above can only be applied when routing is symmetric).

2. Unification of Connection-Oriented and Connectionless Resource Allocation in Support of Multiple Levels of Service

2.1. Resource allocation in a DIF

Connection oriented resource allocation differs from connectionless forwarding in predictability of packet forwarding behaviour. In a non-QoS aware paradigm, a packet is forwarded in the *best* manner possible at the point of time when the packet needs to be handled. The definition of *best* depends on traffic conditions, resource constraints (i.e. queue length) and forwarding policy on each router traversed. To address application requirements for Quality of Service (QoS), a DIF allocates a set of resources out of a pool of available resources under its control. In general, DIFs provide a unified model for resource allocation to the applications, which does not distinguish between a *connectionless* and a *connection-oriented* mode of operation. It is up to the DIF to determine the best forwarding paradigm for a given flow. By requesting a flow, an Application Instance (AI) is requesting a DIF to provide resources to a flow that is conforming to a set of requirements. These requirements are typically related to minimum/maximum rates like packet delay, packet jitter, packet loss, reliable delivery of SDUs, etc.

Establishing a flow within a DIF, the "Error and Flow Control Protocol" is utilized by the DIF associating IPC Processes at the end points. By definition, a single flow is supported by a single EFCP connection at a given time. (De-)multiplexing PDUs from EFCP Instances into N-1 flows is performed by the Relaying and Multiplexing Task (RMT). Considering QoS, the DIF needs to map the flow to a particular QoS cube, which is associated to a predefined set of ranges of parameters that define a region at the QoS space. Consequently a DIF has to classify the flow, matching it with existing QoS cubes and perform the mapping. To facilitate that task, each EFCP connection needs to be associated with a number of policies corresponding to a QoS cube, and assigned the QoS cube’s qos-id. Naturally, the EFCP PDU is put in the corresponding QoS queue before being written to the (N-1) flow.

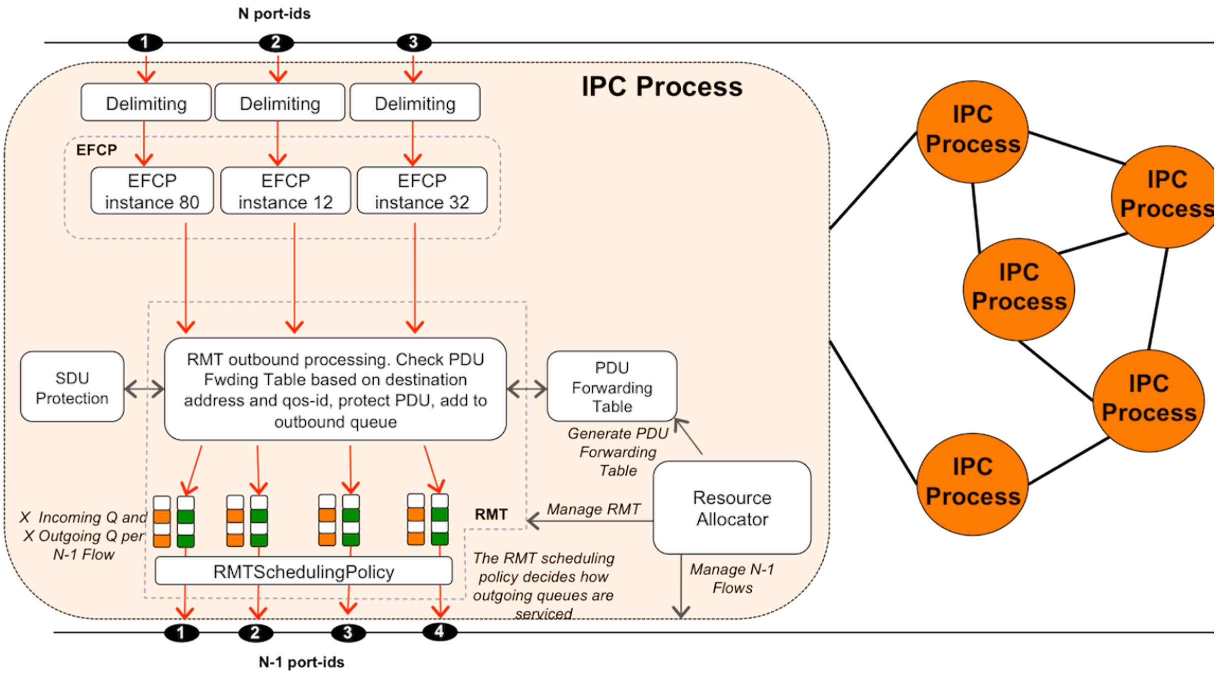


Figure 3. IPC Process, components relevant to resource allocation

The Resource Allocation Task is in charge of managing the allocation of the set of resources needed to provide QoS aware delivery of PDUs. This includes the size and number of QoS cubes available in each IPC Process, as well as the assignment of flows to QoS cubes. In addition, it holds and maintains the PDU forwarding table and is aware of the number of RMT input/output queues per N-1 flow and the policies servicing these queues. A DIF's resource allocation strategy depends on the stochastic properties of the traffic it has to treat, as well as on the different characteristics of the class of flows it wants to support. It is expected that the more stochastic the traffic offered to the DIF is, the more effective strategies that tend towards statistical multiplexing of different flows into shared resources will be, while for more deterministic traffic connection-oriented style resource allocation strategies will have better results.

The DIF's resource allocation strategy influences the number of queues in the RMT, as well as the processing of those queues - as illustrated by the following figure. In one extreme, there can be a single queue per input or output N-1 flow. This queue is shared by all the PDUs of all the N-flows provided by the DIF, therefore treating all the flows the same way and providing no isolation between them. On the other extreme, each flow can be isolated from each other and treated individually by having a dedicated input/ output queue and a proper scheduling algorithm. In-between these two approaches, separate input/output queues per each QoS class provide isolation and differential treatment to flows belonging to different QoS cubes - limiting the resource sharing to flows belonging to the same QoS cube.

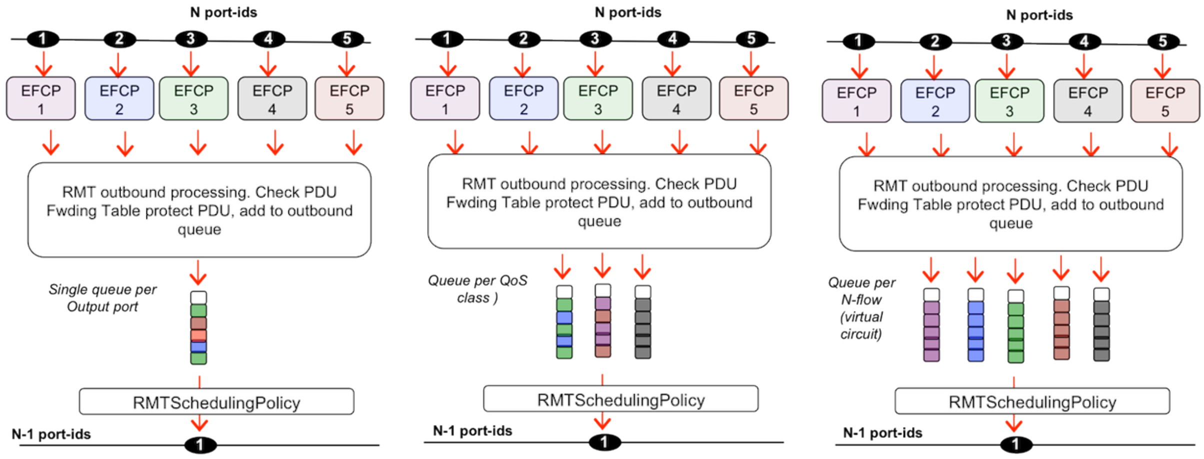

N port-ids
1
2
3
4
5
EFCP 1
EFCP 2
EFCP 3
EFCP 4
EFCP 5
RMT outbound processing. Check PDU Fwding Table protect PDU, add to outbound queue
Single queue per Output port
RMTSchedulingPolicy
N-1 port-ids
1
N port-ids
1
2
3
4
5
EFCP 1
EFCP 2
EFCP 3
EFCP 4
EFCP 5
RMT outbound processing. Check PDU Fwding Table protect PDU, add to outbound queue
Queue per QoS class)
RMTSchedulingPolicy
N-1 port-ids
1
N port-ids
1
2
3
4
5
EFCP 1
EFCP 2
EFCP 3
EFCP 4
EFCP 5
RMT outbound processing. Check PDU Fwding Table protect PDU, add to outbound queue
Queue per N-flow (virtual circuit)
RMTSchedulingPolicy
N-1 port-ids
1

Figure 4. Different options for separating N-flows in different RMT queues (only outgoing processing shown, for simplicity)

There are three main policies in the RMT that control how these queues are serviced:

- **RMT queue monitor policy**. Invoked whenever a PDU is placed in a queue and may keep additional variables that may be of use to the decision process of the RMT scheduling policy and the RMT maximum queue policy.
- **RMT scheduling policy**. This is the meat of the RMT. This is the scheduling algorithm that determines the order input and output queues are serviced. This policy may implement any of the standard scheduling algorithms, FCFS, LIFO, longest queue first, priorities, etc.
- **RMT maximum queue policy**. Invoked when a queue reaches or crosses the threshold or maximum queue lengths allowed for this queue. Note that maximum length may be exceeded.

Once the multiplexing model of the RMT has been decided (number of queues per N-1 port and scheduling algorithms), the next question to answer is how to handle the dimensioning of these queues. At planning time, the DIF designers have to take into account a number of inputs:

- The DIF "traffic matrix". An initial forecast of the volume and stochastic properties of the traffic that will be offered to each of the DIF’s "edge" IPC Processes, as well as the expected outcomes for each flow (or type of flows) is a key information in order to dimension the queues of the RMT. Flows will be matched to different QoS cubes based on their quality expectations (bounds on loss and delay).
- How will the DIF route the offered traffic between edge IPC Processes. This information is required in order to understand the volume and properties of the traffic that will be offered to each "internal" IPC Process in the DIF (that is, the IPC Processes that just relay traffic).

- The amount of resources that the DIF wishes to reserve in order to deal with temporary failures, as well as the strategies that the DIF will use to route around these failures. Depending on the level of service a DIF wants to provide, and the likelihood of different events leading to failures and therefore unavailability of resources, a number of extra resources have to be reserved at the proper IPC Processes.

The initial resource allocation performed at the DIF design time can be kept static and only updated during maintenance events or be dynamically adjusted during operation time. For example, resources assigned to different QoS classes can be dynamically adjusted based on the real utilization of those resources seen by the IPC Processes - maybe there is more traffic of type "A" than initially planned and less of type "B". Or transient failures may force certain IPC Processes to update its resource allocation in order to temporarily process traffic not initially foreseen. The DIF can perform this function in a distributed manner by allowing IPC Processes to exchange resource usage and utilization information via the Resource Allocator tasks.

It is interesting to highlight the strong dependency between routing and resource allocation policies, since the traffic flowing through a DIF should follow the routes through which resources for that traffic have been allocated. If there is a mismatch (due to a temporary failure, for instance) then either routing has to be updated to move traffic through other IPC Processes where resources are available, or the resource allocation has to be updated so that traffic has enough resources while flowing through a certain path.

2.2. Considerations

2.2.1. Distributed resource allocation

Distributed resource allocation requires permanent monitoring of available and used network resources to place flows in a QoS conforming manner. Connection orientation enables to reserve resources along the route before PDUs are actually forwarded. By reserving resources along the path, a more uniform distribution of packet forwarding performance for the flow can be achieved. However in conventional implementations like e.g. MPLS-TE, resource reservation is pretty static and doesn’t adapt to fluctuating flow volumes between EFCP associations. Therefore a self adapting mechanism shall be investigated that can automatically adapt resource usage or resource reservation depending on flow characteristics. The following mechanisms are considered to be evaluated in this work package:

- Assign and release flows to connections based on flow characteristics. Release from a connection means the flow will be forwarded in a connectionless manner.
- Load balance a set of flows across available resources and automatically re-balance flows if QoS classes get compromised
- Grow/shrink reserved bandwidth according to observed traffic demand and QoS policy

There are cases where a flow there is a choice of forwarding a flow over different DIFs e.g. in case where two instances are connected via two physical links in analogy to an Ethernet Link Aggregation Group. In such a cases each of the individual links would form a 0-DIF. As a pre-condition to automatically re-balance traffic across 0-DIFs, an optimum set of parameters needs to be found to cope with the traffic characteristics. Parameters identified are:

- Tolerance: Tolerance defines the allowed degree of imbalance before re-balancing starts. Since traffic can fluctuate heavily in microseconds, reacting too quickly on imbalances is not required. It is planned to study the impact of defining too tight or loose tolerance levels.
- Scan Interval: The interval to be chosen to monitor traffic. Short intervals allow to re-balance more often but may interact with flow control and create positive feedback loops. Longer intervals re-balance less often and may therefore create more imbalances, however flow control will less often be affected by re-balancing actions. It is planned to study the impact of interval choices.

2.2.2. Flow assignment

A DIF always has the option to map an (N-1) flow to a connectionless forwarding if it is able to provide the QoS requested. The connectionless forwarding paradigm is the basic forwarding paradigm as it doesn’t mandate the pre-existence of a connection between two IPC processes and can therefore conveniently be utilized to initiate connection requests. Also in the connectionless paradigm, QoS mechanisms can be enabled to handle occasional resource contention issues by favouring QoS forwarding over best effort traffic. Whether an N-flow is best served in a connectionless or connection-oriented manner depends on various conditions since the RMT needs to balance available resources with QoS needs requested, considering also connectionless traffic. In PRISTINE, a resource allocation model should be investigated to find a suitable strategy to assign N-flows to a connection or connectionless forwarding in a dynamic manner. Points to consider are:

- The purpose of a connection is to reserve certain resources for preferential use of this connection. It therefore sets aside a subset of resources that were available for connectionless forwarding and use by other connections. A policy needs to be applied to avoid resource starvation by reserving too many resources.
- packet flows with high fluctuation of traffic may not be particularly suitable to be mapped on connections with finite resources, while constant flows are. A tie-breaking policy should be investigated to decide where to map those flows.
- creating connections goes along with de-allocating connections. A policy to remove connections when they are no more required needs to be investigated.
- creating and releasing connections takes time before they can be used. This setup delay may violate QoS expectations of (N-1) flows e.g. in cases where interactivity is a key parameter. Hence connectionless forwarding may be required until the connection is established and the flow can be cut over.
- The amount of resources requested for a connection may be disproportionate to the traffic transported by that connection. If actual traffic exceeds the amount of resources requested, QoS can't be guaranteed anymore. Similarly if too many resources are reserved for too little traffic, the network gets under-utilized. A policy shall be investigated to adapt resources assigned to connections to traffic demand.
- To address the needs mentioned, it is required to define the parameters that need to be measured and the time interval to gain those. They eventually need to be fed into a proposed policy to change resource allocation in a more dynamic manner.

2.2.3. Traffic/connection balancing considerations

Balancing traffic needs to be adaptive to flow and network conditions to keep the flow performance high and mitigate different bottlenecks in the network.

A first study aims to gain practical experience in setting parameters needed for a flow assignment policy. Link bundles exist in a certain section of the network (i.e. between two adjacent routers or servers) if traffic expectation exceeds a single interface capacity and therefore two physical parallel links are cabled between adjacent devices. As an example, it is common that several 10G links are bundled together in many data center applications. Although link bundles provide an aggregate capacity equivalent to the sum of the capacity of each member link, there are two limiting factors. First, buffering and forwarding resources are designed per individual member link and second, most flows need to be delivered in sequence. In practice this means that a single flow must be forwarded on a single member link and can't be spread across all of them. As a consequence, although an aggregated link may have enough capacity to forward a new flow, none of the individual member links could accept this flow without compromising existing flows. Therefore flows need to be dynamically re-balanced among the member links to enable uncompromised forwarding of all flows. Such an adaptive load balancing policy for link bundles would be required to evenly distribute data flows across aggregated member links considering connectionless and connection oriented traffic. The balancing algorithm would determine which link to use by considering the scanned packet or bit rate associated with the mapping to a given link. As a trigger to start re-balancing, traffic statistics can be used considering QoS policies. A balancing policy should be based on:

- percentage of allowed deviation, e.g. in the rates among the members of the link bundle. When the tolerance is exceeded, the RMT should perform an adaptive update to re-balance flows.
- scan-interval at which to check the tolerance deviation to adapt quickly to changing conditions but avoiding too frequent RMT interventions consuming computing resources.
- Scan traffic in bits or packets per second, understanding which is more adequate

The second case study is related to a routed network in between two IPC Processes whereby different routes can be chosen to transport the flow. To cover variable bandwidth needs, a mechanism is required allowing a DIF with an established connection to automatically adjust its bandwidth needs based on the volume of traffic flowing through the connection. Again, bandwidth allocation needs to be adjusted according to a specified time interval and considering QoS needs. A possible policy could check at the end of the time interval whether the current maximum average bandwidth usage is in line compared with the allocated bandwidth for the connection. If the connection needs more (or less) bandwidth, an attempt can be made to service the higher/lower demand accordingly.

In contrast to the first study, the bandwidth reservation is dynamic and involves modification of the route of a connection between IPC Processes. In setting up the connection, a negotiation process among all resources along the path is required to reserve adequate capacity along the chosen route, which may well not be the shortest path between two IPC processes. The negotiation process consists of two information exchanges:

- an exchange of information about the amount of resources available in the DIF for every QoS cube. This serves as a basis to find a resource efficient route.
- a resource reservation process that allocates resources along the chosen path. If the resource is allocated successfully it results in an update of the first information

reflecting the new resource situation within the DIF. In case it was rejected, the originator of the request shall be notified and may issue an alternate request to satisfy the demand.

A consideration to be made is related to the amount of already established connections in a DIF. As a DIF may have different connections of finite resources established to reach a certain end-point, a balancing mechanism would need to map flows onto those existing connections. This balancing policy appears to be similar to the first case study and should be validated for suitability.

2.3. Strategies for multiplexing, scheduling and forwarding within a DIF

This section analyzes different approaches for multiplexing, scheduling and forwarding different types of traffic within a DIF, discussing potential policies for the relevant RINA components that explore different possibilities of the problem space. D3.1 introduces the different approaches, discusses its mapping to RINA components as well as its applicability in the project use cases. The next steps will be the simulation of the different policies and the selection of the most promising ones for implementation

2.3.1. Scheduling for resource allocation with multiple levels of QoS

Scheduling is the method by which data flows are given access to system resources. This is usually done to load balance and share system resources effectively or achieve a target quality of service (QoS) [Parekh]. The term scheduling discipline is used when referring to the algorithm used for distributing resources among parties, which simultaneously and asynchronously request them. Most common used disciplines are First In First Out (FIFO), Round-Robin (RR) and Fair Queuing (FR).

In the case of QoS support, a traditional approach is to assign priorities to the distinct QoS classes (strict or probabilistic), such as in fixed-priority pre-emptive scheduling and weighted fair queuing [Ajmone02] schemes. However, although they can provide some QoS differentiation, they do not tightly adapt to the specific flow requirements encountering unforeseen performance issues, become unable to reliably predict network performance and customer experience, and fail to extract the full efficiency gain of statistical multiplexing [Ajmone03].

In scheduling, assigning a higher priority to a flow implies serving its packets before than others’, thus keeping the occupation of the allocated queues low. Indeed, the assigned priorities have an impact on flow QoS experience, particularly on the experienced losses and delay. If we consider these two parameters separately (and not strictly related as in traditional approaches), finer distinction of QoS classes can be achieved. For example, we could configure QoS classes desiring low delays but allowing more losses, others targeting lower losses but allowing some additional delay, etc. Such a distinction between delay and loss offers an additional degree of freedom to the scheduling discipline, and thus providing better discrimination between different flow requirements in terms of QoS.

Working toward an efficient scheduling discipline to be incorporated in RINA IPC processes that effectively accounts for both delay and losses, we will analyse and implement an algorithm based on the ΔQ approach [Davies]. ΔQ is a scheduling model proposed by the company Predictable Network Solutions, based on the idea that a triad of parameters (bandwidth usage, delay and losses) determine the behavior of flows and, by fixing one of these parameters (bandwidth typically), it is possible to provide efficient treatment and predictable QoS experience to the others.

QoS parameters in ΔQ

Adapting the different queues and priorities to the outgoing port state, a scheduling based on ΔQ can distinct two different QoS parameters, delay and losses.

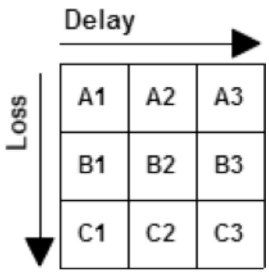


Figure 5. ΔQ QoS queue matrix

The available QoS classes should depend on the level of confidence and control in the network. In a highly controlled network, where we can trust the resource request (i.e. it demands the actual required QoS level), any combination of delay and losses may be admitted. On the other hand, a network with less control on the resource request, QoS would be more likely restricted to some trade-off strategies

Hypothesis

We assume that:

- Packets are marked with the QoS class type (QoS cube) and the destination identifier

- The Resource Allocator (RA) assigns a set of different flows, which may have different QoS class type, to the same output port according to the QoS expectations for these flows
- The RA creates and maintains a matrix of queues per each output port
- The Relaying and Multiplexing Task (RMT) selects the output port and the queue for each incoming packet according to the destination address and the QoS class type

RA and RMT roles in scheduling

In RINA, the RA is the main actor responsible for resource reservation to flows and for the configuration of scheduling policies. The RA manages all the resources of the IPC process and has to assign them to the different queues of each QoS class. The RA needs to take into account all ports and resources reserved and available in order to perform a fair and useful distribution of resources between the different queues in use.

In addition, the RA has to regularly update the different parameters used by the scheduling discipline in order to adjust the policy to the current state of the network. For example, the current bandwidth consumed by each QoS class for each port can be measured and these updates can adjust the different ranges and priorities to be used accordingly.

The RMT module is responsible for executing the scheduling algorithm employing the different policies set by the RA. The scheduling algorithm decides, depending on the different QoS classes and the state of the queues, from which queue take (serve) the next PDU when an output port is available. The proposed scheduling policy for PRISTINE considers 2 different properties for each QoS class given the current state of its queue:

- The **Urgency** of the QoS class: some QoS classes should have strict priority with respect others (e.g., priorization of routing updates and similar messages). Moreover, depending on the state of its queue, a QoS class may require to be served before other QoS classes, e.g., to ensure the targeted packet loss level.
- The **Priority** of the QoS class: a non-strict priority given by the QoS requirements and the current state of its queue.

The RA has assigns an Urgency@Priority pair to each QoS class, based on the current occupation of the queues. In this way, the scheduler can take these values to rapidly serve PDUs from the different queues in the following way:

1. Use the Urgency of the QoS classes to prune the set of QoS queues, leaving only those queues that need to be served quickly (they have the highest Urgency).

2. Taking the selected QoS queues in the previous step, use the Priority of the QoS classes to serve one PDU from one of them (QoS queues with higher Priority are more probable to be selected).

Example: QoS scheduling under different congestion levels

Consider a QoS policy given by the following 2x2 matrix:

A1 Low Delay + Low Losses	B1 High Delay + Low Losses
A2 Low Delay + High Losses	B2 High Delay + High Losses

Figure 6.

Let assume that the RA assigns the following pairs of default Urgency@Priority for the different states to the different QoS queues (higher value means higher urgency/ priority):

	Low usage	Medium usage	High usage
A1	1@8	1@10	2@12
A2	1@8	1@8	1@10
B1	1@2	1@5	2@8
B2	1@2	1@2	1@5

Figure 7.

No congestion

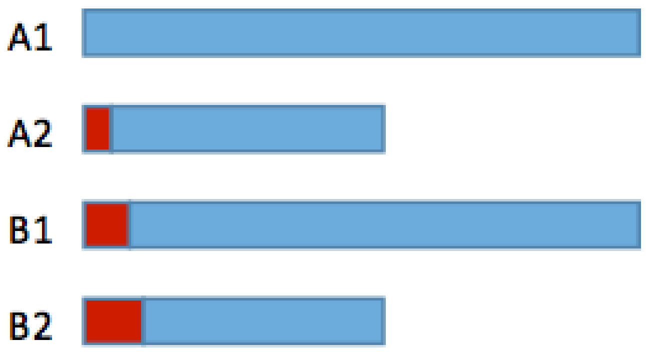


Figure 8.

When no congestion affects the current node, all queues remain almost empty, so the scheduling applies the default values. As queue A1 is empty,it is not considered. For the rest of queues, the priority for Low Delay is set to 8, while for High Delay it is set to 2. Therefore, the scheduling policy assigns service probabilities as follows:

A1 : 0	B1 : 2/12
A2 : 8/12	B2 : 2/12

Figure 9.

Low congestion

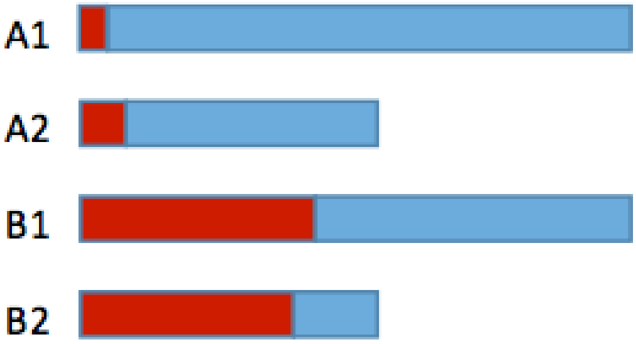


Figure 10.

Queue B1 is starting to be filled and it has Low Losses as a QoS requirement. Nonetheless, since there is no imminent risk to be congested, it keeps the default Urgency value.

Queues A1 and A2 remain almost empty since they are served quickly due to their priority of 8 (as in the previous case). On the other hand, queue B1 is in the risk of having possible losses. Therefore its priority is incremented to 5. Finally, queue B2 shows a similar occupation as B1, but it does not require Low Losses. Therefore, it maintains its priority to 2. Taking all this into account, the scheduling policy assigns the probabilities as follows:

A1 : 8/23	B1 : 5/23
A2 : 8/23	B2 : 2/23

Figure 11.

High congestion

Queues A1 and B1 are almost full and at risk of having imminent losses. Given that, both increment their Urgency to 2. On the other hand, queues A2 and B2 admit losses and therefore their urgency remains 1.

At the same time, since they have imminent risk of losses, A1 increases its priority to 12, while B1 to 8. A2 and B2 are not even considered at this point, given that their urgency is 1. Thus, the scheduling policy assigns probabilities as follows:

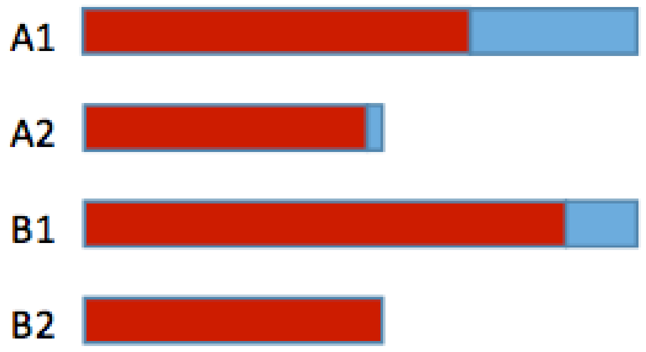
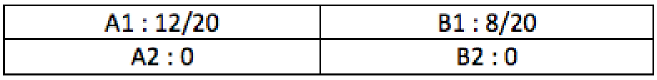


Figure 12.



A1 : 12/20	B1 : 8/20
A2 : 0	B2 : 0

Figure 13.

Analysis of the RMT policies to be adapted for scheduling

According to this idea, the relation of the scheduling with the RMT policies is shown in the following figure.

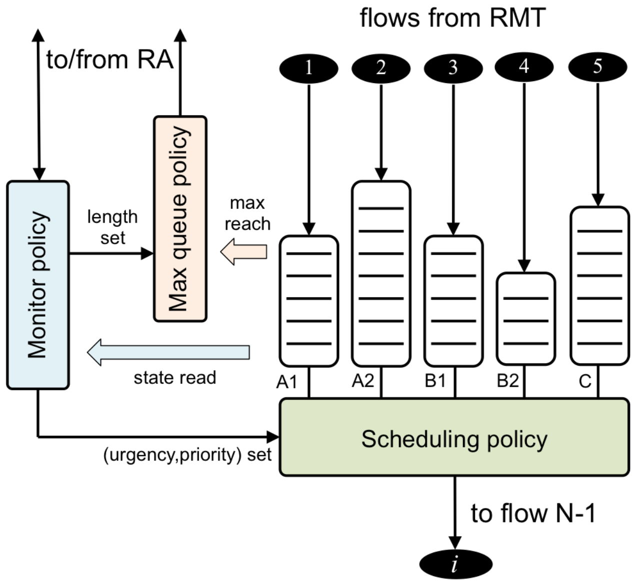


Figure 14. Relation with the RMT policies

- **RMT scheduling policy**. Scheduling uses a queue for each pair output port + QoS cube, whose sizes are managed by the RA. For each queue, the scheduling policy maintains a triad of values, keeping the state of the queue (e.g. occupation of 0%, (0-30]% , (30-60]% , (60-90]%, (90-100]%), Urgency of the queue and Priority. The maximum Urgency among all output queues is also kept in order to speed up the scheduling decisions. The scheduling policy consider two different events:

  - PDU delivered on (N-1)-output port: When a PDU is delivered to an output port, if the port is ready to serve (and there are no other PDU waiting to be served) the PDU is served directly. Otherwise it is stored into the queue for its QoS cube. If the queue is full, then the PDU is dropped by default. A variation here could be to trigger the increase the size of queues for QoS cubes requiring low losses.

  - (N-1)-output port ready to serve: When an output port is ready to serve, the queue from which the next PDU will be served is randomly selected according to their Priority. Only queues matching the maximum Urgency for the port are considered at this moment.

- **RMT monitor policy**. Each time a PDU is inserted or taken from a queue, the RMT monitor policy has to update the values used by the scheduler if needed. This policy only acts when the state of the queue changes. For example, according to some criteria, it can decide to update the triad of values assigned to the queues given the new state. Definition of states and the pair Urgency@Priority for each state is done by the RA and updated depending on the current resource usage. If the Urgency of the queue varies, the maximum urgency for the output port is also recomputed.

- **RMT maximum queue policy**. When a PDU that is inserted in a queue causes that a threshold or the maximum queue length is reached, this policy receives a notification. Such a notification can be then used by the RA to, for example, adjust the length of the queue, reduce the incoming flow rate or re-allocate the flow to a different output port.

In summary, while the scheduling policy focuses on making fast decisions, the RA is in charge of the resource management. All decisions about incrementing buffer size for the queue or update scheduling parameters given the current states of the queues depend completely on RA policies.

Next steps on scheduling

ΔQ has shown good results in controlled environments, but in real cases, like those considered in PRISTINE, the environment does not tend to be so controlled and reliable [PNSol]. Thus, in the previous section, we have introduced some modifications to the original ΔQ scheduling approach so as to permit it to react and adapt the policy to unexpected changes.

In the forthcoming months, we plan to test and verify these modifications through extensive simulation analysis. Different granularities in losses and delay will be studied and evaluated, paying attention to which pairs of these make more sense for each

PRISTINE use case. Once the QoS classes will be defined, different configurations will be tested for different static scenarios (each QoS class using some predefined bandwidth). Taking the outcome of these tests, the RA policy will assign the initial parameters to the RMT scheduling. Into operation, the scheduling policy will monitor the state of the queues and dynamically adapt their Urgency and Priority accordingly.

Additional work will consider more complex scheduling disciplines where distributed decisions can be taken to guarantee multiple levels of QoS. For example, one possible approach to be explored allows the possibility of aggregating specific flows in a higher level entity (which can be referred as channel, virtual connection or flow concatenation) and allocates to it dedicated and fixed distributed resources along the path (either inside a single DIF or inter-DIF). Another approach can explore the possibility of adding information to some packets in order to inform the downstream nodes about the level of accomplishment of the QoS level of a given flow and thus, such downstream nodes, can schedule the packets of this flow in a different manner to compensate the QoS expectation.

2.3.2. Load balancing EFCP connections for the Datacenter Networking use case

In current datacenters the traffic is mainly exchanged among servers and just very little go outside the DC. Some researches show that the amount of intra DC traffic is around 80% of the total DC traffic [benson], and no more of the 20% of the total traffic leaves (or enters) the DC.

To spread the traffic among the datacenter network for incoming, outgoing and internal traffic, load balancing mechanisms are used such as ECMP (Equal Cost Multi Path) and VLB (Valiant Load Balancing) [greenberg]. These mechanisms are based on the assignation of random paths to the different possible routes from a certain source and destination.

The disadvantage of these random approaches is that the probability of assigning paths sharing links is not negligible, thus causing collisions. The next figure shows the case in a fat tree datacenter network, in which server A connects with server C and server B connects with server D. The paths assigned happen to share the link in red, so the total throughput among the servers is reduced to the half.

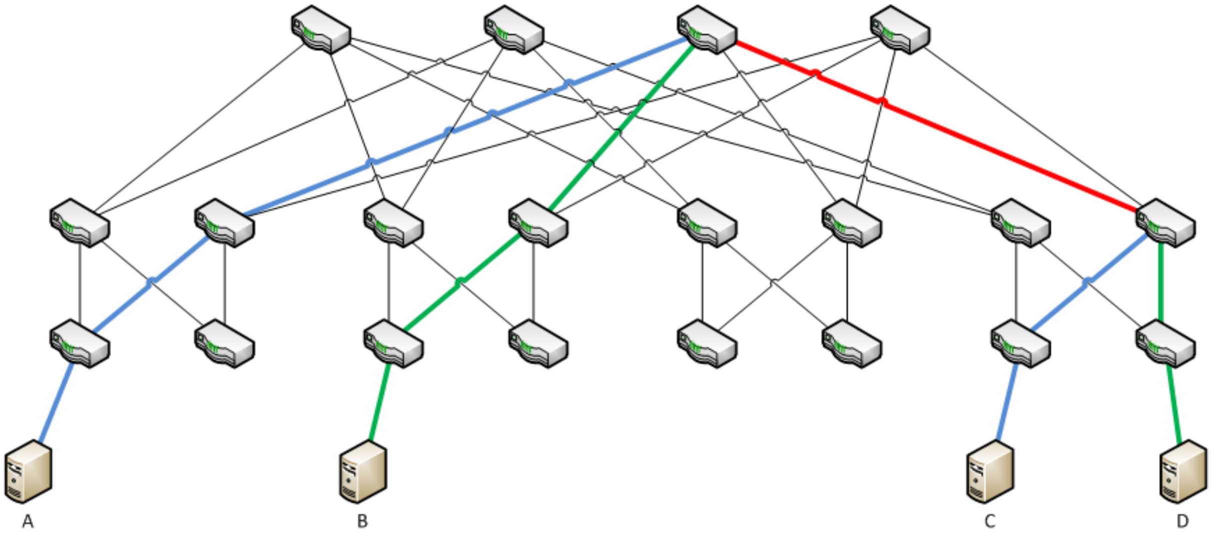


Figure 15. Collisions on path assignment

The study in [dixit] describes the performance of ECMP on a fat tree topology. The study overloads the topology at maximum, with all servers transmitting at full rate. Theoretically, the fat tree has enough capacity to serve all the traffic at full rate, but the inneficiencies of ECMP are shown in the following figure (taken from [dixit]), in which we can see that only around 30% of the links are used at full capacity while the rest of the links are widely under-utilized.

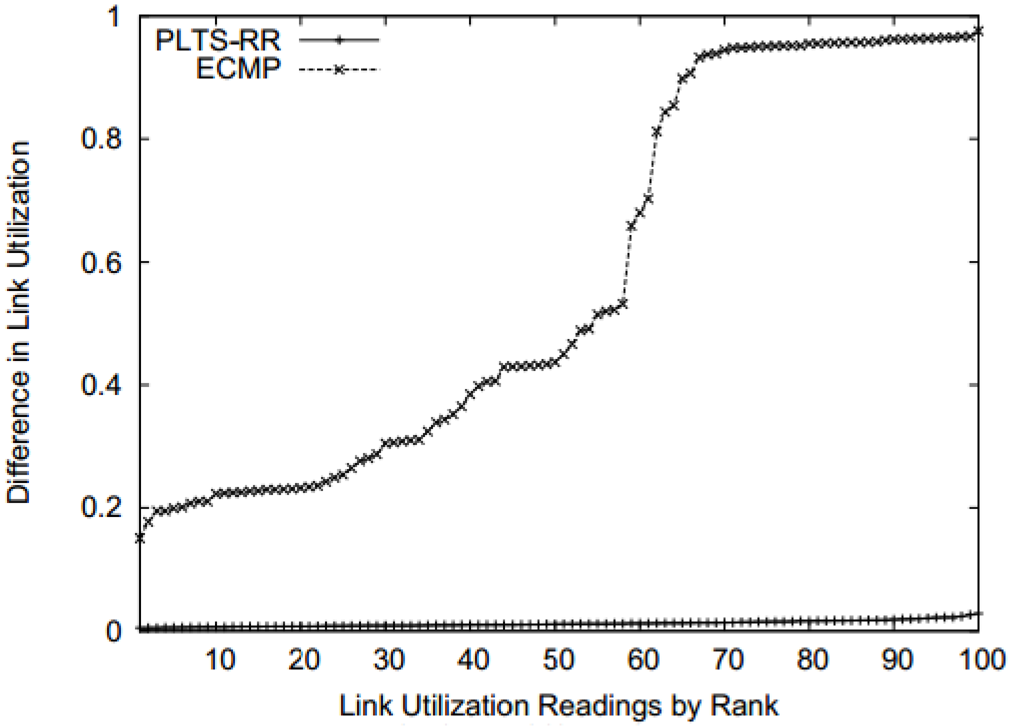


Figure 16. ECMP performance

Some approaches are designed to overcome this issue. One typical example is Multi-path TCP, in which the traffic is spread among sub-flows over different paths. In this way, connections through overloaded links can be rerouted through under-utilized links. The following figure (taken from the work in [raiciu]) shows the MPTCP performance on a similar scenario than the previous case. We can see that the overall throughput, measured as percentile of the optimal, achieve much better results, but still far away from the optimal.

MPTCP is still relaying in the traditional Internet architecture and still burdens its drawbacks, especially those posed by TCP, thus providing a sub-optimal solution for the resource allocation in a datacenter network. More specifically, MPTCP improves the resource allocation by means of opening new sub-flows and sending data through it when a TCP flow gets congested (this also involves performance degradation while the TCP flow reaches the congestion point and MPTCP opens new sub-flows). Due to the decoupling of the TCP (transport) and IP (routing) layers, an optimal resource allocation cannot be achieved by MPTCP since collisions cannot be avoided, i.e. the random routes commanded by ECMP (Equal Cost Multi-Path) will clash and thus degrading the performance.

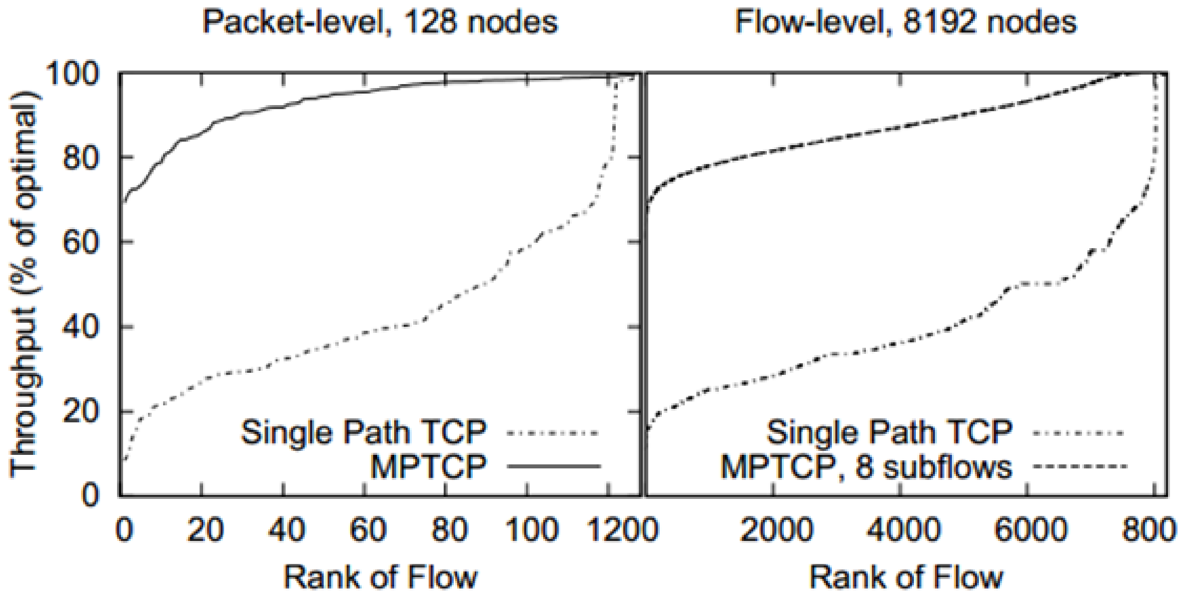


Figure 17. MPTCP performance

With RINA we intend to overcome the issues posed by MPTCP. Since MPTCP represents the state of the art in this matter we will take MPTCP as the performance goal to overcome and we will compare RINA's performance with it. We foresee to achieve a better performance with RINA than with MPTCP since the resource management system will be able to react and reallocate resources before the links get congested. At the same time, as RINA do not separate routing and transport, more efficient resource allocation and optimization mechanisms will be developed.

In fact the MPTCP approach is not required in RINA. MPTCP provides advantages over plain TCP in the Internet because IP addresses are path-dependent and assigned to interfaces of a node, not to the node. A TCP connection defines an association between two interfaces of two nodes (not between two nodes), and fixes the path through which the packets have to arrive at the destination node. If the interface chosen by the TCP connection fails, the TCP connection will fail - even if the node would have other interfaces available. This is the same reason why the packets that leave the source node can only use a single interface (the one identified by the source IP address). MPTCP solves this problem by establishing different TCP connections (called sub-flows) between the same pair of nodes - each sub-flow usually between a different pair of interfaces - and combining the sub-flows into a single flow which is presented to the application.

However EFCP connections in RINA don't suffer from this problem, since they define an association between two nodes (IPC Processes). Addresses in a DIF are assigned to IPC Processes, and are therefore path-independent. This means that PDUs en route to a destination IPC Process can reach it via any N-1 flow (interface) of that IPC Process. Nothing special needs to be done to support multi-homing, since the path to reach a destination IPCP is only chosen by IPCPs that are its nearest neighbors (if a certain path to a neighbor becomes unavailable, its nearest neighbors will quickly detect it and update their PDU forwarding tables).

PDUs of an EFCP connection can also exploit multiple simultaneous paths between source and destination IPCPs. In order to spread the PDUs belonging to an N-EFCP connection amongst one or more N-1 flows in RINA, an adequate PDU forwarding policy must be designed. When PDUs of EFCP connections are passed to the RMT, it queries the PDU forwarding policy to obtain the N-1 port(s) to which the PDU has to be written. If the PDUs of a given flow must be balanced between multiple N-1 flows, it means that:

1. There is more than one valid N-1 flow to reach the next hop.
2. Each valid N-1 flow is assigned a weight that controls how the load of the EFCP connection is spread between the multiple N-1 flows.
3. The PDU forwarding policy executes a function for each PDU that returns the N-1 flow to which the EFCP PDU has to be written.

3. Topological Addressing to Bound Routing Table Sizes

3.1. Introduction

Over the last three decades, the Internet has experienced an exponential growth, interconnecting millions of computers around the world. Moreover, there has been an increasing desire of customer networks to be multi-homed (even modest sized businesses): connected with more than one connection at the same time, which facilitates load-balancing and increases reliability by avoiding single point of failure network connectivity. The large number of new users joining the Internet every day, compounded with the vast multi-homing (and provider-independent address) demands have imposed dramatic stress to the Internet routing system scalability, up to the point that the Internet Assigned Numbers Authority (IANA) Central IPv4 Registry was exhausted in January 2011. That was the first milestone of the IPv4 address pool depletion, which has been followed by the exhaustion of the Asia-Pacific (April 2011), Europe (September 2012) and Latin America (June 2014) Regional Internet Registries (RIRs).

Furthermore, the proliferation of the smartphones, laptops and other mobile devices is already bringing a huge amount of new hand-held terminals into the Internet, sending and receiving large amounts of traffic that has to be efficiently and uninterruptedly handled, even during handover operations over heterogeneous network technologies. IPv6, by changing the address length from 32-bits to 128-bits, is bringing (theoretically) the solution to the inevitable depletion of the pool of unallocated IPv4 addresses. Nevertheless, IPv6 lacks a clear deployment roadmap. In fact, current network operators do not seem to be interested to assume the costs of upgrading their infrastructures (it pays very little to those who have to assume its adoption when compared to IPv4). Moreover, the scalability requirements that a heavily-used IPv6 address space could cause to network devices are also under discussion.

All the aforementioned scalability limitations of the current Internet routing system fundamentally stem from its incomplete naming and addressing scheme. Specifically, the Internet architecture does not provide separate names for the key entities in the architecture, namely, nodes, Points of Attachment to the network (PoAs) and applications. In fact, the only names that are provided are PoA names (IP addresses). Although commonly referred to as “host addresses”, they are interface addresses in reality. Therefore, the network has no means to find that multiple IP addresses of a multi-homed node belong to the same node, thus making multi-homing hard to realize.

Besides, naming the interface instead of the node, forces the Internet to perform routing on the interface level, contributing to unnecessarily large routing tables. This problem became apparent in 1972 already, when Tinker Air Force Base in Oklahoma joined the Net and wanted two connections for reliability. However, it was not changed, since the low penetration of multi-homed access at that time was not considered an issue to the architecture (but it certainly is right now). In addition, mobility, which can be seen as dynamic multi-homing with expected failures, is another feature that suffers from having an incomplete naming and addressing scheme.

3.2. Enhancing routing scalability with RINA

Jerry Saltzer in [Saltzer] documented the entities that make a complete naming and addressing scheme in networks, namely, applications, nodes, PoAs and paths. Any naming and addressing scheme missing one or more of these entities, such as the one of the Internet that only names PoA and routes (it misses application names and node addresses), is incomplete and most probably can lead to increased complexities and scalability problems later on.

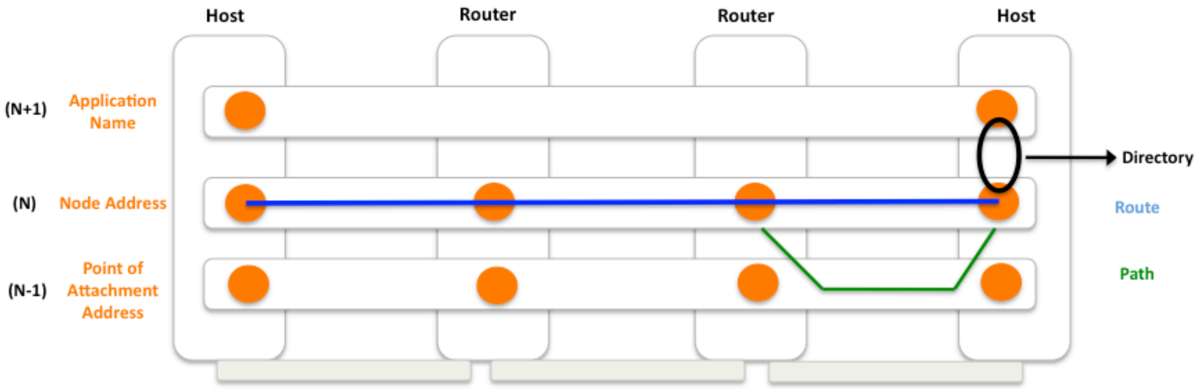


Figure 18. Complete naming and addressing scheme entities according to [Saltzer].

In this framework, application names should be location-independent, allowing them to move from one node to another without loosing their identity. Conversely, node addresses should be location-dependent, but route-independent. In other words, while application names should identify the *what*, addresses should identify the *where* without being tied to the network connectivity graph, as it may change too often (e.g., upon failures, new nodes entering the network, old nodes leaving it etc.). This means that the address space should describe some sort of abstraction of the connectivity graph, where node addresses indicate location but contain no information about how to get there. Routing here can be seen as a two-step process: first, we have to calculate the route, which is a sequence of node addresses (routing is performed to the node instead of to the interface as in the current Internet); then, at each hop, we then choose the appropriate PoA to select the specific path to be traversed.

RINA adopts Saltzer's complete naming and addressing scheme, while bringing it one step further. As RINA layers (DIFs) are recursive, the roles of application, node and PoA are relative to their position in the stack of layers. For example, given a DIF at level N (N-DIF), the IPC processes at level N+1 are application processes to the N-DIF, and the application processes at level N-1 are PoA to the N-DIF. Moreover, in RINA all application processes (including IPC processes) have a name. Within a DIF, each IPC process is assigned a synonym, i.e., an address, whose scope is restricted to the DIF and not visible outside. This has a strong impact on the scalability of the addressing in the architecture, as different DIFs can reuse addresses to identify IPC processes (RINA divides the global scope of the Internet into multiple scopes, completely independent one from another). Moreover, IPC process addresses are optimally generated for routing inside the DIF, thus allowing for efficient and scalable routing schemes, tailored to the particular DIF characteristics.

Topological address spaces

As mentioned above, node addresses in a network should be location-dependent but route-independent. Both characteristics become important toward the design of efficient routing algorithms. Firstly, location-dependent node addresses give us hints in the forwarding decisions in order to select the next hop closest to the destination, up to the point that routers may only need to store the addresses of their adjacent neighbors in their forwarding tables. Of course, this is the ideal scenario (more information may be needed depending on the particular network graph or if connectivity distortions appear). Secondly, route-independent node addresses are not tied to the underlying network graph and do not change if this graph changes. In other words, route-independent node addresses are invariant to the preferred routes to reach them, which can change frequently. This dramatically facilitates multi-homing and mobility; both are cumbersome in the current Internet, where route-dependent (interface) addresses are employed.

A key question that arises here is the following: how do we come up with location-dependent and route-independent node addresses given a certain network graph? At first sight, it seems that an abstraction of the network graph that remains invariant to changes on it is what we need. In this context, the area of the mathematics that concerns abstractions of spatial relations and graphs and properties of invariance is topology. Hence, if we consider addresses to have topological properties (i.e., we use topological addresses), we may be able to create location-dependent and route-independent addresses.

Without entering into the mathematical details, a topological address space is an address space that has a topological structure. Specifically, the topology of the address space maps the elements of the address space to the elements of the network graph through a homeomorphism, that is, a continuous function between topological spaces that is one-to-one and has a continuous inverse function. Any topological space may be metrizable through a distance function and/or have an orientation (partial ordering of the elements). Also the granularity property is interesting in a topological space, which denotes its resolution: topologically speaking, which elements of the topological space can be considered at distance 0 between them (they are "in the same place").

In general, the effectiveness of the routing and resource management for the layer can be greatly enhanced if the topological address space is metrizable and has an orientation. Moreover, better routing scalability can be achieved with coarser granularity, but routing will probably tend to be less optimal, so there is a clear trade-off in this regard.

There are many useful topologies that can be endowed to the (topological) address space, such as grid topologies like the street addressing plan in many US cities, star topologies, hierarchical topologies, meshed topologies etc. All of them can be used to describe sets of graphs with certain invariant interconnection properties. The primary question here is what topologies for address spaces make sense, are easily maintained, scale, have nice properties for routing, and so on. That is, we want to find topologies of address spaces based on the abstractions and aggregation and the topologies of subnets without tying them to the physical topology of the network but at the same time providing a convergence to that physical graph. However, if the graph of the network differs significantly from the topology chosen for the address space, the wrong topology has most probably been chosen. In general, for an effective and scalable routing scheme, the topology of the address space and the graph of the network need to be worked out together.

3.3. Initial specification of scalable routing schemes for RINA DIFs

3.3.1. Dynamic topological routing

This routing solution leverages a single-layer topology of regions (disjoint sub-networks) and targets medium-sized and highly-connected dynamic DIFs, like that in the PRISTINE Distributed Cloud use case. All IPC processes in the DIF are equal in terms or routing and the (N-1)-DIF flows connecting them are divided between in-region and out-region links, depending on whether they provide connectivity to IPC processes belonging to the same or other regions in the network, respectively.

Region assignment is a key issue here, and should be performed in conjunction with the address assignment when a new IPC process enters the DIF. In RINA, the Namespace Manager (NSM) is the responsible for assigning valid addresses to the IPC processes (during the enrollment process) for their operation within the DIF. Hence, it seems a good candidate for managing the region assignment as well, e.g., encoding this information in the address assigned to the new IPC process (i.e., making addresses to be location dependent). While regions could be randomly assigned, it is much preferable to assign the same region to IPC processes near each other, for some definition of “nearness”. In this way, we avoid communications between near IPC processes to travel unnecessarily long paths. To achieve this, information about the “location” of the new IPC processes is required (e.g., (N-1)-DIFs to which they are registered, geographic location, etc.). This information should be made known by the NSM by any means.

Each IPC process needs to maintain reachability information to all the remainder IPC processes inside its region, as well as to the other regions (the granularity of the inter-region routing is the region). For that, it maintains two routing tables, namely, an intra-region one and an inter-region one. Updates of the inter-region routing table are distributed to all neighbours. Conversely, intra-region routing table updates are only distributed through in-region links, leaving IPC processes outside its region unaware of the region’s internal topology.

The protocol uses a distance-vector approach to populate the routing tables. Given the network graph described by the (N-1)-DIF flows, different routing policies (e.g., to be assigned to different QoS classes) could be realized by employing different measures of distance, like bandwidth available, hop count, delay, and so on. It shall be mentioned, though, that IPC processes need to maintain one intra-region and one inter-region routing table for each distinct measure of distance, although they do not need to be simultaneously updated. As routing tables should be relatively small, for each entry we can store the k best next hops (for some k > 0). Moreover, in order to avoid loops, the split-horizon technique is applied to routing updates (as typically employed in RIP protocol configurations), informing of the cost of the secondary option when sending routing updates for the primary option.

Each IPC process is assigned a hierarchical address, defined as the sequence of the region identifier and the node identifier inside the region. This hierarchical address is sufficient to route to any destination IPC process. Routing is performed to the node identifier of the destination IPC process if within the same region, or to the region identifier if within another region.

The following figure gives an example of the hierarchy defined and how routing is done. IPC process A is assigned to region 2 and is given the id 3 in that region. So, its address is A<2.3>. If IPC process B<2.5> wants to deliver a PDU to A, it will forward the PDU to IPC process C<2.4>, assuming that it is the next hop in the shortest path to A<2.3> in the intra-region routing table. In contrast, if the destination IPC process belongs to a different region, the forwarding is done to the next hop of the shortest path to the destination region. Note here that different IPC processes within the same region may not share the same gateway toward a certain out region. For instance, while G<4.4> would be a gateway from region 4 to 2, IPC process P<4.1> would forward to I<3.3> if it is next hop in the shortest path to region 2 (if hop count distance is employed).

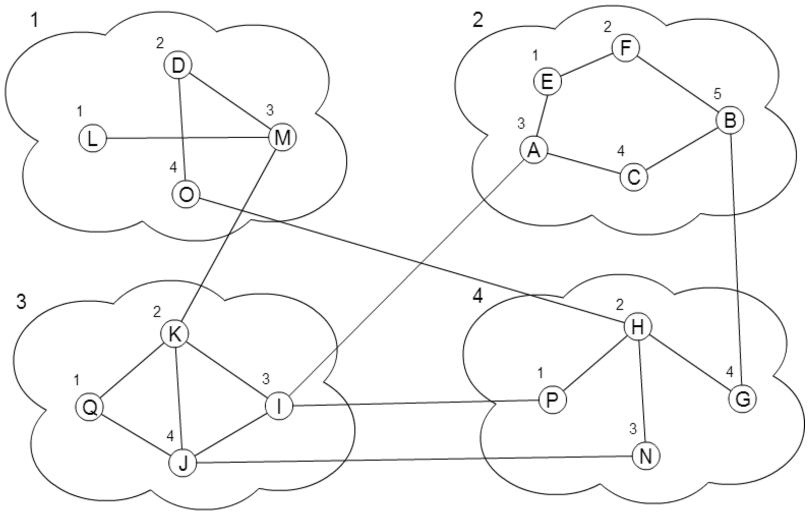


Figure 19. Dynamic topological routing example.

Like other topological routing approaches, the reduction in the routing table size can lead to an increase of the stretch of the resulting end-to-end routes (i.e., ratio between the length of the resulting routes over that of the shortest paths). In this case, a maximum stretch equivalent to the diameter of the destination IPC process region applies. In addition, we cannot argue that this solution is completely scalable, as each IPC process needs to know at least O(√n) nodes and regions (otherwise either the number of regions or the size of them increases largely).

However, these numbers are completely applicable to the PRISTINE Distributed Cloud use case. With respect to the O(√n) routing table size, the relatively small number of nodes of the Distributed Cloud use case (10k nodes) would need for each routing table to store only 100 entries, and, even under a dramatic growth arriving to 1M nodes would not be a problem, as only 1000 entries would need to be stored. The increment of the stretch could initially be a bigger problem in such a case. Nevertheless, while we cannot remove this diametric stretch, the high connectivity expected in the PRISTINE Distributed Cloud use case (each IPC process has a degree around 20 in the current implementation) can keep it to acceptable levels, especially if we ensure a good balance between in-region and out-region adjacent links.

Possible scalability improvement using Greedy Routing

The dynamic topological routing proposed has a main issue with scalability as it needs to maintain information about O(√n) nodes and regions. When dealing with highly dynamic and connected networks, some limitations arise to be deal with, which are related to increasing stretch and scalability. Given that the PRISTINE Distributed Cloud use case needs its highly dynamic behaviour to ensure a good connectivity, we cannot make long term assumptions about its connectivity graph, and so complex calculations that could improve scalability are not possible.

Using the same idea of highly dynamic and connected regions, if a minimum and static inter-zones connectivity can be assured, then it may be possible to use a greedy routing algorithm in order to perform the inter-region routing. Besides, using multiple gateways (instead of single gateway as in common hierarchical solutions), it is possible to provide high intra-region dynamism and some dynamism in the inter-region connectivity.

The greedy approach for inter-region routing implies that at each node we only need to store information about the neighbour regions to our own and the possible gateways to them. In this case, as we can have as much regions as we need, those can stay as small as needed.

Potential issues to be investigated:

- Stretch. A minimum ensured inter-region connectivity graph may allow us to do a greedy embedding that ensures that messages from one region reach any others with a minimum number of regions crossed. That means that, while increasing the inter-region connectivity may decrease the length of some paths, the static address assignment of the embedding does not allow to maintain optimal paths. In addition, as the cost is given by the number of hops between regions, the intra-region paths are not considered so the stretch may increase heavily.
- Fast and accurate embedding. Embedding approaches tend to depend greatly on the inherent topology. In this case, the intra-region graph does not needs to follow any

specific topology, so it’s possible that no good embedding is found for some inter-region graphs.

- Re-embedding. If the number of regions changes or the connectivity has a great change, a re-embedding of the network could be needed to keep a low stretch. In this case, a renumbering of all regions will be performed and it could be expensive. In addition, nodes will need to maintain a mapping between regions and their addresses, so when a re-embedding is done, all nodes needs to synchronize their databases.

3.3.2. Hierarchical Routing For Distributed Cloud

From a routing perspective, the dynamicity and the flexibility of the network are the most challenging constraints that have to be taken into account when designing an efficient routing and addressing scheme. For instance, in PRISTINE’s Distributed Cloud use case, an overlay network is set up between several servers/nodes which can be located in different areas and can be under the control of different ISPs.

Due to the large scale and the high traffic demands of such an application, the network is characterized by a high variation over short time intervals. Virtual tunnels between cloud participants change every 5 minutes and are replaced with other optimized tunnels suggested by neighboring nodes [D2.1]. This will drastically impact the routing performance as routing addresses change frequently. Building a mesh overlay network by the use of topological addresses assigned by the ISP for example would not help to deduce the adjacencies between the nodes participating in the overlay network. Notice that two adjacent nodes may belong to two distant non-neighboring ISPs.

The issue here is that the network interface is addressed, not the node. For example, if an IPC Process decides to move from an ISP to another one, its address also changes even though the node and data stored on the node do not change. This forces use of higher level "identifiers" within the mesh to retain node identity. The challenge will be to find an effective addressing scheme that could maintain a set of "multi-homed" names for each node Naturally, nodes participating in the Distributed Cloud will belong to the same DIF(level N) and have unique names. Also, since each of them belongs geographically to a specific ISP, it will be assigned an address, a topological one which is location-dependent in the DIF (level N-1). When the overlay network is built, we will need another addressing scheme to manage the participating nodes and organize routing and data exchange between them on one hand and to support scalability in the other hand.

As illustrated in the following Figure, an upper DIF ensures the continuity of node identities and builds the overlay network. This (N) DIF is built on top of (N-1) DIF that is providing connectivity to the distributed cloud participants.

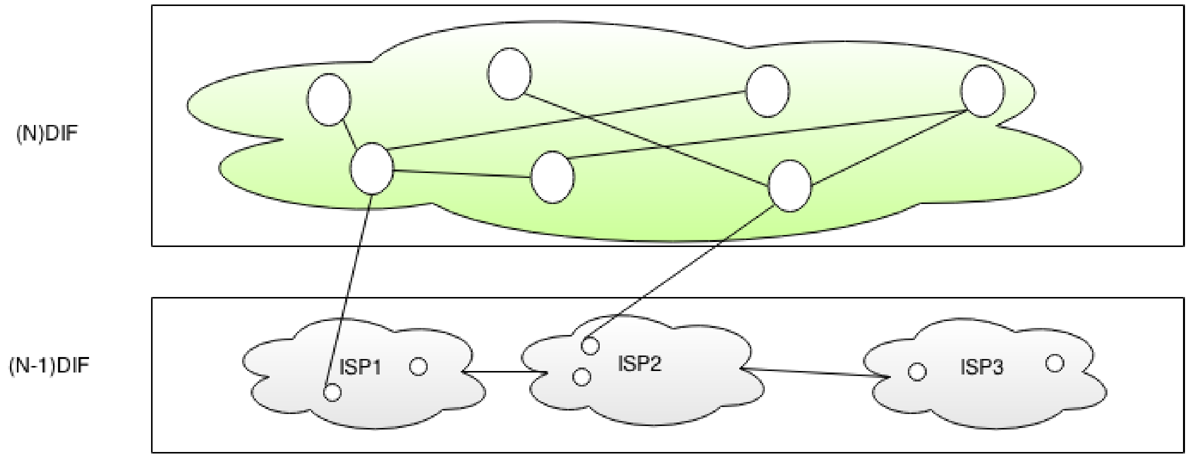


Figure 20. Hierarchical DIF Architecture.

Addressing in the (N) DIF is managed by the association <name, address>. These names must be globally unique and stay the same throughout the lifetime of the IPC process, however the address is topological dependent and should vary when the IPC process moves. In the case of Distributed Cloud the IPC movement is the result of links modifications (new tunnels are established while old ones are expired) between communicating IPCs within the same DIF. Those modifications impact immediate neighborhood and the global topology of the mesh as well.

We base our proposal on the work in [LoPumo]. As discussed above, due to the dynamic aspect of the mesh network, nodes are given unchanged names apart from the topological address. So accordingly, each node is identified by the tuple <Name, Topological Address>. A mechanism based on a distributed hash table (HDHT) is used to manage the associations between nodes names and addresses. A hash function is distributed among nodes to store the mapping between a name and its address.

1. Addressing and Routing

The protocol is based on a hierarchical topology. The network is organized in L levels. Each level is subdivided into groups bounded by a given size S. Groups are recursively subdivided in sub-groups. Moreover, the groups in all levels should be connected. An example is shown in the Figure below where two levels are defined: L-1, L-2 and the size of the group is limited to 3 nodes.

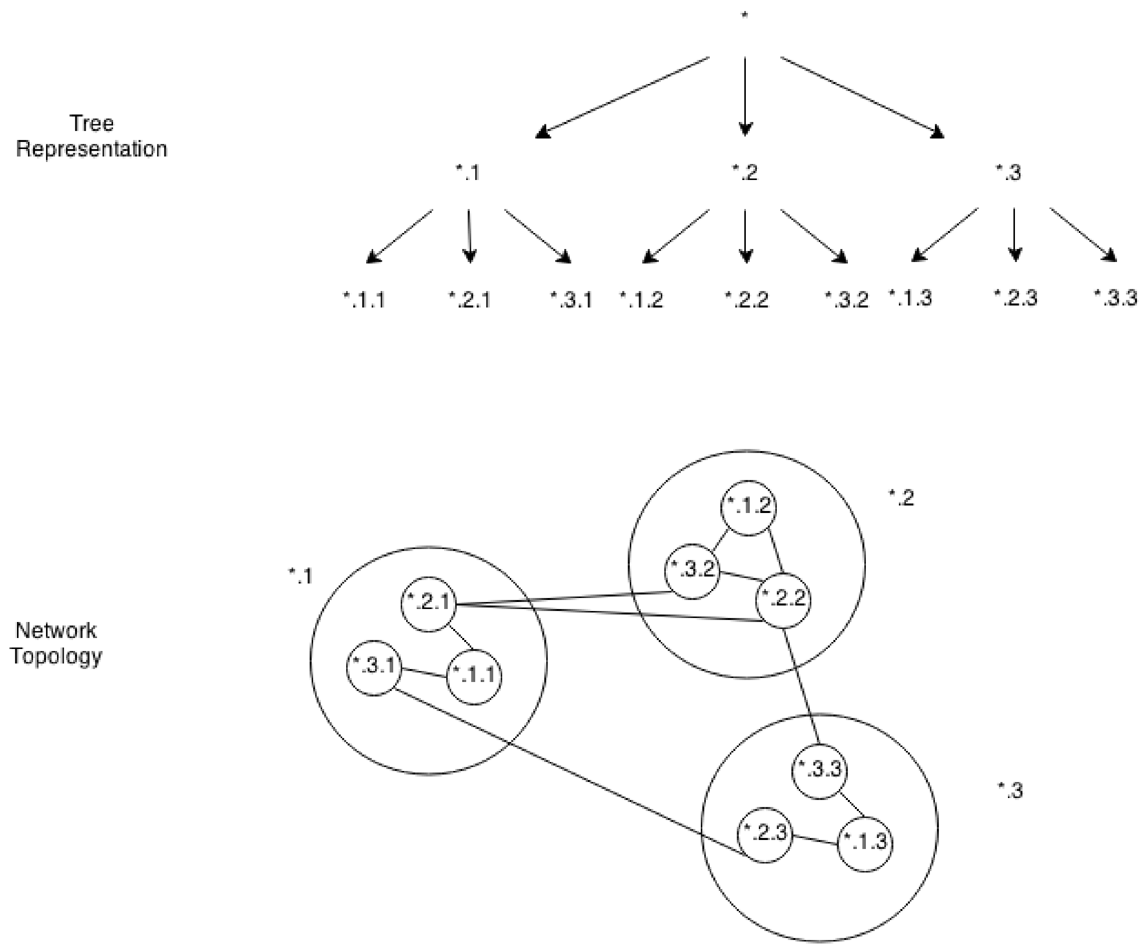


Figure 21. An example of a hierarchical network topology. Figure adapted from [LoPumo].

A node x is assigned a topological address reflecting its location in the hierarchy of the network. It is defined in this format: x0.x1….xL-1. An example is depicted in Figure 2 where three groups are defined, the node *.2.2 for instance is the node 2 in the group 2 and is the child of the node *.2 (level L-1).

Now, having the hierarchical topology set up, the routing between nodes can be done. For each level, a distributed route discovery algorithm is run in order to populate the routing tables in each group. Accordingly, to forward a packet from a node x to a node z, the node should follow the path given from the routing table. If the node z is in another group the problem is reduced to routing the packet from x to any node of the group where z belongs and then to z.

Both distance vector and link-state routing approaches could be applied for the next hop selection. Unlike Distance Vector routing protocols, Link State protocols require additional modification as the weight of the virtual link between two groups should be defined.

1. Balancing the address space

In distributed and dynamic environments, constantly maintaining a coherent hierarchy becomes a very complicated task. A node can decide at any time either to join a group or it can leave a group and can migrate from a group to another. If two nodes are disconnected, the involved nodes are forced to migrate into other groups and consequently, change their address as illustrated in the example in Figure 3. Nevertheless, nodes have not only to change addresses when a group is split, but also they have to apply additional actions in order to enter a new group. The designed protocol proposes a set of balancing rules to deal with this problem and to ensure maintaining a valid hierarchical topology.

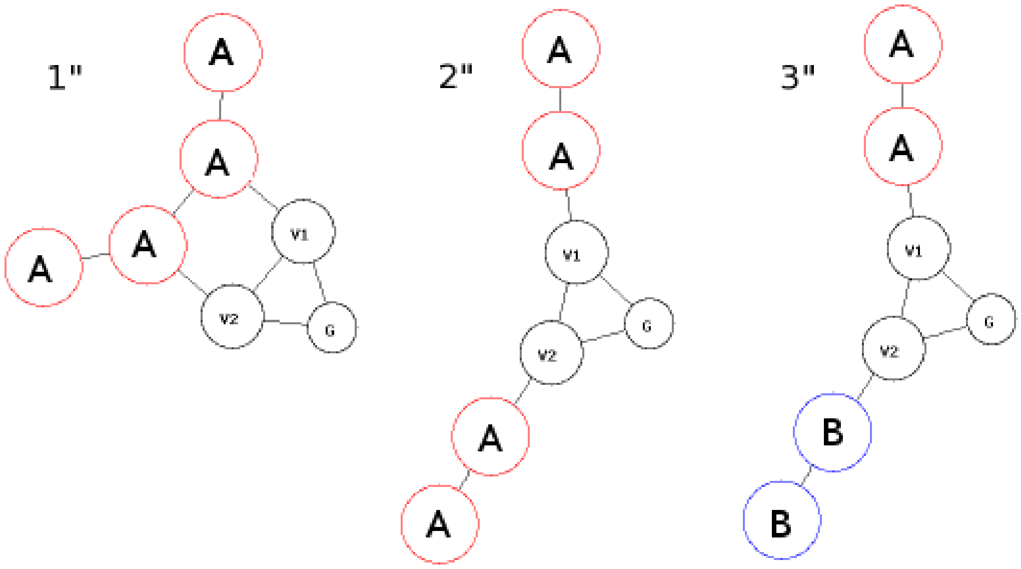


Figure 22. The removal of a link disconnects the group node A in two connected components. The nodes of one component will have to migrate to another group B. Figure taken from [LoPumo].

For instance, if the network is saturated (the size of a group is too large), the new node cannot directly join the group, a border node of the group will be forced to migrate. Furthermore, the coordination among the nodes during the process of the migration is also an issue that has to be considered. An eventual solution is to construct a distributed memory that is shared and accessed by all the nodes in the group in order to avoid conflicts (For instance in the case where two nodes decide to migrate at the same time to the group).

The association between the names and the topological addresses in this proposal helps to keep a node identified during its whole life time. As nodes move, by definition topological addresses change as the geographic location changes, however the names are kept the same. The proposal takes advantage of this topological structure of the network to assist the routing task and build small routing tables for large scale networks. The dynamicity of the nodes is considered as well, a mechanism of address balancing to maintain the hierarchical topology is applied. We believe that this proposal fits the distributed Cloud use case well as both the dynamic and the large scale constraints of such environments are covered.

3.3.3. Addressing and routing in data center network scenarios

The two routing schemes presented before target the PRISTINE distributed use case. Moreover, given their high scalability and flexibility they can easily fit large-scale network service provider scenario as well.

Apart from these scenarios, PRISTINE is also devoted to the evaluation of the RINA benefits within data center environments. Inside data centers, networks traditionally describe tiered hierarchical structures of switches. Hence, hierarchical addressing and routing schemes naturally appear to be the ideal solution in the data center substrate DIF supporting the communication between the internal data center equipment (IPC processes running on hypervisors, Top of Rack, aggregation and core switches, and border router). In some scenarios, a data center provider may own multiple geographically distributed data centers and wants to inter-connect them, e.g., to enable the migration of virtual machines. This requires the deployment of an inter-datacenter DIF on top of the data center substrate DIFs and network service provider DIF (or a TCP/UDP shim-DIF over Internet) inter-connecting them. The inter-datacenter DIF is only aware of the IPC processes on servers and border routers of each data center, which can also be effectively addressed in a hierarchical fashion for efficient hierarchical routing. Figure 23 serves as an example.

In the figure, each data center has a different identifier (from 1 to 8). In the data center substrate DIFs, IPC processes running on hypervisors, switches and border routers are addressed hierarchically, following the hierarchical structure of the data center network (the addresses in blue). In the inter-datacenter DIF, IPC processes on servers and border routers can also be addressed in a hierarchical manner, starting with the identifier of the data center where they belong (addresses in black). For example, assuming that source node 1.1.1 wants to communicate with destination node 1.1.4, it can conclude comparing the addresses that they belong to the same data center (they have the same prefix). Therefore, they can directly communicate over the data center substrate DIF. Conversely, if the same source node wants to communicate to destination node 8.5.5, it will conclude by comparing the addresses that they belong to different data centers. So, communication will be performed over the inter-datacenter DIF, through their gateways (border nodes in their data centers, with addresses 1.0.0 and 8.0.0 in the inter-datacenter DIF).

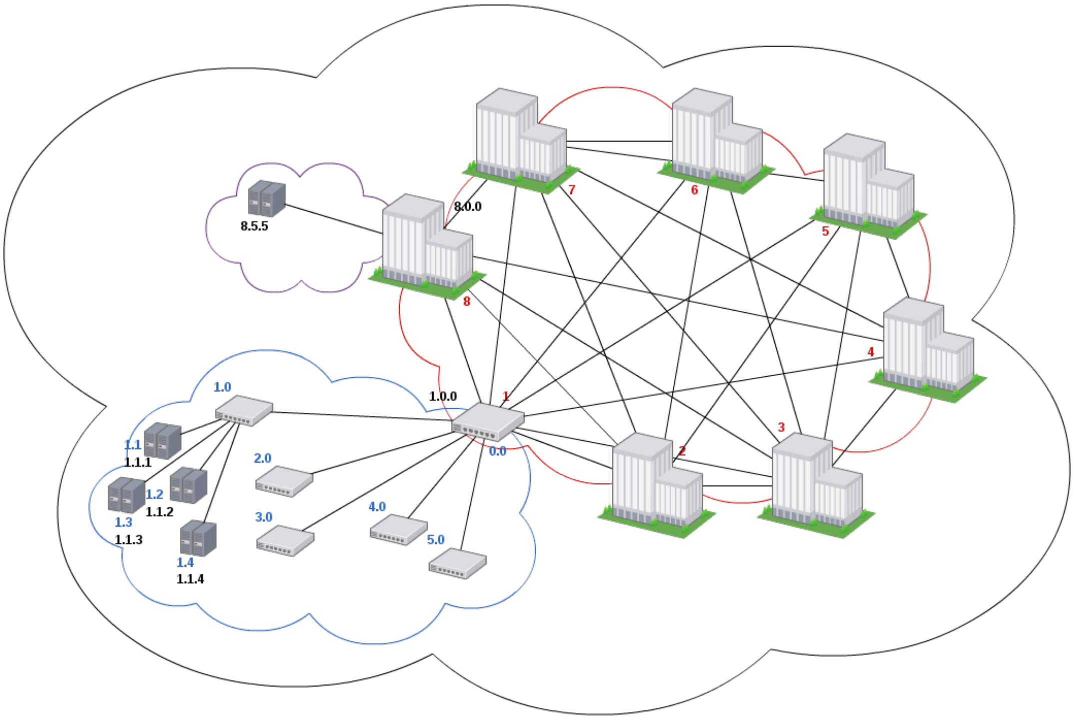


Figure 23. Example of hierarchical addressing and routing in data center network scenarios: addresses in the datacenter substrate and inter-datacenter DIFs are shown in blue and black, respectively.

3.4. Next steps

Hierarchical routing seems to be the most adapted approach to scale in highly dynamic and dense networks. Additional study should be done to analyze the suitability of this approach to PRISTINE use cases and compare it to other possible routing policies. The use of multiple routing policies could be considered within a DIF and should be studied as well.

Moreover, in order to support Quality of Service (QoS) constraints, further investigations have to be conducted to optimize hierarchical routing for instance using different metrics. The relationship between the Resource Allocation and Routing should be defined and analyzed as well.

4. References

[Ajmone02] M.Ajmone Marsan, E.Leonardi, M.Mellia, F.Neri “On the Maximum Throughput Achievable in Multi-Class Input-Queued Switches and Networks of Switches”, in Proc. Infocom 2002, New York, NY, June 2002.

[Ajmone03] M. Ajmone Marsan, M. Franceschinis, E. Leonardi, F. Neri, A. Tarello, “Instability phenomena in underloaded packet networks with QoS schedulers”, in Proc. Infocom 2003, San Francisco, March 2003.

[benson] T. Benson et al. *Network Traffic Characteristics of Data Centers in the Wild.* IMC, 2010.

[D2.1] PRISTINE Consortium. Delivrable-2.1. Use Cases Description and Requirements Analysis Report. May 2014.

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

[dixit] Advait Dixit, Pawan Prakash, Ramana Rao Kompella, “On the Efficacy of Fine-Grained Traffic Splitting Protocols in Data Center Networks”, Purdue e-Pubs, 2011.

[greenberg] A. Greenberg et al. *VL2 A Scalable and Flexible Data Center Network.* SIGCOMM, 2009.

[LoPumo] A. Lo Pumo. Scalable Mesh Networks and the Address Space Balancing Problem. University of Cambridge, 2010.

[Parekh] A.K.Parekh, R.G.Gallager, ”A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks - The Multiple Node Case” IEEE/ACM Transactions on Networking, Vol. 2, n. 2, April 1994, pp. 137-150.

[PNSol] BT Operate Case Study, http://www.pnsol.com/public/CS-PNS-2012-08.pdf

[raiciu] Costin Raiciu, Sebastien Barre, Christopher Pluntke, Adam Greenhalgh, Damon Wischik, Mark Handley, “Improving Datacenter Performance and Robustness with Multipath TCP”, SIGCOMM’11, August 15–19, 2011.

[Saltzer] J. Saltzer. On the Naming and Binding of Network Destinations. RFC 1498 (Informational), August 1993.