Deliverable-6.2

Proof of concept software for the use cases and draft report on the use cases trials and business impact

Deliverable Editors: Felicia Lobillo, ATOS, Roberto Riggio, CREATE-NET

Publication date:	31-September-2015
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, Integration, Trials, Business, Impact

Synopsis: This document reports on the results of the first round of experiments carried out using the PRISTINE SDK. Results show promising technical progresses in all areas. From the business impact perspective, a preliminary qualitative analysis based on the performed experiments has been carried out.

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

List of Contributors

Deliverable Editors: Felicia Lobillo, ATOS, Roberto Riggio, CREATE-NET
i2CAT: Eduard Grasa, Bernat Gaston, Francisco Miguel Tarzan Lorente, Leonardo Bergesio
ATOS: James Ahtes, Felicia Lobillo, Miguel Angel Puente, Javier Garcia
NEX: Gabriel Monnerat
FIT-BUT: Ondrej Rysavy
CN: Roberto Riggio, Kewin Rausch
TSSG: Micheal Crotty, Ehsan Elahi, Jason Barron
iMINDS: Dimitri Staessens, Sander Vrijders
UiO: Peyman Teymoori
NXW: Vincenzo Maffione
TRT: Hamid Asgari

Executive Summary

The first round of experiments carried out as part of the validation work of the PRISTINE SDK has shown promising results. Moreover, it also helped in identifying development issues that have been either corrected during this first development phase or that have been included in the development roadmap of RINA and the PRISTINE SDK for the second phase. Each experiment has covered different research areas (from resilient routing or aggregate congestion control to management aspects) and several RINA policies within each area.

Several of the RINA benefits demonstrated during this first cycle of experiments apply to all three use cases. For example, the Loop Free Alternate (LFA) experiments can be always used by the network administrator to increase IPC service availability. Similarly, Explicit Congestion Notification (ECN) and vertical push-back between layers enable a faster and more efficient response to congestion than the prevalent end-to-end implicit feedback approach used in the Internet. ECN allows congestion effects to be confined closer to where congestion happens instead of affecting the whole network: each DIF manages the congestion in its own resources. This allows network providers to utilize network resources more efficiently and to run networks at higher utilization.

The development, debugging and testing of policies associated to the first cycle of experiments uncovered a number of bugs in the RINA implementation as well as some changes required in the SDK (which have already been addressed and released in the most up to date development branch, pristine-1.3). There are still open issues currently reported that will be dealt with during the second phase, mainly related to design improvements or non blocking bugs.

As far as business aspects are concerned, we have performed a preliminary qualitative analysis based on single experiments and how the KPIs impact on the business process of the use cases.

The work towards the final evaluation of the PRISTINE project results will focus on two main axes. On one hand, the research areas will continue to design and carry out experiments that further validate the usefulness of the PRISTINE SDK as far as RINA benefits are concerned. On the other hand,

we intend to combine several research directions, designing and executing more complex experiments that help validate the SDK and RINA at a larger scale. We will define experiments that gather representative aspects of the three use cases, making sure that the simultaneous use of policies belonging to different research areas do not interfere with one another, but rather show at least the same level of performance they did when they were tested individually. These experiments will provide further feedback towards development regarding the combination of complex scenarios for RINA and the SDK, such as the use cases require.

Finally, we intend to carry out a higher level study focusing on the technical impact per use case and translate the results into the expected business impact.

Table of Contents

List of Figures

List of acronyms

AES	Advanced Encryption Standard
CACE	Common Application Connection Establishment
CBR	Constant Bit Rate
CDAP	Common Distributed Application Protocol
CRC	Cyclic Redundancy Check
DC	Data Centre
DAF	Distributed Application Facility
DFT	Directory Forwarding Table
DIF	Distributed-IPC-Facility
DMS	Distributed Management System
DTCP	Data Transfer and Control Protocol
DTP	Data Transfer Protocol
ECN	Explicit Congestion Notification
EFCP	Error and Flow Control Protocol
ECMP	Equal Cost Multi-Path
IP	Internet Protocol
IPC	Inter-Process Communication
IPCP	Inter-Process Communication Process
IPCM	Inter-Process Communication Manager
ISP	Internet Service Provider
LFA	Loop Free Alternates
MA/MAD	Management Agent (Daemon)
MLS	Multi-Layer Security
NM	Network Management
NMS	Network Management Service
NSM	Name-Space Manager
NSP	Network Service Provider
PDU	Protocol Data Unit
RED	Random Early Detection
RIB	Resource Information Base
RINA	Recursive Inter-Network Architecture
RMT	Relaying and Multiplexing Task
RTT	Round Trip Time
SDU	Service Data Unit
TCP	Transmission Control Protocol

ToR	Top of Rack
TTL	Time To Live
VLAN	Virtual LAN
VM	Virtual Machine
VNF	Virtual Network Function
VPN	Virtual Private Network

1. Introduction

The work achieved in the framework of WP6 during the first phase of the project contemplates several basic experiments covering different individual research areas. The experiments show promising results as far as the benefits expected from RINA and the SDK are concerned. Some blocking issues have also been reported and conveniently corrected whereas design enhancements have been included in the development roadmap.

Concerning business aspects, an initial analysis has been provided for two of the three use cases. As reported in D6.1 "First iteration trials plan for System-level integration and validation" [D61], the Network Service Provider use case analysis has been postponed for the final phase.

The next steps for the validation of RINA and the SDK imply further experimentation within individual research areas but also cross-research areas in order to better serve the different use cases, in which RINA policies belonging to different research domains need to operate simultaneously. This larger experimentation will hopefully reveal the expected results and will also point out enhancements for RINA and the SDK.

1.1. Experiment Coordination and Methodology

A key part of WP6 is the design, setup and execution of experiments trialling RINA and policies implemented by PRISTINE. During the project's first phase, represented by this deliverable, experiment focus was per research area via WP3, WP4 and WP5, following the initial deployment on the Virtual Wall testbed infrastructure and related system tests detailed in D6.1. Each of the implementation WPs were asked to design experiments that would trial their first phase of implementation, whereas the more complex trialling on a per use case context is for the second phase of the project. This initial mapping of research areas, experiments and associated use cases can be found in the table below.

Table 1. Initial mapping of research areas, experiments and associated use cases

Research areas	Experiment	DC Net.	Dist. Cloud	Net. SP
Resource Allocation (WP3)	Policies for resilient routing (NXW, iMinds)	x	x	x

Research areas	Experiment	DC Net.	Dist. Cloud	Net. SP
	Policies for QoS-aware multipath routing (Atos)	x		
	Policies for Cherish/Urgency multiplexing (UPC, i2CAT)	x	x	x
	Alternate protocol on a re6st/ babel virtual Ethernet mesh (Nexedi)		x	
	Load balancing and optimal resource utilization (WIT-TSSG)	x	x	x
Congestion Control (WP3)	Policies for ACC (UiO, i2CAT)	x	x	
	Performance isolation in data-centers (CREATE-NET)	x		
Security coordination (WP4)	Multi-level security (Thales)			x
Configuration Management (WP5)	Policies for the SDU protection module (FIT-BUT)			x
	Deploy IRATI kernel on SlapOS (Nexedi)		x	
	Manager and Management Agent (WIT-TSSG, i2CAT, ATOS)	x	x	x

Descriptors and characteristics to take into account for the design of the initial experiments included:

- Relation to use cases and requirements: The main context of WP6 is to trial PRISTINE results via industry-oriented use cases (Datacenter Networking, Distributed Cloud and Network Service Provider, as described in D2.1). Although the initial phase was centric on the initial output of the research areas, the link and context to the use cases was necessary to provide an initial analysis of potential technical and business impact.
- Experiment goals: Focused on what the experiments aim to achieve, beyond verification tests of WP3-5 implementation.
- Metrics: Influencing the experiment design are the metrics to be measured in the experiment in relation to achieving its goals.
- Process: A step by step process is described for each experiment setup and launch, aimed for reproducibility.

- Configuration: Including both variables to be set (and varied) for the experiment (e.g. different traffic loads, etc.), as well as topology (e.g. deployment of VMs in the VirtualWall infrastructure, DIF structure among them, etc.)

Finally, the presentation of results was planned in a manner that would fit with current WP6 experimentation objectives, as well as to close the project's first development phase and advance plans for its final phase. Apart from the initial results evaluation within the scope of each research area, emphasis is also provided to both the technical and business impact towards the larger scope of each use case. Finally, feedback to development for the second cycle is provided from both the standpoint of the research area of implementation (WP3-5), as well as from aspects of the more holistic scope of the industry use case requirements. The process is depicted in the diagram below.

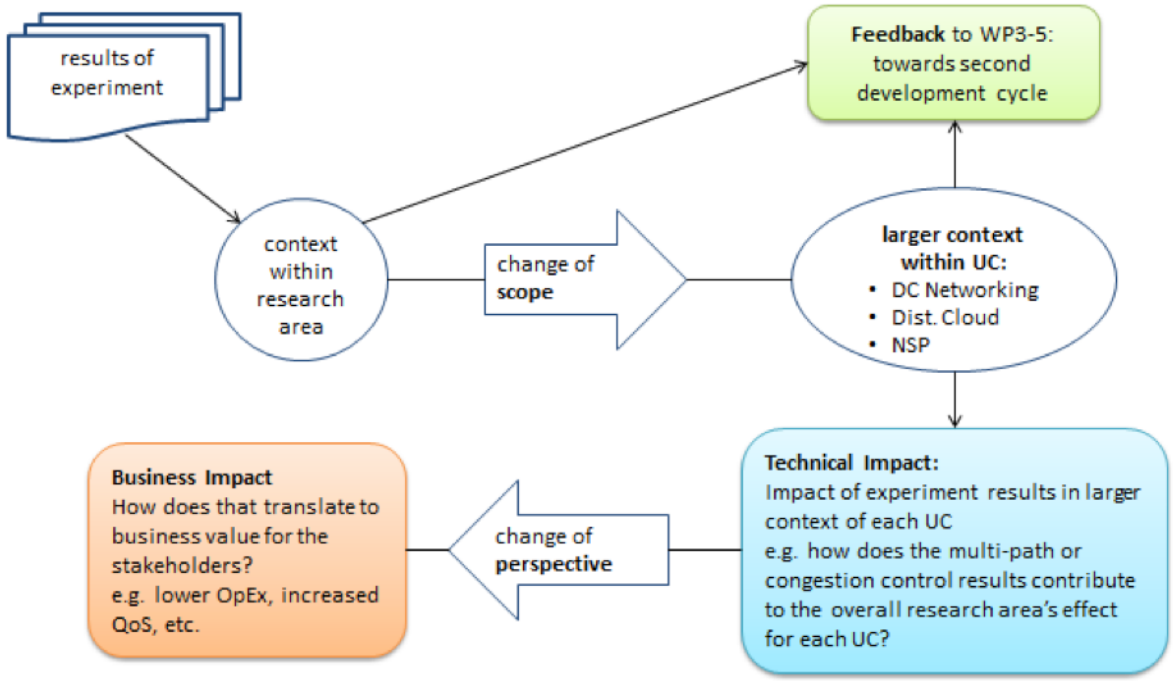

Figure 1. Methodology for initial experimentation phase

1.2. Organization of Deliverable

This deliverable is organized as follows:

Section 2 describes the tools used for testbed setup and experimentation.

Section 3 gathers the experiments carried out per research area.

Section 4 provides a consolidated overview of results for this first phase and outlines the future work until the end of the project as far as validation and business impact analysis are concerned.

2. Testbed Setup and Experimentation tools

2.1. Configurator

The purpose of the configurator is to be able to bootstrap a complete RINA network on the iLab.t Virtual Wall facility. The configurator takes as input the following:

- topology.xml: Contains the physical topology of the network. The different nodes in the network should be provided, and the links between them.

```
<?xml version="1.0"?>
<topology>
 <node id="m"/>
 <node id="n"/>
 <node id="o"/>

 <link id="link0">
   <from node="m"/>
   <to   node="n"/>
 </link>
 <link id="link1">
   <from node="n"/>
   <to   node="o"/>
 </link>
 <link id="link2">
   <from node="m"/>
   <to   node="o"/>
 </link>
</topology>
```

- difs.xml: Contains the different DIFs to be deployed in the network. For every DIF, the type has to be given. In the case of normal DIFs, the DIF template, as defined for the IRATI stack, has to be specified, and also supplied in that directory. In the case of a shim IPC process, the physical (or virtual) link has to specified, on top of which a shim DIF has to be overlayed.

```
<?xml version="1.0"?>
<difs>
```

```
 <dif id="normal" type="normal-ipc" template="default.dif" />
 <dif id="eth.shim0" type="shim-eth-vlan" link="link0"/>
 <dif id="eth.shim1" type="shim-eth-vlan" link="link1"/>
 <dif id="eth.shim2" type="shim-eth-vlan" link="link2"/>
</difs>
```

- ipcps.xml: Contains the placement of the IPCPs on the nodes. For every IPCP, the DIF it will be a part of has to be provided. If the IPCP has to use the IPC services provided by other DIFs, it has to specify this in the element “register-dif”.

```
<?xml version="1.0"?>
<ipcps>
 <node id="m">
   <ipcp ap-name="test-eth" ap-instance="1" dif="eth.shim0"/>
   <ipcp ap-name="test-eth" ap-instance="2" dif="eth.shim2"/>
   <ipcp ap-name="ipcp.m" ap-instance="1" dif="normal">
     <register-dif name="eth.shim0"/>
     <register-dif name="eth.shim2"/>
   </ipcp>
 </node>
 <node id="n">
   <ipcp ap-name="test-eth" ap-instance="1" dif="eth.shim0"/>
   <ipcp ap-name="test-eth" ap-instance="2" dif="eth.shim1"/>
   <ipcp ap-name="ipcp.n" ap-instance="1" dif="normal">
     <register-dif name="eth.shim0"/>
     <register-dif name="eth.shim1"/>
   </ipcp>
 </node>
 <node id="o">
   <ipcp ap-name="test-eth" ap-instance="1" dif="eth.shim1"/>
   <ipcp ap-name="test-eth" ap-instance="2" dif="eth.shim2"/>
   <ipcp ap-name="ipcp.o" ap-instance="1" dif="normal">
     <register-dif name="eth.shim1"/>
     <register-dif name="eth.shim2"/>
   </ipcp>
 </node>
</ipcps>
```

- vwall.ini: Contains configuration parameters to access the iLab.t Virtual Wall facility. This includes the wall to use (wall 1 or wall 2), the user’s credentials (logging in with an SSH key is also supported), the project and experiment name and the image to deploy on every node.

```
[vwall_config]
wall = wall2.ilabt.iminds.be
username = sander
password = <password>
proj_name = PRISTINE
exp_name = 3nodes
image = PRIST-bee1658-NOLOG
```

- apps.xml (optional): Contains the mapping of certain applications to a DIF in order to force them to only use that DIF. A better way is to give this as a parameter to the application at runtime.

```
<?xml version="1.0"?>
<apps>
   <app ap-name="rina.apps.echotime.server" ap-instance="1">
     <node name="m">
       <register dif-name="normal.DIF"/>
     </node>
   </app>
</apps>
```

Once this input is successfully parsed, a new experiment is created on the Virtual Wall based on the specified physical topology. If the experiment already exists, it is not re-created. To achieve this goal, configurator generates an NS script, which is required by the Virtual Wall when creating a new experiment:

```
set ns [new Simulator]
source tb_compat.tcl
set m [$ns node]
tb-set-node-os $m PRIST-dc38de9f-NOLOG
set n [$ns node]
tb-set-node-os $n PRIST-dc38de9f-NOLOG
set o [$ns node]
tb-set-node-os $o PRIST-dc38de9f-NOLOG
set link0 [$ns duplex-link $m $n 1000Mb 0ms DropTail]
set link1 [$ns duplex-link $n $o 1000Mb 0ms DropTail]
set link2 [$ns duplex-link $m $o 1000Mb 0ms DropTail]
$ns run
```

This will create the following topology:

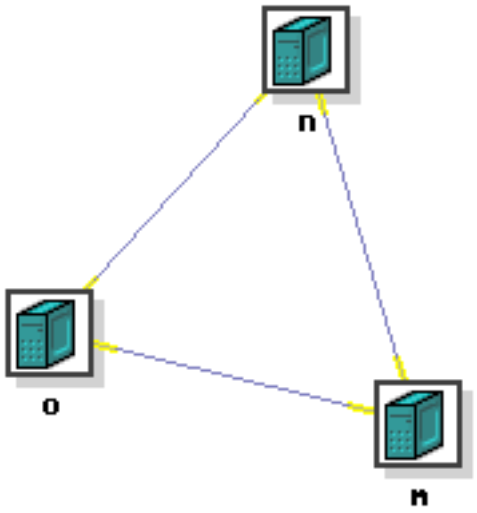

Figure 2. 3-node experiment

Then, the experiment is swapped in. Once again, if it is already swapped in, configurator just continues to the next step. After it is swapped in, configurator will access every node to obtain the OS specific name of the interface for a particular link. As an example, for link0 node m's specific OS name of the interface is eth56. This is needed in order to be able to auto-generate the DIF templates for the shim DIF for Ethernet. Configurator also assigns a VLAN id to every link, since this is required for the shim DIF for Ethernet. It then creates the correct VLAN interface for every VLAN on the appropriate nodes. The kernel modules that are required by the IRATI stack are also inserted on every node. Next, for every node, the configuration files are generated and copied to the configuration directory of the IRATI stack on every node. Finally, on every node, the IPC Manager is started.

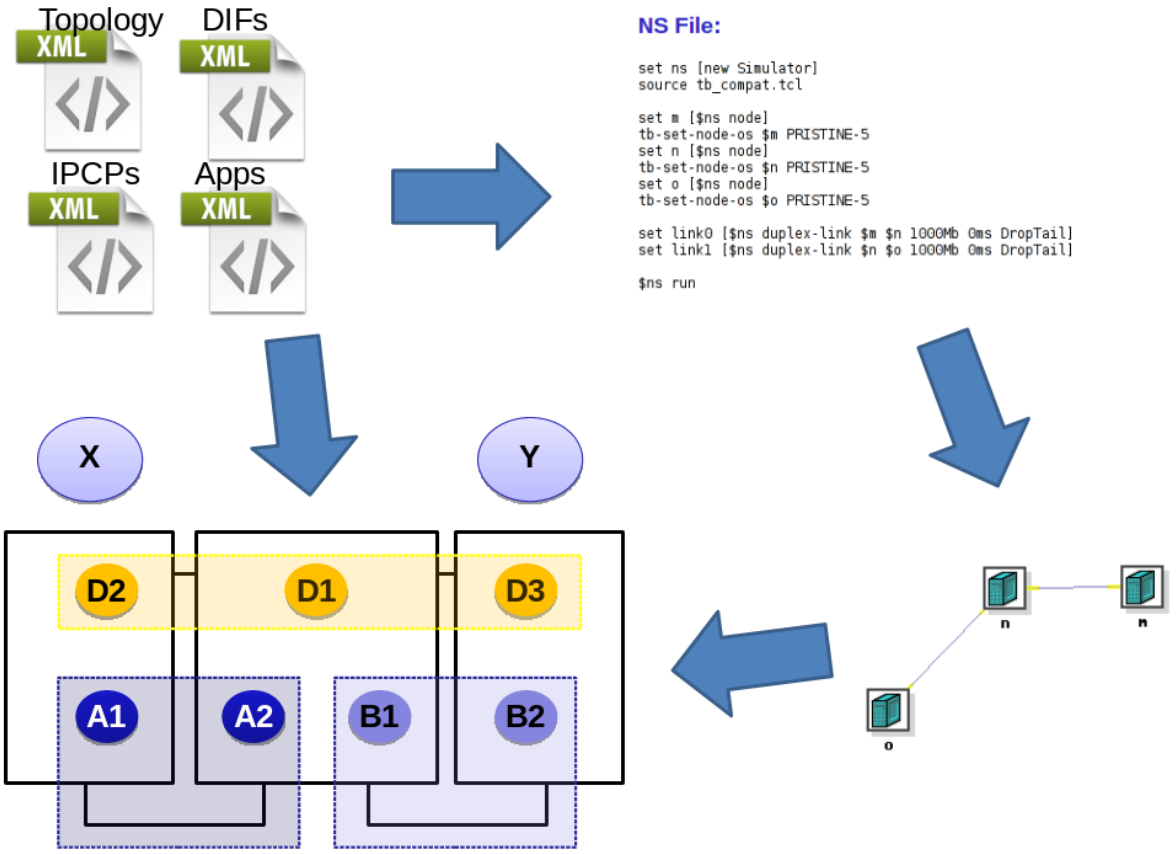

Figure 3. Configurator tool

To build “configurator” one would issue the following commands in the main directory:

```
./bootstrap
./configure --prefix=/path/to/irati/
make
```

To run it, one has to either install it (make install) or go into the src/ directory and run the confgen script. Confgen takes the following command line parameters:

```
[sander@Faust src]$ ./confgen --help
usage: confgen [-h] [--topology FILE] [--ipcps FILE] [--apps FILE]
               [--difs FILE] [--vwall-conf FILE] [--output-dir DIR]
               [--prefix FILE] [--input-dir DIR]

RINA Configuration Generator

optional arguments:
 -h, --help            show this help message and exit
```

```
--topology FILE    the topology XML filename (default: ['topology.xml'])
--ipcps FILE       the IPC Process XML filename (default: ['ipcps.xml'])
--apps FILE        the applications XML filename (default: ['apps.xml'])
--difs FILE        the DIFs XML filename (default: ['difs.xml'])
--vwall-conf FILE  the Virtual Wall INI filename (default: ['vwall.ini'])
--output-dir DIR   the output dir of the XMLs (default: ['configs'])
--prefix FILE      the prefix of the name of the output files (default:
                   ['ipcmanager'])
--input-dir DIR    the input dir of the config files (default:
                   ['inputs/2nodes'])
```

Configurator has the following restrictions:

- Enrollment is not yet automated. You have to perform enrollment yourself once the IPCMs are started.
- Of the shim IPC processes, currently only the shim-eth-vlan is supported. Supporting the shim-tcp-udp would require minimal effort. Supporting the shim-hv would require a bit more effort, since it would also mean instantiating new virtual machines on the Virtual Wall.

2.2. All-in-one-machine Testbed

2.2.1. Introduction

Regression and functionality testing for the IRATI stack are part of the continuous integration activities that PRISTINE developers carry out to validate Pull Requests and test the stack functionalities as a whole.

Those activities require a considerable amount of effort in order to setup the testbed, even when the testbed consists of a set of Virtual Machines (VMs) running on the developer's workstation, which we refer to as all-in-one-machine testbeds. Using VMs on a single workstation allows to build arbitrarily complex network topologies, including L2 switches and RINA router nodes, without having to deal with multiple physical machines.

Testbed setup includes several operations that the developer would normally carry out manually. Some preliminary operation have to be carried out before starting the VMs:

- Creating a disk image for each VM involved in the scenario, each one including the IRATI version to test. This may involve cloning existing images or update them with a different IRATI version.

- Creating a software bridge (e.g. a Linux bridge or an Open V-switch (OVS) bridge) for each L2 domain specified in the scenario topology.
- Destroy the software bridges
- Destroy the disk images. This step is optional, since the developer may want to reuse some image for future tests.

Then, a list of operations have to be executed for each VM involved in the testing scenario:

- Starting the VM.
- Assigning the VM an IP addresses for management and testing purposes, so that the developer can easily ssh into the VM, start tests and collection information.
- Assigning the VM one or multiple network interfaces to be used by IRATI, each one connected on a different L2 domain.
- Creating of one or more VLAN software interfaces on top of each physical interface, which is needed to make use of the Shim DIF over Ethernet.
- Creating of the IPC Manager (IPCM) configuration file to be run on the VM and starting the IPCM daemon
- Shutting down the VMs when the testing activity is completed

Doing all these operations manually may require 20 minutes or more even for scenarios that include four or five VMs. This clearly results in a severe slowdown of the development and test cycle, since the whole setup has to be redone each time a new version of the code has to be tested.

For these reason, T6.1 has developed a tool to automate the testbed setup operations reported above, as described in the following section.

2.2.2. Description of the tool

The PRISTINE all-in-one-machine testing tool is a collection of bash and python scripts pristine-test. QEMU-KVM is the hypervisor used to create the VMs that make up the testbed.

The tool is able to generate testbed scenarios with an arbitrary number of VMs, as allowed by the memory available in the workstation. The VMs are interconnected by L2 networks to form arbitrary topologies. Each VM can

have one or more network interfaces to be used for RINA traffic, while a separate interface is used by the developer and the scripts to ssh into the VM.

The developer specifies the testbed configuration is described by a configuration file (gen.conf), which describes the VMs, the L2 switches and the L2 links that make up the testbed.

The following gen.conf, describes four VMs (nodes), three software switches and six links to connect the VM interfaces to the switches. The topology is depicted in the configuration file itself.

```
# This configuration realizes the following four-nodes topology
#
# M1---[BR0]---M2---[BR1]---M3
#                     |
#                   [BR2]
#                     |
#                     M4

vm m1 ❶
vm m2
vm m3
vm m4
bridge br0 300 ❷
bridge br1 400
bridge br2 500
link br0 m1  ❸
link br0 m2
link br1 m2
link br1 m3
link br2 m2
link br2 m4
```

❶ The vm directive defines a VM, and must specify the VM name.

❷ The bridge directive defines a software switch, and must specify the switch name and the VLAN id to be used by the Shim DIF over Ethernet for all the VMs connected to that bridge.

❸ The link directive defines an L2 link connecting a VM to a switch and must specify the switch name and the VM name.

While the topology can be arbitrary, the current version of the tool always builds a fixed DIF stacking scheme:

1. A Shim DIF over Ethernet for each software switch, that includes all the VM nodes connected to the switch. The VLAN id to be used is specified by the bridge directive.
2. A single normal DIF that includes all the nodes in the scenario

Therefore, scenarios with more than one normal DIF stacked arbitrarily are not currently supported. In spite of this limitation, the tool offers coverage for a large number of functional and regression tests, and was found to be effective to intercept regressions introduced by new Pull Requests, before those regression could make their way in the OpenIRATI development branch.

2.2.3. How to use the tool

In the following we assume that QEMU-KVM and the python interpreter is installed on the developer's machine. The tool can be downloaded by github:

```
$ git clone git@github.com:vmaffione/pristine-test.git
```

Before starting to use the tool, the developer has to prepare a single qcow2 disk image containing an installation of OpenIRATI. In the following, this image will be referred to as the base disk image. To create an empty image, use the following command:

```
$ qemu-img create -f qcow2 disk.qcow2 6G
```

To install a Linux distribution on the image (e.g. Debian) using an ISO installation image, issue the following command (assuming the workstation CPUs have x86_64 architecture)

```
$ qemu-system-x86_64 -enable-kvm -cdrom XX.iso -hda disk.qcow2 -boot d
```

This will boot a QEMU VM with the ISO image available in the emulated CD-ROM slot, so that it is possible to proceed with the installation. Once the distribution installation is complete, the VM can be shut down.

The configure script contained in the tool repository must be run to set some variables that are needed to configure the tool

```
$ ./configure /home/developer/irati/local /home/foo/disk.qcow2 developer
```

The first argument specifies the path in the VM where IRATI userspace programs and libraries are (or will be) installed. This must be the same as the argument passed to the install-user-from-scratch script (available as part of the RINA stack), if that script is used to install the librina, rinad and rina-tools packages. The second argument specifies the path on the developer's workstation where the base disk image is stored. The third argument specifies the username created on the base disk image that will be used to ssh into the VM.

QEMU-KVM has a snapshot feature that allows a VM to be started in Copy-On-Write mode, so that all changes to the VM disk are not written back to the image, but kept in memory and thrown away when the VM is shut down. All the VMs in the scenario are started in snapshot mode, and associated to the very same base disk image. When the scenario is shut down, therefore, all the disk modifications VMs are just discarded. This methodology has multiple advantages:

- The developer has to update and maintain a single disk image, even if the testbed contains tens of VMs.
- Cloning images, which usually takes tens of seconds, is never necessary.
- If something goes wrong during the tests (kernel crashes, disk corruption, etc.) this has no effect on the base disk image.

The configure script creates the following configuration files and scripts:

- `program.sh`, that launches a VM without the snapshot feature, to be used by the developer to update and maintain the base disk image. Thanks to QEMU port forwarding, the developer can ssh into the VM, addressing the port 2222 on the localhost destination (127.0.0.1).
- `gen.py`, to be used to generate the testbed bootstrap and shutdown scripts, based on the gen.conf configuration file.
- `template.conf`, the IPCM configuration file template to be used on the VMs.

To complete the base disk image preparation, the following operations must be performed:

- Run the `program.sh` script to start the VM (with persistent disk modifications).
- Copy `template.conf`, `default.dif`, `shimeth1.dif`, `shimeth2.dif` and `shimeth3.dif` to the `/etc/`` directory on the VM.
- Copy `enroll.py` to the `/usr/bin` directory (or to a different $PATH folder) on the VM.
- Compile `mac2ifname.c` (gcc -o mac2ifname mac2ifname.c) and copy the executable to the /usr/bin directory on the VM.

Once the base disk image is ready, the developer can specify the desired topology in the gen.conf configuration file and run the `gen.py` script. The latter scripts generates the bootstrap script (`up.sh`) and shutdown script (down.sh), based on the scenario configuration.

The `up.sh` script automatically carries out the following operations:

- Create the Linux bridges that implement the software switches specified by gen.conf bridge directives.
- Create the TAP interfaces[TAP] required to implement the VM interfaces, and attach them to the Linux bridges as specified by gen.conf link directives.
- Start (in snapshot mode) the VMs specified by the vm directives. Each VM is given a management interface for SSH access and as many TAP interfaces as specified by `gen.conf`.
- On each VM, use SSH in non-interactive mode to perform the following tasks:
  - Finalize the IPCM configuration file and DIF templates, filling in the `` `/etc/template.conf `` and `/etc/{star}.dif` files.
  - Create and configure the VLAN interfaces as as required for the shim DIF over Ethernet.
  - Load the OpenIRATI kernel modules
  - Start the IPCM deamon.
- For each Shim DIF over Ethernet (e.g. for each switch) choose a VM as the enrollment pivot for that Shim DIF and perform the enrollment of all the other VMs against the pivot VM into the normal DIF.

The `down.sh` script automatically carries out the following operations:

- Shut down all the VMs in the testbed.
- Destroy all the TAP interfaces created.
- Destroy all the Linux bridges created.

Once the `up.sh` script is run to bootstrap the scenario, the developer can access the VMs through their management interfaces, using SSH towards the localhost destination (127.0.0.1). The first VM can be accessed at port 2223, the second one at port 2224, and so on.

A simple test that can be performed to test the connectivity provided by the normal DIF in the example scenario reported in the previous section, is to run the rina-echo-time application in server mode on node m1, and the same application in client mode on node m4.

The whole process described in this section is summarized in the following figure.

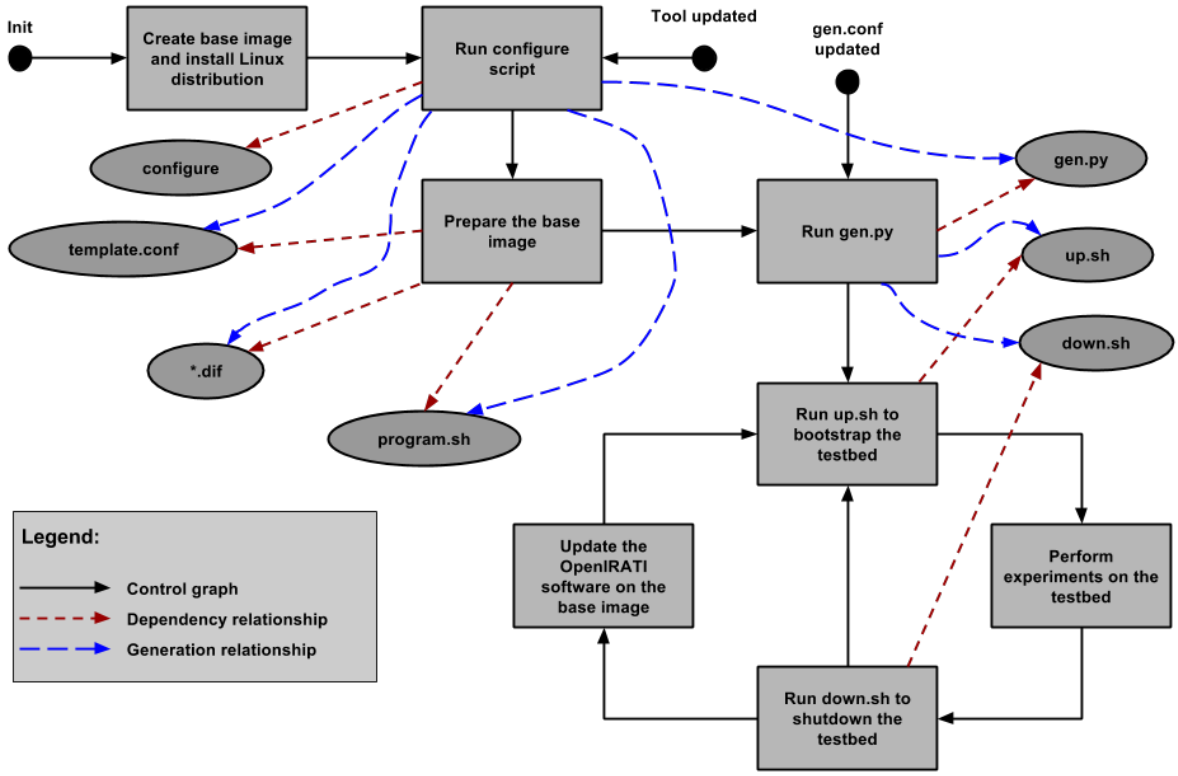

Figure 4. All in one machine - testbed

2.3. RINA traffic generator tool

The RINA traffic generator (rina-tgen) was inherited from the GN3+ IRINA OC project [IRINA]. PRISTINE experimentation sets new requirements for rina-tgen, for which it has undergone some changes:

- During PRISTINE, a change in the IPC API was included that has the flow allocation function return a `<port_id>` instead of a `<Flow *>`. rina-tgen was updated for compatibility with this new API.
- A major code refactor was conducted to make the tool more extensible. It includes function prototypes for registering the application with multiple DIFs (multi-homing).
- Statistics are now reported per port_id, over a timed interval and a packets per second (p/s) metric was added
- An option was added to write output as comma-separated-values (.csv) for easy parsing and analysis, which generates a `.csv` file per connected client and per performed test for easily tracking multiple concurrent clients connected to a single server.
- The build system was improved with improved dependency detection of the BOOST C++ libraries.
- The client and server were extended with non-blocking I/O options for improved accuracy in the server application and the handling of packet loss. To include this functionality, the non-blocking I/O in IRATI needed revision, performed in WP2.

```
USAGE:

   ./rina-tgen  [-o <string>] [-l] [--interval <unsigned integer>] [-c
                <unsigned integer>] [--duration <unsigned integer>] [--
rate
                <unsigned integer>] [--timeout <unsigned integer>] [-s
                <unsigned integer>] [--distribution <string>]
                [--poissonmean <double>] [--qoscube <string>] [-d
 <string>]
                [--client-api <string>] [--client-apn <string>]
                [--server-api <string>] [--server-apn <string>] [-r]
                [--sleep] [--] [--version] [-h]


Where:
```

```
  -o <string>,  --output-path <string>
    Write csv files per client to the specified directory, default = no
    csv output.

  -l,  --listen
    Run in server (consumer) mode, default = client.

  --interval <unsigned integer>
    report statistics every x ms (server), default = 1000.

  -c <unsigned integer>,  --count <unsigned integer>
    Number of SDUs to send, 0 = unlimited, default = unlimited.

  --duration <unsigned integer>
    Duration of the test (seconds), 0 = unlimited, default = 60 s IF
count
    is unlimited.

  --rate <unsigned integer>
    Bitrate to send the SDUs, in kb/s, 0 = no limit, default = no limit.

  --timeout <unsigned integer>
    Time for a test to timeout from client inactivity (ms), default =
    10000 ms

  -s <unsigned integer>,  --size <unsigned integer>
    Size of the SDUs to send (bytes), default = 500.

  --distribution <string>
    Distribution type: CBR, poisson, default = CBR.

  --poissonmean <double>
    The mean value for the poisson distribution used to generate
    interarrival times, default is 1.0.

  --qoscube <string>
    Specify the qos cube to use for flow allocation, default =
unreliable.

  -d <string>,  --dif <string>
    The name of the DIF to use, empty for any DIF, default = empty (any
    DIF).

  --client-api <string>
    Application process instance for the client, default = 1.
```

```
--client-apn <string>
  Application process name for the client, default =
  traffic.generator.client.

--server-api <string>
  Application process instance for the server, default = 1.

--server-apn <string>
  Application process name for the server, default =
  traffic.generator.server.

-r,  --register
  Register the application with the DIF, default = false.

--sleep
  sleep instead of busywait between sending SDUs, default = false.

--,  --ignore_rest
  Ignores the rest of the labeled arguments following this flag.

--version
  Displays version information and exits.

-h,  --help
  Displays usage information and exits.
```

Future improvements depend on requirements from the experiments planned in PRISTINE. Currently they include:

- functional split in tgen-server and tgen-client
- server-side traffic generation for bidirectional traffic tests
- use separate flows for experiment control traffic and actual experiment traffic
- support for a single flow must remain for experiments directly over the shim-eth-vlan
- implementation of multi-homing the server in multiple DIFs
- implementation of multi-homing clients in multiple DIFs
- loopback functionality

3. Experiments Per Research Area

3.1. Loop Free Alternates in RINA

3.1.1. Short Description

This experiment validates the Loop Free Alternates (LFA) implementation and propagation of error detection between DIFs. It shows how PDUs belonging to active flows on a physically resilient network are quickly rerouted through an alternate path after a failure of a node or a link.

3.1.2. Experiment goals

Assuming the link/node failure does not partition the network, (i.e. we have at least a 2-connected graph), the primary goal is to make sure that flows affected by this failure are restored to the extent allowed by the LFA mechanism [RFC5286] . The second goal is to make sure that recovery time is at least one order of magnitude shorter than the time needed for convergence of the routing algorithm.

3.1.3. Use Cases and requirements addressed

Failures can occur in all network scenarios, so this experiment spans all PRISTINE use cases.

- Internet Service Provider Use Case: solution to meet requirements ST-DIF-2, SC-DIF-2, SC-DAF-2, VNF-DIF-2 and VNF-DAF-2, MT-DIF-2, M-DAF-2 and ST-DIF-6.
- Data Center Use Case: Solution for requirement DCF-DIF-7.
- Distributed Cloud use case: Provides a fast switchover mechanism over resilient graph topologies (complements requirement SOS-DIF-1).

3.1.4. Metrics and KPIs

The main metric is the recovery time (ms) needed to restore complete connectivity. Recovery time is expected to be in the order of ~10 ms, with a target for the prototype set at 100ms, which is at least one order of magnitude faster than the routing convergence. We aim for low packet loss as well, aiming for at most 1% packet loss measured in the second following the failure.

Metric	target (use case)	target (prototype)
Recovery time (s)	<50ms	100 ms
Packet loss	~0	1%

3.1.5. Experiment steps:

The experiment consists of the following actions:

1. **Setup the a RINA test network with redundant physical connections.** A single node or link failure must not result in a network partition. The minimal topology which has this property is a ring, however, large rings are not suitable for the LFA algorithm, so larger experiments will require mesh topologies.
2. **Configure a single normal DIF.** This normal DIF will be supported by a number of shim DIFs. The IPCPs of the normal DIF will need to be enrolled.
3. **Run application flows (over the normal DIF).** A number of long running application flows (rina-tgen) need to be established over the normal DIF. Some traffic must be routed over the physical link that will be submitted to failure conditions.
4. **Submit the network to failure conditions.** While SDUs are being exchanged, unplug an Ethernet cable in the network, or configure a network interface as down, in order to trigger the LFA fast reroute.
5. **Verify operation during failure conditions.** Check that application flows keep working normally after LFA has restored the routing tables. It may happen that some SDUs are lost during the recovery, but this is acceptable and can be handled by retransmission (DTCP), if needed.
6. **Restore normal operating conditions.** Plug the cable back (or put the network interface up) and check that LFA restores the old paths.

3.1.6. Experiment configuration

We performed this functional test in a single normal DIF overlaying 3 shim DIFs over Ethernet (802.1Q). Introducing resiliency in lower level DIFs may make more sense than solving it in higher level DIFs, since the time for detection may be higher in higher level DIFs and single failures in lower ranked DIFs may trigger many more repair action at higher ranked DIFs.

Hence, this basic scenario of a single normal DIF at the first level is a good candidate for a proof of concept. We used the following topology, so that every node has a Loop Free Alternate in case of a link failure:

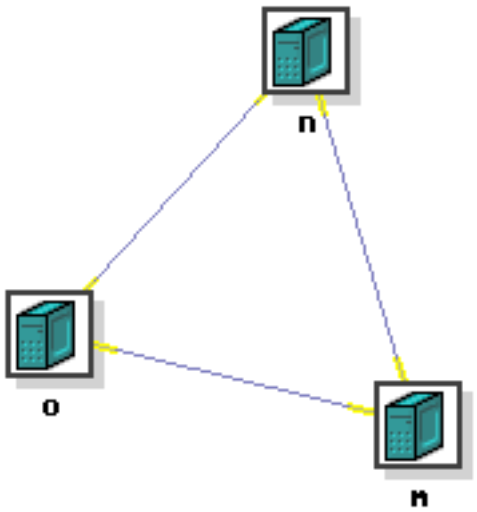

Figure 5. 3 node experiment

The normal DIF has the following configuration for its EFCP component:

```
"name" : "unreliable",
 "id" : 1,
 "partialDelivery" : false,
 "orderedDelivery" : false,
 "efcpPolicies" : {
      "dtpPolicySet" : {
        "name" : "default",
        "version" : "0"
      },
      "initialATimer" : 0,
      "dtcpPresent" : false
 }
```

Which means that flows will be created without flow control and without retransmission control. After the complete network has been bootstrapped, e.g. all IPCPs instantiated on all nodes and enrolled in their respective DIFs, we launched the rina-tgen application. On node m, we launched the application in server mode.

```
./rina-tgen -l
```

On node o, we started the traffic generator in client mode, and asked it to send SDUs of size 1450 at a rate of 900 Mbps at constant bit rate (CBR). Other modes than CBR are available, such as a Poisson distribution, which mimics better the traffic shape of a video stream. However, for this proof of concept experiment, CBR is sufficient.

```
./rina-tgen -s 1450 --rate 900000
```

After some time, we brought down the interface on node o that corresponds to the direct link to node m. The LFA policy in kernel space then successfully switches to the LFA, which is the interface that corresponds to the link going to node n.

3.1.7. Result evaluation

The result when the physical interface is brought down is shown in the figure below.

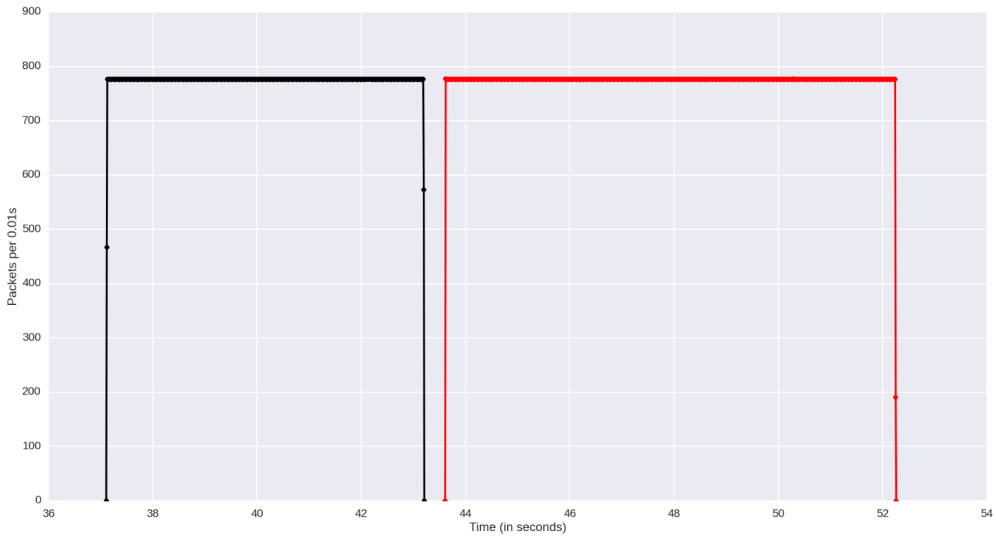

Figure 6. Recovery after disabling the physical interface

The black line shows the initial interface on which SDUs are sent. The red line shows the interface that is used once the stack recovers from the failure. We measured the time between the first packet being sent on the backup NIC and the last packet being sent on the original NIC. It takes approximately 400 milliseconds to recover from the failure. This seems

to differ a bit from the expected outcome. However, we also performed another experiment where we only brought down the virtual interface that the shim DIF for Ethernet was using. This is depicted in the figure below.

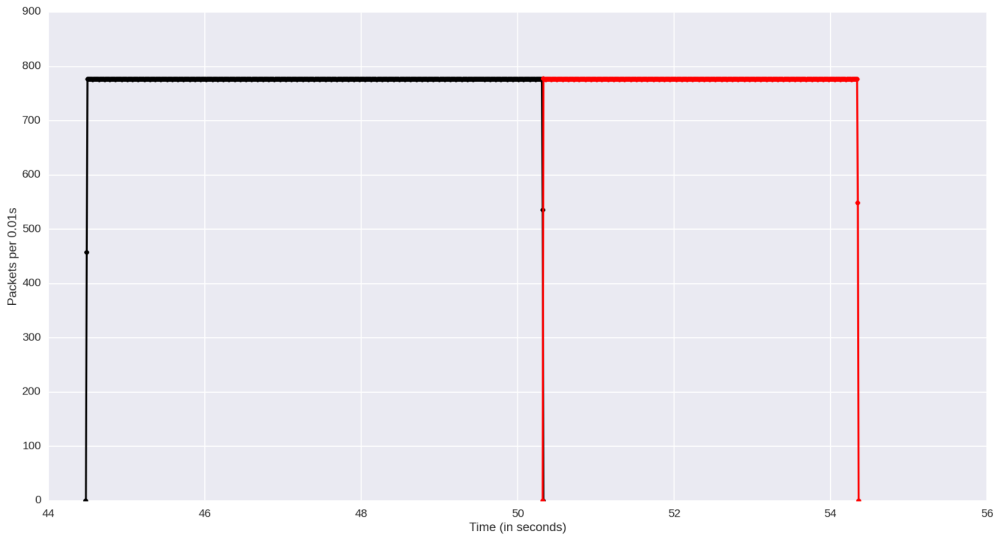

Figure 7. Recovery after disabling the VLAN interface

Here it can be seen that the recovery is almost instantaneous. In fact, it takes 1.7 ms to recover, which is well within the projected outcome of the order of 100ms for the prototype. After some investigation, we found the reason for this huge difference in recovery time. When a physical interface is brought down the driver is contacted to actually bring the interface down, meanwhile dropping any new packets that are trying to use the Network Interface Card (NIC). On our test machine, it takes the driver 400 ms to perform this, which is exactly the difference between the two recovery times. Using a different NIC may decrease this time. So in conclusion, the failure detection time is currently 400 ms, whereas the time to recover after detection is 1.7 ms.

3.1.8. Feedback to development

During the implementation and experimentation of the LFA policy, we revealed several issues to be resolved in the IRATI stack:

- Currently we can only use flows without flow control, since the EFCP component seems to stop working when too many control PDUs are lost. We have shared this information with WP2, where it is currently being debugged.

- The Qdisc, which is used in the shim DIF over Ethernet to prevent the rate of PDUs that are sent to exceed the line rate of the NIC, was only attached to one transmission queue, which works on a VM, but not on real hardware in the Virtual Wall. This resulted in the shim DIF for Ethernet blocking its ports continuously when a packet was dropped.
- Several locking problems were found in the RMT component in kernel space, which were debugged and fixed.
- Non-blocking I/O seems to be a better alternative when drops may occur in the network, since it decouples the applications more from the quirks of the network stack.

3.1.9. Feedback to use-cases requirements

The Loop Free Alternate experiments described previously have no recommendations towards use cases requirements. In effect, the routing resiliency feature spans all three use cases, and can be always used by the network administrator to increase IPC service availability.

3.2. Simple multipath routing within a POD

3.2.1. Short Description

This experiment aims at proving the correct functioning of the simple multipath routing strategy and at obtaining load balancing measures of its operation using a pod topology of 4 nodes.

3.2.2. Experiment goals

This experiment aims at validating the proper functioning of the simple multipath policies over a simple datacenter pod topology. The goal, in this phase, is to test the correct traffic balance between two different paths with the same cost, in order to maximize the network resources utilization.

Evaluating the performance of the multipath policies within a pod is a key step towards the evaluation of the full datacenter topology. Different results may be observed and used for the refinement of the policies implementation. Besides, within the full datacenter network, some flows will be routed just within a pod without traversing the core layer, so the results obtained in this experiment allow testing that scenario.

3.2.3. Use Cases and requirements addressed

This experiment is specifically focused in the Data Center Use Case, in which all nodes are interconnected by means of equal cost links. The experiment covers the requirements DCF-DIF-1, DCF-DIF-12 and POD-DIF-1, which reflect the need for using any available capacity among all the possible paths for the Datacenter Fabric DIF and the POD-DIF respectively.

3.2.4. Brief description of the policies

This experiment uses two policies for testing the multipath routing. The first one is the Multipath Routing policy, located in the user space of the stack, which in charge of identifying all the possible paths to any other node in the network and generating the routing entries with the next hop node for each of those paths. The algorithm used for the path identification is the ECMP Dijkstra, a variation of the original Dijkstra routing algorithm in which all paths of equal cost to reach the destination are considered and not simply discarded once one has been selected.

The multipath routing is complemented with a specific PDU Forwarding policy in the RMT, located in the kernel space of the stack. This policy implements a hash-threshold algorithm to select the specific port among the available ones to send the PDU. The process is as follows: A single RINA flow is identified by the source and destination addresses and connection endpoint ids and the QoS. Using that information, a fast hash function such as the CRC16 implemented in the Linux kernel is performed over those parameters to get a unique identifier. On the other hand, the whole output space of the hash function (16 bits in the case of CRC16) is divided among the available ports, therefore having a specific range of hash values for each port. Then, the port associated to the range containing the flow hash value is the chosen port.

The figure below shows a schematic diagram of the routing process, marked in red the policies used for this experiment.

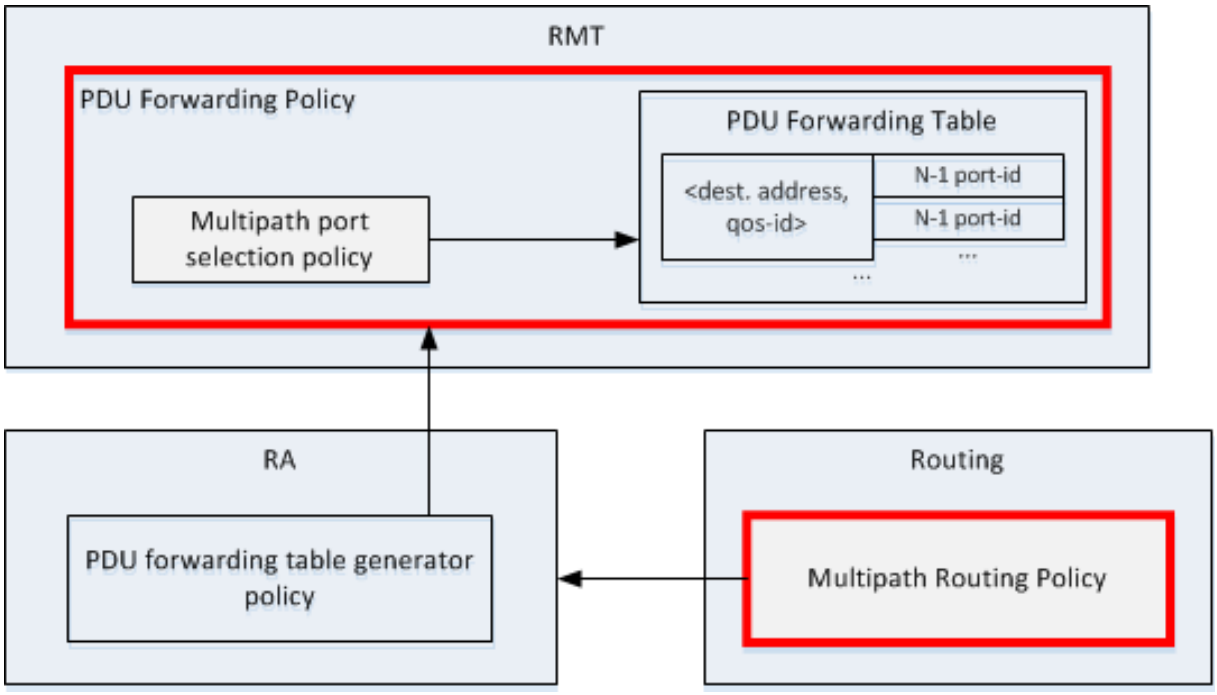

Figure 8. Schematic diagram of multipath routing components and policies involved in this experiment

Those two previous policies combined find multiple available paths to reach the destination IPC Process and distribute traffic equally among those paths, contributing to the optimal utilization of the date-center resources.

3.2.5. Metrics and KPIs

The main metric to measure the validity of multipath routing is the percentage of traffic routed for each of the available paths between the two top of Rack Switches. In this case, with the 4 nodes topology deployed in the experiment there will be two available paths. Therefore the expected percentage of traffic is 50% in each path (equal distribution).

3.2.6. Experiment configuration

Using a datacenter pod topology of 4 nodes, one of the edge routers is configured as traffic source and the other edge router as traffic sink. The connectivity between those routers is done through two aggregation routers.

The communication between nodes is achieved using two levels of DIFs: at the bottom level Shim DIFs over VLANs link every router with each other.

On top of this level, there is the DC Fabric DIF which covers the four nodes as a whole.

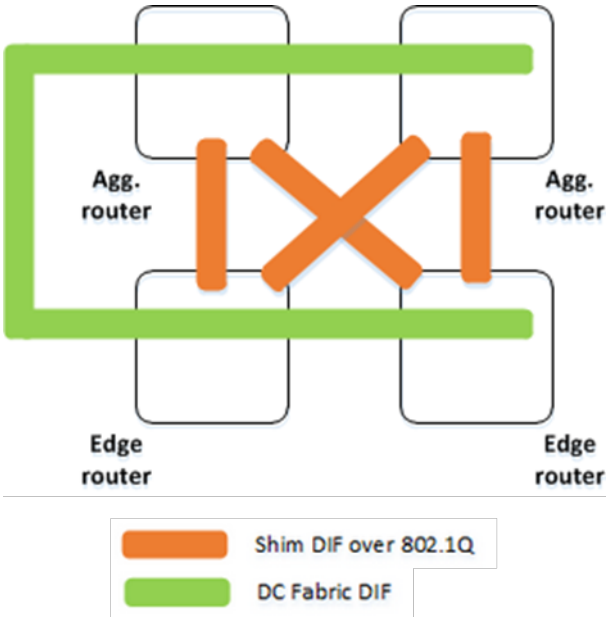

Figure 9. DIF configuration

3.2.7. Experiment steps

The configuration of the nodes and the enrollment of the IPCPs follow the usual procedure through the IPCManager on each node. On the router defined as traffic source, several instances of the rina-echo-time client are launched, in order to simulate different flows from the origin to the source, using the following bash script.

```
#!/bin/bash
for i in {1..100}
./rina-echo-time –c 10 &
done
exit
```

The rina-echo-time listener is launched on the router defined as traffic sink. Once the script is launched, one hundred different flows are created which should be split equally between the two available paths.

Then, the total number of PDUs routed to each path is measured on the traffic source node.

3.2.8. Results evaluation

The experiment was performed four times obtaining the result shown in the figure below.

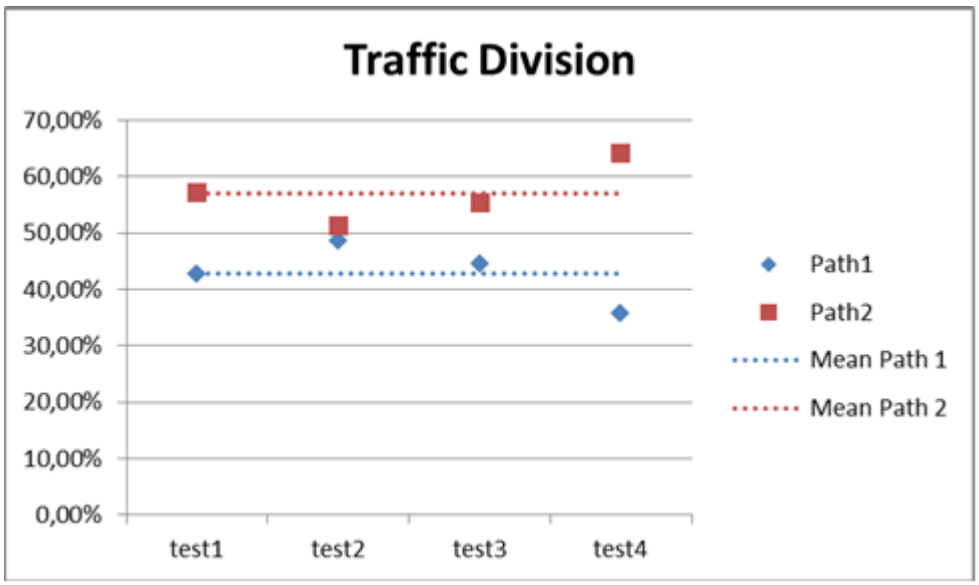

Figure 10. Experiment result

The blue dots show the percentage of traffic routed throw the first path whereas the red ones represent the percentage routed throw the second path.

Taking the average traffic of each path among the different tests, it can be seen that around a 42% of the traffic is sent through the first path and the remaining 58% through the second path. This is close to the expected 50%-50% balanced distribution of the hash-threshold algorithm for forwarding decisions, whose purpose is just to send the traffic evenly among the available paths, without taking into account any link characteristics such as capacity or required QoS.

3.2.9. Feedback to development

For the execution of this experiment no specific bugs have been found in the RINA stack implementation. The multipath policies have been introduced in the SDK.

However, despite the good implementation of the SDK, it is worth commenting the few sources of documentation still available for the development. This is a known fact for the developer community that

requires more attention, since it is considered a potential issue as it may discourage newcomers willing to contribute to the stack implementation.

3.2.10. Technical impact towards use cases

This experiment proves that the use of multipath routing is perfectly achievable in RINA, being a key aspect in the Datacenter Use Case. Future performance experiments using more advanced forwarding decision algorithms will be used to obtain a more specific technical impact on use cases, comparing current Internet implementations against RINA counterparts.

3.3. Aggregate Congestion Control

3.3.1. Short description

This set of experiments analyzes the behavior of RINA networks dealing with congestion at recursive layers (hence the Aggregate Congestion Control name), investigating performance and fairness aspects.

3.3.2. Experiment goals

The experiments build on the results of the congestion control simulations performed within task T3.1 and reported in [D32]. Therefore, the first goal is to experimentally verify some of the simulation results.

Secondly, the experiments aim to validate both the different aspects of the RINA architecture related to congestion control (flow control, congestion detection and pushback between layers) and the correct behavior of all the components of the RINA implementation which play a role in congestion control (mainly EFCP and the RMT).

Finally, the third goal is to demonstrate that each DIF can have a different set of congestion control policies - leveraging PRISTINE's SDK. To this end, two different congestion management policies are used in all the scenarios, which are later studied in terms of throughput, resource usage, and stability. The policies are as follows:

1. The first policy tries to mimic as much as possible the behavior of the TCP congestion avoidance algorithm [Jacobson] working with Explicit Congestion Notification (ECN) provided by intermediate IPCPs with a RED (Random Early Detection) style policy [RFC2309] for marking

PDUs. This policy-set is used as the "default" case and accomplishes two purposes: i) facilitating the comparison with the TCP/IP world, since the measured benefits are due to the RINA structure and not to an optimal congestion management policy; ii) allows for the comparison with other policies.

2. The second policy is a congestion management policy that adapts Raj Jain's binary-feedback congestion avoidance scheme [Jain] to the DIF environment.

3.3.2.1. Brief description of the policies

The two congestion management policies used in these experiments are based on a explicit feedback system with three main components: i) EFCP at the sender IPC Process (IPCP), which is flow-controlled by the receiver IPCP using window-based flow control; ii) RMTs of IPCPs between the sender and the receiver, which relay PDUs and may decide to mark the PDU with an ECN flag under certain conditions, and iii) the EFCP receiver which increases/decreases the credit available to the sender based on the PDUs received and the feedback from the relaying IPCPs (in the form of ECN-marked PDUs).

Explicit congestion detection with binary feedback

Jain et al designed a binary feedback congestion avoidance scheme [Jain] that tries to keep the network operating in a region known as 'the knee'. The 'knee' is the region where the power is maximized, with the power being defined as (throughput^alpha)/delay. After passing the point of the 'knee' small increments of throughput result in high increments in the response time (delay). Congestion control schemes are designed to protect the network once it reaches the congestion collapse point (or 'cliff'), the point at which throughput quickly approaches zero and response time tends to infinity. Therefore, Jain's scheme is designed to keep the network operating in the most efficient zone.

- RMT policies (when to mark PDUs): RMT queues have to be monitored in order to identify when the network load is growing. Jain et al showed that for most traffic distributions a network resource is at the 'knee' when the average queue length is of one PDU. When the average queue length is greater than one, the RMT will set the ECN flag of the PDU to true. Jain et al. describe a procedure to compute the queue length ensuring

that the measured load level lasts enough to be worth signaling the user [Jain].

- EFCP policies (when to reduce/increase the window size): The EFCP receiver must quantify the number of signals it receives from the network and take a decision. A critical aspect of this policy is when to take a decision: taking it too often may lead to unnecessary oscillations, taking it too seldom will make the network less responsive to congestion. Jain et al propose that windows should be adjusted only once every two Round Trip Times (RTT), and that only the PDUs belonging to the last cycle should be used for taking a decision: if 50% or more than the PDUs received during the last cycle have the ECN flag set the window size should be decreased; otherwise it should be increased. The increase/decrease policy follows an Additive Increase Multiplicative Decrease (AIMD) approach, increasing the window by one PDU and reducing it by a factor of 7/8 (see [Jain] for a full discussion).

Random Early Detection (RED) + TCP congestion avoidance algorithm

This policy set combines Active Queue Management techniques to mark PDUs in the RMT queues with the TCP-Tahoe congestion avoidance algorithm (but working with explicit feedback rather than with packet loss).

- RMT policies (when to mark PDUs): Active queue management schemes make packet dropping or 'marking' decisions based on the queue behavior, with the goal of avoiding phase effects as well as keeping the queue size low while allowing occasional bursts. RED is one of such schemes. Every time a packet arrives to an output RMT queue, RED calculates the average queue length and takes a decision:
  - If the average queue length is below a minimum threshold (*minthres*) it does nothing.
  - It the average queue length is between *minthres* and the maximum threshold (*maxthres*) the probability of marking a PDU will be between zero and the maximum marking probability (*maxp*), and it will directly be proportional to the average queue length.
  - If the average queue length is over *maxthres*, the PDU is always marked.
- EFCP policies (when to reduce/increase the window size): The EFCP policy acts in the DTCP component. It tries to mimic the TCP-Tahoe

algorithm in a RINA-adapted way. The EFCP receiver starts from the *slow start* mode, with a configurable initial credit, and a configurable *sshtresh* which determines when to switch to *congestion avoidance* mode. In the *slow start* mode, every time it receives a not ECN-marked PDU from the sender, the credit is increased by one. In the *congestion avoidance* mode, the credit is increased by *1/credit*. The switch to the *congestion avoidance* mode can occur either because a PDU arrives with the ECN flag set to true or because the current credit (window size) reaches *sshtresh*. In any mode, if a PDU with ECN is received, the current credit is halved and the ECN bit will not be considered again until the next RTT. RTT is approximated at a certain moment by the reception of a whole window size at that moment number of PDUs.

3.3.3. Use Cases and requirements addressed

Distributed cloud and data-centre networking. Congestion avoidance techniques enable the network provider to operate the network close to an optimal performance point, keeping a good balance between maximum throughput and a minimum delay. Congestion control techniques also allow the network provider to keep the network in an operational state when the offered load approaches the maximum network load.

3.3.4. Metrics and KPIs

Throughput vs. experiment time
: This metric will help analyzing the stability of the congestion management policies by looking at the oscillations of the throughput vs. time in the presence of congestion.

Window size vs. experiment time
: This metric will also contribute to the stability analysis of the congestion management policies.

3.3.5. Experiment configuration

In this experiment named "Two hosts experiment", there are two levels of DIFs: two bottom DIFs providing IPC over an Ethernet links whose throughput is limited to 250 Mbps and 50 Mbps, and the resulting link delay is approximately 130ms and 220ms respectively; and an upper DIF that connects together the two hosts via the router. A single flow provided by the upper DIF connects two instances of the "*rina_tgen*" application,

which tries to send data in SDUs of 1450 bytes through a flow with a certain rate during an amount of time specified by the experimenter. The experiment variables are the number of flows between the two hosts.

The described experimental scenario is depicted in Figure 11. The configuration files required for each of the three systems are provided in Section A.1

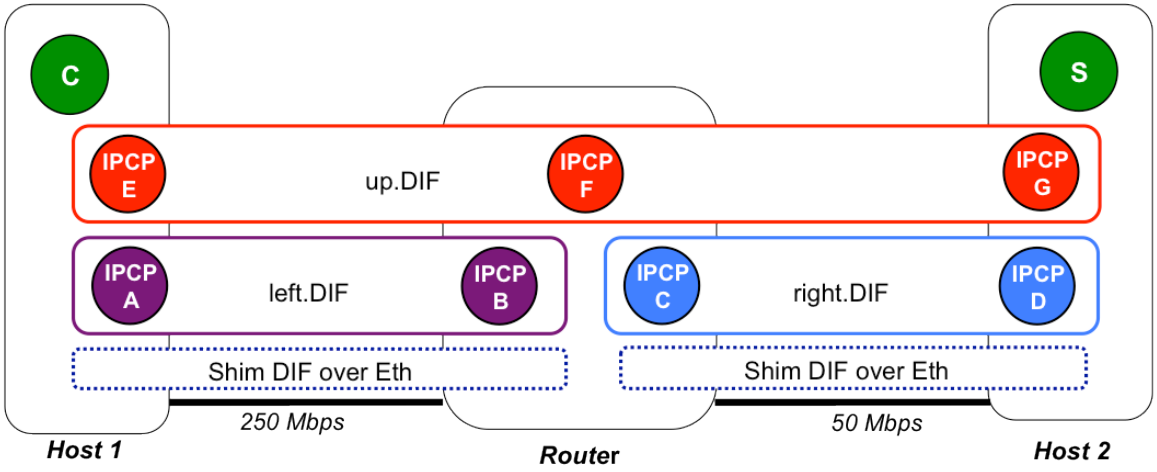

Figure 11. Two hosts experiment

3.3.5.1. Experiment steps

Once the three systems are ready, the IPCManager needs to be started in each system by typing the following command.

```
./ipcm -c ../etc/ipcmanager.conf
```

Then, from a console in system *Host 2* type the following commands to enroll IPCPs *C* and *D* in the DIF *right DIF* and IPCPs *F* and *G* in the DIF *up.DIF*.

```
telnet localhost 32766
IPCM >>> enroll-to-dif 2 right.DIF 100 C 1
DIF enrollment succesfully completed
IPCM >>> enroll-to-dif 3 up.DIF right.DIF F 1
DIF enrollment succesfully completed
IPCM >>> exit
```

Now start the *rina-tgen* application in server mode by typing:ms and

```
./rina-tgen -l --interval 500
```

Now, start a console in system *Host 1* and type the following commands to enroll IPCPs *A* and *B* in the DIF *left DIF* and IPCPs *E* and *F* in the DIF *up.DIF*.

```
telnet localhost 32766
IPCM >>> enroll-to-dif 2 left.DIF 110 B 1
DIF enrollment succesfully completed
IPCM >>> enroll-to-dif 3 up.DIF left.DIF F 1
DIF enrollment succesfully completed
IPCM >>> exit
```

Now start the *rina-tgen* application in client mode by typing (-s defines de SDU size in bytes):

```
./rina-tgen --duration 60 --rate 0 -s 1450 --distribution CBR
```

3.3.6. Evaluation results

3.3.6.1. 1 flow

Figure 12 shows the throughput achieved by one flow vs. the experiment time in three cases: iperf using the Linux TCP/IP stack and RINA with the configuration depicted in the experiment's description figure using the two previously described congestion management policy sets: Jain's binary feedback scheme and the combination of RED with TCP's congestion avoidance scheme.

Figure 12. Two hosts experiment, single flow: Throughput (Mbps) vs. Experiment time (s)

The figure shows that native Linux TCP achieves a slightly higher throughput that the two RINA-based solutions (which we attribute to the not performance-optimized RINA implementation used for the experiments), but, as expected, it also has a more unstable behaviour than the flows provided by RINA. This is due to the fact that TCP works only with end-to-end implicit signals (packet lost), and needs to cause packet loss in order to get feedback and adjust the transmission rate. In contrast, the two levels of DIFs and the pushback between layers allow RINA to react sooner to congestion. Combining this feature with the explicit feedback (ECN flag), and we obtain the smoother behaviour shown in the graph: the flow reacts sooner to congestion and the throughput corrections don't need to be as large as in the TCP case.

Figure 13 depicts the evolution of the window size in the case of the TCP-RED policy set. In this case, the window size oscillates more than Jain's case reflected in Figure 14. The reason for this is that, in contrast to Jain's policy where the PDU is marked with ECN when the average occupation of the queue in 1 cycle is greater or equal to 1, in TCP-RED the ECN flag that advertises congestion is marked following a certain probability when the average occupation of the queue from the creation of the flow is between 2 thresholds or always when it is beyond the *max tresh*. Moreover, when

the credit has to be decreased, it is halved instead of decreased by 7/8 as in Jain's.

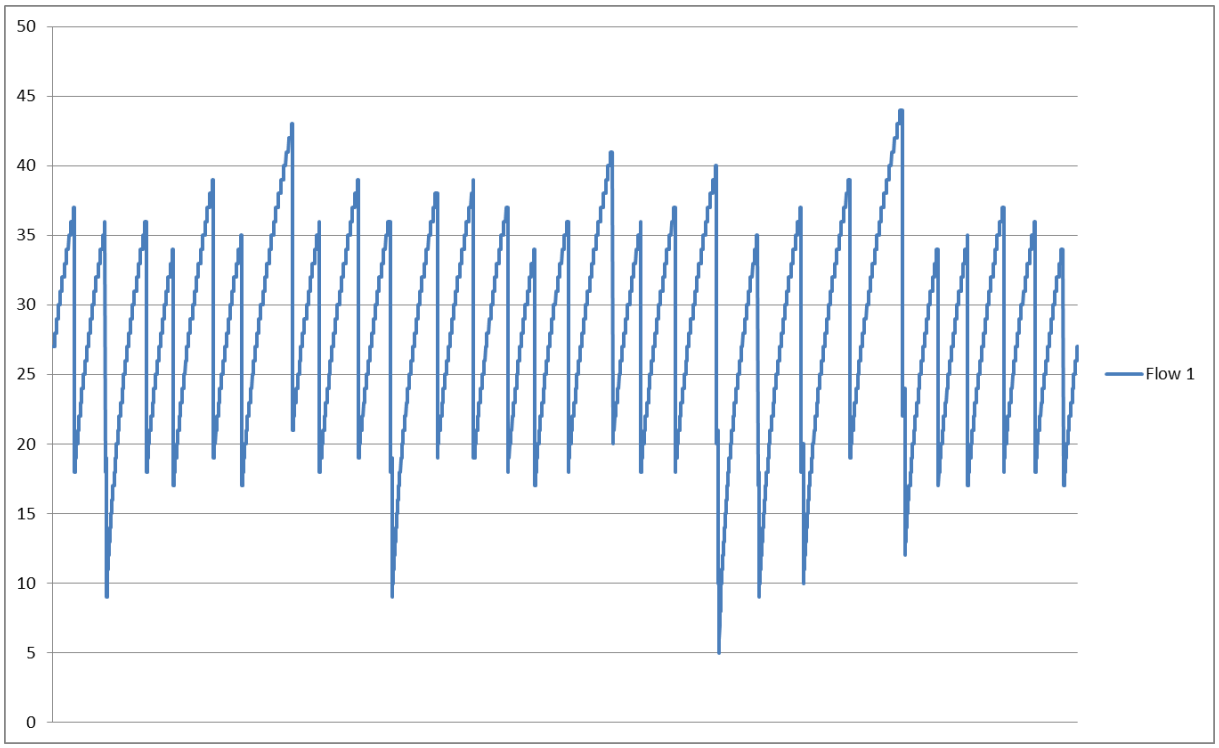

Figure 13. Two hosts experiment, single flow, rina-tcp-red-ps: Evolution of window size (PDUs)

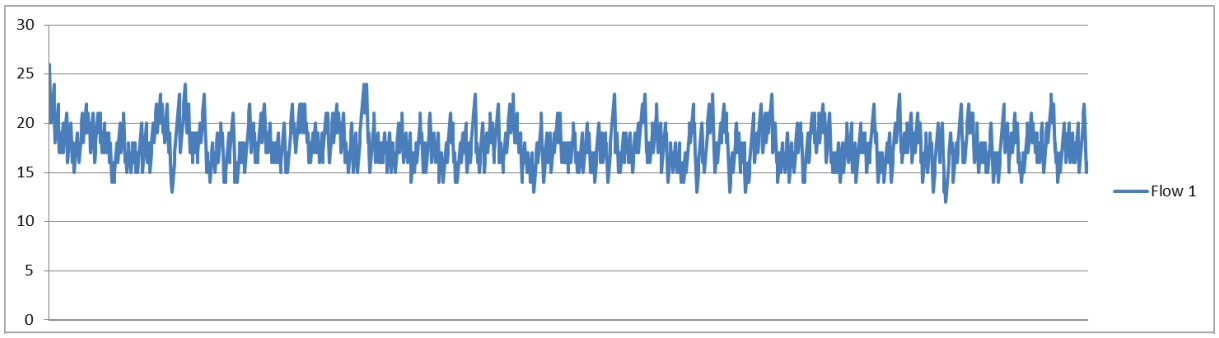

Figure 14. Two hosts experiment, single flow, rina-jain-ps: Evolution of window size (PDUs)

Figure 15 and Figure 16 depicts the occupation of the queue in the router in case of TCP-RED policy. The blue line corresponds to the actual amount of PDUs on each moment, while the red line corresponds to the average occupation calculated at each time. Figure 16 zooms a portion of Figure 15 in order to better see the average occupation. Correspondingly, Figure 17 and Figure 18 depict the same variable for the case of Jain's scheme. It can clearly be observed that in the latter case the occupation of the queue is lower than in the TCP-RED case since PDUs are marked sooner in Jain (queue average size of 1 or more). It is important to stress that, even the

calculation of the average queue occupation is done using fixed point operations, the value that is considered to take the decision on marking the PDU, and the one depicted in the graph, is an integer. This is the reason why sometimes the average value deviates a bit.

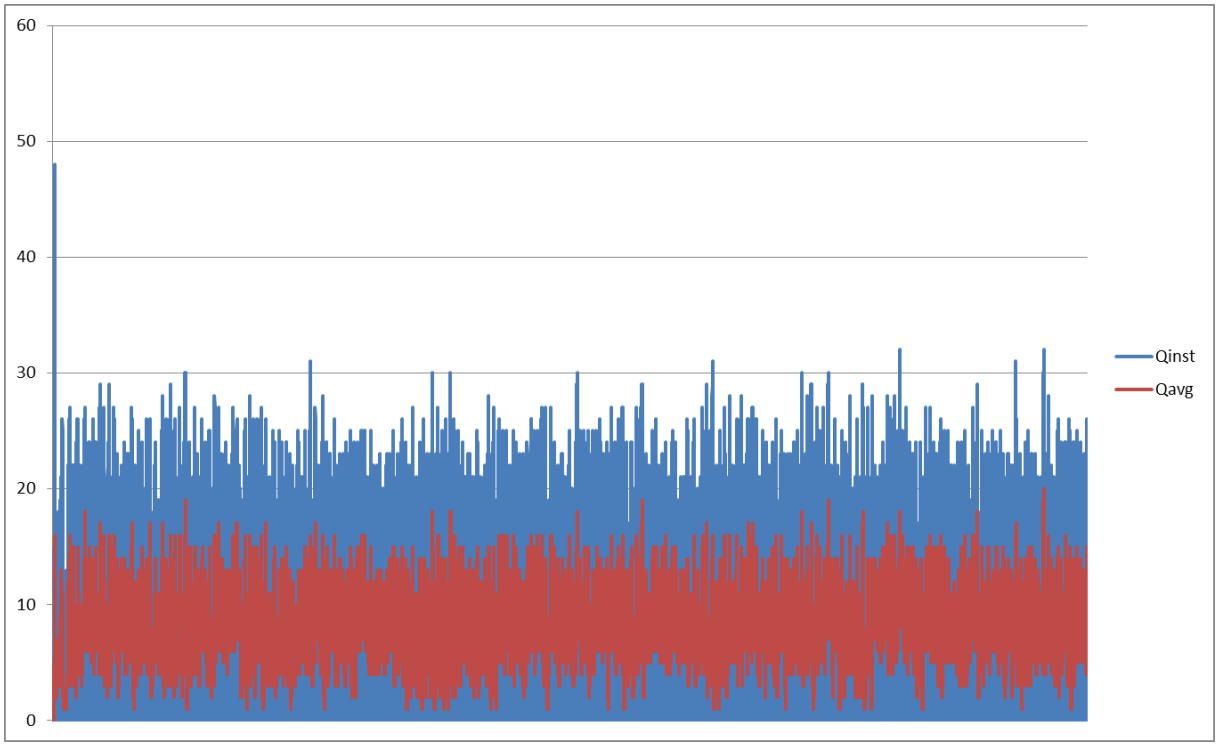

Figure 15. Two hosts experiment, single flow, rina-tcp-red-ps: Instant and average queue occupation (PDUs)

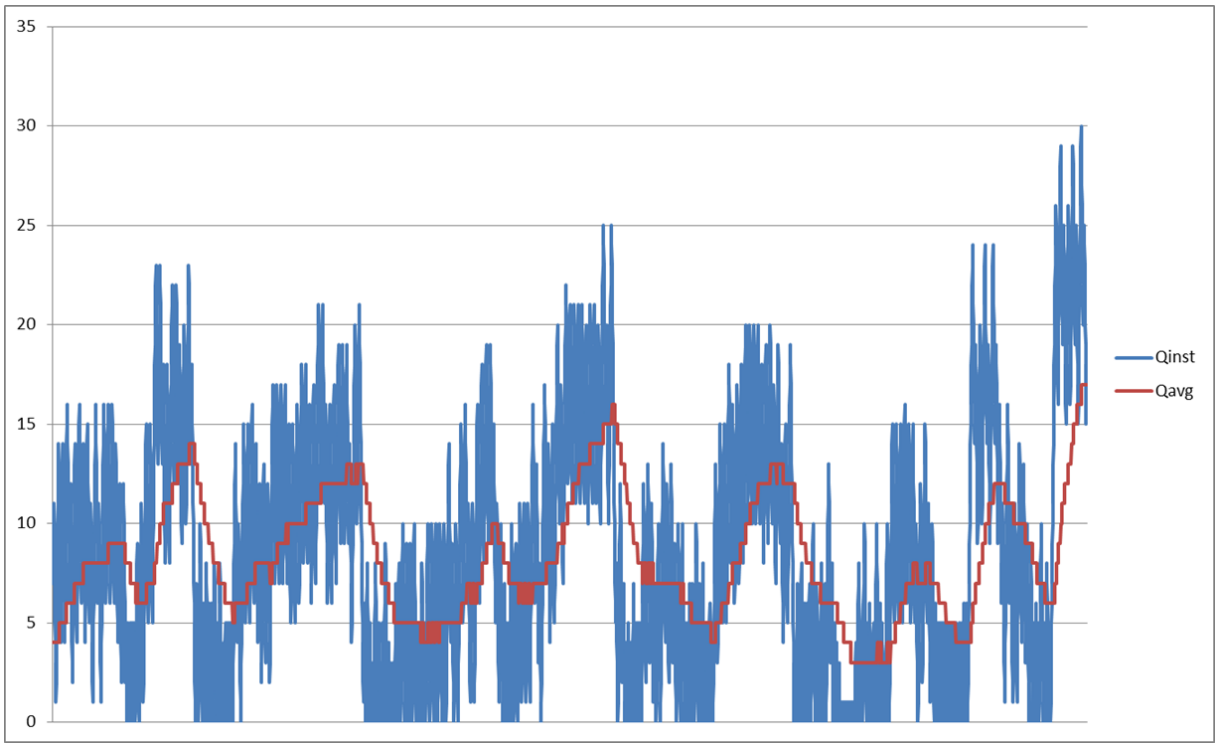

Figure 16. Two hosts experiment, single flow, rina-tcp-red-ps: Instant and average queue occupation - detail (PDUs)

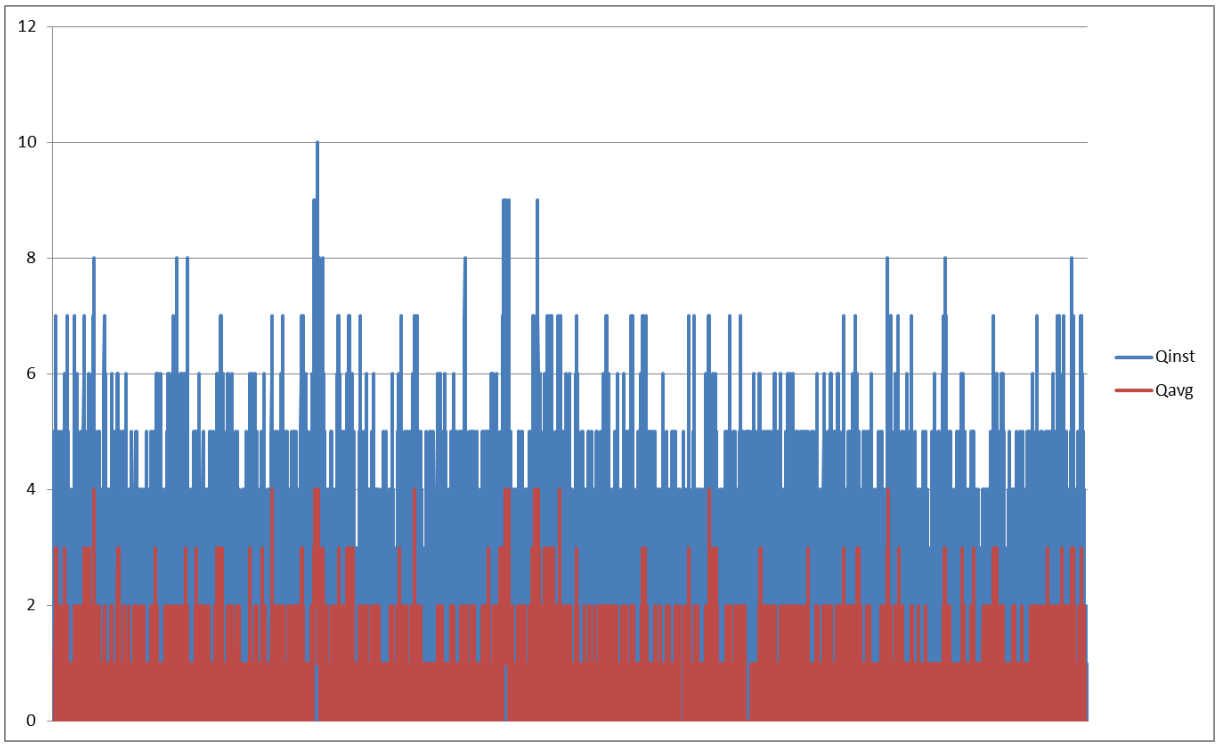

Figure 17. Two hosts experiment, single flow, rina-jain-ps: Instant and average queue occupation (PDUs)

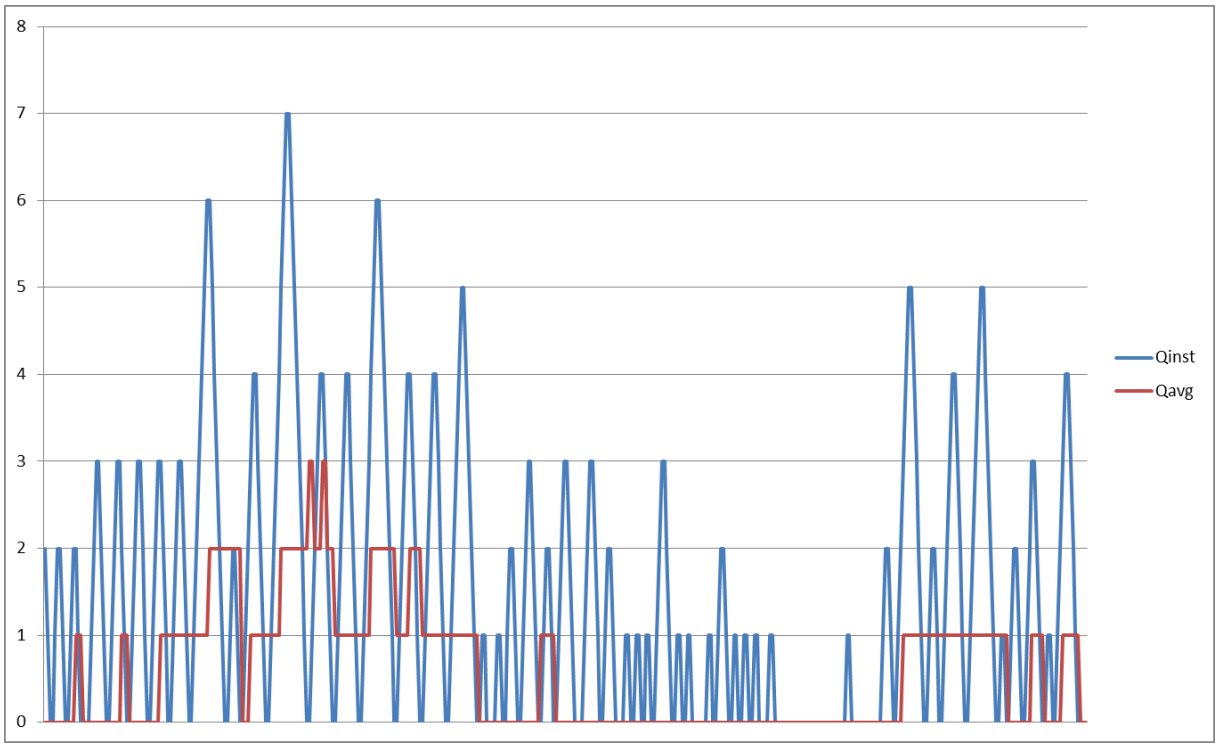

Figure 18. Two hosts experiment, single flow, rina-jain-ps: Instant and average queue occupation - detail (PDUs)

Concluding remarks

The above scenario which was reported in D3.2 is a basic topology of evaluating congestion control.

Our evaluation on this scenario showed that RINA is capable of running different congestion control policies on different pieces of a connection. Implementing two policy sets – TCP/RED and Jain PS – confirmed that per-DIF congestion control, as the third goals in this experiment, is automatically possible in RINA no matter if the DIF is operating on a single link or it is an end-to-end one.

Figures 3 and 4 additionally illustrated that our implementations of a TCP-like congestion controller and the Jain PS method were valid, and referring to Figure 2, their throughput was close to the bandwidth of the bottleneck link.

Figures 5 to 8 also validated the correct operation of the RINA SDK and our implementations of TCP/RED and Jain PS since the results were similar to those in the literature about queue size properties of these two methods.

3.3.6.2. 4 parallel flows

Figure 19 shows the experiment results using iperf and TCP with four simultaneous flows, while Figure 20 does the same for RINA using the TCP-RED congestion avoidance policy set and Figure 21 does it using the Jain's policy set. These three figures highlight the higher instabilities of TCP flows compared to utilizing RINA as concluded in the case of single flow experiments. Since there are now four flows competing against each other for the same shared resource without explicit congestion feedback, the behaviour of the TCP flows is more chaotic than in the single flow case. In contrast, the flows provided by RINA continue having similar oscillations as in the single flow case. Figure 23 illustrates the evolution of the window sizes of the four flows during the experiment using RINA-Jain policy set, showing that they all oscillate around the `knee´, getting a fair share of the bottleneck resource. Figure 22 shows the evolution of the window sizes in the case of RINA TCP-RED policy set. In this case we can see a bit higher oscillation due to the combination of halving the credit when an ECN flag is received and periods of constant credit (during the time the credit is increased by *1/credit*) until it is finally increased 1 or halved.

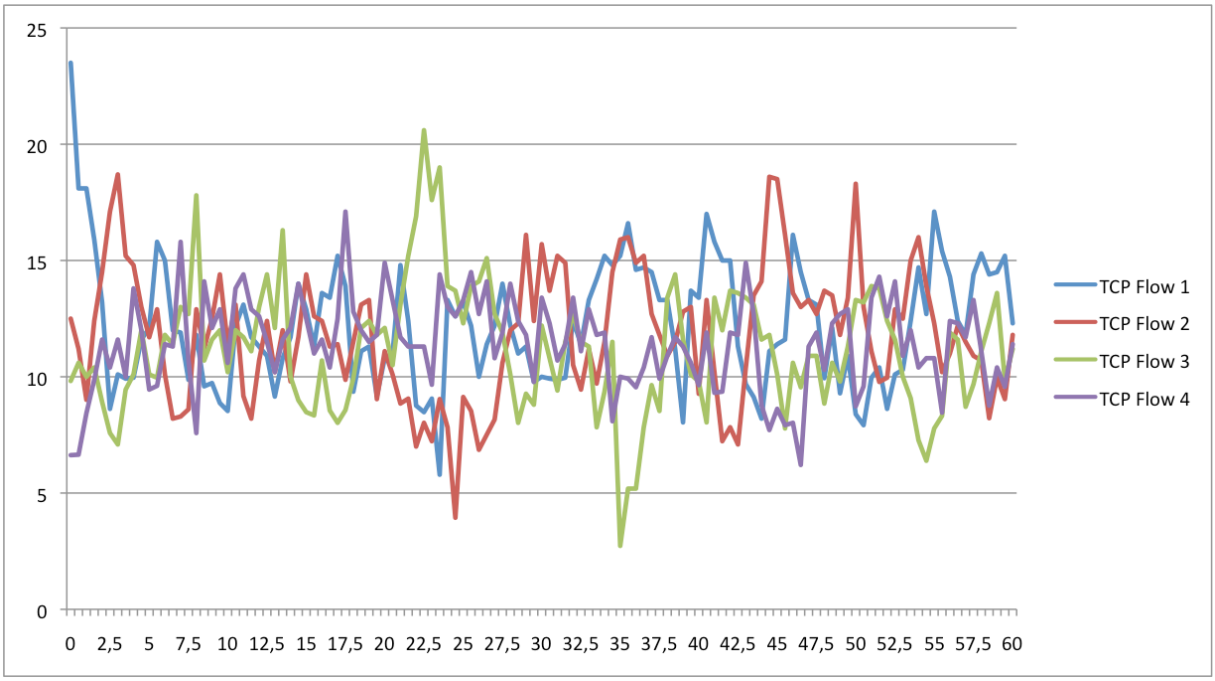

Figure 19. Two hosts experiment, four flows,
TCP: Throughput (Mbps) vs. Experiment time (s)

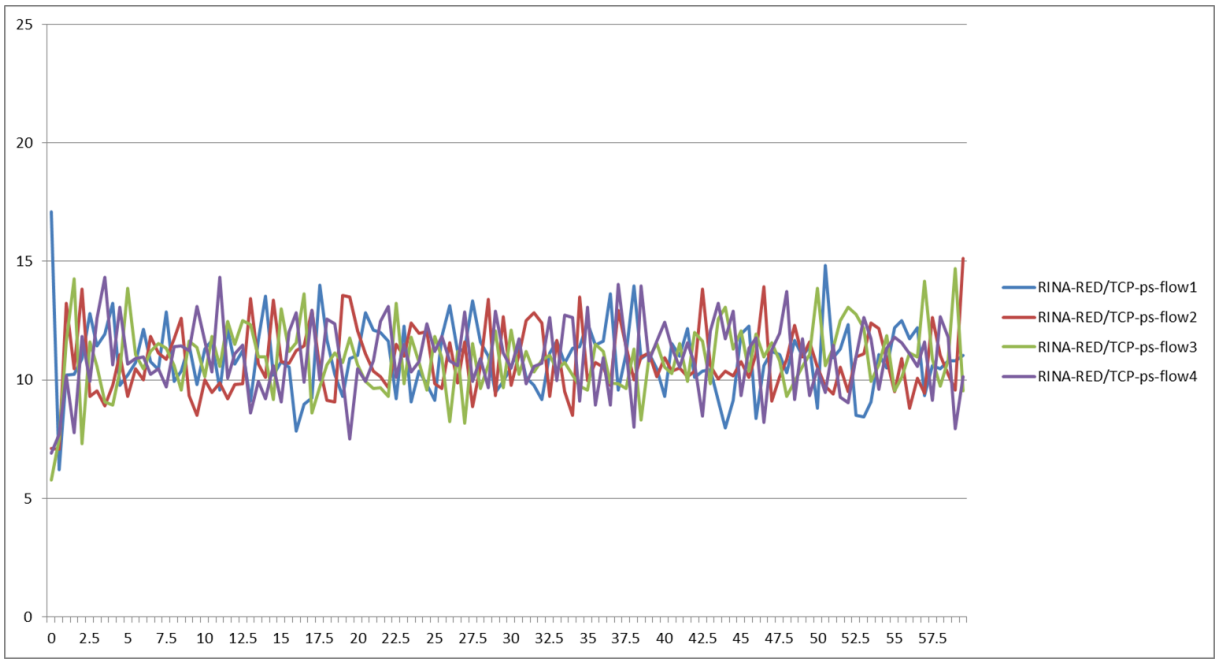

Figure 20. Two hosts experiment, four flows, RINA-tcp-red-ps: Throughput (Mbps) vs. Experiment time (s)

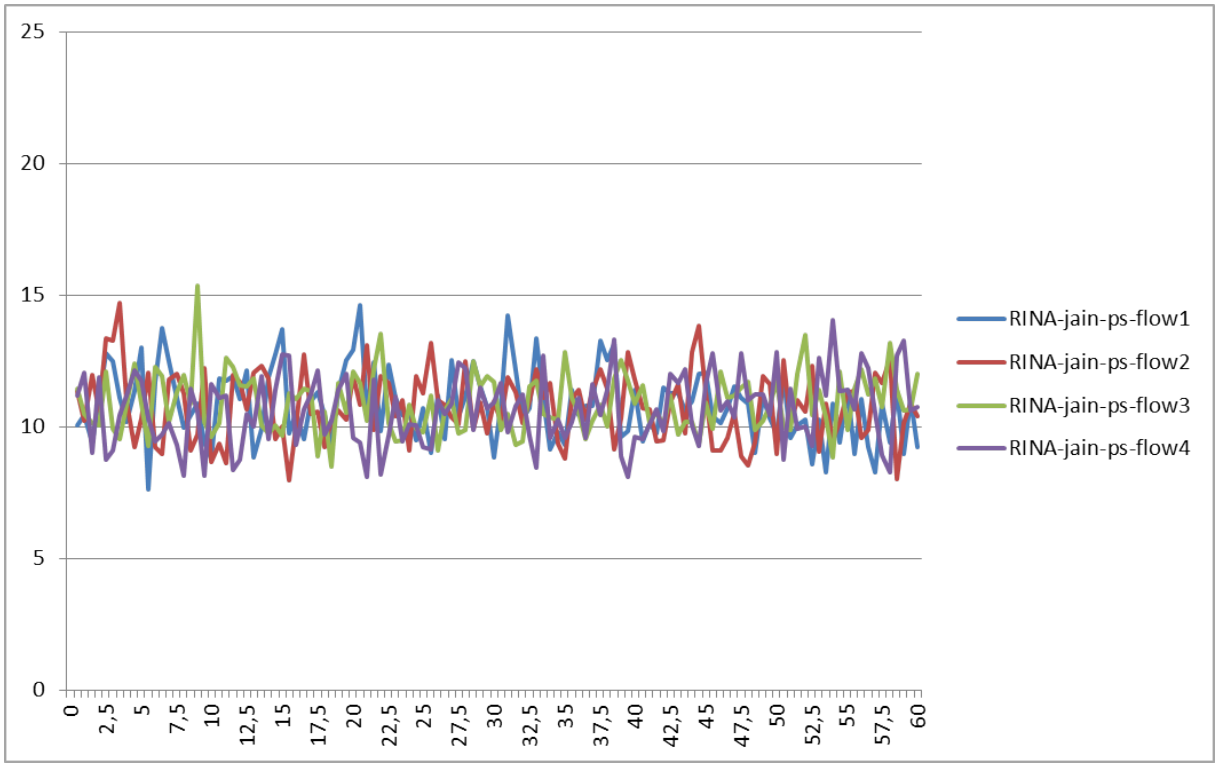

Figure 21. Two hosts experiment, four flows, RINA-jain-ps: Throughput (Mbps) vs. Experiment time (s)

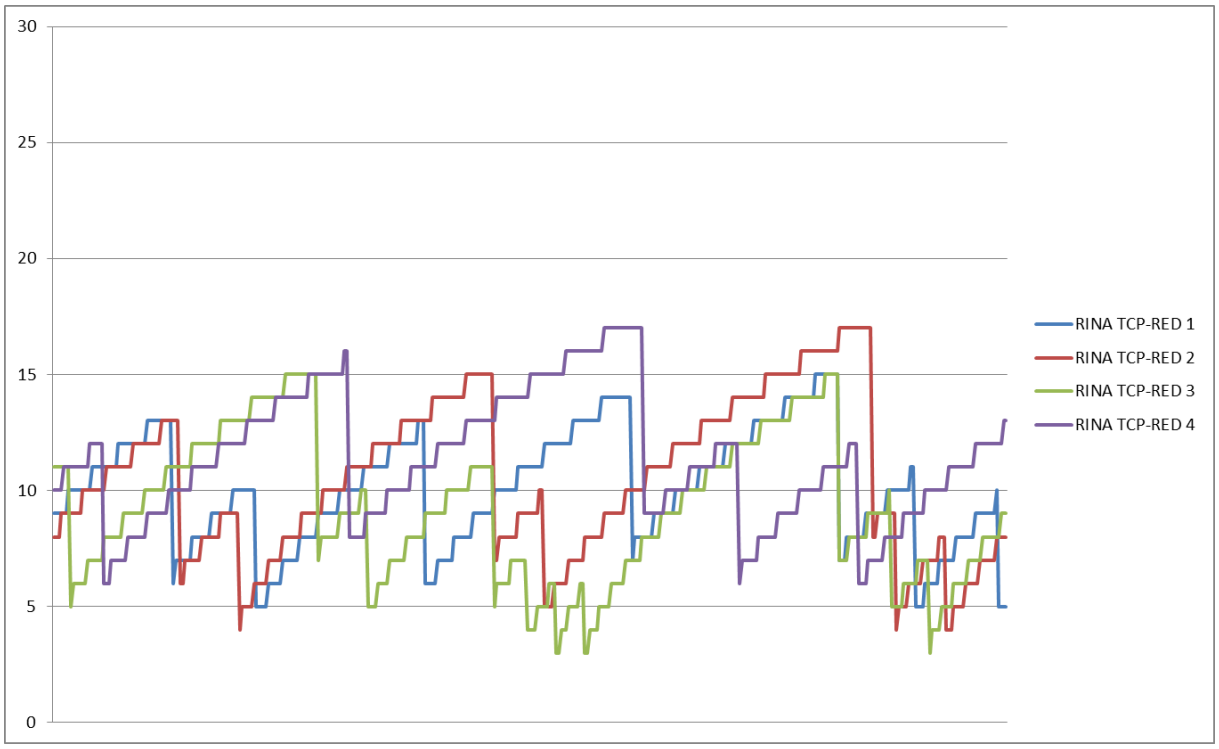

Figure 22. Two hosts experiment, four flows, rina-tcp-red-ps: Evolution of window size in time(PDUs)

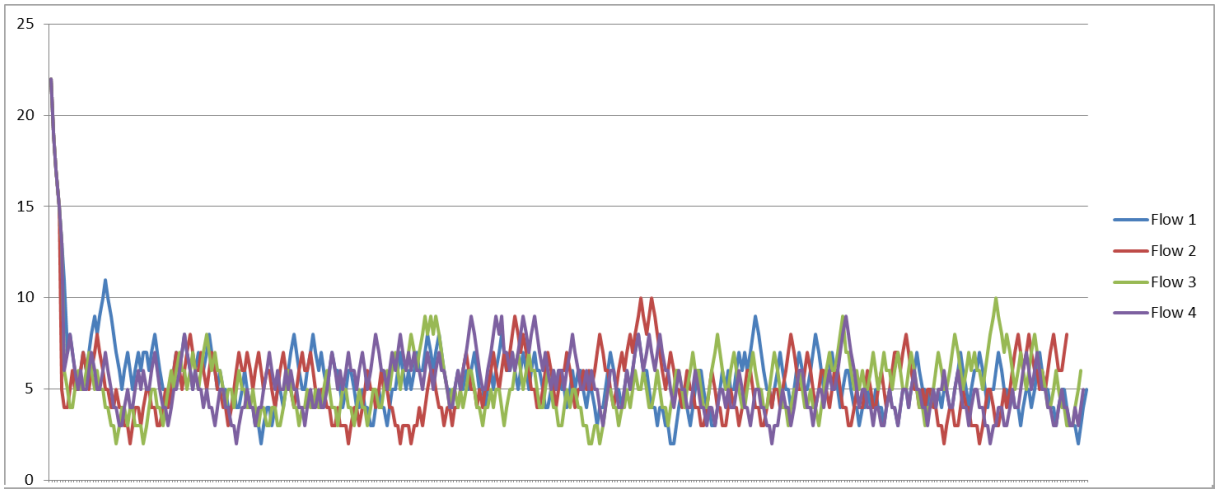

Figure 23. Two hosts experiment, four flows, rina-jain-ps: Evolution of window size in time(PDUs)

Figure 24 and Figure 26 depict the queue occupation in the Router for the case of TCP-RED and Jain respectively when being fed by 4 concurrent source flows. As expected, it can be observed that the average queue occupation with both policy sets is higher than the respective case with only one flow. Figure 25 and Figure 27 highlight the average queue occupation curve.

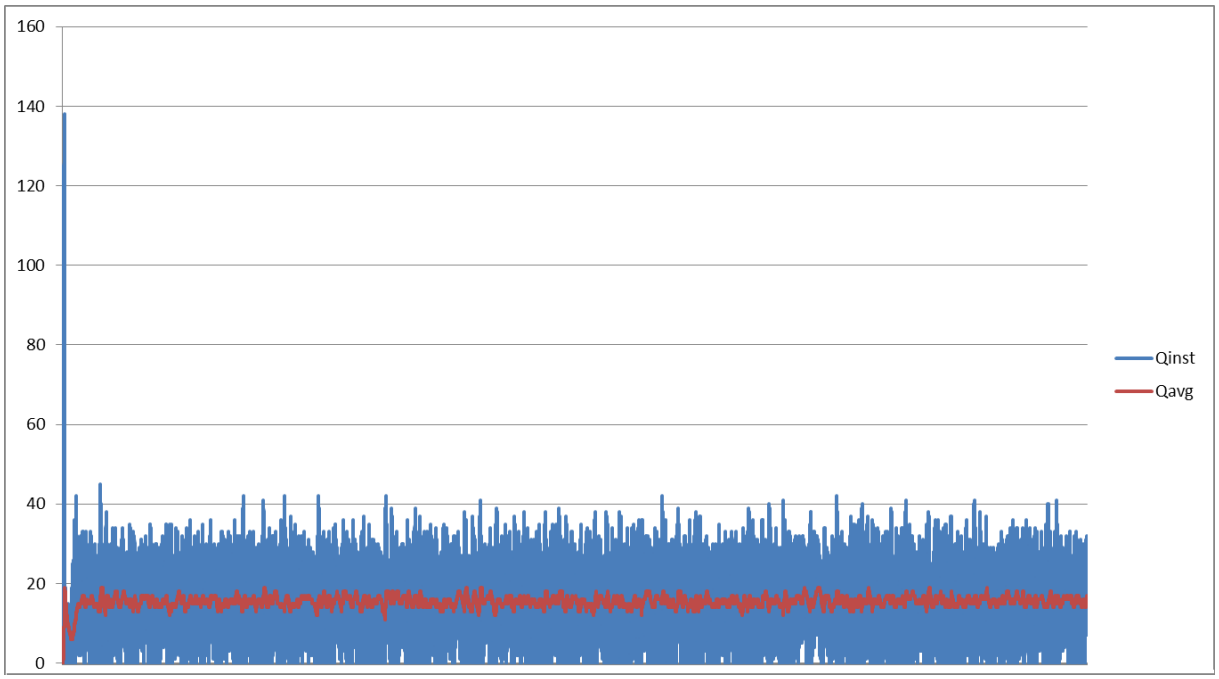

Figure 24. Two hosts experiment, four flows, rina-tcp-red-ps: Instant and average queue occupation in time(PDUs)

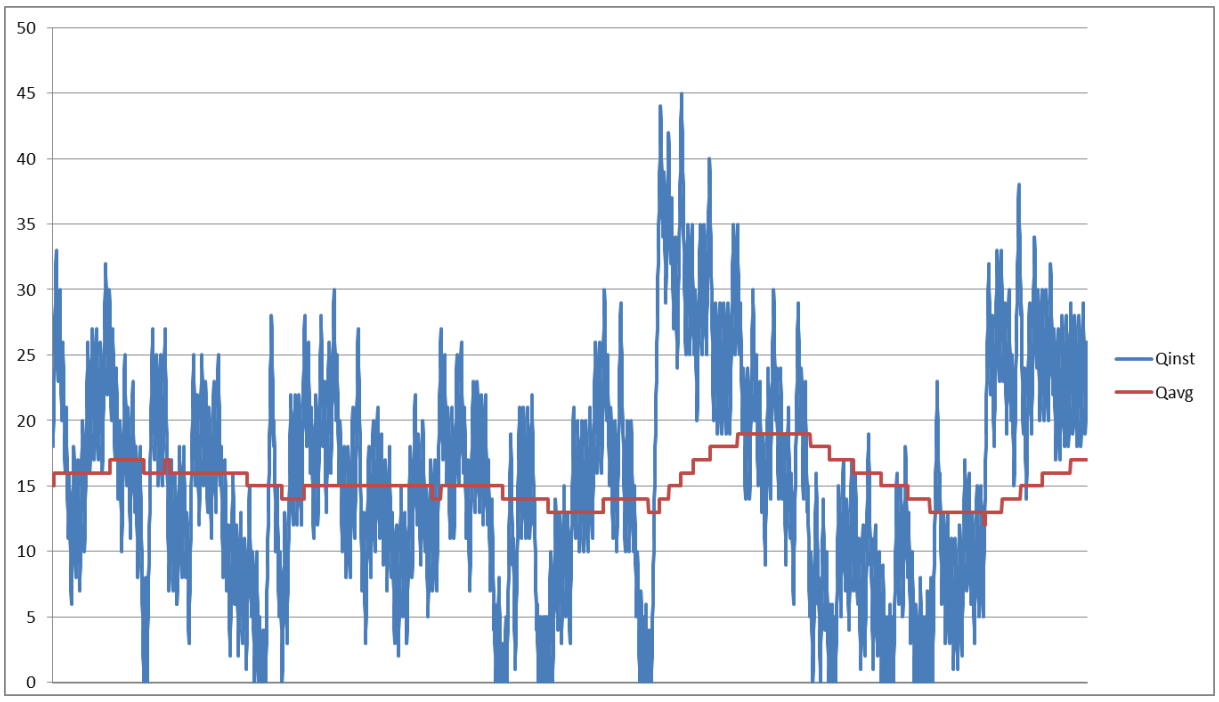

Figure 25. Two hosts experiment, four flows, rina-tcp-red-ps: Instant and average queue occupation in time - detail (PDUs)

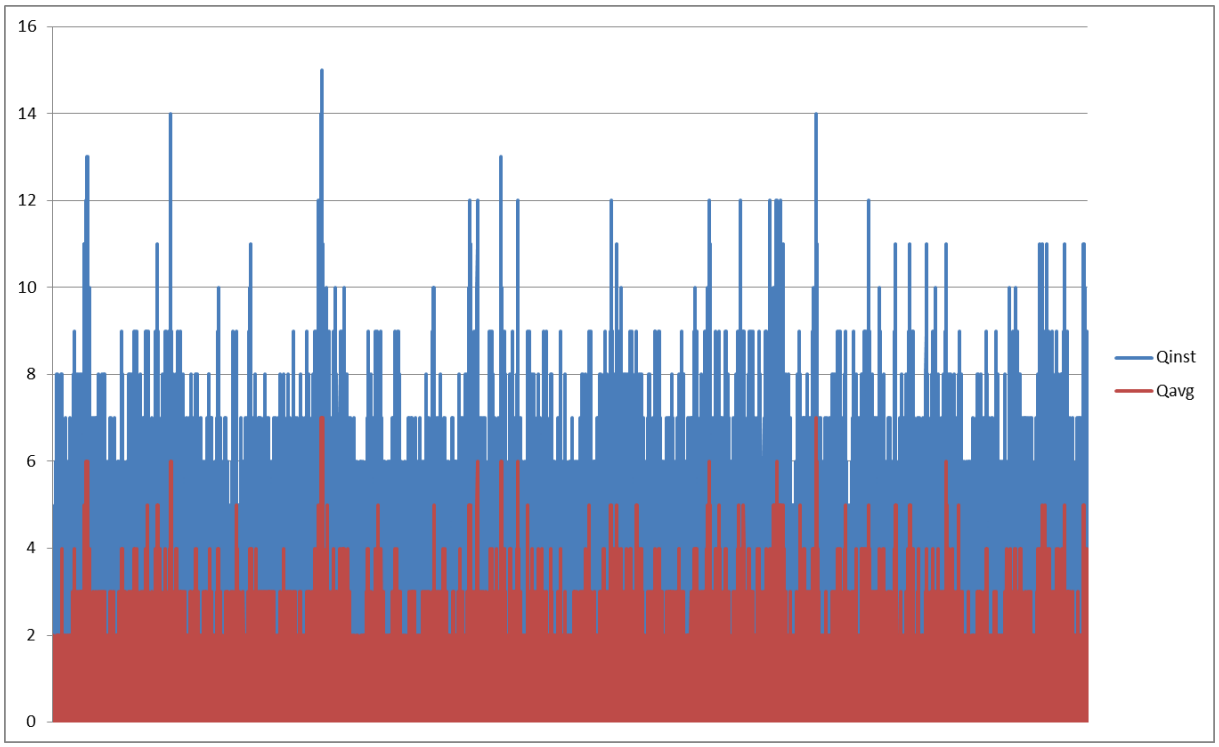

Figure 26. Two hosts experiment, four flows, rina-jain-ps: Instant and average queue occupation in time (PDUs)

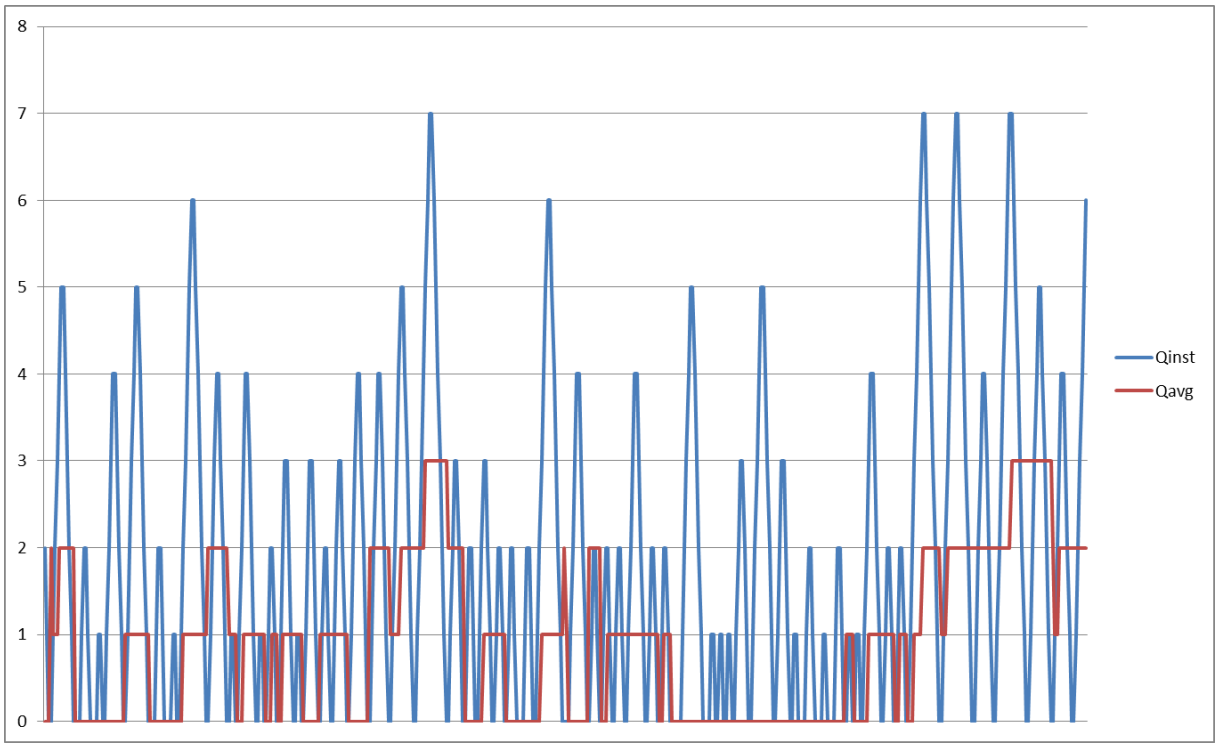

Figure 27. Two hosts experiment, four flows, rina-jain-ps: Instant and average queue occupation in time - detail (PDUs)

Concluding remarks

The above scenario is the extension of the previous one which helps evaluate how multiple flows compete for the bottleneck link in the network topology.

Referring to Figure 9 to 11, although all the four flows transmitted almost the same volume of data in the end, TCP showed a higher flow throughput oscillation in short-term; this was, however, mitigated in the TCP/RED and Jain PS policy implementations of RINA, leading to a higher short-term fairness among flows.

Investigating the congestion window size behaviour in Figures 12 and 13 also confirmed the correct operation of the two implemented policy sets in case of parallel flows. Moreover, Figures 14 to 17 implied that our congestion control policy implementations were capable of keeping the queue size of the bottleneck link small.

3.3.7. Feedback towards development

The development, debugging and testing of these congestion management related policy sets uncovered a number of bugs in the RINA implementation as well as some changes required in the SDK. The following bullet points summarize the main feedback items towards the RINA prototype development (which have already been addressed and released in the most up to date development branch, *pristine-1.3*[1]).

- A number of DTCP policy hooks have been updated to accept an entire PCI struct as an input parameter instead of only a sequence number, implemented by PR 682[2]. This allows for greater flexibility in the related policy implementations, since more information about the PDU is known.
- One of the functions of the RMT policy set (*rmt-q-monitor-tx-policy*, which was called when a PDU was enqueued or dequeued by the RMT in an output port for transmission) has been split into two separate functions: *rmt_q_monitor_policy_tx_enq* (called when the PDU is enqueued) and *rmt_q_monitor_policy_tx_deq* (called when the PDU

[1] https://github.com/irati/stack/tree/pristine-1.3
[2] https://github.com/IRATI/stack/pull/682

is dequeued). This eases or even enables the work of the policy implementations when calculating different parameters of the n1_port queues. The fixes have been implemented by PR 687[3].

- DTCP policy sets can set functions that are always used by the main code to NULL (such as the RTT estimator policy), opening up the opportunity to cause kernel bugs. The proper solution is for each component to define a number of mandatory functions that have to be implemented by all policy sets, and for the SDK to enforce this upon loading the plugin containing the policy set functions. This feature will be implemented in the second version of the SDK; meanwhile a temporary solution to avoid kernel bugs has been introduced by PR 715[4].

Moreover, some design improvements have been identified and will be scheduled for the next iteration of the prototype in version 1.4:

- The API of the RMT and the RMT policy set need to be refactored to provide a better integration.
- A new method for retrieving operation data is required in order to avoid logging to a file on every PDU operation since this highly decreases the prototype's performance and the veracity of the results. Using sysfs/procfs can be an option as it was already used in the case of the RINA TCP-RED policy-set.

3.3.8. Technical impact towards use cases

In spite of the simplicity of the experiment scenarios, it already illustrates two main benefits of the RINA architecture. The first one is that Explicit Congestion Notification and vertical pushback between layers enable a faster and more efficient response to congestion than the prevalent end-to-end implicit feedback approach used in the Internet. ECN allows congestion effects to be confined closer to where congestion happens instead of affecting the whole network: each DIF manages the congestion in its own resources. This allows network providers to utilize network resources more efficiently and to run networks at higher utilization.

The second advantage inferred from this experiment is that each DIF can use a different congestion management solution, the one that is better

[3] https://github.com/IRATI/stack/pull/687
[4] https://github.com/IRATI/stack/pull/715

suited to its operational environment. There is not a single congestion management solution that will be effective in all networking environments, therefore being able to deploy a variety of solutions in different parts of the network using the same architectural framework is a great advantage over the state of the art. This feature will allow for a customized and higher-efficient response to congestion which, combined with the first feature, will lead to networks that adapt better to congestion and can therefore be safely run at a higher utilization.

Last but not least, since flows are aggregated at lower layer DIFs, as also reported by [D32], the use cases can gain a lot from this property by lowering the number of competing flows significantly. Here, we can see that, for example, in the 4 parallel flows scenario, all the flows were aggregated into one flow when they were competing for the bandwidth in the bottleneck link. Our next step is to examine this feature in further details on larger network topologies.

3.4. Performance Isolation in Multi-Tenants Data-Centres

3.4.1. Short description

This set of experiments aims at analyzing the behavior of RINA in multi-tenants data-centers, investigating performance isolation and network fairness aspects. The experiment aims also at validating the behavior of different RINA components, namely EFCP, and RMT. Moreover, this experiment aims at validating the implementation of a rate-based flow control mechanism that was added to the IRATI stack implementation as a pre-requisite for the performance isolation experiment.

3.4.1.1. Experiment goals

This set of experiments will span both development cycles. In the first cycles the congestion control policies will be tested in a scaled-down 4-nodes topology. The goal, in this phase, is to test the correct behavior of the policies as well as to validate the RINA stack and the testing facilities. This phase will provide testbeds owner and SDK developers with feedback about missing features and bugs. In the second cycle the experiment will be extended to larger fat-tree topology and the Equal-Cost Multiple-Path routing policies will also be implemented providing an holistic solution for performance isolation in multi-tenants data-centres.

The DIF configuration used in this set of experiments is depicted below in Figure 28.

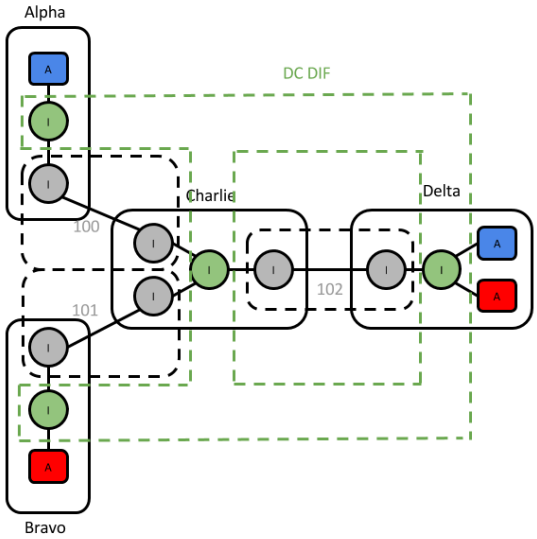

Figure 28. Data-centre congestion control experiments: DIF Configuration

3.4.1.2. Brief description of the policies

The performance isolation policies used in this experiment are based on explicit feedback. Three main logical steps can be identified: rate-based flow control at the sender IPC Process (IPCP), ECN marking at the point of congestion, and receiver feedback.

- RMT Policies. RMT queues at the ToR switches (DC-DIF) are monitored in order to identify possible congestion situations. When the average queue length is higher than a certain threshold then the passing flows are ECN-marked. It is worth noticing that the full-bisection bandwidth assumption ensures us that the only point in the network where congestion can occur is at the edges, i.e. in the link between ToR and servers.
- EFCP Policies. Receiver IPCPs at the servers (DC-DIF) look for ECN-marked flows. If an ECN marked flow is found then the optimal transmission rate for the incoming flows is computed. Notice that this information is available at the server in that tenants are admitted only if there are enough resources for their VMs (bandwidth hose model is used).

- ECMP Policies. Equal-Cost Multi-Path (ECMP) routing policies will be implemented in the second development to randomize routing at the level of flows ensuring optimal utilization of the date-centre resources.

3.4.1.3. Use Cases and requirements addressed

Data-centre networking. In multi-tenants data-centers computing and networking resources are shared. This allows for several economies of scale but, at the same time, it raises concerns about performance predictability for CPU, memory, disk and network bandwidth. It is then critical that tenants running various jobs (e.g. MapReduce, Storage, etc.) are isolated from one another so that the network activity of one tenant does not adversely impact other tenants. Moreover, it is also important that tenants can use spare network resources (when they are available) and that a simple resource request model is employed. We tackle the first requirement by implementing work-conserving policies and the later by relaying on the hose model for specifying the tenants' bandwidth requirements.

3.4.2. Experiment 1: Rate-based flow control

3.4.2.1. Short description

This experiment aims at demonstrating rate-based flow control mechanisms in the data-centre use case. It is worth noticing that, although in this experiment the rate-based flow control is used within the data-centres use case, it implementation is completely general purpose and is effectively available to all the other policies and applications implemented using the PRISTINE SDK.

In the envisioned performance isolation solution, traffic shaping at the network edges is assumed in order to enforce bandwidth demands. Based on this model, tenants would need to specify just one additional parameter, i.e., the virtual network interface card (VNIC) bandwidth, alongside with the amount of memory and the number of CPU cores. However the IRATI stack delivered to PRISTINE could only support window-based flow control. The first step of the experiment was then to extend the IRATI stack with the required rate-based flow control as a prerequisite for the performance isolation policies.

3.4.2.2. Experiment goals

Evaluating the performances of the rate-based flow developed by CREATE-NET. The experiment aims at validating the implementation of an extension to the RINA stack which adds supports for rate-constrained EFCP flows. The experiment exploits multiple flows while assessing the correctness and the stability of the implemented rate-based flow control mechanism.

3.4.2.3. Metrics and KPIs

1. Throughput between pairs of servers
2. Throughput stability

3.4.2.4. Experiment configuration

Figure 29 depicts the experiment setup. In this scenario three servers (S1, S2, and S3) are attached to a fourth relaying node T. The scenario assume two tenants (Red and Blue). The bandwidth requests made by the Red and the Blue tenants are, respectively, 4 Mb/s and 2 Mb/s. All links are 10 Mb/s.

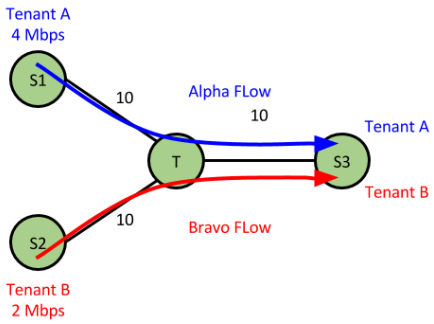

Figure 29. Experiment 1: Network setup

1. Experiments are carried out for different application payload lengths (from 100 bytes to 1400 bytes in steps of 100 bytes).

2. Results are analyzed in order to ensure that the target end-to-end bitrate is respected and that the rate shaping is stable.
3. Results are also compared with a deployment with window-based flow control.

3.4.2.5. Experiment steps

1. Setup a basic topology with 4 nodes (2 transmitter, 1 receiver, and 1 relaying node)
2. Deploy a DIF on top of the three shim DIFs
3. Start one flow from the transmitter to the receiver with target rate of 4 Mb/s
4. Wait 30s
5. Start the second flow from the transmitter to the receiver with target rate of 2 Mb/s
6. Throughput and latency measurements are taken.
7. Repeat for different payload lengths

3.4.2.6. Result evaluation

Figure 30 shows the throughput of two flows (alpha and bravo) vs. the experiment time in the case of 1400 bytes-long datagrams. In this scenario the target throughput for the alpha flow has been set to 4 Mb/s while the target throughput for the bravo flow has been set to 2 Mb/s. Both transmitters are operating in saturation regime. As it can be seen from the figure, the target bit-rates are closely followed by the two flows.

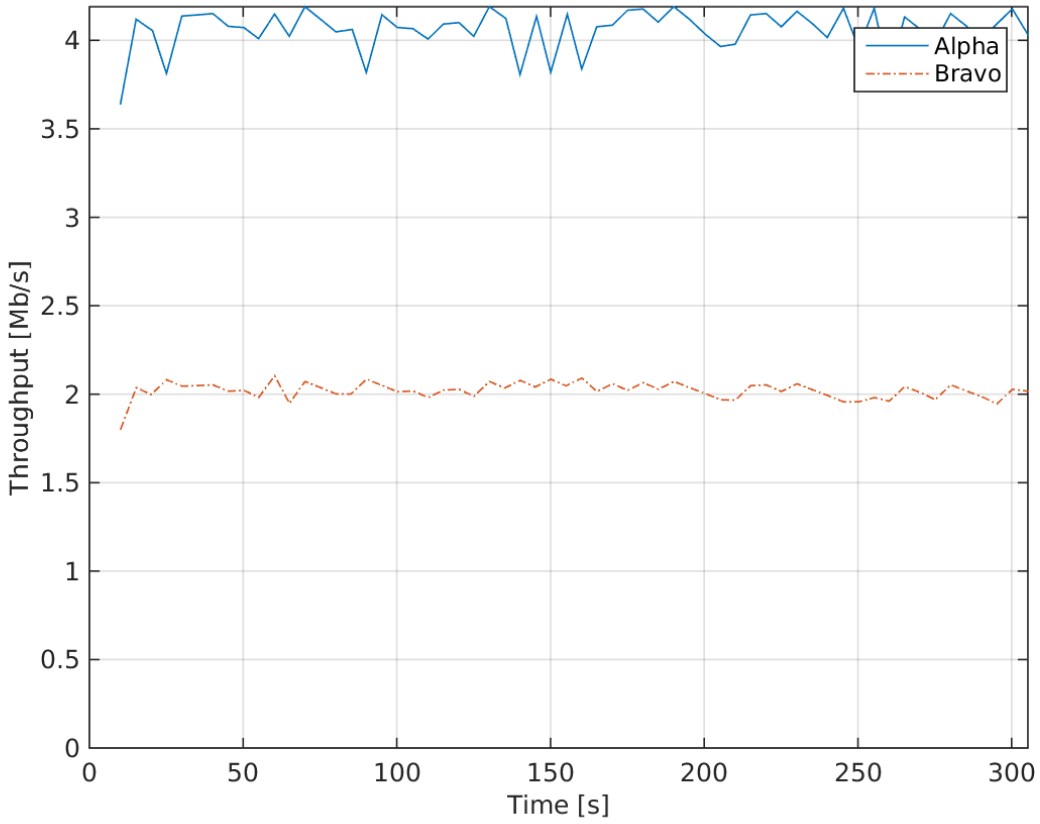

Figure 30. Rate-based flow control: Two flows (alpha and bravo). Results are in Mb/s (s)

Figure 31 shows the throughput distribution for alpha vs. the experiment time using different payload sizes. As it can be seen the median throughput stabilizes around the target throughput for payloads longer than 400 bytes. We ascribe this behavior to the preliminary nature of the rate-based flow control (which we remind the reader was not available in the IRATI stack when this experimentation begun). We plan to address this performance pitfall during the second development cycle. A similar consideration can be made also for the bravo flow whose results are plotted in Figure 32.

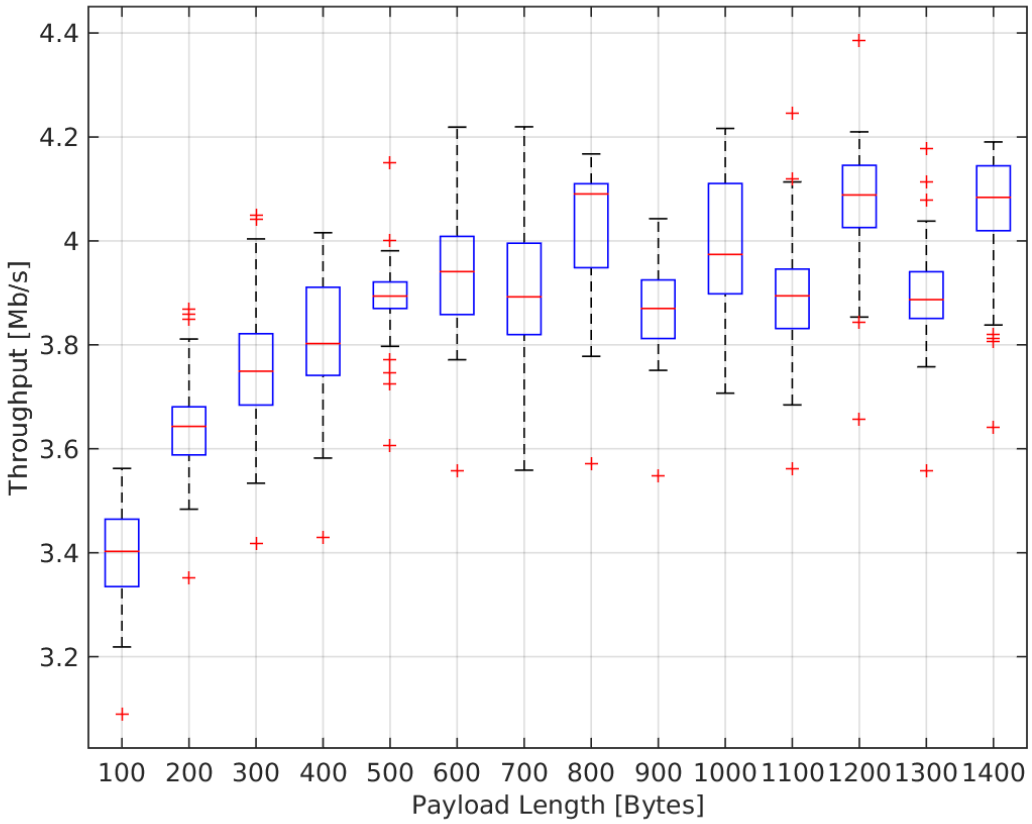

Figure 31. Rate-based flow control: Throughput distribution for different payload sizes (alpha flow). Results are in Mb/s (s)

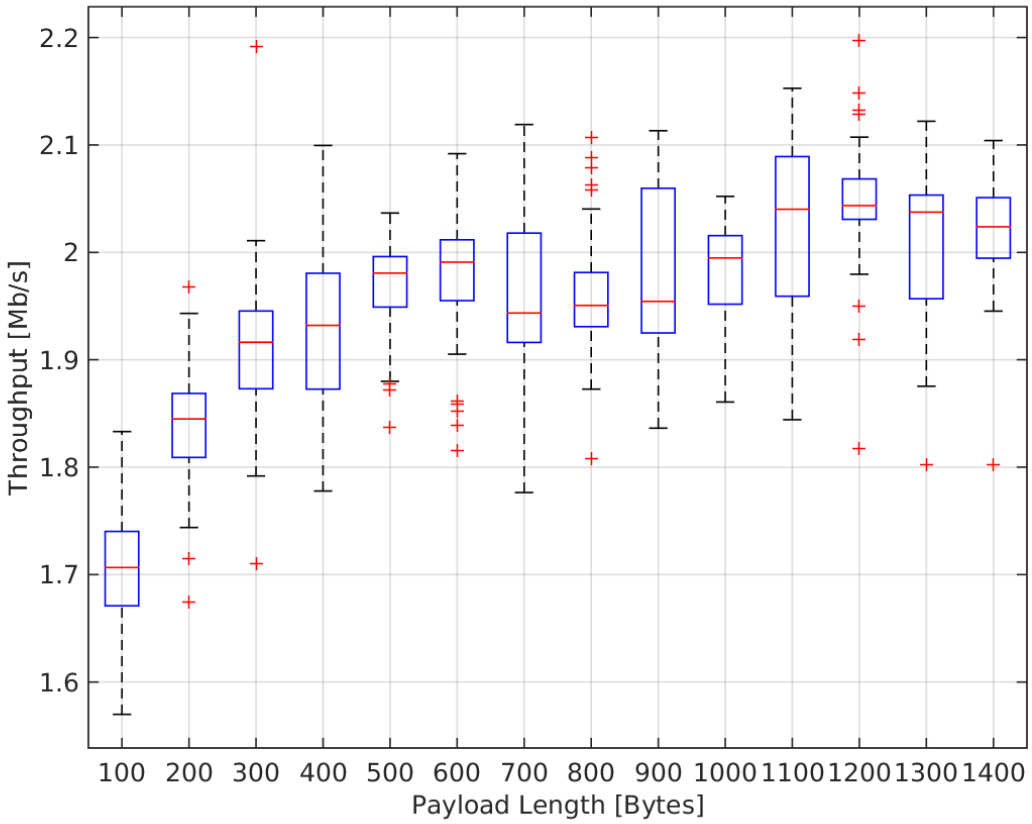

Figure 32. Rate-based flow control: Throughput distribution for different payload sizes (bravo flow). Results are in Mb/s (s)

Finally, in Figure 33 we plot the results of another experiment, using the same topology, performed using rate-based flow control with two concurrent flows. Each flow is configured to respect a rate of 10 Mb/s. As expected given the TCP-like behavior of the rate-based rate control, the two equally share the available bandwidth. It is worth noticing that the actual sum of the alpha and bravo flows throughput is slightly lower than the link capacity. This behavior can be ascribed to maturity level of the code base.

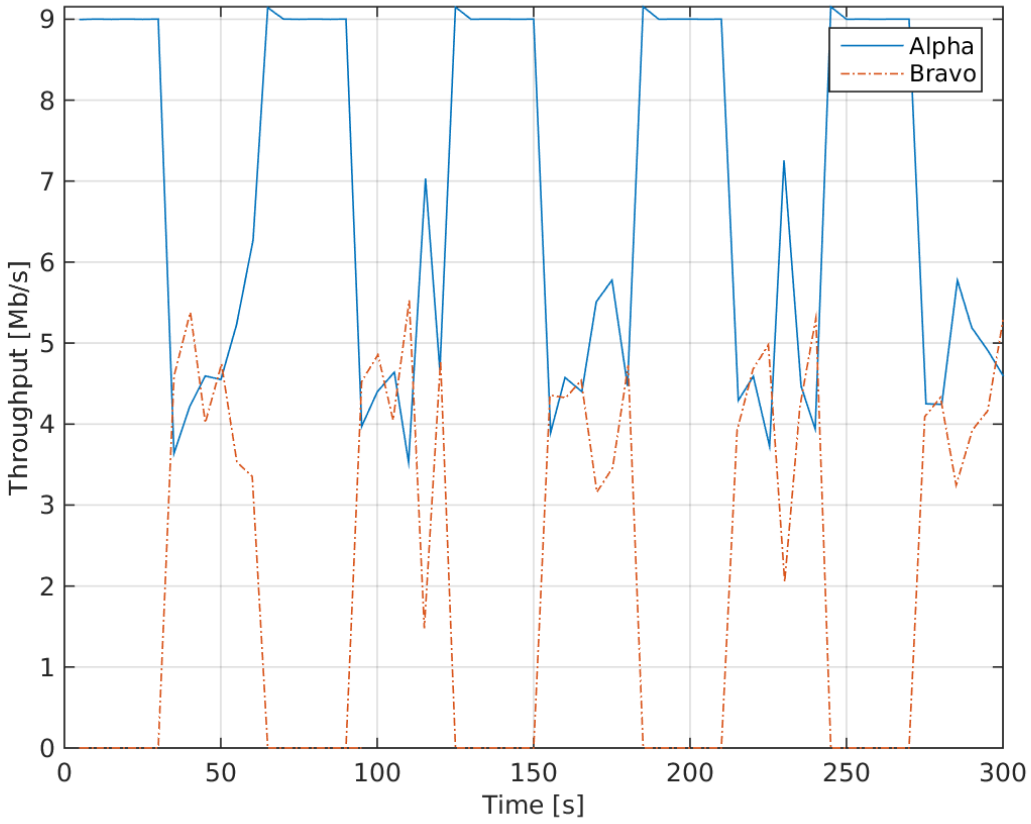

Figure 33. Rate-based flow control: Two flows (alpha and bravo). Results are in Mb/s (s)

3.4.3. Experiment 2: Performance isolation in data-centres (basic scenario with no routing)

3.4.3.1. Short description

This experiment aims at demonstrating ECN-based congestion control mechanisms in the data-centre use case. This experiment shall demonstrate how by leveraging on RINA recursive architecture it is possible to achieve basic performance isolation in multi-tenant data-centres. As opposed to solutions available in the literature that require quite complex modification to the Linux networking stack (e.g. EyeQ). RINA allows to achieve the same results with simple and atomic policies consisting of a few hundreds LOC. This improves both re-usability and security. Moreover, being the final system more homogeneous, not ad-hoc management tools are necessary. The later advantage is expected to play a key role w.r.t the business impact of RINA.

In the envisioned performance isolation scheme (inspired by EyeQ), tenants require a minimum bandwidth for each of their VMs (in this preliminary study we assume that the same minimum bandwidth is specified for each network slice). The tenants expect the system to be work-preserving meaning that unused bandwidth shall be allocated to the VM hosted in a certain server. Target bandwidths must be recomputed when a previously inactive VM starts generating or receiving traffic.

3.4.3.2. Requirements

Uniform high capacity, Performance isolation, Full-bisection bandwidth topology, intra-data-centre traffic, continuous traffic, rate-based flow control

3.4.3.3. Goals

Evaluating the performances of the congestion control mechanism developed by CREATE-NET is a key step toward the goal of achieving a full performance isolation solution for data-centres. The final solution will include also a traffic aware ECMP implementation as well as a centralized manager for network configuration. The main expected advantages of the proposed solutions are: (i) faster reaction to congestion situations (compared to legacy TCP/IP networks); (ii) dramatically simpler code base (compared to solutions like EyeQ; and (iii) standardized network management.

3.4.3.4. Metrics and KPIs

1. Maximum throughput between pairs of servers

3.4.3.5. Experiment configuration

We consider the scenario depicted in Figure 34. VMs of tenants Red and Blue are given a minimum bandwidth guarantee of 2Mb/s and 8Mb/s respectively. The Red tenant starts a flow from S1 to S3. In the absence of contention, this flow is allocated the full line rate (i.e. 10Mb/s). While the flow is in progress a second flow is started by the Blue tenant from S2 to S3 creating congestion at the shared destination. In this scenario the node T will detect a possible congestion and will start ECN marking the affected flows. The IPCP at the destination server will detect the ECN marked flows

and will then compute the new target bit-rates for the congested flows. The new target bit-rates are then send back to the source IPCP using a DTP management message.

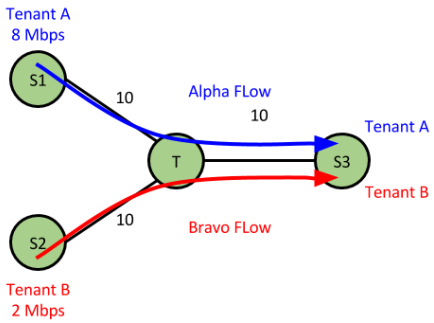

Figure 34. Experiment 2: Network setup

Results will be analyzed in order to ensure that the target end-to-end bitrate is respected and that the rate shaping is stable.

3.4.3.6. Experiment steps

1. Setup a basic full-bisection bandwidth topology with 4 nodes (3 servers and 1 Top of Rack switch (ToR))
2. Deploy two tenant DIF on top of the DC DIF
3. Each Tenant is assigned a minimum bandwidth such that the nominal speed of the links between server and TOR is not exceeded. In this experiment all links are assumed to be 10 Mb/s links and the Red Tenant is assigned 2 Mb/s as minimum guaranteed bandwidth while the Blue Tenant is assigned 8 Mb/s as minimum bandwidth.
4. The Red Tenant begins transmitting a flow between S1 and S3. Since this is the only flow in the network the tenant should be able to exploit the full line rate.
5. The Blue Tenant begins transmitting a flow between S2 and S3.

6. A congestion situation is detected at the node T.
7. Both flows are ECN-marked and the receiver IPCP instructs the transmitter to reduce the rate.
8. Throughput and latency measurements are taken.

3.4.3.7. Result evaluation

The results of this experiment are plotted in Figure 35. As it can be seen when only the Blue tenant is inactive, the full capacity of the path between S1 and S3 is allocated to the Red tenant. However, as soon as the Blue tenant start generating traffic the minimum bandwidths for the two tenants are enforced.

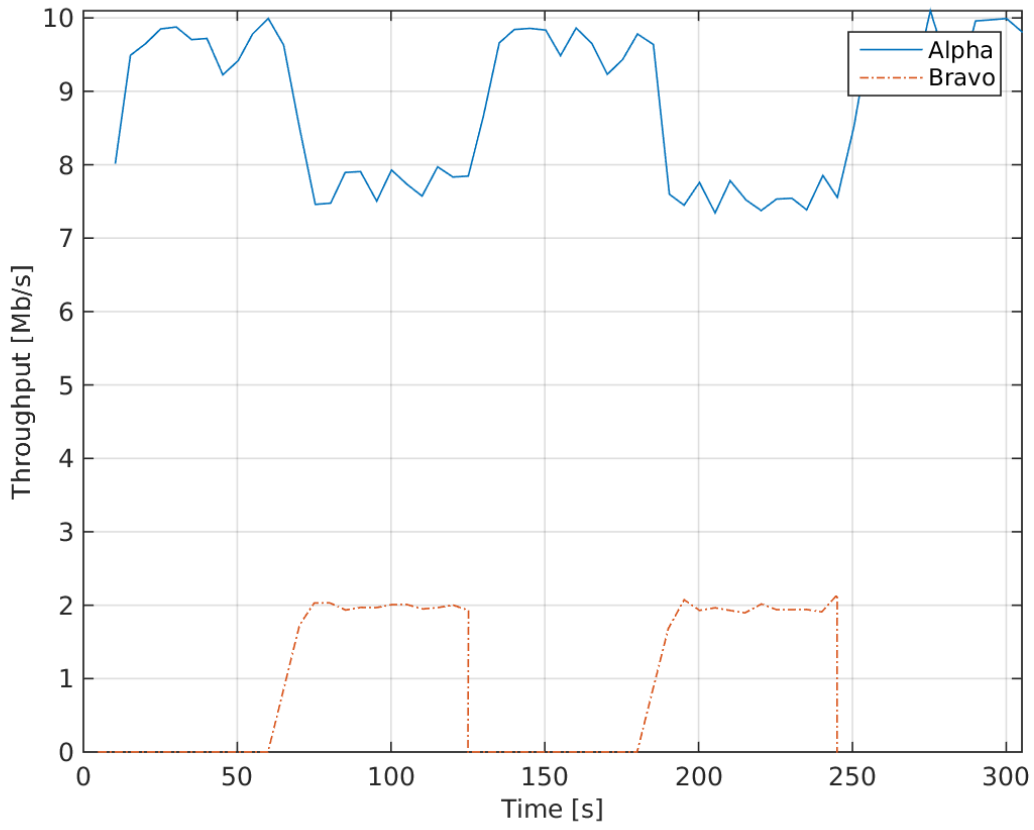

Figure 35. Congestion control policies: Two tenants (Red and Blue). Results are in Mb/s (s)

3.4.3.8. Feedback towards development

The development, debugging and testing of both the rate-based flow control and of the policies for congestion control helped uncover a number of bugs in the RINA implementation. The following bullet

points summarize the main feedback items towards the RINA prototype development (which have already been addressed and released in the most up to date development branch, *pristine-1.3*[5]).

- Various memory leaks, at kernel level, in the RMT default policies module has been discovered and reported to the OpenIRATI team in order to fix them.
- Locking problem which cause the Shim Ethernet module to fail during kernel unloading.
- The unloading failure forced the user to reboot the entire node in order to correctly load again the module.
- Found and reported another problem in the Shim Ethernet module which caused a malfunction in Linux transmission control mechanism called qdisc.
- Found and reported a problem in the user-level IPC Manager which prevented the EFCP to load a custom DTP/DTCP policy.

Open issues currently reported but still not fixed:

- The Shim Ethernet module can't be removed from the kernel once it has loaded and used by an IPCP. This is probably due the kernel workers not being correctly freed.
- Sometimes the IPCP hangs and it's marked as a defunct process, but still persists and it's not de-allocated. The problem has not yet being correctly identified.

Moreover, some design improvements have been identified and will be scheduled for the next iteration of the prototype in version 1.4:

- Stacking multiple DIFs on top of each other leads to both a throughput degradation (in the order of 1 Mb/s per stacked DIF) and in a more unstable throughput when rate-based congestion control is used. We ascribe this behavior to the multiple buffers that must be traverse by the data before reaching the actual wire and to synchronization issues between the various timers.

[5] https://github.com/irati/stack/tree/pristine-1.3

- The use of ethtool to limit the bandwidth of the physical Ethernet interface available in the Virtual Wall is not consistent across the various servers.
- The virtual link feature provided by the Virtual Wall has an unpredictable behavior that can lead to packets being dropped at intermediate nodes.

Due to the limitation described above we implemented a custom link emulation system within the RINA DTP. We plan to remove such code once the problems described above have been tackled.

3.4.3.9. Technical impact towards use cases

The experiments described in this section illustrate two major benefits of the RINA architecture in the data-centre use case. The first one is that by using ECN notification is possible to quickly react to congestion in a data-centre topology while always making optimal use of the available capacity (work-preserving resource allocation).

The second advantage highlighted by this experiment lies in the fact that a complex behaviour such as the one described by the authors of the EyeQ system can be actually implemented using two simple policies (RMT and EFCP) consisting of a few hundreds of LOC as opposed to the several thousands required for the original EyeQ codebase. This results in a better code reuse and maintainability.

It is worth noticing that, although the impact described above refers to the data-centre use case, the policies developed and tested are effectively use case agnostic and the same benefits can also be envisioned for other use cases as well

3.5. Experimenting with SDU Protection Policies on service provider RINA networks

3.5.1. Short Description

The experiments provided in this section measure performance impact of applied SDU protection policy applied at different DIFs on the network traffic.

These experiments analyze the performance impact of SDU Protection policies in service provider RINA networks. The experiments will be set in an environment of a service provider network that leverages the recursion features of the RINA architecture to minimize the exposure of its internal assets outside of its network. As it is shown in Figure 36, since most of the provider DIFs will be internal, the amount of damage that an external attacker can cause is limited, unless the attacker manages to compromise the physical assets of the provider. To secure communication in service provider network SDU protection can be applied at different DIFs. The experiments provided in this section measure performance impact of applied SDU protection policy applied at different DIFs on the network traffic.

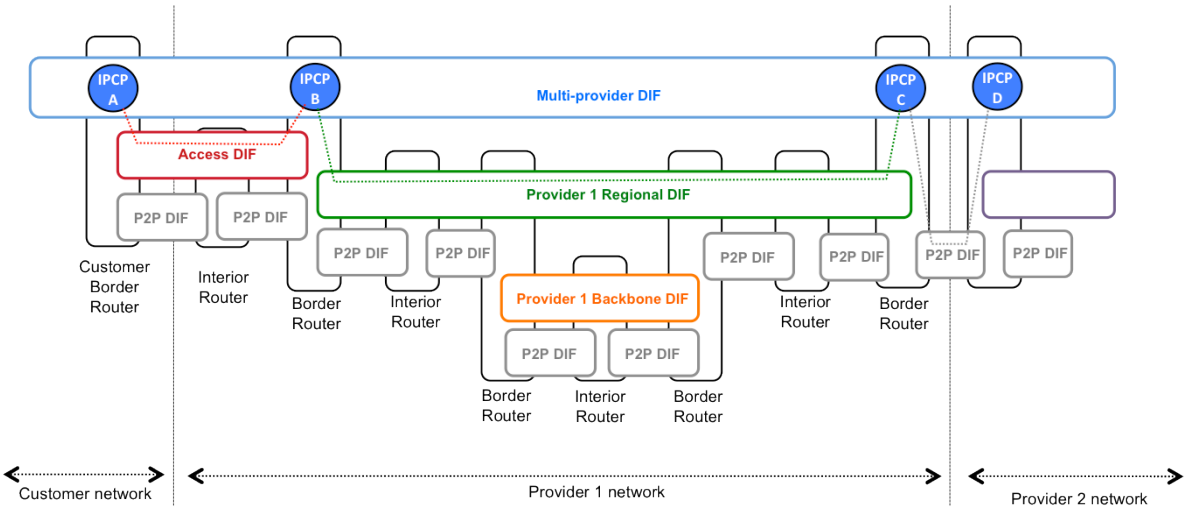

Figure 36. DIF configuration of a network service provider

3.5.2. Experiment goals

This experiment aims to analyze the performance impact of SDU protection applied between peer nodes. The Cryptographic SDU Protection Policy is required to protect the communication between neighbor routers. The results of this evaluation provide information on direct overhead of SDU processing when applying AES and SHA algorithms.

This experiment aims to prove the correct functioning of the SDU protection mechanism and to perform the performance evaluation by measuring SDU protection overhead:

1. To verify correct functionality of the SDU protection implementation

2. To determine performance characteristics of two SDU Protection Policies implemented

Traffic: varying traffic load is assumed to be generated and throughput and latency parameters will be measured. Because of the measured values are strongly dependent on hardware platform capabilities, the results will be evaluated with respect to measured baselines. As a baseline, default SDU protection is considered. This SDU protection policy is the simplest SDU protection policy. It only computes CRC and TTL values. The aim of the experiment is to evaluate overhead of crypto SDU protection policy. Both available SDU protection policies will be analyzed using different traffic load pattern and for crypto SDU protection, different settings will be applied.

3.5.3. Use Cases and requirements addressed

These experiments address the network service provider use case. Performance evaluation of crypto-based SDU protection policy implementation is provided. The performance evaluation determines an overhead incurred when the Crypto-based SDU protection policy is enabled. Because this depends on resources available, the performance evaluation will be given relatively to the baseline defined by the default SDU protection policy. The default SDU protection policy applies only CRC algorithm to ensure that SDU's were not corrupted during transmission. On the other hand Crypto-based SDU protection policy applies cryptographic algorithms for providing confidentiality and integrity of SDUs. The PoC implementation of Crypto-based SDU protection policy is evaluated. This implementation currently only support AES and SHA algorithms.

3.5.4. Metrics and KPIs

1. Comparison of maximum throughput from the source node to the destination node for various SDU protection policies and traffic characteristics
2. Latency statistics from the source node to the destination node for various SDU protection policies

Ideally, the measured values should show that the imposed overhead of crypto-based SDU protection has acceptable overhead.

3.5.5. Experiment configuration

The recursion feature of RINA makes RINA networks inherently more secure that current Internet networks, independently of the security policies used to secure those networks: current ISPs internal routers are in the same layer (the public Internet) as other ISPs internal routers or customer's routers. There are various options where traffic protection can be applied. In the experiments, the performance analysis of SDU protection applied at different DIFs will be studied. All scenarios are derived from the service provider RINA network topology in Figure Figure 36.

All experiments are carried out in virtual environment The environment runs on a host, which is Linux OS with KVM hypervisor. Host station is Intel Core i5-3210M CPU @ 2.50Ghz with four cores and 16GB Memory. All RINA nodes (guests) employ x86 64-bit SMP architecture running Linux Debian, kernel version 3.15.0.

3.5.6. Experiment steps

1. Setup the experiment environment which depends on the case.
2. Apply the configuration files according the case.
3. Generate a set of traffic loads with different parameters.
4. Measure end-to-end throughput and latency for the following SDU policies: 1) basic, 2) Crypto without AES instruction set support and 3) Crypto with AES instruction support enabled.

3.5.7. Results Evaluation

3.5.7.1. SDU Protection Performance Impact over Shim DIF point-to-point connection

This case represents the simplest scenario, where two nodes communicate directly over the common supporting Ethernet Shim DIF as shown in Figure 37. Obtained results will consist of data on throughput and latency between a pair of nodes for various SDU protection policies applied. Clearly, the basic SDU protection is the fastest. Crypto SDU protection profile without AES support in CPU instruction set achieve 60-70% of performance of the basic SDU protection. Crypto SDU protection

accelerated with AES support can achieve around 80% of the performance of the basic SDU protection. Also, while the packet rate remains constant for the basic SDU protection, it decreases in the case of the crypto SDU protection regardless of whether AES support is enabled or disabled.

When comparing the presented results with Internet security protocols, several factors need to be considered. The performance of an IPsec system depends on CPU, RAM, NICs, switches, kernel and configuration. As presented for Libreswan Project, which provides IPSec implementation, the performance of IPSec implementation is between 12% - 53% of unencrypted traffic. This depends on the algorithms used, where AES128 is the fastest because of hardware support.

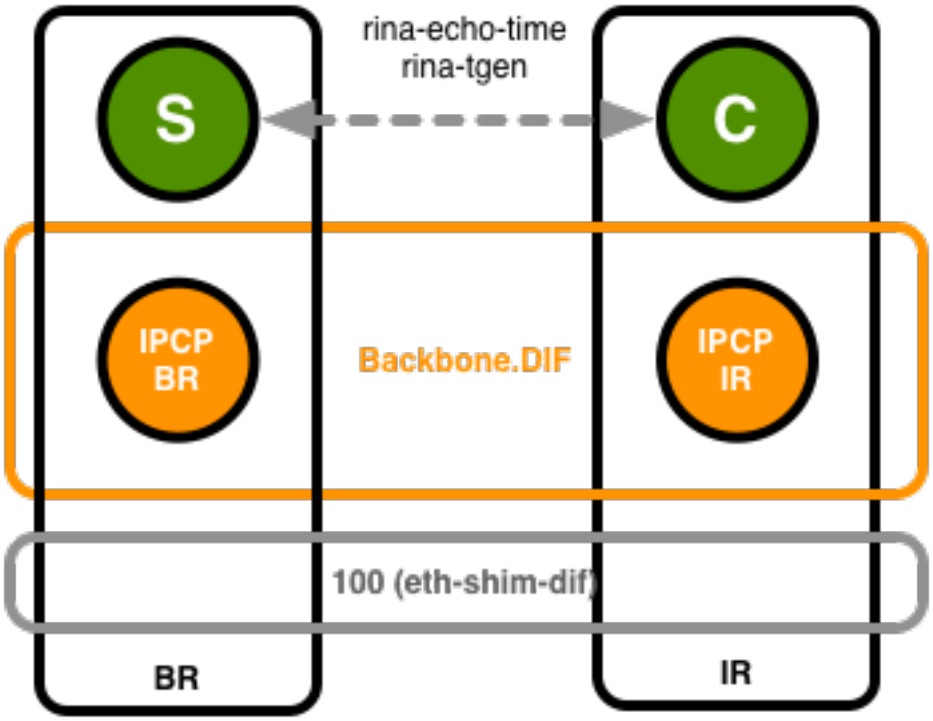

Figure 37. Provider Backbone DIF

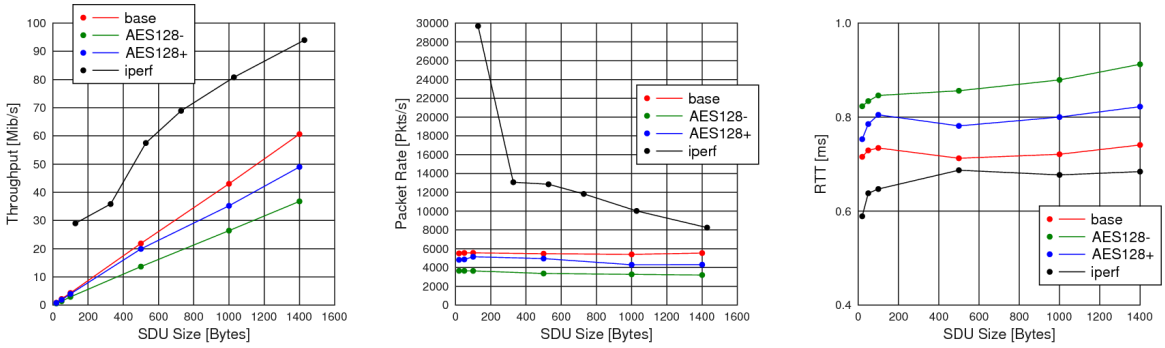

Figure 38. Throughput, Packet Rate and Delay Results

3.5.7.2. SDU Protection Performance Impact with SDU protection applied in Access or Provider DIF

This case consists of three nodes as show in figure Figure 39 and it is a part of the Figure 36 topology. The middle node forwards the traffic between the other two nodes. In this experiment we compare three cases:

- Only Basic SDU Protection is applied.
- Cryptographic SDU protection is applied in Provider DIF.
- Cryptographic SDU protection is applied in Access DIF.

Obtained results consist of data on throughput and latency between a pair of nodes for these three cases. As the performance baseline, the first case is considered. The second case requires that expensive Cryptographic SDU protection is applied for each transmitted application SDU only once. The third case mean that Cryptographic SDU protection is applied to each application SDU twice. Firstly, it is applied when it leaves the Access DIF at source node. Secondly, it is applied at the middle node that forwards the data to the target node.

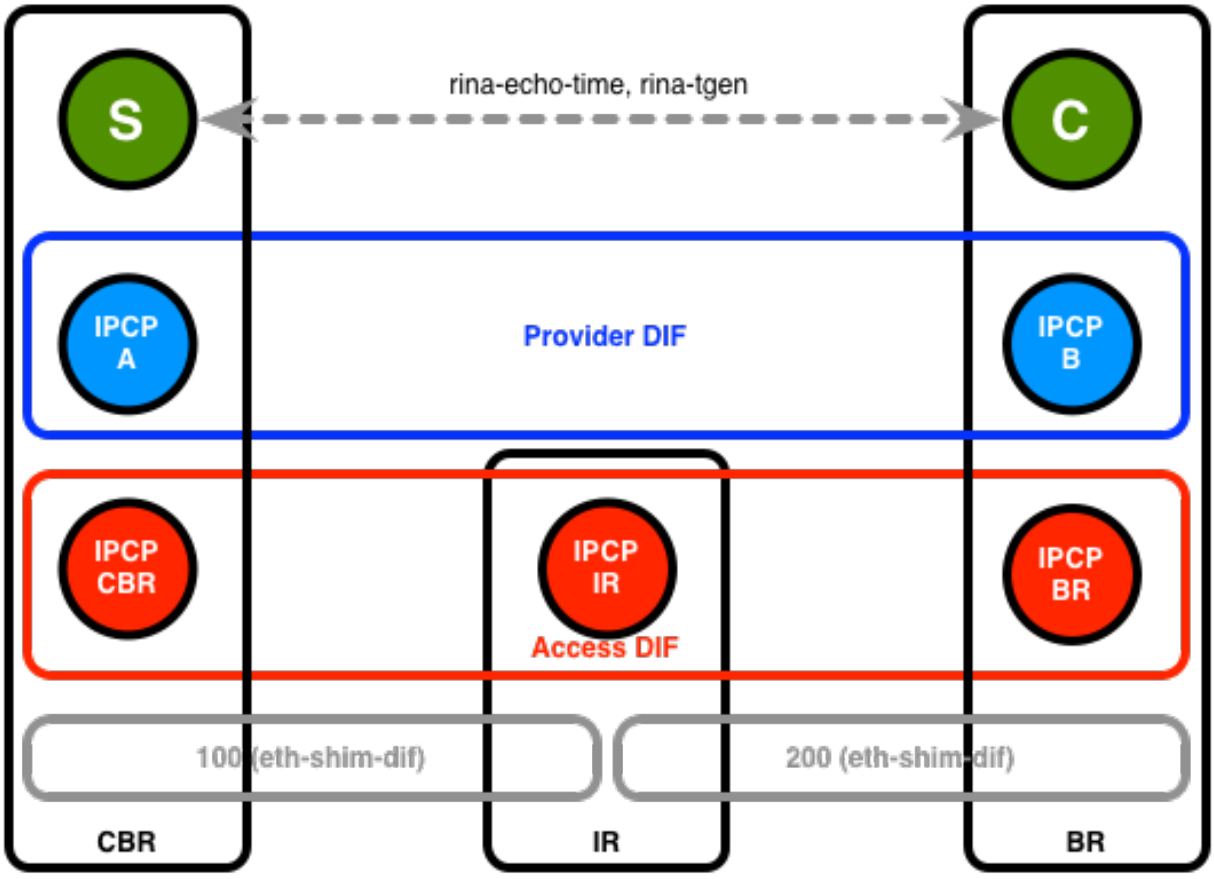

Figure 39. Multi-Provider DIF

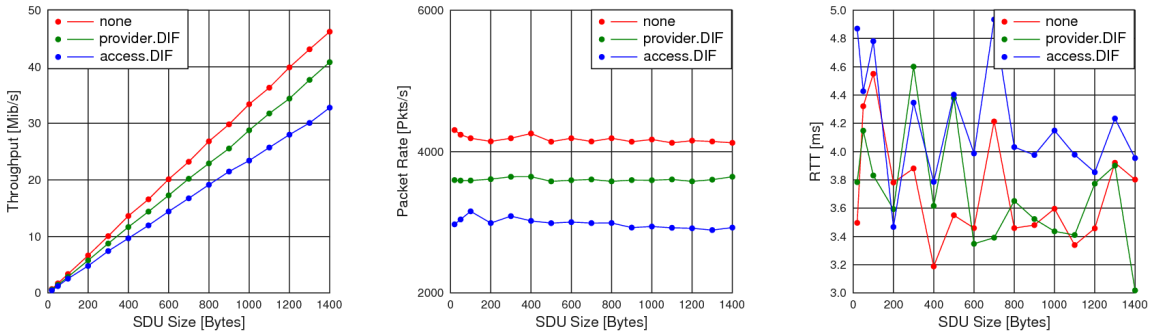

Figure 40. Throughput, Packet Rate and Delay Results

3.5.8. Feedback towards development

Preparing and conducting experiments generated feedback towards development of RINA Stack. The following points were identified:

- We have found a problem with setting environment where the lower DIF is encrypting traffic and this DIF contains forwarding nodes. In this setting the nodes in the upper DIF cannot enroll and often this leads to a crash of the system. This issue is probably in kernel implementation related to SDU protection component.
- Log files can become quite huge when running RINA for some time. It is because of limited possibilities to debug kernel code and thus event logging is used very often. This is not an issue but code implementation should be check carefully if all the log information is necessary at this stage of development.
- When something is going wrong, RINA offers brief error description. This information is enough for RINA stack developer. For an ordinary user it should contain additional information on how to identify and possibly solve the problem.
- Because RINA stack is intensively developing, there is not any compact reference on how to write configuration files. To provide configuration one has to follow the examples and try what works and what does not.
- To evaluate the performance of the implementation one may try to use iperf to get reference values of TCP/IP stack. In RINA tools like rina-tgen and rina-echo-time can be used to get information on throughput and latency. While these tools are supposed to achieve the same, there is not evidence that iperf and rina-tgen results are comparable.

- When using rine-tgen tool, it seems that -d parameter (registering in selected DIF) used at the client side does not work as expected. When there are more than a one DIF over a Shim DIF, then rina-tgen client cannot connect to the server. For rina-echo-time this works. There is a workaround that requires to put registration of rina-tgen application to configuration files.

3.5.9. Technical impact towards use case

Although the experiments were conducted in simplified environment due to limitation of the host where virtual machines were run, the experiments itself and obtained results gave the following information about RINA architecture and PoC implementation of SDU protection:

- Proof of Concept implementation of the Cryptographic SDU protection policy reliably protects and verifies SDU. By capturing the traffic it can be seen that the whole SDU is encrypted by the AES algorithm using the symmetric key negotiated during authentication phase or predefined in IPCP configuration files. From the standard techniques the PoC implementation lacks protection against replay attack. The success of the replay attack depends on the parameters of the flow. If the flow supports ordered delivery then disordered packets will be automatically discarded. When ordered delivery is not used then user of the flow needs to resolve the cases when the duplicate SDU is delivered.
- As expected, applying more complex SDU protection policy incorporates some overhead. Cryptographic SDU Protection policy can achieve up to 80% performance of Basic SUD Protection policy considering 100Mbps point-to-point links. It may be possible that for faster connection the cryptographic SDU Protection policy can be slower. On the other hand, the PoC implementation was evaluated and this implementation does not contain any performance optimisation. The implementation relies on cryptographic library available in Linux kernel and thus its performance strongly depends on the performance of this kernel library.
- Since each layer is responsible for protecting its own communication it may be possible that application communication is encrypted multiple times. The result of this is that throughput is significantly smaller in comparison to the case when SDU protection is only applied in the DIF supporting the application. The advantage of RINA with respect

to application of security policies is that SDU protection policy can be configured on per flow basis and it is completely transparent to the applications. This is quite different approach to the current Internet, where securing the data communication means to re-implement the application protocol to use secure layer library or to deploy secured virtual private network.

3.6. Deploy IRATI kernel on SlapOS infrastructure

3.6.1. Short Description

This experiment validates the automation of IRATI kernel build through ansible profile and deployment in SlapOS host.

3.6.2. Experiment General Information

- **Experiment owner (partner) and individual contact**: Nexedi (Aaron Chen - aaron.chen@nexedi.com[6])
- **System**: RINA stack, IRATI, SlapOS

3.6.3. Experiment Goals

The time consuming to deploy an IRATI kernel manually can be high, because if we have machines not well configured, it can take more time if you start to get issues when installing packages. Also, different machines can generate different results, for example, different package versions or installed differently.

The goal of this experiment is automate IRATI kernel deployment in machines with clean setup hosted by SlapOS.

3.6.4. Use Cases and Requirements Addressed

This experiment addresses the Distributed Cloud use case.

3.6.5. Metrics and KPIs

N/A

[6] mailto:aaron.chen@nexedi.com

3.6.6. Experiment steps

1. **Install a SlapOS host**.
2. **Follow IRATI tutorials to enable host with RINA**.
3. **Automate tutorial through ansible script to check repeatability**. With the ansible script, will be possible request KVMs in SlapOS and reproduce IRATI tutorials to install the IRATI kernel in large scale.

3.6.7. Experiment Configuration

We reproduced this experiment in a KVM with Debian 7 and all steps to install the KVM in SlapOS can be found here: developer-Allocate.SlapOS.KVM.Instance[7].

To setup a clean environment is recommended to install only SSH Server in the KVM. Desktop environment or any other service are not required.

With the KVM deployed and few commands, the IRATI is installed. To install, you just need download and run the ansible script:

```
su
apt-get update && apt-get install ansible
wget https://lab.nexedi.cn/AaronChen/RINA-Quick/blob/master/rina.yml
ansible-playbook -i "localhost," -c local rina.yml
```

When the ansible script is finished, the new kernel will be compiled but will not be set as the default boot kernel. We are using the grub bootloader on a fresh default Debian install. So this can be done by editing the grub config file located at:

```
/etc/default/grub
```

Editing GRUB_DEFAULT to "1>2"

Save the file and then run update-grub to put the change into effect:

```
update-grub
```

[7] http://www.osoe-project.org/developer-Allocate.SlapOS.KVM.Instance

After reboot the system, is possible check if the correct kernel is loaded by attempting to load a rina module:

```
modprobe normal-ipcp
```

If this completes without any warnings, the RINA kernel is now being used.

3.6.8. Result evaluation

Once completed, you now have a KVM in SlapOS using RINA kernel, ready to use and develop.

With the automation of all process, you gain in many aspects:

- Avoid repeat the same process many times;
- Avoid human mistakes;
- Reproduce the same result several times;
- Speed up of install IRATI kernel in large scale;

The average time spent to install IRATI kernel is small when you have to install in a single machine. But, is starts to be a problem when you have to install in two or more places. The chance to mistakes can increase drastically.

3.6.9. Feedback towards Development

RINA in its current state is not ready for mainstream use. Its stability issues need to be addressed and core features still need to be worked on. Both network shims currently in use are both limited, the Ethernet shim in its current form only supports a single DIF while the IP shim requires manual mapping for all addresses. Manual enrollment of DIFs also needs to be addressed along with IPC address allocation.

3.7. Support an alternate protocol on a re6st/babel virtual Ethernet mesh

3.7.1. Short Description

SlapOS is currently deployed using re6st networking. Re6st is a worldwide random mesh of Ethernet tunnels on top of which we route IPv6. Routing tables are computed by babel daemon to minimize latency.

This mesh infrastructure solves the lacked of reliability of Internet nowadays caused by government interference and by traffic optimization by telecommunication companies. The same infrastructure could be used to deploy RINA worldwide based on IRATI's Ethernet shim, as long as re6st can support multiple network protocols.

3.7.2. Experiment Goals

The goal of this trial is to find out how easy it is to extend re6st with another protocol and build a hybrid infrastructure which can support multiple protocols and at the same time use the routing metrics computing for one protocol to feed the routing tables of another protocol. In this trial, we will extend re6st with IPv4 and map existing IPv6 routing tables to IPv4 routing tables. The same approach can later be applied to IRATI based on the results of other trials related to IRATI's Ethernet shim.

3.7.3. Use cases and requirements addressed

Distributed Cloud Use Case https://wiki.ict-pristine.eu/Distributed-Cloud-use-case

3.7.4. Metrics and KPIs

N/A

3.7.5. Experiment Steps

1. Extend re6st registry with IPv4 address registry.
2. Deploy IPv4 addresses on every node.
3. Configure babel to map IPv6 routes to IPv4 routes.

3.7.6. Experiment Configuration

re6stnet[8] is distributed as a Python egg, and is also packaged for DEB & RPM based distributions:

We reproduced this experiment in a machine with Debian 7 installed.

[8] https://pypi.python.org/pypi/re6stnet/0.431

To get re6stnet installed you can follow this tutorial vifib-Install.IPv6.FAQ[9].

The patch to support IPv4 can be found here: re6stnet.git[10]

After installed you can check if re6stnet is installed:

```
$ dpkg -l | grep re6stnet
ii  re6stnet         0-376.g254dd5c     all    resilient, scalable, IPv6
network application
```

3.7.7. Result Evaluation

Results from this experiment:

1. Support of IPv4 in re6st
2. Package to automate re6st installation
   - In order to build DEB snapshot package whose version is derived from current Git revision, the debian/changelog file must be generated automatically, that's why you can't use dpkg-buildpackage directly: run debian/rules instead. RPM does not have this limitation: do rpmbuild -bb re6stnet.spec` as usual.
3. Extension of babel to monitor / update route metrics in real time
   - It was implemented to destroy tunnels gracefully, otherwise there are often routes broken for a minute. In addiction, it also allowed us to implement parts in a more cleaner way, and interacting with babel will permit more features in the future.
4. Short plan to apply babel route metrics of re6st to IRATI global deployment
   - The same approach used to extend re6st with IPv4 and map existing IPv6 routing tables to IPv4 routing tables, can be applied to IRATI based on the results of other trials related to IRATI's Ethernet shim.

In this experiment we tested the idea of using routes for one protocol (IPv6) to replace routes in another protocol (IPv4) and we plan to do same in RINA.

[9] http://www.nexedi.com/vifib-Install.IPv6.FAQ
[10] http://git.erp5.org/gitweb/re6stnet.git/commitdiff/2fb63515d602b77c684c30dfc9b8e680ae427bbc?js=1

This is an interesting goal if we look at the history of IPv6, the first providers were providing IPv6 on top of IPv4 rather than native and even after 20 years it is still the same. We plan to do a similar approach, in a way that will be transparent to RINA applications.

3.7.8. Feedback towards Development

"ipv4" is an option to set on the registry (re6st-registry.conf) instead of on re6stnet.conf, because every node has to be a router for any packet, ipv4 is a network option, not a node one.

3.8. Manager and management agent

This section outlines the current results for a set of experiments used to validate both the DMS manager and the Managed Agent Daemon (MA). This is done by using a default policy set for performing typical management functions by the DMS. Where not otherwise stated, the experiments have been performed with pristine v1.2[11] branch.

3.8.1. Experiment 1: Bootstrap of an IPCP with local flows

3.8.1.1. Short Description

In this experiment one single node is used. There is only one IPCP which is using the local flow functionality that allows it to create, internally, a reliable connection, simulating in this way a connection between two IPCPs in the same DIF.

3.8.1.2. Experiment goals

The following management strategies need to be created and deployed, which correspond to generic use-cases for the DMS system (DMS Default policy set).

3.8.1.3. Use Cases and requirements addressed

Use of the DMS default policy set is related to all three use cases: 1. Internet Service Provider Use Case 2. Data Center Use Case 3. Distributed Cloud use case.

[11] https://github.com/IRATI/stack/tree/v1.2.0

1. Internet Service Provider Use Case: **SC-DIF-*5***, **SC-DAF-*5*,VNF-DIF-*5*,VNF-DAF-*5***
2. Data Center Use Case: **NM-DC-6**
3. Distributed Cloud use case: **NM-DISCLOUD-6**, **NM-DISCLOUD-7**, **NM-DISCLOUD-8**

3.8.1.4. Experiment configuration

The experiment configuration is outlined in the figure below.

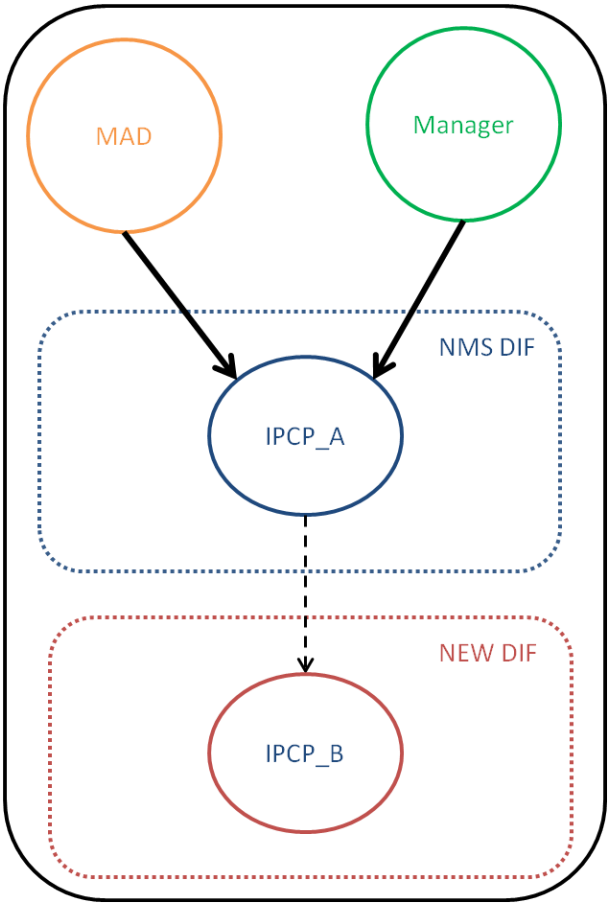

Figure 41. Manager and MA, Experiment 1 configuration

As for the DIF configuration, the NMS DIF is the DIF used to communicate the manager with the MAD. We have used a reliable connection to assure that the communication successful. Using a non reliable connection has no sense unless some kind of reliability techniques are implemented in both, the MAD and the Manager. Otherwise, ordering of information requests or the responses may be lost. The configuration used is given in Appendix A.

3.8.1.5. Experiment steps

The following steps are required to run the experiment.

Step 1: When the systems bootstraps, it only creates one normal IPCP called IPCP_A (within one normal DIF called NMS DIF) using the scripting of the pristine stack (an automated process that loads configurations and creates predefined IPCPs). It also starts the Management Agent daemon (MAD) as an application and registers it to the created IPCP. This is done also with the scripting engine that interprets a new section in the configuration file (see Section A.2)

```
nodeA$ ipcm -c ../etc/ipcmanager.conf -a "console,scripting,mad"
```

Step 2: Once registered, the MAD starts to look for the Manager, querying the Directory Forwarding Table (DFT) periodically.

Step 3: Start the Manager, by executing the following application.

```
nodeA$ manager
```

Once the Manager is registered within the NMS DIF, the MAD establishes a connection with the manager following the Common Application Connection Establishment Phase (CACEP) procedures.

Step 4: The Manager sends a CREATE CDAP message targeting the IPCProcess object of the RIB. Depending on the information provided in the CDAP message, the MAD will also assign the new IPCP (IPCP_B) to a DIF with the required parameters.

Step 5: The Manager now sends a READ CDAP message to the MAD targeting the RIBDaemon object that has been created under the new IPC process. The MAD responds with the rib information that can be checked by the Manager to assure that the created IPC process has been bootstrapped as expected.

The following figure summaries the communication exchange between the Manager and Management Agent.

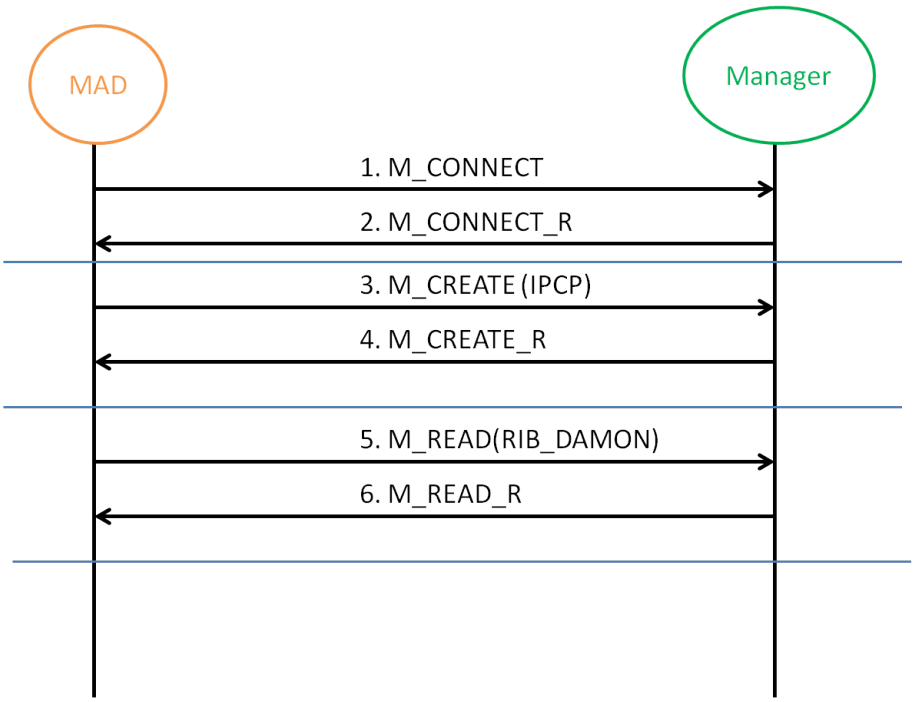

Figure 42. Manager and MA, Experiment 1 message flow

3.8.1.6. Result evaluation

The successful integration and performance of this experiment, confirms the bootstrapping process of the management agent daemon (MAD) works as expected. To validate the management agent daemon we have developed a C++ manager only for testing purposes. This experiment has been re-run with the DMS manager has replicated these results.

1. Creation of a DIF implies the requires creation of a new IPCP and assign it to a DIF.
2. Monitoring is achieved through the Manager requesting the state of an IPCP. This is performed by sending a READ CDAP message targeting the RIBDaemon object of the IPCP.

3.8.1.7. Feed back towards development

While we have proved the successfully result of the experiment, it has been done using a limited C++ manager designed only for testing purposes. Due to various technical reasons that have delayed the integration, specially the JAVA bindings using SWIG.

3.8.1.8. Technical impact towards use-cases

The successful performance of this experiment, confirms that the existing use-cases are valid.

3.8.2. Experiment 2: Bootstrap of an IPCP with local flows (DMS Manager version)

3.8.2.1. Short Description

In this experiment one single node is used. There is only one IPCP which is using the local flow functionality that allows it to create, internally, a reliable connection, simulating in this way a connection between two IPCPs in the same DIF.

3.8.2.2. Experiment goals

The following management strategies need to be created and deployed, which correspond to generic use-cases for the DMS system.

3.8.2.3. Use Cases and requirements addressed

Use of the DMS default policy set is related to all three use cases: 1. Internet Service Provider Use Case 2. Data Center Use Case 3. Distributed Cloud use case.

1. Internet Service Provider Use Case: **SC-DIF-5**, **SC-DAF-5**,**VNF-DIF-5**,**VNF-DAF-5**
2. Data Center Use Case: **NM-DC-6**
3. Distributed Cloud use case: **NM-DISCLOUD-6**, **NM-DISCLOUD-7**, **NM-DISCLOUD-8**

3.8.2.4. Experiment configuration

The experiment configuration is exactly the same as for experiment 1. (See Section 3.8.1.4)

3.8.2.5. Experiment steps

The following steps are required to run the experiment.

Step 1: When the systems bootstraps, it only creates one normal IPCP called IPCP_A (within one normal DIF called NMS DIF) and starts the Management Agent daemon (MAD) as an application and registers it to the created IPCP.

```
nodeA$ ipcm -c ../etc/ipcmanager.conf -a "console,scripting,mad"
```

Step 2: Once registered, the MAD starts to look for the Manager, querying the Directory Forwarding Table (DFT) periodically.

Step 3: Start the DMS Manager, by executing the following application.

```
nodeA$ dms/bin/sh/dms.sh
```

Once the Manager is registered within the NMS DIF, the MAD establishes a connection with the manager following the Common Application Connection Establishment Phase (CACEP) procedures.

Step 4: The Manager sends a CREATE CDAP message targeting the IPCProcess object of the RIB. Depending on the information provided in the CDAP message, the MAD will also assign the new IPCP (IPCP_B) to a DIF with the required parameters.

```
dms-shell> scrun setup ❶
dms-shell> sendTrigger dialect:ts,id:tcreatenormalipcp ❷
```

❶ Setup of the DMS Manager strategies
❷ Triggers the creation of a normal IPC Process

Step 5: The Manager now sends a READ CDAP message to the MAD targeting the RIBDaemon object that has been created under the new IPC process. The MAD responds with the rib information that can be checked by the Manager to assure that the created IPC process has been bootstrapped as expected.

3.8.2.6. Result evaluation

The successful integration and performance of this experiment, confirms the bootstrapping process of the MAD and DMS Manager works as

expected. This verifies the operations outlined in experiment 1, with the Java bindings and DMS manager.

The inclusion of a strategy based management system allows more complex scenarios and deployments to be created by adjusting the strategies used in the DMS Manager.

3.8.2.7. Feed back towards development

The creation of the normal IPC Process in the agent proves the successfully result of the experiment. This has been done using the *pristine-1.3*[12] branch and demonstrates the improvements made on the JAVA bindings using SWIG.

3.8.2.8. Technical impact towards use-cases

The successful performance of this experiment, confirms that the existing use-cases are valid.

3.8.3. Experiment 3: Bootstrap and configuration of a new DIF

3.8.3.1. Short Description

In this experiment three nodes are used. There is only one IPCP which is using the local flow functionality that allows it to create, internally, a reliable connection, simulating in this way a connection between two IPCPs in the same DIF.

3.8.3.2. Experiment goals

The following management strategies need to be created and deployed, which correspond to generic use-cases for the DMS system.

3.8.3.3. Use Cases and requirements addressed

Use of the DMS default policy set is related to all three use cases: 1. Internet Service Provider Use Case 2. Data Center Use Case 3. Distributed Cloud use case.

1. Internet Service Provider Use Case: **SC-DIF-5**, **SC-DAF-5**,**VNF-DIF-5**,**VNF-DAF-5**

[12] https://github.com/irati/stack/tree/pristine-1.3

2. Data Center Use Case: **NM-DC-6**
3. Distributed Cloud use case: **NM-DISCLOUD-6**, **NM-DISCLOUD-7**, **NM-DISCLOUD-8**

3.8.3.4. Experiment configuration

This experiment is composed by 3 nodes, shown in the following figure.

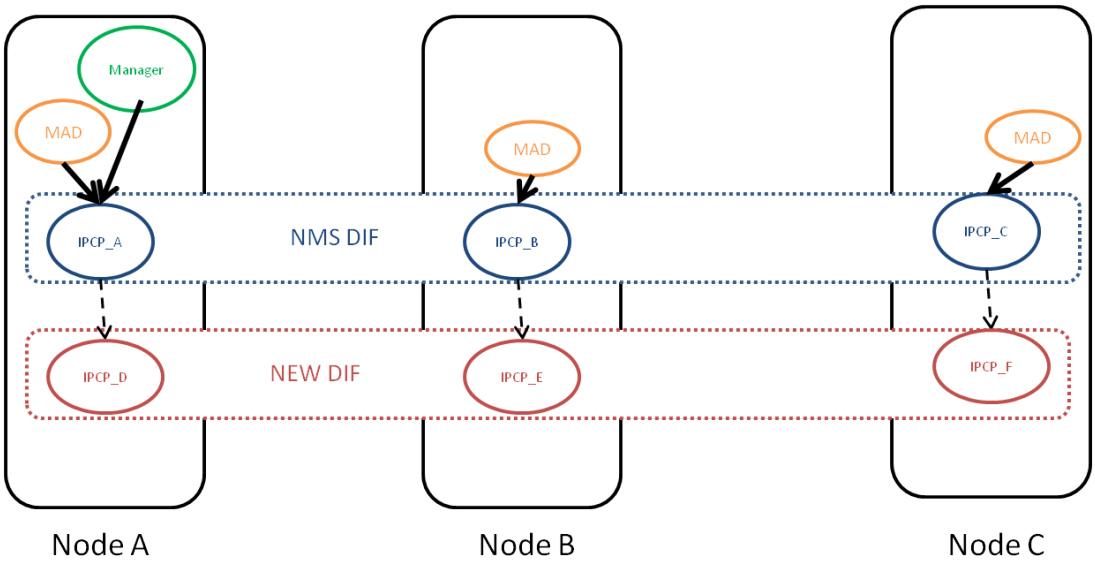

Figure 43. Manager and MA, three node configuration

3.8.3.5. Experiment steps

The steps needed to recreate this scenario are as follows:

1. System bootstrap in the three nodes and create a NMS DIF connecting the three nodes.
2. The Manager registers to the NMS DIF, causing all the MADs to start establishing a connection with it.
3. The Manager does in the node A the same procedure than in the Experiment 1.
4. The Manager asks MAD in node B to create an IPCP_E and to start the enrollment to the NEW DIF through the IPCP_D.
5. The Manager repeats the procedure for node C.
6. Finally it asks for the RIB information to the three nodes.

More detailed instructions are provided below.

Step 1: All nodes are started with the management agent enabled.

```
nodeA$ ipcm -c ../etc/ipc-manager.conf-nodea -a "console,scripting,mad" ❶
nodeB$ ipcm -c ../etc/ipc-manager.conf-nodeb -a "console,scripting,mad" ❷
nodeC$ ipcm -c ../etc/ipc-manager.conf-nodec -a "console,scripting,mad" ❸
```

❶ Start the IPC Manager on node A
❷ Start the IPC Manager on node B
❸ Start the IPC Manager on node C

Step 2: Once registered, the three MADs start to look for the Manager, querying the Directory Forwarding Table (DFT) periodically.

Step 3: Start the Manager, by executing the following application in one of the nodes.

```
nodeA$ manager
```

Once the Manager is registered with the NMS DIF (through the corresponding IPCP of the system), the MADs establish a connection following the Common Application Connection Establishment Phase (CACEP) procedures.

Step 4: The Manager sends a CREATE CDAP message to the MAD targeting the IPCProcess object of the RIB of node 1. The MAD will also assign the new IPCP (IPCP_B) to a DIF with the required parameters.

Step 5: The Manager now sends a READ CDAP message to the MAD targeting the RIBDaemon object that has been created under the new IPC process. The MAD responds with the RIB information that can be checked by the Manager to assure that the created IPC process has been bootstrapped as expected.

Step 6: The Manager sends a CREATE CDAP message to the MAD targeting the IPCProcess object of the RIB of node 2. In this case, no DIF information is provided, but the created IPCP in the node 1 is given. With this information the new IPCP of node 2 will join the created DIF in node 1 by enrolling to the IPCP of the node 1.

Step 7: The Manager repeats the step 5 for the node 2 and repeats the same procedure for node 3.

3.8.3.6. Result evaluation

The experiment shows that the MA and DMS Manager are working as expected for the requirements of the first prototype. The functionalities tested and validated through this experiment have a direct impact on the three uses cases. All use-cases require that a new DIF can be instantiated and provisioned on an arbitrary number of nodes.

1. Creation of a DIF implies the requires creation of a new IPCP and assign it to a DIF.
2. Monitoring is achieved through the Manager requesting the state of a an object, in this case the RIBDaemon object state.

The inclusion of an agent, which can execute orders remotely is a major advance in the stack and also in the adoption of RINA.

3.8.3.7. Feedback towards development

The MA daemon is validated using an ad-hoc limited manager (as per experiment 1).

3.8.3.8. Technical impact towards use-cases

The successful performance of this experiment confirms that the existing use-cases are valid and there is no perceived impact on the use-cases.

4. Conclusions and Impact

4.1. Technical impact towards use cases

In spite of the simplicity of the experiment scenarios considered, the experiments carried out in this cycle have already illustrated several benefits of the RINA architecture.

Several of the RINA benefits demonstrated during this first cycle of experiments effectively apply to all three use cases. For example, the Loop Free Alternate experiments can be always used by the network administrator to increase IPC service availability. Similarly, explicit Congestion Notification and vertical pushback between layers enable a faster and more efficient response to congestion than the prevalent end-to-end implicit feedback approach used in the Internet. ECN allows congestion effects to be confined closer to where congestion happens instead of affecting the whole network: each DIF manages the congestion in its own resources. This allows network providers to utilize network resources more efficiently and to run networks at higher utilization.

RINA allows each DIF (and thus each scope) to use the congestion management solution that is best suited to its operational environment. There is not a single congestion management solution that will be effective in all networking environments, therefore being able to deploy a variety of solutions in different parts of the network using the same architectural framework is a great advantage over the state of the art. This feature will allow for a customized and higher-efficient response to congestion which, combined with the first feature, will lead to networks that adapt better to congestion and can therefore be safely run at a higher utilization.

Moreover, since flows are aggregated at lower layer DIFs, as also reported by [D32], the use cases can gain a lot from this property by lowering the number of competing flows significantly. Here, we can see that, for example, in the 4 parallel flows scenario, all the flows were aggregated into one flow when they were competing for the bandwidth in the bottleneck link. Our next step is to examine this feature in further details on larger network topologies.

4.2. Feedback towards development

4.2.1. Improvements already implemented and released

The development, debugging and testing of policies associated to the first cycle of experiments uncovered a number of bugs in the RINA implementation as well as some changes required in the SDK. The following bullet points summarize the feedback towards the RINA prototype development (which have already been addressed and released in the most up to date development branch, *pristine-1.3*[13]).

- A number of DTCP policy hooks have been updated to accept an entire PCI struct as an input parameter instead of only a sequence number, implemented by PR 682[14]. This allows for greater flexibility in the related policy implementations, since more information about the PDU are known.
- One of the functions of the RMT policy set (*rmt-q-monitor-tx-policy*, which was called when a PDU was enqueued or dequeued by the RMT in an output port for transmission) has been splitted into two separate functions: *rmt_q_monitor_policy_tx_enq* (called when the PDU is enqueued) and *rmt_q_monitor_policy_tx_deq* (called when the PDU is dequeued). This eases or even enables the work of the policy implementations when calculating different parameters of the n1_port queues. The fixes have been implemented by PR 687[15].
- DTCP policy sets can set functions that are always used by the main code to NULL (such as the RTT estimator policy), opening up the opportunity to cause kernel bugs. The proper solution is for each component to define a number of mandatory functions that have to be implemented by all policy sets, and for the SDK to enforce this upon loading the plugin containing the policy set functions. This feature will be implemented in the second version of the SDK; meanwhile a temporary solution to avoid kernel bugs has been introduced by PR 715[16].
- Several locking problems were found and fixed in the RMT and in the Shim Ethernet components in kernel space.

[13] https://github.com/irati/stack/tree/pristine-1.3
[14] https://github.com/IRATI/stack/pull/682
[15] https://github.com/IRATI/stack/pull/687
[16] https://github.com/IRATI/stack/pull/715

- Various memory leaks, at kernel level, in the RMT default policies module has been discovered and reported and fixed.
- The unloading failure which forced users to reboot the entire node in order to correctly load again the module has been debugged and fixed.
- We found, reported, and fixed another problem in the Shim Ethernet module which caused a malfunction in the Linux transmission control mechanism called qdisc.
- Found and reported a problem in the user-level IPC Manager which prevented the EFCP to load a custom DTP/DTCP policy.

4.2.2. Open issues

Open issues currently reported but still not fixed:

- The Shim Ethernet module can't be removed from the kernel once it has loaded and used by an IPCP. This is probably due the kernel workers not being correctly freed.
- Sometimes the IPCP hangs and it's marked as a defunct process, but still persists and it's not de-allocated. The problem has not yet being correctly identified.
- Despite the good implementation of the SDK, it is worth commenting the few sources of documentation still available for the development. This is a known fact for the developer community that requires more attention, since it is considered a potential issue as it may discourage newcomers willing to contribute to the stack implementation.

4.2.3. Design improvements

Some design improvements have been identified and will be scheduled for the next iteration of the prototype in version 1.4:

- The API of the RMT and the RMT policy set need to be refactored targeting a better integration.
- A new method for retrieving operation data is required in order to avoid logging to a file on every PDU operation since this highly decreases the prototype's performance and the veracity of the results. Using sysfs/

procfs can be an option as it was already used in the case of the RINA TCP-RED policy-set.

- Currently we can only use flows without flow control, since the EFCP component seems to stop working when too many control PDUs are lost. We have shared this information with WP2, where it is currently being debugged.
- Stacking multiple DIFs on top of each other leads to both a throughput degradation (in the order of 1 Mb/s per stacked DIF) and in a more unstable throughput when rate-based congestion control is used. We ascribe this behavior to the multiple buffers that must be traverse by the data before reaching the actual wire and to synchronization issues between the various timers. Due to these limitations some experiments implemented a custom link emulation system within the RINA DTP implementation. We plan to remove such code once the problems described above have been tackled.
- Non-blocking I/O seems to be a better alternative when drops may occur in the network, since it decouples the applications from the quirks of the network stack.
- RINA in it's current state is not ready for mainstream use due to stability issues which will be addressed during the second development cycle and core features which need to be implemented.
- The currently available shims suffer of several limitations. The Ethernet shim in its current form only supports a single DIF while the IP shim requires manual mapping for all addresses.
- Manual enrolment of DIFs also needs to be addressed along with IPC address allocation.
- Proof of Concept implementation of the Cryptographic SDU protection policy reliably protects and verifies SDU. By capturing the traffic it can be seen that the whole SDU is encrypted by the AES algorithm using the symmetric key negotiated during authentication phase or predefined in IPCP configuration files. From the standard techniques the PoC implementation lacks protection against replay attack. The success of the replay attack depends on the parameters of the flow. If the flow supports ordered delivery then disordered packets will be automatically discarded. When ordered delivery is not used then user of the flow needs to resolve the cases when the duplicate SDU is delivered.

- As expected, applying more complex SDU protection policy incorporates some overhead. Cryptographic SDU Protection policy can achieve up to 80% performance of Basic SUD Protection policy considering 100Mbps point-to-point links. It may be possible that for faster connection the cryptographic SDU Protection policy can be slower. On the other hand, the PoC implementation was evaluated and this implementation does not contain any performance optimisation. The implementation relies on cryptographic library available in Linux kernel and thus its performance strongly depends on the performance of this kernel library.
- Deployment of a strong SDU protection policy should be at the highest possible DIF with respect to the application context. This is due to the fact the when SDU protection is used in lower DIFs it may not protect the application communication. Also as shown in another experiment, when the protection is applied in lower DIFs the application throughput is significantly smaller in comparison to the case when SDU protection is applied in the DIF supporting the application. The advantage of RINA with respect to application of security policies is that SDU protection policy can be configured on per flow basis and it is completely transparent to the applications. This is quite different approach to the current Internet, where securing the data communication means to reimplement the application protocol to use secure layer library or to deploy secured virtual private network.

4.2.4. Feedback for testbed owners

Finally, this first cycle of experiments provided also significant feedback for testbed owners, in particular:

- The virtual link feature provided by the Virtual Wall has an unpredictable behaviour that can lead to packets being dropped at intermediate nodes.
- The use of ethtool to limit the bandwidth of the physical Ethernet interface available in the Virtual Wall is not consistent across the various servers.
- The qdisc, which is used in the shim DIF over Ethernet to prevent the rate of PDUs that are sent to exceed the line rate of the NIC, was only attached to one transmission queue, which works on a VM, but not on

real hardware in the Virtual Wall. This resulted in the shim DIF for Ethernet blocking its ports continuously when a packet was dropped.

4.3. Expected Business Impact

4.3.1. Distributed Cloud Use Case

4.3.1.1. Stakeholder Overview

VIFIB is a distributed cloud system offering resilient computing services, maintaining 3 clones and backups on 35 different sites to achieve high levels of resiliency. In order to maximize the resiliency, most of VIFIB servers are hosted in homes and offices with broadband Internet access. The stakeholders of the distributed cloud use case are:

- **Distributed cloud Manager**. Manages the infrastructure of the distributed cloud, providing and maintaining the systems and software required to provide the resilient computing services to users. VIFIB is the distributed cloud Manager.
- **Home/office providers**. Owners of homes or offices that host one or more VIFIB servers. VIFIB subsidizes their IPv6 Internet connectivity.
- **Resilient computing service users (application providers)**. The users that deploy their applications on VIFIB's distributed infrastructure. Application providers are the tenants of the distributed cloud.
- **Users of applications deployed in the distributed cloud**. These stakeholders use the services provided by the applications deployed in the distributed cloud. It may be the tenants themselves if the software is for auto-consumption (for example a company using VIFIB's infrastructure as an extension of their private cloud) or third parties (for example companies using a hosted ERP software suite).

4.3.1.2. Business Impact Overview

All the stakeholders could be affected by RINA adoption to some degree with the exception of the home/office providers, since they do not have any visibility into the software that the VIFIB nodes are running internally, nor they use any of the services provided by VIFIB. VIFIB, as the distributed cloud Manager, would be the stakeholder more affected by RINA adoption.

VIFIB would have to deploy the software implementing RINA capabilities in all of their servers and gateways, upgrade its Management System and train their operations personnel in managing RINA networks. No VIFIB hardware would need to be replaced.

Tenants deploying applications on VIFIB's infrastructure may or may not be affected by RINA deployment. Tenants may want to adapt their applications to use the native RINA IPC API (instead of the sockets API) if they see benefits that outweigh the cost of the adaptation. One driver for this adaptation to happen would be that the systems of the end users of these applications also supported RINA and requested application providers to include RINA support in their applications. Nontheless application providers could choose to stick to the sockets API, since the RINA implementation deployed by VIFIB would support native sockets emulation (it is a strict requirement for adoption, otherwise all applications would have to be adopted to run over RINA networks). Therefore, RINA adoption impact would range from zero to an adaptation of the application code dealing with communications.

Finally end users of applications deployed in the cloud are in a similar situation to the distributed cloud application providers. End user systems (laptops, smartphones, home computers, etc.) may be upgraded with the software implementing RINA if the advantages in doing so (see next section) outweigh the drawbacks. Upgrading to RINA is not a requirement, since end users can access services deployed in the distributed cloud via VIFIB gateways, which would convert from IPv4/IPv6 to RINA and viceversa. However, native RINA support in end user's system could provide enhanced multi-homing, mobility, QoS support and security. Applications running in end users' systems may or may not be upgraded to the native RINA API depending on the decision taken by the providers of the applications deployed in the distributed cloud.

The table below summarizes the RINA adoption impact to the different stakeholders, for three different incremental adoption scenarios.

Scenario	**DC Manager**	**App providers**	**End Users**
RINA in DC only	Deploy RINA implementation in internal systems, update NMS, train personnel	No impact	No impact

RINA in DC and end-users systems	Same as before	No impact	Install RINA implementation in internal systems, learn how to configure
RINA in DC, end users systems and applications	Same as before	Update applications to IPC API	Same as before, deploy upgraded applications

The following table summarizes the RINA adoption benefits for the stakeholders of the Distributed Cloud use case.

Distributed Cloud (DC) Priorities & Trends		**PRISTINE Research Areas**	**Business Impact towards UC**
Scalability	• global routing DC on flat IPv6 addresses, addresses assigned to interfaces	Addressing and routing, configuration management	• Minimize routing table sizes by: assigning addresses to nodes, use of topological addressing, partitioning the DC network in multiple layers. • DC network (single layer) scales to higher number of nodes, routing convergence is faster, providing enhanced resiliency in case of failures • Agile creation and configuration of multiple layers increases the scalability of the DC (enabling flow aggregation at border routers between layers)
Security	• Authentication and access control of DC network nodes • Encryption of all traffic exchanged between DC network nodes • Identification and isolation of	Security, security management	• Simpler security model which increases probability of identifying rogue members • Better tooling to isolate rogue members (isolate rogue members in a specific "jail" DIF)

Distributed Cloud (DC) Priorities & Trends		PRISTINE Research Areas	Business Impact towards UC
	rogue nodes in the DC network		
Performance isolation	• Keep DC network in an optimum operating point • Manage congestion, sharing bottlenecks in a fair way between QoS classes • Allocate loss and delay between QoS classes	Congestion management, resource allocation, performance management	• Allow the DC operator to keep the network stable at higher loads, maximizing the utilization of the infrastructure • Allows the DC operator to offer differentiated SLAs to customers, supporting loss-sensitive and delay-sensitive applications over the same infrastructure

4.3.2. Datacentre Networking Use Case

4.3.2.1. Stakeholder Overview

In the market context, the priority stakeholder in PRISTINE's Datacenter (DC) Networking use case is of course the datacenter operator itself. However, supporting stakeholders exist in the value chain that is relevant when considering RINA's impact. The primary actors include:

- Datacenter operator: In the most simplified business model the operating tenant is also the facility owner. This could also include colocation at the same facility and more complex models.
- Datacenter infrastructure management (DCIM) solution providers: supplying their DC operator customers via integrated software suites (e.g. full facility management and monitoring) or stand-alone solutions (e.g. power management solutions, sensor system, etc.).
- Network provider: Supply the network and bandwidth between interconnected data centers.
- Hosted tenants: The operator's serviced user, whether of the same organization or an external customer. Extremely diverse depending on the datacenter's profile, e.g. public cloud provider vs. a large enterprise's

private DC. These could include anything from an application service provider with its deployed SaaS and end-user customer data, to the underlying infrastructure of a network operator.

Other areas of the value chain exist, but discarded from the focus of this summary as its business impact is more indirect. For example, more efficient energy consumption leads to lower revenue for the energy provider; a more CapEx/OpEx efficient software-driven datacenter would affect construction design.

The direct adopter of RINA technology would be the DC operator. The DCIM, system integrator, etc. role in the market, for example, would be very relevant as a vehicle to deliverer RINA technology in a business scenario, but impact would still start at the operations of the datacenter itself.

A simplified typology of datacenters is presented for context (modified and extended from [Gigaom]):

1. Public Cloud Providers, such as Infrastructure as a Service (IaaS) for Software as a Service (SaaS) hosting, web hosting, etc. These datacenters are focused on multi-tenant hosting for a variety of outsourced applications and data from its customers.
2. Co-location Centers, essentially renting servers to different tenants, servicing them with the needed support of space, power, cooling, security, etc.
3. Enterprise Datacenters, “in-house” facilities that are operated by a single organization, most associated to the Large Enterprise profile due to the capital needed. The model pre-dates IaaS outsourcing, yet remains a relevant target for PRISTINE in terms of scaling target size for RINA adoption (vs. larger public datacenters).
4. Scientific Computing Centers, such as national laboratories, HPC for science, etc. Larger multi-site networks exist in these communities, as well.

All profiles are relevant for PRISTINE, which targets RINA for both inter- and intra-communications for datacenters. For the later, a market segment for solutions has risen due the rise of DC multi-site deployment and complexity, termed Data Center Interconnect (DCI). This can include various scenarios, including:

- multiple DCs for the same organization, e.g. a multi-national or cloud provider with multiple deployments across its geographic markets
- to federate DCs of different organizations to shared data or resources
- to allow workload sharing and scalability between sites, e.g. consolidating a process at one site in particular or combine resources
- disaster recovery between data centers

4.3.2.2. Business Impact Overview

As the chief stakeholder of the Datacenter Networking use case, DC operators, regardless of profile, look for common improvements in their costs, operation and service around a multitude of common issues. These include traffic management (within and between DCs), energy consumption, performance, reliability, hardware optimization and longevity, network scalability and security, all of which have spurred adoption for software-defined management of the network and a heavier reliance on virtualization.

RINA's potential impact addresses DC operator core business objectives:

- Lowering OpEx (operational expenditure) with better resource & network optimization, traffic management, etc.
- Lowering OpEx and environmental footprint with better energy consumption (e.g. Power Usage Effectiveness, PUE). Energy is a huge cost of any DC's operating budget, as well as an environmental issue that receives pressure from regulation and industry certification, including Green IT.
- Lowering CapEx (capital expenditure) with better use of existing hardware and future procurement of hardware with better longevity and modularity (e.g. more generic hardware with more reliance on software-defined management)
- Increasing performance, reliability and QoS towards both their hosted customers between other connected DCs, fulfilling internal or SLA-related metrics, and increasing competitive standing.

The above impact would translate to more specific business benefits based on the target objectives of that datacenter in question. For example, an increase in QoS for hosted applications in a cloud provider's datacenter;

more efficient business processes and data analytics in a large enterprise tenet of a datacenter; better end-user services due to related improvements in a network operator's datacenter infrastructure; etc.

A mapping has been drafted for D6.2 to derive the business impact of the potential RINA adoption towards datacenters, linked to DC operator needs and the research areas that served the basis for PRISTINE policies and experimentation.

Datacenter Priorities & Trends		**PRISTINE Research Areas**	**Business Impact towards UC**
Traffic Management	• need better routing and load-balancing to manage increasing peaks	resource allocation; congestion control	• optimized traffic management leads to lower OpEx, maintain QoS at peaks
Energy Consumption	• lower expenditure (power usage effectiveness) and carbon footprint	resource allocation; congestion control; resiliency, high availability; performance management; configuration management	• lower DC energy-related costs, cooling, etc. (due to better management and optimization of network and related HW) • aid sustainability (environment) policy, compliance, etc.
Performance & Reliability	• increasing QoS challenges; fault avoidance		• higher QoS, better B2B customer satisfaction • lower failure rate • better integration between DCs in network
HW Optimization & Longevity	• modular HW; better use of existing HW	performance management	lower CapEx, higher re-use of existing investments
Network Scalability	• adoption of SW-defined net. architecture	performance management; configuration management	• long-term competitiveness, • lower maintenance/ upgrade costs
Security	• network security within and between DC sites	authentication, access control, encryption; security coordination	• higher security for multi-site DC operator, • compliance to certificates, regulation

Datacenter Priorities & Trends	PRISTINE Research Areas	Business Impact towards UC
		• higher customer trust

4.4. Next Steps

The work towards the final evaluation of the PRISTINE project results will focus on two main axes. On one hand, the research areas will continue to design and carry out experiments that further validate the usefulness of the PRISTINE SDK as far as RINA benefits are concerned. On the other hand, we intend to combine several research areas, designing and executing more complex experiments that help validate the SDK and RINA at a larger scale. We will define experiments that gather representative aspects of the three use cases, making sure that the simultaneous use of policies belonging to different research areas do not interfere with one another, but rather show at least the same level of performance they did when they were tested individually. These experiments will provide further feedback towards development regarding the combination of complex scenarios for RINA and the SDK, such as the use cases require.

As far as economic aspects are concerned, the preliminary analysis will be extended to match the larger scope introduced by the more complex experiments combining different research areas and providing a more holistic approach from the perspective of the three industry use cases. The business impact exercise will provide a feasibility analysis for initial RINA adoption in the scope of those use cases, as well as a potential roadmap for larger penetration of the technology towards those stakeholders, such as datacenter operators, network service providers, cloud infrastructure/software providers, large enterprise, etc. This will take into account not only the potential benefits and relative advantage, but also compatibility and complexity considerations with existing (e.g. previous CapEx investments) and developing technology (e.g. adoption paths for SDN, NFV, etc.)

Finally, experimentation with the Network Service Provider use case, which has been left for the second iteration of the project, will be completed and reported as far as technical and business impacts are concerned.

References

[Allin1] PRISTINE all-in-one-machine testing tool. Available online[17].

[D32] PRISTINE Consortium. Deliverable 3.2. Initial specification and proof of concept implementation of techniques to enhance performance and resource utilization in networks. June 2015. Available online[18].

[D42] PRISTINE Consortium. Deliverable 4.2 version 2. Initial specification and proof of concept implementation of innovative security and reliability enablers. July 2015. Available online[19].

[D61] PRISTINE Consortium. Deliverable 6.1. First iteration trials plan for System-level integration and validation. Available online[20].

[Gigaom] Adam Lesser, Four types of data-centers, available online[21].

[IRINA] GN3+ IRINA Open Call project. Resources available online[22].

[Jacobson] Jacobson, V. Congestion avoidance and control. In Proceedings of SIGCOMM '88 (Stanford, CA, Aug. 1988), ACM.

[Jain] Jain R and Ramakrishnan K (1988) Congestion avoidance in computer networks with A connectionless network layer: concepts, goals, and methodology. In Computer Networking Symposium: Proceedings, April 11–13, 1988 (ed. IEEE), pp. 134–143.

[RFC5286] A. Atlas and A. Zinin. "Basic Specification for IP Fast Reroute: Loop-Free Alternates". RFC 5286. Sept 2008.

[RFC2309] Braden et al. "Recommendations on Queue Management and Congestion Avoidance in the Internet", RFC 2309, April 1998.

[17] https://github.com/vmaffione/pristine-test
[18] http://ict-pristine.eu/wp-content/uploads/2013/12/pristine-d32-enhance-performance-and-resource-utilization-in-networks-v1_0.pdf
[19] http://ict-pristine.eu/wp-content/uploads/2013/12/Pristine-D4.2-2nd-Version.pdf
[20] http://ict-pristine.eu/wp-content/uploads/2013/12/pristine_d61_first_iteration_trials_plan_v1_0-1.pdf
[21] http://research.gigaom.com/2012/10/4-types-of-data-centers/
[22] http://www.geant.net/opencall/Optical/Pages/IRINA.aspx

[Libreswan] Libre SWAN project. Accessed September 2015. Available online[23]

[TAP] Universal TUN/TAP device driver. online[24]

[23] https://libreswan.org/wiki/Benchmarking_and_Performance_testing
[24] https://www.kernel.org/doc/Documentation/networking/tuntap.txt

A. Experiment configuration files

A.1. ACC experiments

A.1.1. RED + TCP-Tahoe experiment

A.1.1.1. Host 2

ipcmanager.conf

```
{
  "localConfiguration" : {
    "installationPath" : "/usr/local/irati/bin",
    "libraryPath" : "/usr/local/irati/lib",
    "logPath" : "/usr/local/irati/var/log",
    "consolePort" : 32766,
    "pluginsPaths" : ["/usr/local/irati/lib/rinad/ipcp"]
  },
  "applicationToDIFMappings" : [ {
    "encodedAppName" : "traffic.generator.client-1--",
    "difName" : "up.DIF"
  }, {
    "encodedAppName" : "traffic.generator.client-2--",
    "difName" : "up.DIF"
  }, {
    "encodedAppName" : "traffic.generator.client-3--",
    "difName" : "up.DIF"
  }, {
    "encodedAppName" : "traffic.generator.client-4--",
    "difName" : "up.DIF"
  }, {
    "encodedAppName" : "traffic.generator.server-1--",
    "difName" : "up.DIF"
  }],
  "ipcProcessesToCreate" : [ {
    "type" : "shim-eth-vlan",
    "apName" : "test-eth-vlan",
    "apInstance" : "1",
    "difName" : "100"
   }, {
    "type" : "normal-ipc",
    "apName" : "D",
    "apInstance" : "1",
```

```
    "difName" : "right.DIF",
    "difsToRegisterAt" : ["100"]
   }, {
    "type" : "normal-ipc",
    "apName" : "G",
    "apInstance" : "1",
    "difName" : "up.DIF",
    "difsToRegisterAt" : ["right.DIF"]
   } ],
  "difConfigurations" : [ {
    "name" : "100",
    "template" : "shim-eth-vlan.dif"
  }, {
    "name" : "right.DIF",
    "template" : "accr.dif"
  }, {
    "name" : "up.DIF",
    "template" : "acc.dif"
  }  ]
}
```

shim-eth-vlan.dif

```
{
    "difType" : "shim-eth-vlan",
    "configParameters" : {
        "interface-name" : "eth1"
      }
}
```

A.1.1.2. Router

ipcmanager.conf

```
{
  "localConfiguration" : {
    "installationPath" : "/usr/local/irati/bin",
    "libraryPath" : "/usr/local/irati/lib",
    "logPath" : "/usr/local/irati/var/log",
    "consolePort" : 32766,
    "pluginsPaths" : ["/usr/local/irati/lib/rinad/ipcp"]
  },
  "applicationToDIFMappings" : [ {
```

```
    "encodedAppName" : "traffic.generator.client-1--",
    "difName" : "right.DIF"
  }, {
    "encodedAppName" : "traffic.generator.client-2--",
    "difName" : "right.DIF"
  }, {
    "encodedAppName" : "traffic.generator.server-1--",
    "difName" : "right.DIF"
  }],
  "ipcProcessesToCreate" : [ {
    "type" : "shim-eth-vlan",
    "apName" : "test-eth-vlan",
    "apInstance" : "1",
    "difName" : "100"
   }, {
    "type" : "shim-eth-vlan",
    "apName" : "test-eth-vlan2",
    "apInstance" : "1",
    "difName" : "110"
   }, {
    "type" : "normal-ipc",
    "apName" : "B",
    "apInstance" : "1",
    "difName" : "left.DIF",
    "difsToRegisterAt" : ["110"]
   }, {
    "type" : "normal-ipc",
    "apName" : "C",
    "apInstance" : "1",
    "difName" : "right.DIF",
    "difsToRegisterAt" : ["100"]
   }, {
    "type" : "normal-ipc",
    "apName" : "F",
    "apInstance" : "1",
    "difName" : "up.DIF",
    "difsToRegisterAt" : ["left.DIF", "right.DIF"]
   } ],
  "difConfigurations" : [ {
    "name" : "100",
    "template" : "shim-eth-vlan.dif"
  }, {
    "name" : "110",
    "template" : "shim-eth-vlan2.dif"
  }, {
```

```
    "name" : "left.DIF",
    "template" : "accl.dif"
  }, {
    "name" : "right.DIF",
    "template" : "accr.dif"
  }, {
    "name" : "up.DIF",
    "template" : "acc.dif"
  } ]
}
```

shim-eth-vlan.dif

```
{
    "difType" : "shim-eth-vlan",
    "configParameters" : {
        "interface-name" : "eth1"
      }
}
```

shim-eth-vlan2.dif

```
{
    "difType" : "shim-eth-vlan",
    "configParameters" : {
        "interface-name" : "eth2"
      }
}
```

A.1.1.3. Host 1

ipcmanager.conf

```
{
  "localConfiguration" : {
    "installationPath" : "/usr/local/irati/bin",
    "libraryPath" : "/usr/local/irati/lib",
    "logPath" : "/usr/local/irati/var/log",
    "consolePort" : 32766,
    "pluginsPaths" : ["/usr/local/irati/lib/rinad/ipcp"]
  },
  "applicationToDIFMappings" : [ {
    "encodedAppName" : "traffic.generator.client-1--",
```

```
    "difName" : "up.DIF"
  }, {
    "encodedAppName" : "traffic.generator.client-2--",
    "difName" : "up.DIF"
  }, {
    "encodedAppName" : "traffic.generator.client-3--",
    "difName" : "up.DIF"
  }, {
    "encodedAppName" : "traffic.generator.client-4--",
    "difName" : "up.DIF"
  }, {
    "encodedAppName" : "traffic.generator.server-1--",
    "difName" : "up.DIF"
  }],
  "ipcProcessesToCreate" : [ {
    "type" : "shim-eth-vlan",
    "apName" : "test-eth-vlan",
    "apInstance" : "1",
    "difName" : "110"
   }, {
    "type" : "normal-ipc",
    "apName" : "A",
    "apInstance" : "1",
    "difName" : "left.DIF",
    "difsToRegisterAt" : ["110"]
   }, {
    "type" : "normal-ipc",
    "apName" : "E",
    "apInstance" : "1",
    "difName" : "up.DIF",
    "difsToRegisterAt" : ["left.DIF"]
   } ],
  "difConfigurations" : [ {
    "name" : "110",
    "template" : "shim-eth-vlan.dif"
  }, {
    "name" : "left.DIF",
    "template" : "accl.dif"
  }, {
    "name" : "up.DIF",
    "template" : "acc.dif"
  }  ]
}
```

shim-eth-vlan.dif

```
{
    "difType" : "shim-eth-vlan",
    "configParameters" : {
        "interface-name" : "eth1"
      }
}
```

A.1.1.4. DIF configurations

accr.dif

```
{
    "difType" : "normal-ipc",
     "knownIPCProcessAddresses" : [ {
         "apName" : "C",
         "apInstance" : "1",
         "address" : 1
          }, {
         "apName" : "D",
         "apInstance" : "1",
         "address" : 2
        } ],
        "addressPrefixes" : [ {
         "addressPrefix" : 0,
         "organization" : "N.Bourbaki"
          }, {
         "addressPrefix" : 16,
         "organization" : "IRATI"
      } ]
}
```

accl.dif

```
{
    "difType" : "normal-ipc",
     "knownIPCProcessAddresses" : [ {
         "apName" : "A",
         "apInstance" : "1",
         "address" : 1
          }, {
```

```
          "apName" : "B",
          "apInstance" : "1",
          "address" : 2
         } ],
         "addressPrefixes" : [ {
          "addressPrefix" : 0,
          "organization" : "N.Bourbaki"
           }, {
          "addressPrefix" : 16,
          "organization" : "IRATI"
       } ]
}
```

acc.dif

```
{
    "difType" : "normal-ipc",
    "dataTransferConstants" : {
     "addressLength" : 2,
     "cepIdLength" : 2,
     "lengthLength" : 2,
     "portIdLength" : 2,
     "qosIdLength" : 2,
     "sequenceNumberLength" : 4,
     "maxPduSize" : 10000,
     "maxPduLifetime" : 60000
    },
    "qosCubes" : [ {
  "name" : "unreliablewithflowcontrol",
          "id" : 1,
          "partialDelivery" : false,
          "orderedDelivery" : true,
          "efcpPolicies" : {
               "dtpPolicySet" : {
                 "name" : "default",
                 "version" : "0"
               },
               "initialATimer" : 300,
               "dtcpPresent" : true,
               "dtcpConfiguration" : {
                    "dtcpPolicySet" : {
                      "name" : "red-ps",
                      "version" : "0"
                    },
```

```
                "rtxControl" : false,
                "flowControl" : true,
                "flowControlConfig" : {
                    "rateBased" : false,
                    "windowBased" : true,
                    "windowBasedConfig" : {
                      "maxClosedWindowQueueLength" : 50,
                      "initialCredit" : 3
                     }
                }
            }
       }
    }, {
    "name" : "reliablewithflowcontrol",
      "id" : 2,
      "partialDelivery" : false,
      "orderedDelivery" : true,
      "maxAllowableGap": 0,
      "efcpPolicies" : {
           "dtpPolicySet" : {
             "name" : "default",
             "version" : "0"
           },
           "initialATimer" : 300,
           "dtcpPresent" : true,
           "dtcpConfiguration" : {
                "dtcpPolicySet" : {
                  "name" : "red-ps",
                  "version" : "0"
                },
                "rtxControl" : true,
                "rtxControlConfig" : {
                    "dataRxmsNmax" : 5,
                    "initialRtxTime" : 1000
                },
                "flowControl" : true,
                "flowControlConfig" : {
                    "rateBased" : false,
                    "windowBased" : true,
                    "windowBasedConfig" : {
                      "maxClosedWindowQueueLength" : 50,
                      "initialCredit" : 25
                    }
             }
           }
```

```
        }
    } ],
    "knownIPCProcessAddresses" : [ {
     "apName" : "E",
     "apInstance" : "1",
     "address" : 1
    }, {
     "apName" : "F",
     "apInstance" : "1",
     "address" : 2
        }, {
        "apName" : "G",
        "apInstance" : "1",
        "address" : 3
  } ],
  "addressPrefixes" : [ {
     "addressPrefix" : 0,
     "organization" : "N.Bourbaki"
    }, {
     "addressPrefix" : 16,
     "organization" : "IRATI"
     } ],
    "rmtConfiguration" : {
       "pftConfiguration" : {
         "policySet" : {
           "name" : "default",
           "version" : "0"
         }
       },
       "policySet" : {
         "name" : "red-ps",
         "version" : "1",
         "parameters": [{
            "name"  : "qmax_p",
            "value" : "600"
            }, {
            "name"  : "qth_min_p",
            "value" : "10"
            }, {
            "name"  : "qth_max_p",
            "value" : "30"
            }, {
            "name"  : "Wlog_p",
            "value" : "10"
            }, {
```

```
            "name"  : "Plog_p",
            "value" : "9"
        }]
    }
},
"enrollmentTaskConfiguration" : {
    "policySet" : {
        "name" : "default",
        "version" : "1",
        "parameters" : [{
            "name"  : "enrollTimeoutInMs",
            "value" : "10000"
          },{
            "name"  : "watchdogPeriodInMs",
            "value" : "30000"
          },{
            "name"  : "declaredDeadIntervalInMs",
            "value" : "120000"
          },{
            "name"  : "neighborsEnrollerPeriodInMs",
            "value" : "30000"
          },{
            "name"  : "maxEnrollmentRetries",
            "value" : "3"
          }]
    }
},
"flowAllocatorConfiguration" : {
    "policySet" : {
      "name" : "default",
      "version" : "1"
     }
},
"namespaceManagerConfiguration" : {
    "policySet" : {
      "name" : "default",
      "version" : "1"
      }
},
"securityManagerConfiguration" : {
    "policySet" : {
      "name" : "default",
      "version" : "1"
      }
},
```

```
      "resourceAllocatorConfiguration" : {
          "pduftgConfiguration" : {
            "policySet" : {
              "name" : "default",
              "version" : "0"
            }
          }
      },
      "routingConfiguration" : {
          "policySet" : {
            "name" : "link-state",
            "version" : "1",
            "parameters" : [{
              "name"  : "objectMaximumAge",
              "value" : "10000"
            },{
              "name"  : "waitUntilReadCDAP",
              "value" : "5001"
            },{
              "name"  : "waitUntilError",
              "value" : "5001"
            },{
              "name"  : "waitUntilPDUFTComputation",
              "value" : "103"
            },{
              "name"  : "waitUntilFSODBPropagation",
              "value" : "101"
            },{
              "name"  : "waitUntilAgeIncrement",
              "value" : "997"
            },{
              "name"  : "routingAlgorithm",
              "value" : "Dijkstra"
            }]
      }
    }
}
```

A.1.2. Explicit congestion detection with binary feedback (Jain et al) experiment

A.1.2.1. Host 2

ipcmanager.conf

```
{
  "localConfiguration" : {
    "installationPath" : "/usr/local/irati/bin",
    "libraryPath" : "/usr/local/irati/lib",
    "logPath" : "/usr/local/irati/var/log",
    "consolePort" : 32766,
    "pluginsPaths" : ["/usr/local/irati/lib/rinad/ipcp"]
  },
  "applicationToDIFMappings" : [ {
    "encodedAppName" : "traffic.generator.client-1--",
    "difName" : "up.DIF"
  }, {
    "encodedAppName" : "traffic.generator.client-2--",
    "difName" : "up.DIF"
  }, {
    "encodedAppName" : "traffic.generator.client-3--",
    "difName" : "up.DIF"
  }, {
    "encodedAppName" : "traffic.generator.client-4--",
    "difName" : "up.DIF"
  }, {
    "encodedAppName" : "traffic.generator.server-1--",
    "difName" : "up.DIF"
  }],
  "ipcProcessesToCreate" : [ {
    "type" : "shim-eth-vlan",
    "apName" : "test-eth-vlan",
    "apInstance" : "1",
    "difName" : "100"
   }, {
    "type" : "normal-ipc",
    "apName" : "D",
    "apInstance" : "1",
    "difName" : "right.DIF",
    "difsToRegisterAt" : ["100"]
   }, {
    "type" : "normal-ipc",
```

```
    "apName" : "G",
    "apInstance" : "1",
    "difName" : "up.DIF",
    "difsToRegisterAt" : ["right.DIF"]
   } ],
  "difConfigurations" : [ {
    "name" : "100",
    "template" : "shim-eth-vlan.dif"
  }, {
    "name" : "right.DIF",
    "template" : "accr.dif"
  }, {
    "name" : "up.DIF",
    "template" : "acc.dif"
  }  ]
}
```

shim-eth-vlan.dif

```
{
    "difType" : "shim-eth-vlan",
    "configParameters" : {
        "interface-name" : "eth1"
      }
}
```

A.1.2.2. Router

ipcmanager.conf

```
{
  "localConfiguration" : {
    "installationPath" : "/usr/local/irati/bin",
    "libraryPath" : "/usr/local/irati/lib",
    "logPath" : "/usr/local/irati/var/log",
    "consolePort" : 32766,
    "pluginsPaths" : ["/usr/local/irati/lib/rinad/ipcp"]
  },
  "applicationToDIFMappings" : [ {
    "encodedAppName" : "traffic.generator.client-1--",
    "difName" : "right.DIF"
  }, {
    "encodedAppName" : "traffic.generator.client-2--",
```

```
    "difName" : "right.DIF"
  }, {
    "encodedAppName" : "traffic.generator.server-1--",
    "difName" : "right.DIF"
  }],
  "ipcProcessesToCreate" : [ {
    "type" : "shim-eth-vlan",
    "apName" : "test-eth-vlan",
    "apInstance" : "1",
    "difName" : "100"
   }, {
    "type" : "shim-eth-vlan",
    "apName" : "test-eth-vlan2",
    "apInstance" : "1",
    "difName" : "110"
   }, {
    "type" : "normal-ipc",
    "apName" : "B",
    "apInstance" : "1",
    "difName" : "left.DIF",
    "difsToRegisterAt" : ["110"]
   }, {
    "type" : "normal-ipc",
    "apName" : "C",
    "apInstance" : "1",
    "difName" : "right.DIF",
    "difsToRegisterAt" : ["100"]
   }, {
    "type" : "normal-ipc",
    "apName" : "F",
    "apInstance" : "1",
    "difName" : "up.DIF",
    "difsToRegisterAt" : ["left.DIF", "right.DIF"]
   } ],
  "difConfigurations" : [ {
    "name" : "100",
    "template" : "shim-eth-vlan.dif"
  }, {
    "name" : "110",
    "template" : "shim-eth-vlan2.dif"
  }, {
    "name" : "left.DIF",
    "template" : "accl.dif"
  }, {
    "name" : "right.DIF",
```

```
    "template" : "accr.dif"
  }, {
    "name" : "up.DIF",
    "template" : "acc.dif"
  } ]
}
```

shim-eth-vlan.dif

```
{
    "difType" : "shim-eth-vlan",
    "configParameters" : {
        "interface-name" : "eth1"
      }
}
```

shim-eth-vlan2.dif

```
{
    "difType" : "shim-eth-vlan",
    "configParameters" : {
        "interface-name" : "eth2"
      }
}
```

A.1.2.3. Host 1

ipcmanager.conf

```
{
  "localConfiguration" : {
    "installationPath" : "/usr/local/irati/bin",
    "libraryPath" : "/usr/local/irati/lib",
    "logPath" : "/usr/local/irati/var/log",
    "consolePort" : 32766,
    "pluginsPaths" : ["/usr/local/irati/lib/rinad/ipcp"]
  },
  "applicationToDIFMappings" : [ {
    "encodedAppName" : "traffic.generator.client-1--",
    "difName" : "up.DIF"
  }, {
    "encodedAppName" : "traffic.generator.client-2--",
    "difName" : "up.DIF"
```

```
  }, {
    "encodedAppName" : "traffic.generator.client-3--",
    "difName" : "up.DIF"
  }, {
    "encodedAppName" : "traffic.generator.client-4--",
    "difName" : "up.DIF"
  }, {
    "encodedAppName" : "traffic.generator.server-1--",
    "difName" : "up.DIF"
  }],
  "ipcProcessesToCreate" : [ {
    "type" : "shim-eth-vlan",
    "apName" : "test-eth-vlan",
    "apInstance" : "1",
    "difName" : "110"
   }, {
    "type" : "normal-ipc",
    "apName" : "A",
    "apInstance" : "1",
    "difName" : "left.DIF",
    "difsToRegisterAt" : ["110"]
   }, {
    "type" : "normal-ipc",
    "apName" : "E",
    "apInstance" : "1",
    "difName" : "up.DIF",
    "difsToRegisterAt" : ["left.DIF"]
   } ],
  "difConfigurations" : [ {
    "name" : "110",
    "template" : "shim-eth-vlan.dif"
  }, {
    "name" : "left.DIF",
    "template" : "accl.dif"
  }, {
    "name" : "up.DIF",
    "template" : "acc.dif"
  }  ]
}
```

shim-eth-vlan.dif

```
{
    "difType" : "shim-eth-vlan",
```

```
    "configParameters" : {
        "interface-name" : "eth1"
      }
}
```

A.1.2.4. DIF configurations

accr.dif

```
{
    "difType" : "normal-ipc",
     "knownIPCProcessAddresses" : [ {
         "apName" : "C",
         "apInstance" : "1",
         "address" : 1
          }, {
         "apName" : "D",
         "apInstance" : "1",
         "address" : 2
        } ],
        "addressPrefixes" : [ {
         "addressPrefix" : 0,
         "organization" : "N.Bourbaki"
          }, {
         "addressPrefix" : 16,
         "organization" : "IRATI"
      } ]
}
```

accl.dif

```
{
    "difType" : "normal-ipc",
     "knownIPCProcessAddresses" : [ {
         "apName" : "A",
         "apInstance" : "1",
         "address" : 1
          }, {
         "apName" : "B",
         "apInstance" : "1",
         "address" : 2
        } ],
        "addressPrefixes" : [ {
```

```
          "addressPrefix" : 0,
          "organization" : "N.Bourbaki"
           }, {
          "addressPrefix" : 16,
          "organization" : "IRATI"
       } ]
}
```

acc.dif

```
{
    "difType" : "normal-ipc",
    "dataTransferConstants" : {
     "addressLength" : 2,
     "cepIdLength" : 2,
     "lengthLength" : 2,
     "portIdLength" : 2,
     "qosIdLength" : 2,
     "sequenceNumberLength" : 4,
     "maxPduSize" : 10000,
     "maxPduLifetime" : 60000
    },
    "qosCubes" : [ {
  "name" : "unreliablewithflowcontrol",
          "id" : 1,
          "partialDelivery" : false,
          "orderedDelivery" : true,
          "efcpPolicies" : {
               "dtpPolicySet" : {
                 "name" : "default",
                 "version" : "0"
               },
               "initialATimer" : 300,
               "dtcpPresent" : true,
               "dtcpConfiguration" : {
                    "dtcpPolicySet" : {
                      "name" : "cas-ps",
                      "version" : "0"
                    },
                    "rtxControl" : false,
                    "flowControl" : true,
                    "flowControlConfig" : {
                        "rateBased" : false,
                        "windowBased" : true,
```

```
                    "windowBasedConfig" : {
                      "maxClosedWindowQueueLength" : 50,
                      "initialCredit" : 20
                     }
                }
            }
        }
    }, {
    "name" : "reliablewithflowcontrol",
      "id" : 2,
      "partialDelivery" : false,
      "orderedDelivery" : true,
      "maxAllowableGap": 0,
      "efcpPolicies" : {
           "dtpPolicySet" : {
             "name" : "default",
             "version" : "0"
           },
           "initialATimer" : 300,
           "dtcpPresent" : true,
           "dtcpConfiguration" : {
                "dtcpPolicySet" : {
                  "name" : "cas-ps",
                  "version" : "0"
                },
                "rtxControl" : true,
                "rtxControlConfig" : {
                    "dataRxmsNmax" : 5,
                    "initialRtxTime" : 1000
                },
                "flowControl" : true,
                "flowControlConfig" : {
                    "rateBased" : false,
                    "windowBased" : true,
                    "windowBasedConfig" : {
                      "maxClosedWindowQueueLength" : 50,
                      "initialCredit" : 20
                    }
             }
           }
      }
} ],
"knownIPCProcessAddresses" : [ {
 "apName" : "E",
 "apInstance" : "1",
```

```
      "address" : 1
    }, {
      "apName" : "F",
      "apInstance" : "1",
      "address" : 2
        }, {
        "apName" : "G",
        "apInstance" : "1",
        "address" : 3
  } ],
  "addressPrefixes" : [ {
     "addressPrefix" : 0,
     "organization" : "N.Bourbaki"
    }, {
     "addressPrefix" : 16,
     "organization" : "IRATI"
     } ],
    "rmtConfiguration" : {
       "pftConfiguration" : {
         "policySet" : {
           "name" : "default",
           "version" : "0"
         }
       },
       "policySet" : {
         "name" : "cas-ps",
         "version" : "1"
       }
    },
    "enrollmentTaskConfiguration" : {
       "policySet" : {
          "name" : "default",
          "version" : "1",
          "parameters" : [{
              "name"  : "enrollTimeoutInMs",
              "value" : "10000"
           },{
             "name"  : "watchdogPeriodInMs",
             "value" : "30000"
           },{
             "name"  : "declaredDeadIntervalInMs",
             "value" : "120000"
           },{
             "name"  : "neighborsEnrollerPeriodInMs",
             "value" : "30000"
```

```
        },{
          "name"  : "maxEnrollmentRetries",
          "value" : "3"
        }]
    }
},
"flowAllocatorConfiguration" : {
    "policySet" : {
      "name" : "default",
      "version" : "1"
     }
},
"namespaceManagerConfiguration" : {
    "policySet" : {
      "name" : "default",
      "version" : "1"
      }
},
"securityManagerConfiguration" : {
    "policySet" : {
      "name" : "default",
      "version" : "1"
      }
},
"resourceAllocatorConfiguration" : {
    "pduftgConfiguration" : {
      "policySet" : {
        "name" : "default",
        "version" : "0"
      }
    }
},
"routingConfiguration" : {
    "policySet" : {
      "name" : "link-state",
      "version" : "1",
      "parameters" : [{
        "name"  : "objectMaximumAge",
        "value" : "10000"
      },{
        "name"  : "waitUntilReadCDAP",
        "value" : "5001"
      },{
        "name"  : "waitUntilError",
        "value" : "5001"
```

```
            },{
              "name"  : "waitUntilPDUFTComputation",
              "value" : "103"
            },{
              "name"  : "waitUntilFSODBPropagation",
              "value" : "101"
            },{
              "name"  : "waitUntilAgeIncrement",
              "value" : "997"
            },{
              "name"  : "routingAlgorithm",
              "value" : "Dijkstra"
            }]
      }
  }
}
```

A.2. MAD-MANAGER experiments

A.2.1. ipcmanager.conf

```
{
  "localConfiguration" : {
    "installationPath" : "/home/irati/iratis/bin",
    "libraryPath" : "/home/irati/iratis/lib",
    "logPath" : "/home/irati/iratis/var/log",
    "consolePort" : 32766,
    "pluginsPaths" : ["/home/irati/iratis/lib/rinad/ipcp"]
  },
  "applicationToDIFMappings" : [ {
    "encodedAppName" : "rina.utils.apps.echo.server-1--",
    "difName" : "normal.DIF"
  }, {
    "encodedAppName" : "rina.utils.apps.echo.client-1--",
    "difName" : "normal.DIF"
  }, {
    "encodedAppName" : "rina.utils.apps.rinaperf.server-1--",
    "difName" : "normal.DIF"
  }, {
    "encodedAppName" : "rina.utils.apps.rinaperf.client-1--",
    "difName" : "normal.DIF"
  }],
  "ipcProcessesToCreate" : [ {
    "type" : "shim-eth-vlan",
```

```
    "apName" : "eth-vlan-300",
    "apInstance" : "1",
    "difName" : "300"
   }, {
    "type" : "shim-eth-vlan",
    "apName" : "eth-vlan-400",
    "apInstance" : "1",
    "difName" : "400"
   }, {
    "type" : "normal-ipc",
    "apName" : "NMS1",
    "apInstance" : "1",
    "difName" : "NMS.DIF",
    "difsToRegisterAt" : ["400"]
   } ],
  "difConfigurations" : [ {
    "name" : "300",
    "template" : "shim-eth-vlan.dif"
  },  {
    "name" : "400",
    "template" : "shim-eth-vlan.dif"
  }, {
    "name" : "NMS.DIF",
    "template" : "NMS.dif"
  }, {
    "name" : "normal.DIF",
    "template" : "normal.dif"
  }  ],
    "addons" : {
        "mad" : {
            "managerAppName" : "rina.apps.mad.1-1--",
            "NMSDIFs" : [
                {
                    "DIF": "NMS.DIF"
                }
            ],
            "managerConnections" : [
                {
                    "managerAppName" : "rina.apps.manager-1--",
                    "DIF" : "NMS.DIF"
                } ]
            }
    }
}
```

A.2.2. NMS DIF

```
{
    "difType" : "normal-ipc",
    "dataTransferConstants" : {
     "addressLength" : 2,
     "cepIdLength" : 2,
     "lengthLength" : 2,
     "portIdLength" : 2,
     "qosIdLength" : 2,
     "sequenceNumberLength" : 4,
     "maxPduSize" : 10000,
     "maxPduLifetime" : 60000
    },
    "qosCubes" : [ {
  "name" : "unreliablewithflowcontrol",
        "id" : 1,
        "partialDelivery" : false,
        "orderedDelivery" : true,
        "efcpPolicies" : {
             "dtpPolicySet" : {
               "name" : "default",
               "version" : "0"
             },
             "initialATimer" : 300,
             "dtcpPresent" : true,
             "dtcpConfiguration" : {
                  "dtcpPolicySet" : {
                    "name" : "default",
                    "version" : "0"
                  },
                  "rtxControl" : false,
                  "flowControl" : true,
                  "flowControlConfig" : {
                      "rateBased" : false,
                      "windowBased" : true,
                      "windowBasedConfig" : {
                        "maxClosedWindowQueueLength" : 50,
                        "initialCredit" : 50
                       }
                  }
             }
          }
       }, {
```

```
      "name" : "reliablewithflowcontrol",
        "id" : 2,
        "partialDelivery" : false,
        "orderedDelivery" : true,
        "maxAllowableGap": 0,
        "efcpPolicies" : {
             "dtpPolicySet" : {
               "name" : "default",
               "version" : "0"
             },
             "initialATimer" : 300,
             "dtcpPresent" : true,
             "dtcpConfiguration" : {
                  "dtcpPolicySet" : {
                    "name" : "default",
                    "version" : "0"
                  },
                  "rtxControl" : true,
                  "rtxControlConfig" : {
                      "dataRxmsNmax" : 5,
                      "initialRtxTime" : 1000
                  },
                  "flowControl" : true,
                  "flowControlConfig" : {
                      "rateBased" : false,
                      "windowBased" : true,
                      "windowBasedConfig" : {
                        "maxClosedWindowQueueLength" : 50,
                        "initialCredit" : 50
                      }
                  }
             }
        }
    } ],
    "knownIPCProcessAddresses" : [ {
     "apName" : "NMS1",
     "apInstance" : "1",
     "address" : 16
    }, {
     "apName" : "NMS2",
     "apInstance" : "1",
     "address" : 17
  } ],
  "addressPrefixes" : [ {
     "addressPrefix" : 0,
```

```
 "organization" : "N.Bourbaki"
}, {
 "addressPrefix" : 16,
 "organization" : "IRATI"
 } ],
"rmtConfiguration" : {
   "pftConfiguration" : {
     "policySet" : {
       "name" : "default",
       "version" : "0"
     }
   },
   "policySet" : {
     "name" : "default",
     "version" : "1"
   }
},
"enrollmentTaskConfiguration" : {
   "policySet" : {
      "name" : "default",
      "version" : "1",
      "parameters" : [{
          "name"  : "enrollTimeoutInMs",
          "value" : "10000"
        },{
          "name"  : "watchdogPeriodInMs",
          "value" : "30000"
        },{
          "name"  : "declaredDeadIntervalInMs",
          "value" : "120000"
        },{
          "name"  : "neighborsEnrollerPeriodInMs",
          "value" : "30000"
        },{
          "name"  : "maxEnrollmentRetries",
          "value" : "3"
        }]
   }
},
"flowAllocatorConfiguration" : {
    "policySet" : {
      "name" : "default",
      "version" : "1"
     }
},
```

```
"namespaceManagerConfiguration" : {
    "policySet" : {
      "name" : "default",
      "version" : "1"
      }
},
"securityManagerConfiguration" : {
    "policySet" : {
      "name" : "default",
      "version" : "1"
      }
},
"resourceAllocatorConfiguration" : {
    "pduftgConfiguration" : {
      "policySet" : {
        "name" : "default",
        "version" : "0"
      }
    }
},
"routingConfiguration" : {
    "policySet" : {
      "name" : "link-state",
      "version" : "1",
      "parameters" : [{
        "name"  : "objectMaximumAge",
        "value" : "10000"
      },{
        "name"  : "waitUntilReadCDAP",
        "value" : "5001"
      },{
        "name"  : "waitUntilError",
        "value" : "5001"
      },{
        "name"  : "waitUntilPDUFTComputation",
        "value" : "103"
      },{
        "name"  : "waitUntilFSODBPropagation",
        "value" : "101"
      },{
        "name"  : "waitUntilAgeIncrement",
        "value" : "997"
      },{
        "name"  : "routingAlgorithm",
        "value" : "Dijkstra"
```

```
            }]
        }
    }
}
```