Deliverable-2.1

Use cases description and requirements analysis report

Deliverable Editor: Diego LopezTelefonica I+D

Publication date:	31-May-2014
Deliverable Nature:	Report
Dissemination level (Confidentiality):	PU (Public)
Project acronym:	PRISTINE
Project full title:	Programmability In RINA for European supremacy of virTualised NEtworks
Website:	www.ict-pristine.eu
Keywords:	use cases, requirements analysis, RINA, distributed cloud, datacentre networking, network functions virtualization (NFV), SlapOS
Synopsis:	This deliverable provides a detailed view of PRISTINE's reference scenarios - Distributed Cloud, Datacenter Networking and Network Service Provider - and analyzes the requirements that need to be supported in order to successfully fulfill the different use cases.

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

Disclaimer

Executive Summary

The goal of this document is to analyze the use cases that will shape the requirements of the research, development and experimentation activities that will be carried out by the PRISTINE partners. Each use case provides an opportunity to explore and understand how RINA can be applied in a real-world scenario, as well as to assess the advantages it provides over existing alternatives. The main outcome of D2.1 is a set of draft designs of RINA-based solutions for the Distributed Cloud, Datacenter Networking and Network Service Provider use cases. Each design specification includes the type of DIFs in the scenario, the requirements each DIF has to fulfill and the relationships between DIFs; the requirements for the Network Management System and the requirements for supporting tools and subsystems that allow RINA to interoperate with current technologies and applications.

PRISTINE use cases are based on three well-defined scenarios, targeting different realistic deployments of RINA interoperating with some of the current computer networking technologies. The following paragraphs provide a short summary of these scenarios.

Distributed Cloud. SlapOS is a decentralized cloud technology used to build a physically distributed cloud. Customer's applications are run in traditional datacenters, but also in servers from offices and home users. SlapOS is in charge of managing the overall cloud from a logically centralized location: the SlapOS master (a distributed approach is currently under development). The SlapOS master controls the different computers running SlapOS slaves. In terms of networking, the master and the nodes at different locations are interconnected through multiple IPv6 providers. In order to guarantee a high reliability (99.999%), SlapOS uses an overlay called re6st, which creates a mesh network of OpenVPN tunnels on top of several IPv6 providers and uses the Babel protocol for choosing the best routes between nodes. PRISTINE will provide an alternative to the re6st overlay, by using RINA.

Datacentre Networking. The datacenter space is one of the areas that has seen more virtual networking innovations during the last few years, fueled by the flexibility requirements of cloud computing. A myriad of SDN-based virtual network solutions, usually providing L2 over L3 or L4 tunnels and a control plane, are available in the market (VXLAN, NVGRE, STT, etc). PRISTINE will investigate and trial the use of RINA-based solutions for datacenter networking. Important issues to be addressed in a datacenter environment are the mobility of Virtual Machines to allow an efficient utilization of datacenter resources as well as high reliability; multi-homing support;

guaranteeing the level of service in inter-data center communications and flexible allocation of flows supporting computer and storage resources. RINA provides an excellent framework to tackle these issues, and the PRISTINE project will exploit them as explained in task T2.1.

Network Service Provider. The goals of this scenario are to investigate and trial the benefits of the use of the RINA technology by a Network Service Provider (NSP), and to analyze RINA as a essential component of the Network Functions Virtualization (NFV) concept within an operator network. It is obvious that a disruptive, clean-slate technology like RINA would have a difficult way within a NSP environment, very much oriented towards service provisioning and stability, and the phased incorporation of thoroughly tested technologies. But the recent advent of the NFV proposal opens a very interesting window of opportunity for demonstrating RINA in the NSP environment, and showcase the advantages that the combination of both approaches (NFV and RINA) can bring to network service design, management, and operation.

Table of Contents

Glossary

AP
Application Process

API
Application Programming Interface

ARP
Address Resolution Protocol

AS
Autonomous System

CDN
Content Distribution Network

CDP
Cisco Discovery Protocol

CEP
Customer Premises Equipment

DB
Database

DC
Datacenter

DCTCP
Datacenter TCP

DHT
Distributed Hash Table

DIF
Distributed IPC Facility

DMS
Distributed Management System

DNS
Domain Name System

DS
Differentiated Services

DSDV
Destination-Sequenced Distance-Vector

ECMP
Equal-Cost Multi Path

ECN
Explicit Congestion Notification

EIGRP
Enhanced Interior Gateway Routing Protocol

ERP
Enterprise Resource Planning

FTTH
Fiber to the Home

HMAC
Hash-message authentication code

HPC
High Performance Computing

HTTP
Hyper Text Transfer Protocol

HV
HyperVisor

IDE
Integrated Development Environment

IHU
I Heard You

I/O
Input Output

IP
Internet Protocol

IPC
Inter Process Communication

IS
Integrated Services

IS-IS
Intermediate System to Intermediate System

ISP
Internet Service Provider

LAN
Local Area Network

LLDP
Link Layer Discovery Protocol

MA
Management Agent

MAC
Media Access Control

MANO
Management and Orchestration

NAT
Network Address Translation

NM-DMS
Network Management Distributed Management System

NIC
Network Interface Card

NFV
Network Functions Virtualization

NFVI
NFV Infrastructure

NFVO
NFV Orchestrator

NVGRE
Network Virtualization using Generic Routing Encapsulation

OSPF
Open Shortest Path First

P2P
Peer to Peer

PaaS
Platform as a Service

PBB
Provider Backbone Bridges

PDU
Protocol Data Unit

PHB
Per Hop Behaviour

PM
Pool Manager

QoS
Quality of Service

RB
Routing Bridges

RINA
Recursive InterNetwork Architecture

RSVP
Resource Reservation Protocol

RTT
Round Trip Time

SAN
Storage Area Network

SDDC
Software Defined Datacenter

SDN
Software Defined Networking

SF
Service Function

SFC
Service Function Chaining

SFF
Service Function Forwarder

SLAPOS
Simple Languate for Accounting and Provisioning Operating System

SPB
Shortest Path Bridging

SSL
Secure Sockets Layer

STP
Spanning Tree Protocol

STT
Stateless Tunneling Transport

TCP
Transmission Control Protocol

TLS
Transport Layer Security

TOS
Type of Service

ToR
Top of the Rack switch

TRILL
Transparent Interconnection of Lots of Links

UBM
Unified Business Model

UDP
User Datagram Protocol

UPNP
Universal Plug and Play

VIM
Virtual Infrastructure Manager

VLAN
Virtual Local Area Network

VLB
Valiant Load-Balancing

VLL
Virtual Leased Line

VM
Virtual Machine

VNF
Virtual Network Function

VNFC
Virtual Network Function Component

VNFM
Virtual Network function Manager

VPN
Virtual Private Network

VTEP
Virtual Tunnel EndPoint

VXLAN
Virtual eXtensible LAN

WAN
Widre Area Network

XML
eXtensible Markup Language

1. Introduction

This report defines the use cases that PRISTINE solutions will address. The report provides a detailed view of PRISTINE's reference scenarios (distributed cloud, datacentre networking and network service provider), describes a number of use cases within the reference scenarios and analyzes the requirements that need to be supported in order to successfully fulfill the different use cases.

1.1. Methodology

PRISTINE researchers have followed the same methodology to analyze each of the three scenarios considered by the project. The first task has been to carry out a detailed study of how the scenario is addressed today using the currently available technologies. After that, an analysis of the main limitations of current technologies applied to the scenario is performed, identifying the main sources of those limitations. Finally, the last step is to analyze how to best apply RINA to solve the aforementioned limitations. Doing this exercise allows PRISTINE researchers: i) to understand the different DIFs present in the solution; ii) to study and describe the requirements that each DIF has to fulfill; iii) to analyze the role and requirements of the Network Management System and iv) to understand what other systems and tools are required for allowing RINA to interoperate with current technologies and applications - such as gateways, shim DIFs or RINA APIs exposed to applications.

1.2. Use cases overview

1.2.1. Distributed Cloud

The SlapOS (Simple Language for Accounting and Provisioning Operating System) is a decentralized Cloud Computing technology. It is designed to automate the deployment and configuration of applications in a heterogeneous environment, and it relies on servers located in people's home and now also in offices, data centers or even a smartphone, tablet or TV. One of its main applications is to create a disaster recovery cloud which can resist any force majeure event (ex. war, terrorism, political instability, software bug) which does affect traditional clouds from time to time. It is also much cheaper and environmentally friendly. SlapOS is based on a Master and Slave design. Slave nodes request to the Master node which software they should install, which software they should run and report to the Master node how much resources each running software has been using for a certain period of time. The Master node keeps track of available slave node capacity and available software. The Master node also

acts as a Web portal and Web service so that end users and software bots can request software instances which are instantiated and run on Slave nodes. Master nodes are stateful while Slave nodes are stateless. More precisely, all information required to rebuild a Slave node is stored in the Master node. This may includes the URL of a backup service which keeps an online copy of data so that in case of failure of a Slave node, a replacement Slave node can be rebuilt with the same data. It is thus very important to make sure that the state data present in the Master node is well protected.

Right now, SlapOS relies on IPv6 in order to interconnect all nodes. Each node is allocated usually 100 global IPv6 addresses or more. Unfortunately, all IPv6 providers that have been tried were unable to provide reliable connectivity (providers from France, Germany, Japan, Norway). For example, in France among 200 IPv6 addresses provided by a Freebox (Free ISP), 3 becomes unreachable from time to time, during a couple of minutes or hours. OVH routers sometimes no longer route packets to Free, but only for IPv6, during a couple of hours. Telia routers sometimes “eat” a few bytes during the initialization of a session. Overall, the use of native IPv6 of ISPs lead to a service availability of 99% or worse which is not convenient for a distributed cloud system. This has led to build an overlay communicating system that is in charge of building dynamically a mesh network between different participants. The mesh network is in charge of finding routes that must be secure and meet the QoS requirements. The overlay mesh network is also in charge of the routing of the packets as well.

re6st is an open source tunnel generator created to build a mesh of tunnels over IPv4 networks. Each node for example starts tunnels to 10 other nodes. Once the tunnels are set, distance vector routing is used on each node to guarantee connectivity. We use babel at this point as our routing protocol. Every day, 3 tunnels out of 10 are replaced with other tunnels suggested by neighboring nodes. The process of creating initial tunnels and replacing them based on neighbor suggestion is very similar to the process of P2P systems such as Bittorrent. Overall, re6st can be used for many applications: network resiliency, bandwidth aggregation, latency optimization, etc. It is used by the government of Ivory Coast for its sovereign Cloud Computing platform. PRISTINE will develop an alternative strategy to re6st. With the RINA architecture, the cloud participants are seen as application processes which use different DIFs to communicate together. The security policy to follow as well as the required QoS policy are examples of the used DIFs for the distributed cloud system that are of great importance. In fact, the system must be able to provide high security level to prevent one participant in the mesh to act against other participants. Moreover, the data flows between the participants should respect a certain level of QoS (loss, delay).

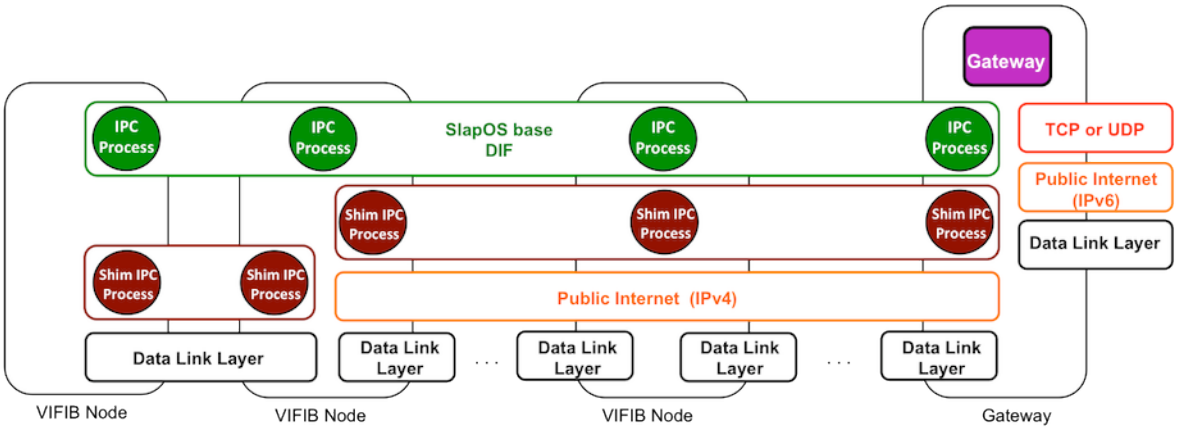

Figure 1. Replacing the re6net overlay with a DIF

Figure 1 illustrates the most direct way of using RINA in VIFIB's environment: replacing the re6net overlay by a DIF (called Slap-OS based DIF in the picture). This DIF has to be able to operate over IPv4 as well as directly over Ethernet environments, allowing all the VIFIB nodes distributed through the world to be connected together forming a single resource pool. The policies of the DIF could be directly reflecting the current configuration of the re6net overlay (hop-by-hop encryption, random and resilient connectivity graph, distance-vector routing taking delay into account), augmented with other functions such as congestion avoidance or enhanced resource allocation. Alternative policies could also be considered, allowing a comparison with the re6net base case.

1.2.2. Datacentre (DC) Networking

DC networks today are moving away from a tiered, hierarchical structure to a structure that resembles more and more that of a supercomputer; where all the nodes can be interconnected amongst them with high bandwidth and minimum delay. However, supercomputers are a far more controlled, static an homogeneous environment than a DC: usually external access is not allowed, there's one single tenant that executes HPC applications on behalf of users and all the systems are at one single location. In contrast, DCs are multi-tenant by nature, must support access from the outside (if providing a public cloud service) and the DC may be actually distributed within multiple locations. Moreover, the support for cloud computing demands flexibility, as applications with different networking requirements are dynamically instantiated and destroyed. These characteristics make DC networking a challenge, which is very complex to meet with the current technologies:

- Multi-tenancy demands strict flow isolation, both from a security and resource allocation point of view. TCP provides poor flow isolation, as by design flows compete for the same resources, interfering with each other. Security is complicated

since it is expressed in terms of IP addresses and ports, instead of application names (updating the rules is cumbersome in a changing environment, such as the DC one).

- The support of different applications with changing requirements implies the ability for the network to provide different levels of services, backed by different resource allocation techniques, which IP doesn't support. For the DC to make an efficient use of its resources and to support the high availability of applications, it is necessary to relocate running VMs to different physical machines, sometimes in another physical DC. The fact that IP doesn't easily support mobility complicates VM mobility a lot, usually restricting the movement of a VM within the same IP subnet.

Some separate solutions to the different issues have been proposed and deployed, such as DTCP to provide better flow isolation; or virtual networking to create L2 overlays on top of L3 networks (VXLAN, NVGRE, STT), thus allowing VMs to move. But all these solutions are add-ons that only address the issues partially, and further complicate the management of the DC (which in turn makes flexibility and dynamicity harder). In contrast, RINA provides a framework in which most of these problems are non-issues (access control rules are defined based on application names, congestion control and resource allocation techniques can be utilized to provide strong isolation between flows, mobility is inherently supported by the structure); therefore building upon RINA enables simpler, more efficient, easier to manage and more responsive DCs.

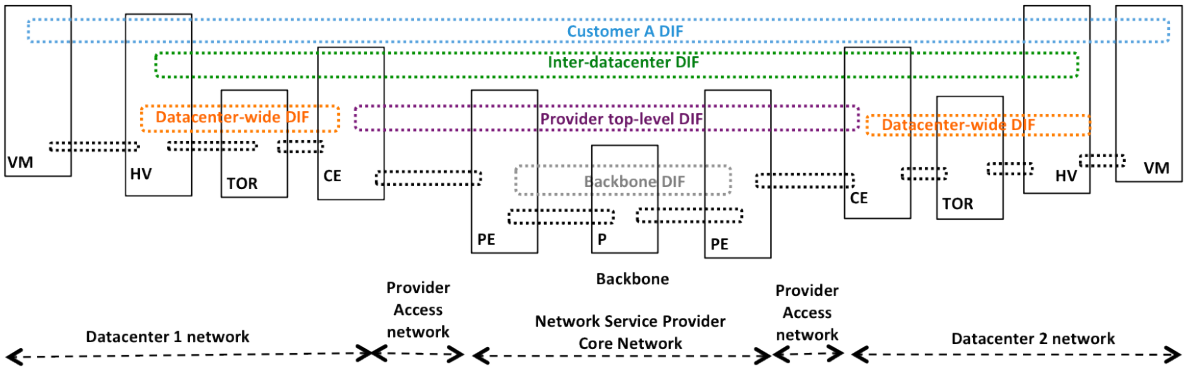

Figure 2. DC networking with RINA

Figure 2 shows how RINA can be applied to a distributed DC environment, where the DC owner provides IaaS services to customers, by instantiating several VMs in the two DCs and connecting them together. The DCs are interconnected by a provider network (it could also be through the Internet and the design wouldn't change). The following types of DIFs can be seen in the figure:

- DC-wide DIF. This DC "substrate DIF" enables the direct communication between all the physical systems within a single DC (Hypervisors, Storage, Top of Rack switches and border routers). No user applications or Virtual Machines can run

directly on top of this DIF, only other DIFs that directly support customers. Therefore VMs will not be even aware of the existence of this DIF, enhancing the security of the DC resources.

- Inter-DC DIF. If the same provider owns the two DCs, it is advantageous to create one or more inter-datacenter DIFs that connect together all the compute and storage resources at both DCs. This will simplify DC management, since customer DIFs will only need to be configured at the Hyper- visor and Storage machines, leaving the configuration of Top of Rack switches and border routers untouched when new applications are instantiated.
- Customer DIF. Dedicated to connect together two or more VMs dedicated to a single customer. The DIF essentially creates an individual security and resource allocation domain, separating the networking resources allocated to that customer from all the other ones. The DIF behavior can be highly tailored to optimally support each particular customer requirements (authentication, access control, data transfer, routing, etc).

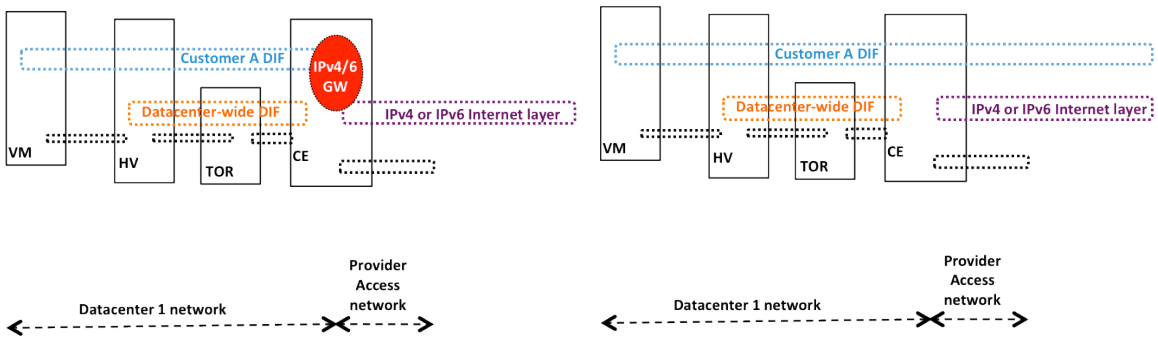

Figure 3. DC networking with RINA, (left) customer does not support RINA (right) customer supports RINA

Figure 3 illustrates how external client applications (e.g. browsers or mobile apps) can access the applications running in a customer VM, for example through the Internet. If the client application runs in a system that is not RINA aware, the client just uses regular IPv4 or IPv6 until the datacenter border router, which redirects the traffic to the right DIF through a gateway (RINA-IPv4/v6 gateway). If the client application runs in a system that is RINA aware it has two options: i) join the “customer A DIF” or ii) join another DIF on top of the “customer A DIF”, that further restricts the access to the customer A applications running in the DC - option ii) is not shown in the figure.

1.2.3. Network Service Provider

The goals of this scenario are to investigate and trial the benefits of the use of the RINA technology by a Network Service Provider (NSP), and to analyze RINA as an

essential component of the Network Functions Virtualization (NFV) concept within an operator network. It is obvious that a disruptive, clean-slate technology like RINA would have a difficult way within a NSP environment, very much oriented towards service provisioning and stability, and the phased incorporation of thoroughly tested technologies. But the recent advent of the NFV proposal opens a very interesting window of opportunity for demonstrating RINA in the NSP environment, and showcase the advantages that the combination of both approaches (NFV and RINA) can bring to network service design, management, and operation.

NFV implies a recursive structure in network service design: Virtual Network Function Components (VNFCs) are connected to build Virtual Network Functions (VNFs), that are in turn connected to build services. Since nothing prohibits that one or more of these services could be used to build VNFs or network services at a higher layer, recursiveness becomes an inherent property of VNF scenarios. Therefore the application of RINA to the construction of VNFs and virtualized network services seems promising.

Beyond the direct link in what relates to its recursive nature, NFV elements impose new requirements on their underlying network mechanisms in terms of security, resiliency and elasticity, as well as the need of a coherent abstraction layer supporting a unified interaction mechanism providing the base of a uniform VNF development and execution environment. The DIF model seems well suited for this.

It is worth noting that this use case focuses on the internals of NFV-based service provisioning. The new way of constructing services by function virtualization is the gist of NFV, without requiring any disruptive change in external service usage and operation. Focusing RINA applicability in these internals will translate into leveraging the NFV thrust, as well as providing an affordable acceptance threshold to actual deployments. An example of this approach can be seen in Figure 4, where "service-chain DIFs" (in yellow) are dedicated to support individual chains of services (blue DAF, distributed application) that process a subset of the operators traffic by applying a number of network functions in a certain order.

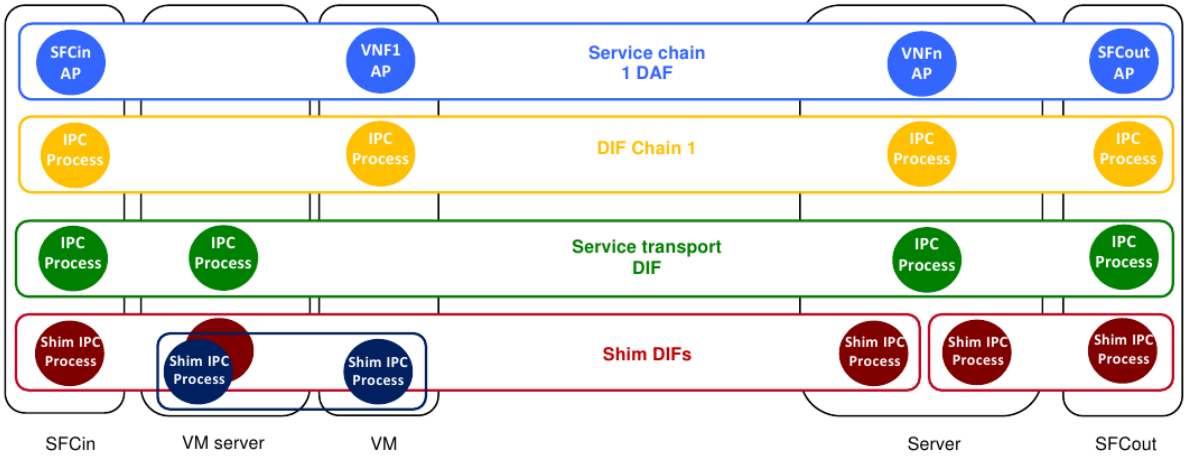

Figure 4. RINA supporting chains of virtual network functions

2. Distributed cloud

SlapOS is a decentralized cloud technology used to build a physically distributed cloud. Customer's applications are run in traditional datacenters, but also in servers from offices and home users. SlapOS is in charge of managing the overall cloud from a logically centralized location: the SlapOS master (a distributed approach is currently under development). The SlapOS master controls the different computers running SlapOS slaves. In terms of networking, the master and the nodes at different locations are interconnected through multiple IPv6 providers. In order to guarantee a high reliability (99.999%), SlapOS uses an overlay called re6st, which creates a mesh network of OpenVPN tunnels on top of several IPv6 providers and uses the Babel protocol for choosing the best routes between nodes. PRISTINE will provide an alternative to the re6st overlay, by using RINA on top of IPv6.

2.1. Introduction and Motivation

VIFIB[1] is a decentralized cloud system, also known as resilient computing. It consists of computers that are located in people's home, in offices, etc. By hosting PCs in many different places and by copying each associated database in at least three different distant sites, the probability of mass destruction of the whole infrastructure becomes extremely low.

On each computer VIFIB allocates 100 IPv6 addresses and 100 IPv4 addresses. Each service running in the computer is attached to a dedicated IPv4 address (ex. 10.0.42.124), as well as a global IPv6 address (ex. 2001:67c:1254:6::1/64). All services are interconnected across PCs using "tunnels" (stunnel software[2]) that redirect local IPv4 to global IPv6, encrypt flows and redirect IPv6 to IPv4. This way, two services running in different homes, compatible or not with IPv6, can be interconnected through a secure link that also provides mutual authentication through TLS X509 certificates. Even insecure services such as *memcached* can be deployed over insecure networks through this approach, as if they were deployed in a local area network.

Initial VIFIB was using native IPv6 of providers (Free, OVH, NTT, etc.). However, this appeared to be unreliable for various reasons. Therefore, VIFIB created an overlay network: re6stnet[3]. re6stnet provides reliable IPv6 over randomly generated IPv4 "tap" tunnels based on openvpn. Routing between is provided by babeld[4] ++ distance

[1] http://www.vifib.com
[2] https://www.stunnel.org/index.html
[3] http://git.erp5.org/gitweb/re6stnet.git/blob/HEAD:/README?js=1
[4] http://www.pps.univ-paris-diderot.fr/~jch/software/babel/

vector protocol. Overall, VIFIB is quite happy now with this solution: all local services are interconnected between computers; babeld routing makes the interconnection quite reliable; in case of failure of one link (ex. Orange FTTH) traffic is redirected on another link (ex. SFR FTTH). VIFIB customers are happy too. Its services in China can communicate with VIFIB servers in Japan or Europe. Services in different homes can interconnect and authenticate each other.

Tunnels between computers change every 5 minutes: some are trashed and replaced. babeld updates about every minute the routing table. Overall, the probability of losing IPv6 connectivity from China to France for a VIFIB customer is very low. This creates an excellent situation to host for example an ERP in France, an HTTP reverse proxy in China, and make sure the HTTP reverse proxy in China can always access the ERP in France.

There is however one big problem: if any of VIFIB hosts' is attacked, then the attacker can broadcast in re6stnet "wrong" babeld routing information and propagate wrong routes that will lead to the disconnection of services. Babeld recently added a latency based "distance vector" that partly solves this problem. But in reality, this problem is severe. There is no way to guarantee that what the routing protocol says is what really happens. The problem is even worse if VIFIB is opened to the public and let people add their PC to re6stnet mesh. Anyone can then join re6stnet and put it down by broadcasting wrong babeld protocol information.

IPv6 is mostly used as an overlay with good interoperability (an AS hosts VIFIB's border gateways so that re6stnet IPv6 hosts can access any IPv6 hosts in the world). It would thus be possible to replace IPv6 with RINA. The problem to solve is actually to create a public overaly network where every node helps other nodes to find the best possible route to interconnect two services on two VIFIB nodes. The system should be able to detect and isolate nodes that have been compromised.

2.2. Detailed description

2.2.1. SlapOS introduction

SlapOS is a commercial grade cloud computing solution that powers the VIFIB decentralized cloud system. Thanks to its distributed nature, SlapOS can provide higher resiliency and higher privacy than Cloud Computing solutions hosted in datacenters. Another interesting aspect of SlapOS is that it supports multiple modes of isolation:

- process based: provides the fastest I/O performance and the lowest deployment cost

- virtualization mode: provides higher isolation but lower I/O performance and adds costs for deployment
- ZeroVM mode: provides isolation nearly as high as virtualization and performance nearly as good as process

SlapOS includes a complete PaaS (Platform as a Service) solution and orchestrator. SlapOS PaaS (also known as WebRunner) provides full web based IDE that can run any complex tree of automatically configured services (database, web front end, application server, backup, monitor, etc.). A video that demonstrates WebRunner can be found here[5].

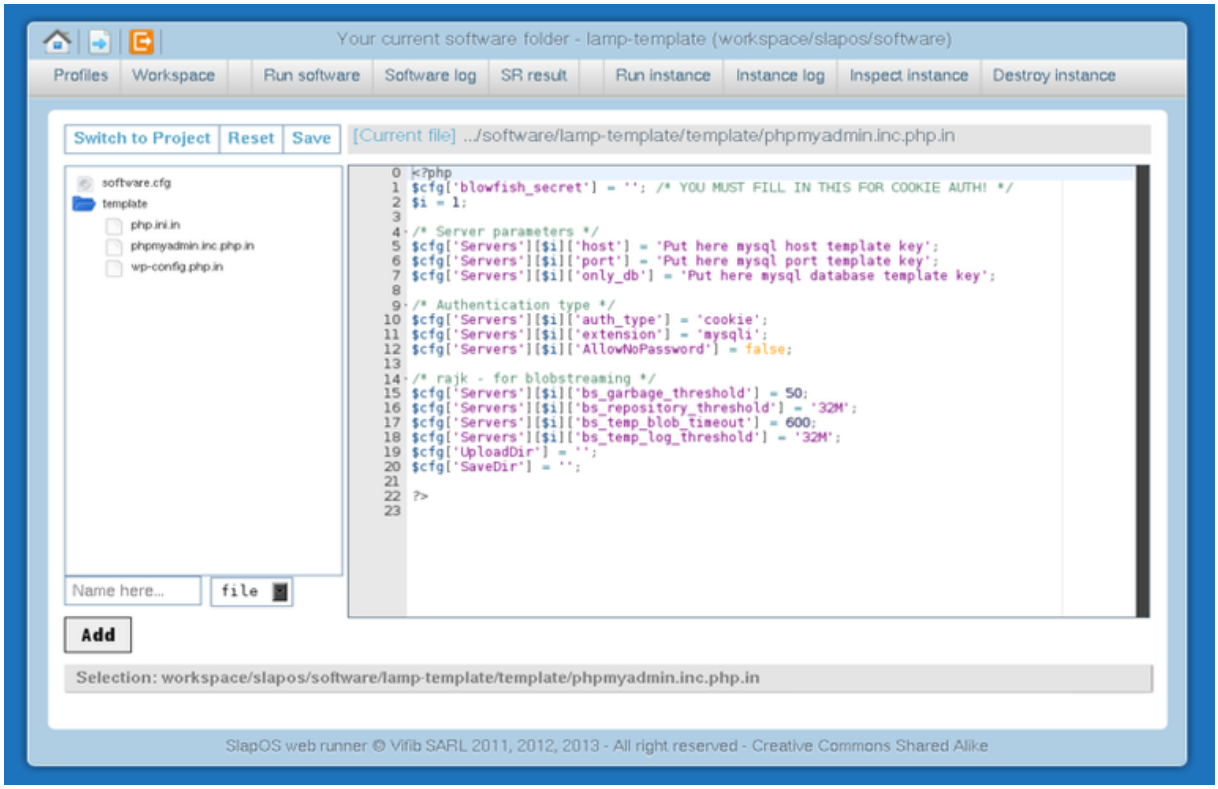

Figure 5. SlapOS Webrunner

SlapOS was inspired by recent research in Grid Computing and in particular by BonjourGrid [abbes08] - [abbes10], a meta Desktop Grid middleware for the coordination of multiple instances of Desktop Grid middleware. It is based on the moto that ”everything is a process”. SlapOS is now an OW2 project. The Figure below shows the current architecture.

5 https://www.youtube.com/watch?v=nmEBGqGNab8

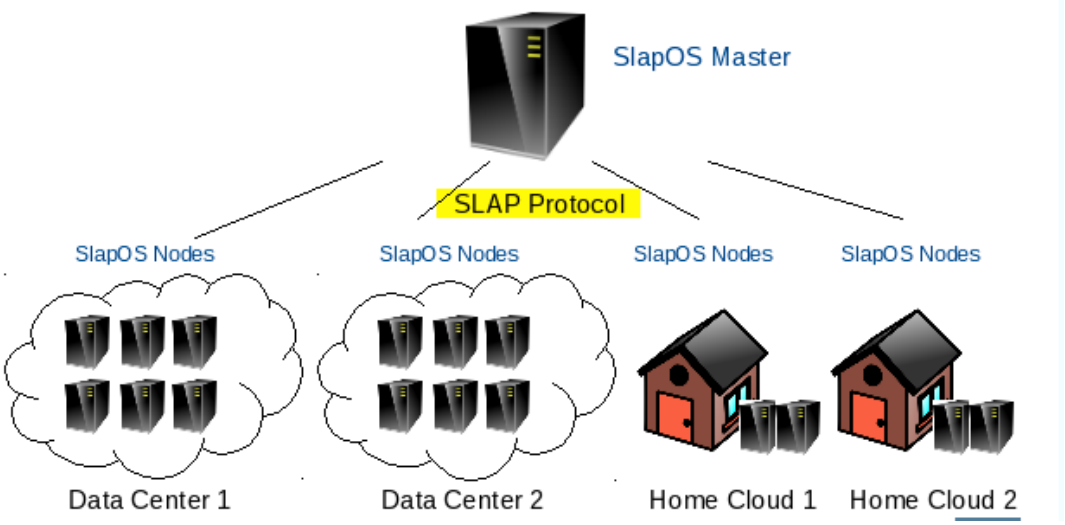

Figure 6. SlapOS architecture

SlapOS defines two types of servers: SlapOS Nodes and SlapOS Master. SlapOS Nodes can be installed inside data centers or at home. Their role is to install software and run processes. SlapOS Master acts as a central directory of all SlapOS Nodes, knowing where each SlapOS Node is located and which software can be installed on each node. The role of SlapOS Master is to allocate processes to SlapOS Nodes.

SlapOS Nodes and SlapOS Master exchange are interconnected through the HTTP and XML based SLAP (Simple Language for Accounting and Provisioning) protocol. SlapOS Master sends to each SlapOS Node a description of which software should be installed and executed. Each SlapOS Node sends to SlapOS Master a description of how much resources were used during a given period of time for accounting and billing purpose.

Current implementation of SlapOS

SlapOS is implemented as an extension of widely adopted open source software: GNU/Linux, Buildout and Supervisord. The client requirements are communicated via a Buildout profile. Buildout - a build system for creating, assembling and deploying applications from multiple parts - can be used to build software written in different languages on a variety of Operating Systems. Supervisord is a system that controls processes on UNIX-like operating systems. SlapOS builds on these base elements introducing the Slapgrid daemon, which implements the SLAP protocol on each SlapOS Node. Slapgrid is the component that handles the requests from the SlapOS master to install new software on the SlapOS Node on behalf of the client. Master node requests are processed following these steps:

- Slapgrid passes the buildout profile to a buildout processor, which creates all the required configuration files.

- Slapgrid then invokes supervisord, which triggers the invocation of all the processes required to implement the customer service.

As shown in the Figure below, after some time a typical SlapOS Node will include multiple software applications. Each software application will be instantiated multiple times - for multiple clients, for example - each of these instances running in a different OS process. For example, both Mediawiki and OS Commerce could be installed onto the same SlapOS Node, with six instances of each being run as processes. SlapOS follows the strategy of instantiating different software in different processes instead of dedicating full Virtual Machines to separate software instances. This approach allows SlapOS to use hardware resources and RAM in particular more efficiently.

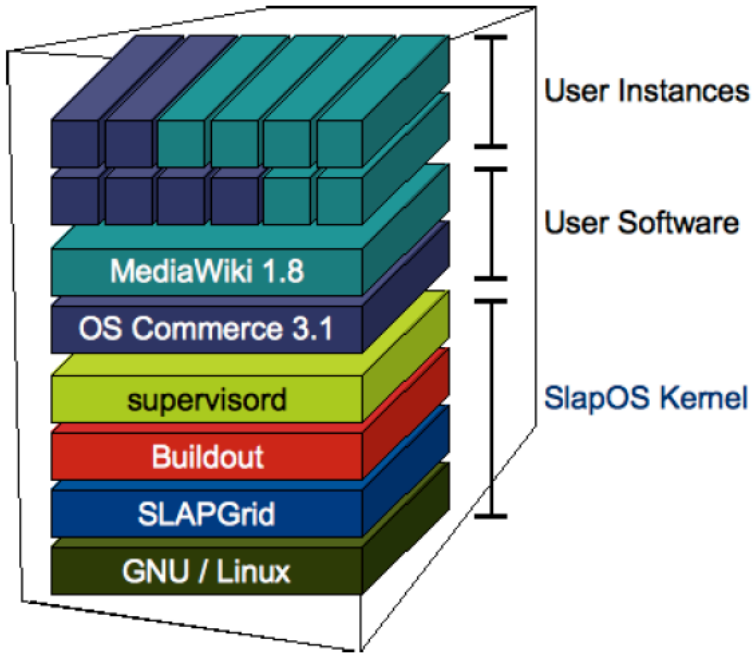

Figure 7. SlapOS node

SlapOS Master runs ERP5 Cloud Engine, a version of ERP5 open source ERP capable of allocating processes in relation with accounting and billing rules. The role of the Master node is to centrally manage the nodes in the SlapOS Grid, using the ERP5 Cloud engine to perform the following functions:

- Maintain an up-to-date inventory of the SlapOS nodes: state, availability, resource usage, etc.
- Maintain the inventory of software that can be installed on the SlapOS nodes, as well as the recipes (buildout profiles) to install each software component.

- Handle customer requests by i) deciding which software has to be instantiated in which SlapOS node and ii) requesting each SlapOS node the installation of one or more software components.

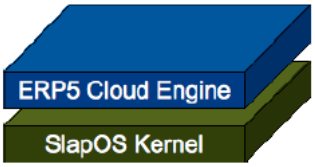

Figure 8. SlapOS master

Accounting model

The accounting model of SlapOS is an application of ERP5 Unified Business Model (UBM) [smets]. ERP5 defines 5 concepts which match with any business problem: Resource, Movement, Item, Node and Path. The ERP5 model is based on the idea that everything in business is a matter of flow, stock, traceability and planning of future flows. In the case of SlapOS, **Nodes** are the people and organisations who register to the SlapOS portal. Once they are registered, they can request any type of resource. They can also add to a SlapOS global Cloud their own servers in order to contribute to total amount of processing resources. SlapOS **Resources** are the Software Products such as MySQL, KVM, Xwiki, etc. which people and organisations can subscribe to and for which they will be later invoiced.

Movements represent the billing information such as the amount of GHz, the amount of storage or the number of users which have been used over a period of time by each process in the SlapOS cloud. The core part of SlapOS: traceability. In order to know which version of which software is being run on which computer and IP address for who and with which parameters, SlapOS defines 5 types of UBM **Item**:

- Computer. A Computer Item is created each time a server is added to the SlapOS cloud.
- Computer Partition. Each Computer Item is divided into a given number of so-called Computer Partition. The number of independent services (Software Products) which can be instantiated on a given Computer is at most equal to the number Computer Partitions of the Computer. Computer Partitions are usually implemented as subdirectories of the Computer. On an IPv6 network, each Computer Partition can be associated to a different IPv6 address. On an IPv4 network, each Computer Partition can be associated to a different port range for the same IPv4 address. Some Computer Partitions can also be associated to a tap virtual ethernet interface which is then bridged and used by virtual machines.

- A Software Release Item defines precisely how to build a Software and install it on a Computer. Currently, SlapOS uses buildout profiles to define a Software Release. Multiple Software Release can exist for the same Software Product.
- A Software Instance Item contains all parameters to configure an instance of Software Release. It also defines the type of service to run from Software Release.
- The Subscription Item is used to group all accounting information for a given consistent set of Software Instances. It is also used to generate daily, weekly, monthly or yearly subscription billing.

Nodes, services and networking between services

Node definition

Every SlapOS node is made usually of:

- a single CPU (ex. Intel i7)
- a single SSD drive (Intel or Samsung, 120 GB to 1 TB)
- 16 GB to 32 GB RAM
- single or dual Gb Ethernet
- a GNU/Linux distribution (Debian, OpenSuSE, etc.)
- a single or dual IPv4 address, that is usually non-routable (ex. 192.168.234.10)
- Internet access through FTTH (50 Mbps upload or better)

Allocation of services

SlapOS can allocate services (software instances) on nodes (computer partitions). The typical services deployed in the VIFIB infrastructure are:

- databases
- web servers
- application servers
- "kvm" processes (very few)
- file conversion servers
- webrunners (a kind of PaaS made of a webserver and some CGI)
- reverse proxies

A service itself can allocate another service. Services form a forest of trees. Each tree consists of a root service and child services.

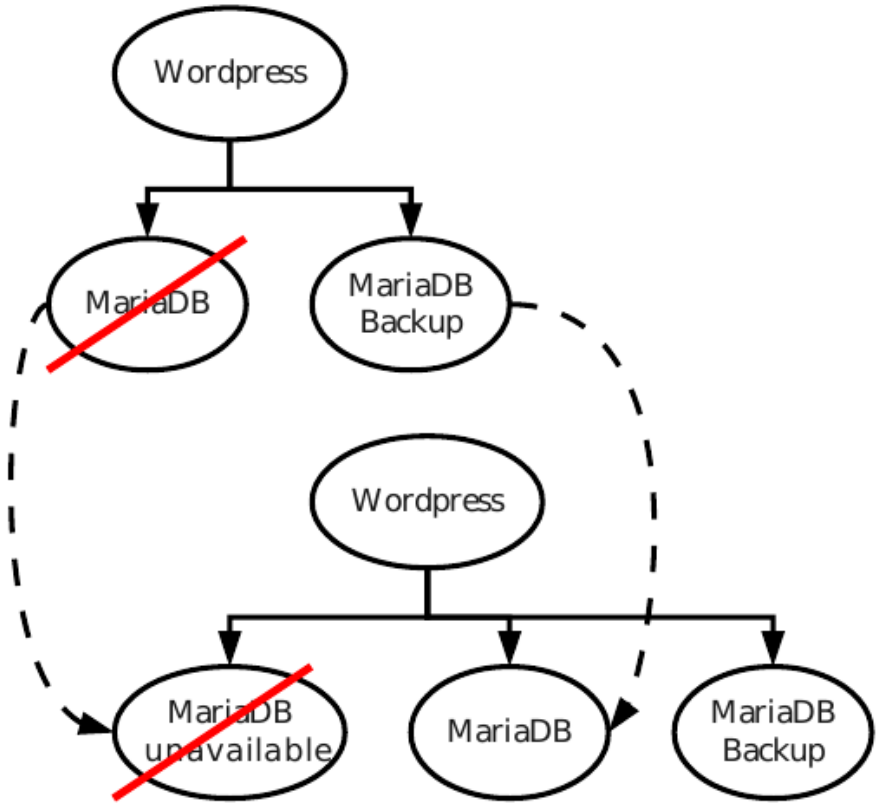

Figure 9. Service tree

SlapOS is itself a service that can be allocated by SlapOS. A certain form of recursivity thus exists. Certain services are called "slave services". This is used for sharing resources among different services. A typical example of slave service is a CDN that shares a single resource (public IPv4 address) among different services (front-end for multiple web domains). Overall, SlapOS services are structured with 3 relations:

- instance of (a SlapOS master service)
- child of (any service)
- slave of (any service that supports slaves)

Each service can access any other service information (e.g. address) of another service that shares a common parent. All services of the same tree must be able to communicate with each other.

Communication between services

In current SlapOS, each service is attached:

- one public IPv6 address
- one loopback IPv4 address

Communication between services can either rely on

- direct connection between IPv6 addresses
- redirection of IPv4 loopbacks to IPv6 using stunnel[6].

After startup each VIFIB PC gets allocated 100 IPv6 addresses and 100 IPv4 addresses. Each service is attached to a dedicated IPv4 address (ex. 10.0.42.124). Then each service is also attached to a global IPv6 address (ex. 2001:67c:1254:6::1/64). All services are interconnected across PCs either directly through IPv6 - if the services are in the same VLAN - or using "tunnels" (stunnel software) that redirects local IPv4 to global IPv6, encrypts flows and redirects IPv6 to IPv4. This way, two services running in different homes, compatible or not with IPv6, can be interconnected through a secure link that also provides mutual authentication through TLS X509 certificates. Even insecure services such as memcached can be deployed over insecure networks through this approach, as if it were a local area network.

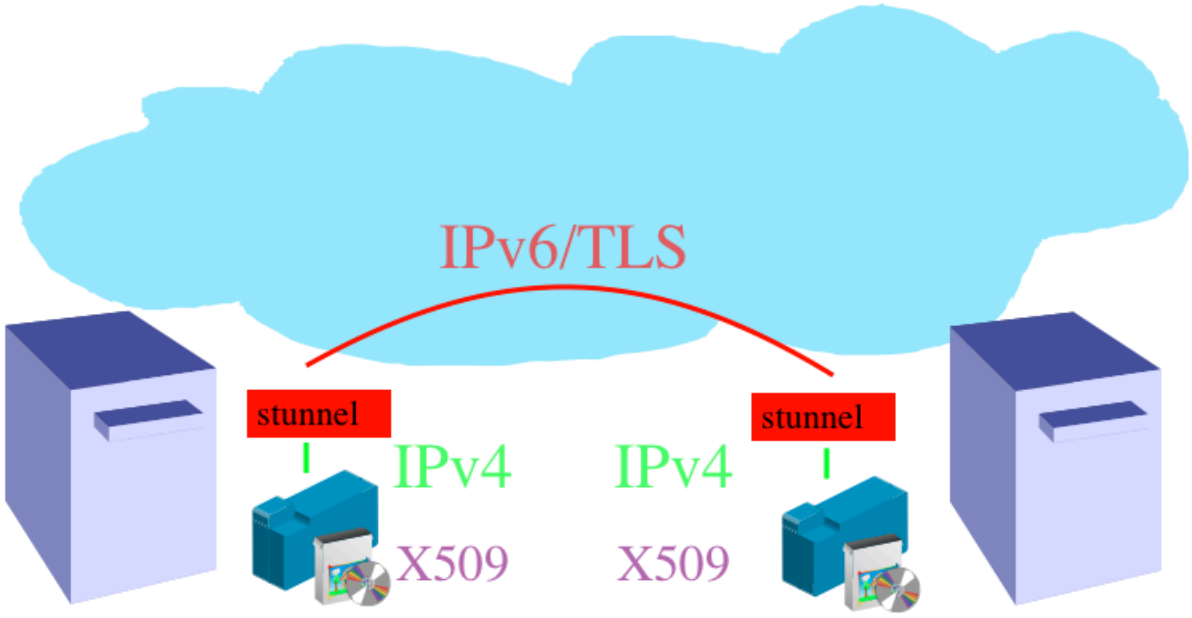

Figure 10. Use of stunnel

2.2.2. Detailed description of the res6net overlay

Motivation and introduction

The initial VIFIB infrastructure was using native IPv6 from different providers (Free, OVH, NTT, etc.). However, this appeared to be unreliable for various reasons. Many

[6] https://www.stunnel.org/index.html

houses have no access to it and native IPv6 networks su#er from a lot of problems. Using native IPv6 in houses is impossible for any professional usage. The common workaround is to create tunnels from each peer to a server with a good IPv6 connection. This solution doesn't scale well and is not resilient at all. It also requires to have multiple servers around the world to reduce the latency overhead. The solution of VIFIB was to build a scalable, decentralized and resilient peer-to-peer (p2p) overlay network by creating tunnels between peers to answer those problems. P2p overlay networks improve reliability of connections, even for IPv4 [spring]. Since the routing tables of the overlay are under the control of VIFIB -and not the ISPs-, the overlay can recover faster from a link failure than BGP or other algorithms used by Internet providers. This also improves latency : because of routing policies, the path chosen is not always the shortest and the triangular inequality is not respected among peers. Using a detour route reduces latency [hoffman].

Therefore, VIFIB created an overaly network: re6stnet. re6stnet provides reliable IPv6 over randomly generated IPv4 "tap" tunnels based on openvpn. Routing between is provided by babeld ++ distance vector protocol. All local services are interconnected between PCs; babeld routing makes the interconnection quite reliable; in case of failure of one link (ex. Orange FTTH) the traffic is redirected on another link (ex. SFR FTTH). VIFB customers are happy too. VIFIB services in China can communicate with servers in Japan or Europe. Services in different homes can interconnect and authenticate each other.

Structure

The following Figures provide an overview of the structure of the re6net overlay. IPv6 connectivity between nodes can be provided i) through Ethernet if two nodes share the same LAN; ii) through a VPN tunnel (OpenVPN [openvpn]) that simulates an Ethernet LAN between nodes and iii) through a sequence of two previous cases thanks to the Babel routing protocol.

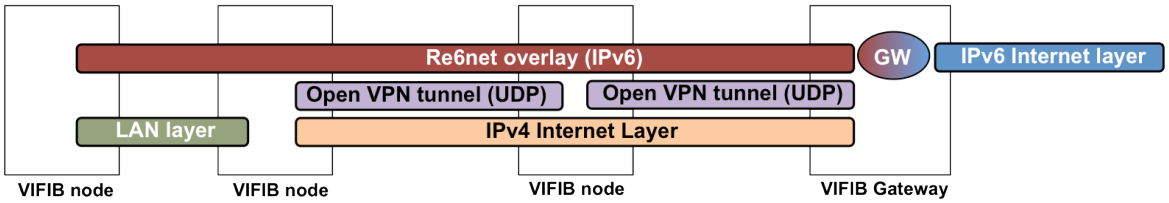

Figure 11. Schematic view of the re6net overlay (I)

There are four types of systems in the VIFIB overlay: regular nodes running customer's services (as introduced in the previous section); gateways to the IPv6 Internet; reverse proxies for web services (HTTP and HTTPS) that allow IPv4 users to access the services

deployed in the overlay; and the registry, which provides the initial point of contact for nodes joining the overlay (as described in the enrollments section).

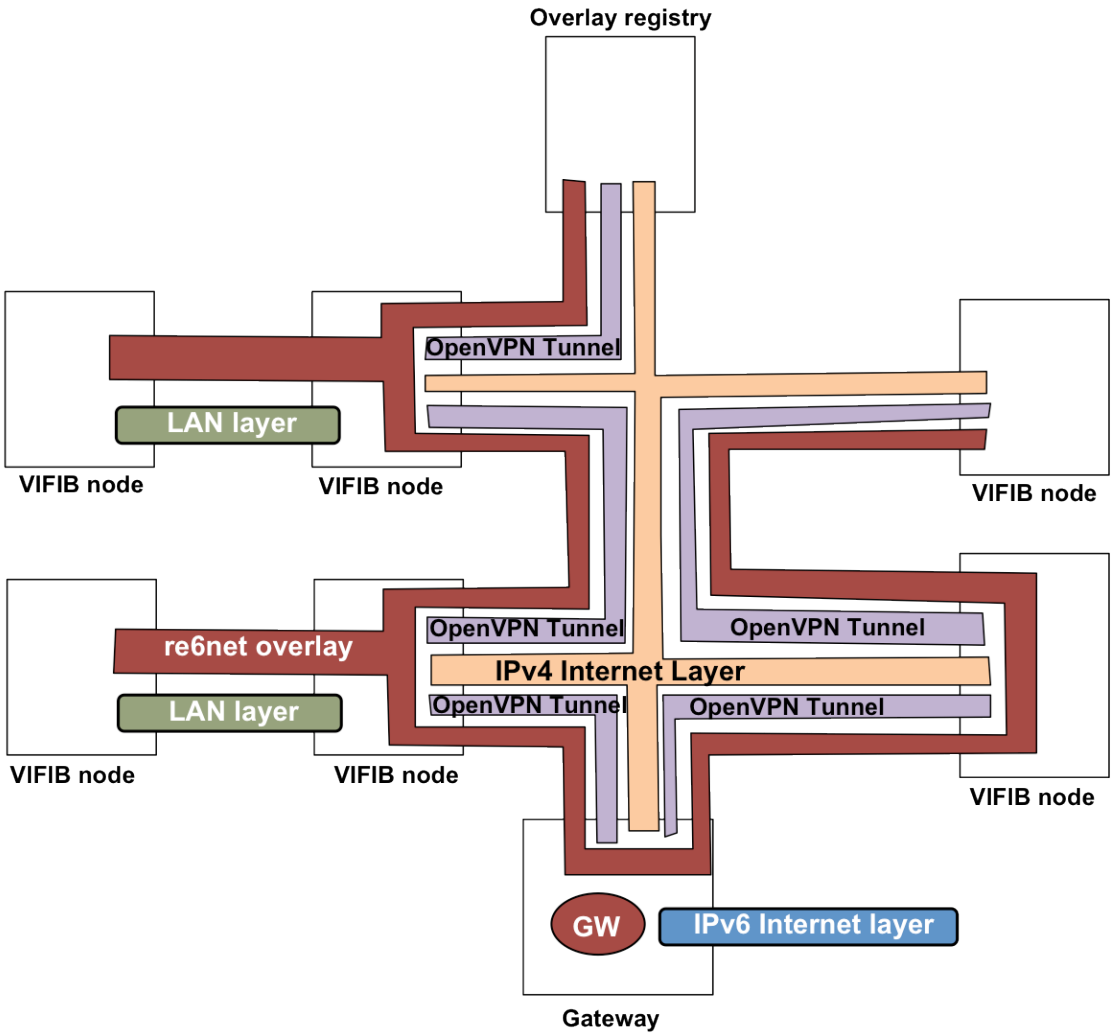

Figure 12. Schematic view of the re6net overlay (II)

Joining the overlay (enrollmnent), and the registry

R6net nodes join the overlay with some loose rules and no previous knowledge about its connectivity graph. To join the network a node has to contact the registry in order to get an SSL certificate (required to create the OpenVPN tunnels), the static IPv4 address where the OpenVPN server of the node has to be listening to and a random list of peers with its contact information (IPv4 address and UDP port of the OpenVPN server on that peers).

In order to keep the list of peers up to date, each network node contacts the registry twice a day, renewing the list of peers. The registry keeps a cache of the network nodes that have contacted it recently, so that nodes that are not active can be removed from the cache (they are assumed to be down).

The only information stored on each node is a partial list of other peers of constant size and no information has to be exchanged between peers. They only need to contact the registry a few times a day. Thus, the overhead generated by the maintenance of the network is very small and the network is scalable. Except for the registry, the network is decentralized. The registry can fail for a short time since every node has a local DB and can survive alone as long as it is not empty. If the registry is not reachable from a node (for example if it is censored in a particular country), the local DB can be initially filled by hand and then the registry reached through the overlay network. This makes the network resilient.

Re6net uses the registry as a central point to exchange informations. If it is down for a long time, peers cannot discover each others anymore. Its neighbor might also have to forward a lot of traffic if the network grows too big. Thus, the registry should only be used when a peer knows no other peers (for example when it first enter the network) so there is no central point of failure and the network will scale better.

To allow others to connect to it, a node randomly chooses a few addresses belonging to the network in its routing table and sends them the data required to connect to him (external IPv4 address and port) so the receivers can store them in their local DB. A new node has to send its reachability information to a number of peers equal to the size of the local DB so it has as many chances as other to be chosen for establishing a tunnel.

Address assignment

The address assignment procedure is automated as much as possible. In order to get IPv6 addresses, the following approaches are defined:

- Approach 1 (re6st): the registry allocates a subrange of IPv6 addresses to VIFIB nodes.
- Approach 2 (native IPv6): by requesting many times "one more address" and making sure it does not conflict.

For IPv4 address, the procedure is the following:

- Approach 1 (local, loopback addresses): generated randomly.
- Approach 2 (public, routable): Manual assignment of IPv4 on machine or on router, and if needed NAT setup on router.

Connectivity graph

P2p networks are not merely robust, they have been proven to be practically indestructible: As long as the peers remain connected, the network services can be provided. This leads to a crucial and fundamental problem of connectivity in p2p overlay networks [mahlmann]. If a group of peers loses contact to the rest of the network because the connecting peers leave the network, then the network is partitioned. Without outside information like from the users or from a central server it is impossible to rejoin a once broken network. Clearly, increasing the interconnectivity, i.e. the number of peers a node knows, alleviates the danger. Yet, it comes with the problem to continuously check a large number of neighbors and it is not clear whether the implied graph structure is robust under churn.

The re6net overlay organizes nodes in a flat random graph, using its own algorithm. The goal of this algorithm is to construct a robust network structure with a small diameter, in order to minimize the latency between nodes. Since every node can host a limited number of tunnels, VIFIB only considered algorithms based on k-regular undirected graphs [bollobas] - where k is the number of tunnels that each peer creates. Since re6net relies on peers it doesn't control, the overlay has to assume they may fail or leave the network at any moment [gribble]. An attacker might also try to kill selected peers to damage the network. Thus resiliency must be considered in order to avoid losing connectivity. A way to measure resilience is to count the number of nodes or edges that must be removed to disconnect two nodes in the graph. Since each node has k neighbors, the optimum is to have k paths with no common nodes except the first and the last one between each pair of peers. Modified De Bruijn graphs can achieve these constraints without altering their almost optimal average distance and diameter [rai], [ngo].

Having a complete view of the network on each peer is impossible for a scalable network. Peers might often leave and enter it. Broadcasting information about their status each time would saturate the links. Structured overlay network, used in distributed hash tables like Chord or Kademlia [mazieres], [lopumo], are able to build a network with O(ln(n)) information on each node. But this still require some overhead and links are established according to a fixed topology that doesn't take into account the underlying network parameters. This is acceptable when the overlay does not consider latency but in VIFIB's case it might lead to bad connectivity graphs. Since no pertinent information can be collected without too much overhead, re6net uses random graphs to generate the connectivity graph of the network. The only information a peer needs is the external IPv4 and UDP port of a few randomly chosen peers to be able to connect to them.

Constructing the connectivity graph

Random k-regular graphs have good diameter [delavega] and resilience proprieties [bollobas]; however they are difficult to generate, requiring a lot of coordination between the overlay nodes to establish the tunnels. Instead of that, re6net generates a graph where the arity (number of tunnels) of a node is k and close to 2*k with high probability. Each peer relies on a local list of possible nodes to connect to (the local DB) to establish a tunnel to k randomly chosen peers. Since each peer creates k tunnels, a node has in average 2*k tunnels.

To avoid that two nodes both establish a tunnel to the other, a node remembers peers connected to it. If two nodes both establish a tunnel to the other at the same time, they compare their SSL certificate serial number to chose which tunnel to keep. If a tunnel is detected to be down, it is immediately replaced by a new randomly chosen one. Since a peer only cares about the tunnels it has established, no cooperation is needed among nodes. To include new peers into the network, each peer replaces a randomly chosen tunnel among the ones it has established every 300 seconds.

[beaugnon] provides a study on the properties of VIFIB's connectivity graph generation algorithm, taking into account several factors in function of k (number of tunnels that each node establishes) and n (number of nodes in the overlay): the probability that the network becomes partitioned, the average diameter of the overlay, its fault-tolerance and availability. This study led to the following conclusions:

- k=10 was chosen as a good compromise between achieving a reasonable resilience - probability of unreachability between 2 nodes is 8% with only 20% of the network alive - and network diameter (average distance between 2 nodes is 1.7 hops for n=100) and keeping a low minimizing the number of tunnels established by each node.
- To avoid having too many tunnels on the same node, a maximum number of tunnels in each node (arity) of $3 \cdot k$ was set. If a node tries to connect to a node which is already full, then the connection will be refused and the node will try to connect to someone else. This only affects a small number of peers (0.3% of nodes with n=1000), with almost negligible effects.

Tunnel replacement strategy to optimize the connectivity graph

Even if re6net has good performances, it can still be improved since some tunnels are almost useless and could be replaced by more useful ones. For example, if a node has a really bad connection no traffic will transit through it and it is useless for other to

connect to it. Tunnels established by the node itself are sufficient to ensure a good connectivity with the rest of the overlay.

It is difficult to choose the best tunnel to create without a lot of information about the network. But once the tunnel is established, it is possible to make local measurements on it to estimate its usefulness. Instead of randomly choosing a tunnel to replace every 5 minutes, the less useful one is chosen. After several iterations, only the most useful tunnels will remain, incrementally optimizing the overlay network. There are several ways to measure the usefulness of a tunnel:

- The most obvious way to measure a tunnel usefulness is to look at the amount of traffic going through it. This can be verified very easily through the kernel stats. Favoring such tunnels will favor the most used routes and tends to make faster connections between nodes that needs to communicate a lot to each other. This seems to be a good idea but it also raises some problems. Firstly, if a group of nodes communicate a lot together and never to the other nodes, they will end up being disconnected from the rest. Secondly, it brings some instabilities in the network connectivity graph, since the most used routes are not always the same.
- Another way is to try to keep tunnels with a good latency. Favoring those tunnels will avoid the ones with a high latency, but a node will tend to stay connected only to close peers and a group of machines that are well-connected will eventually end up being disconnected from the rest.
- Finally, a third way to evaluate the usefulness of a tunnel is to count the number of peers reached through it. This information is retrieved from the routing table. This is the measure VIFIB has used to choose which tunnel should be replaced at every iteration of the algorithm.

Replacing tunnels with the less peers reached tends to sort peers in two equally-sized categories : nodes with an almost maximal arity (30 tunnels) and nodes with an almost minimal arity (10 tunnels). Nodes with an almost maximal arity are the more central nodes in the overlay network. They allow peers connected to them reach a lot of other peers so others stay connected to them. On the other side, less central nodes have an almost minimal arity : others have dropped tunnels they had established to them. Since a high arity increases the centrality, central nodes tends to become even more central. This is why the two categories of peers are so well separated. Since each node still has at least 10 tunnels and that not only one or two but half of the peers are more important (30 tunnels), the connectivity graph of the network still provides good resiliency properties.

Many iterations have to be made before the average latency is almost optimal. Since each iteration generates some traffic to update routes, the tunnel replacement frequency cannot be increased too much. Almost optimal latency will be achieved when the network is stable, but when peers often leave and enter the overlay performances are decreased : the almost optimal connectivity graph has no time to be established. It still leads to better latency than direct connections but not as good that what is achievable with a stable network [beaugnon]. As an additional benefit, replacing only tunnels with the less peers reached also reduce the traffic generated by the routing algorithm since less routes are changed.

On the use of OpenVPN tunnels

The re6net implementation is based on a Python program to contact the registry and manage tunnels. The program launches an OpenVPN server and several OpenVPN clients [2] to establish the tunnels with the neighbours. It also launches the routing daemon. The script is able to automatically ask for a port forward to a router using the UPnP-IGD protocol. VIFIB's implementation also detects when some other peers are available in the LAN to avoid establishing tunnels with them and instead uses a direct connection.

re6net uses OpenVPN tunnels over UDP and the SSL/TLS authentication mode. In SSL/TLS mode, an SSL session is established with bidirectional authentication (i.e. each side of the connection must present its own certificate). If the SSL/TLS authentication succeeds, encryption/decryption and HMAC key source material is then randomly generated by OpenSSL's RAND_bytes function and exchanged over the SSL/TLS connection. Both sides of the connection contribute random source material. This mode never uses any key bidirectionally, so each peer has a distinct send HMAC, receive HMAC, packet encrypt, and packet decrypt key.

The encrypted packet is formatted as follows:

- HMAC(explicit IV, encrypted envelope)
- Explicit IV
- Encrypted Envelope

The plaintext of the encrypted envelope is formatted as follows:

- 64 bit sequence number
- Payload data, i.e. IP packet or Ethernet frame
- The HMAC and explicit IV are outside of the encrypted envelope.

The per-packet IV is randomized using a nonce-based PRNG that is initially seeded from the OpenSSL RAND_bytes function.

Because SSL/TLS is designed to operate over a reliable transport, OpenVPN provides a reliable transport layer on top of UDP (see the diagram below). OpenVPN multiplexes the SSL/TLS session used for authentication and key exchange with the actual encrypted tunnel data stream. OpenVPN provides the SSL/TLS connection with a reliable transport layer (as it is designed to operate over). The actual IP packets, after being encrypted and signed with an HMAC, are tunneled over UDP without any reliability layer.

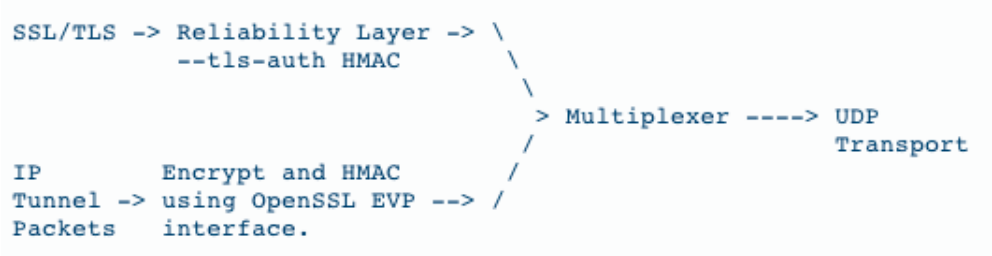

Figure 13. OpenVPN operation (I)

Routing between nodes of the overlay: the Babel protocol

Babel is intended to work in wireless mesh networks as well as in classical wired networks, and has been extended with support for overlay networks. It is a distance-vector routing protocol based on Bellman-Ford algorithm. It inherits some techniques from Destination-Sequenced Distance-Vector (DSDV) [perkins] and EIGRP [albrightson] to prevent count-to-infinity routing loops, and to quickly converge on loop free paths.

Even after a mobility event is detected, a Babel network usually remains loop-free. Babel then quickly reconverges to a configuration that preserves the loop-freedom and connectedness of the network, but is not necessarily optimal; in many cases, this operation requires no packet exchanges at all. Babel then slowly converges, in a time on the scale of minutes, to an optimal configuration. This is achieved by using sequenced routes, a technique pioneered by DSDV routing [perkins].

Neighbors are detected by the mean of Hello and IHU (I Heard You) messages exchange. Hello messages are basically used to declare itself to the immediate neighbors whereas IHU are used to confirm the bidirectionality of the link and contain the link quality as well.

Like classical distance vector routing protocols, routing information is exchanged only between neighbors by means of update messages. The information received in those

messages is checked against the Babel feasibility condition, which ensures that the route does not create a routing loop. If the feasibility condition is not satisfied, the update is either ignored or treated as a retraction, depending on some other conditions. If the feasibility condition is satisfied, then the update cannot possibly cause a routing loop, and the update is accepted.

The measure of distance in hops gives a good idea of how things are but some tunnels might have a much bigger latency than others. Computing the shortest path using latency is necessary to achieve good performances. Therefore re6net continuously measures the RTT of each tunnel, and this metric is used by the Babel daemon to compute the shortest path taking into account the latency constraint.

Accessing re6net from the IPv6 public Internet: the gateway

In order to communicate with the Internet through the IPv6 Internet VIFIB has requested a /48 IPv6 subnet so IPs given to nodes are routable on the Internet and can be accessed through a gateway; making the link between the real IPv6 Internet and the overlay network.

2.3. Issues and limitations

2.3.1. Naming and addressing complexity, renumbering

The network interface is addressed not the node. As IPv6 also assigns an IPv6 address to a network interface, then if an alternate network is used the network interface address also changes (even though the node and data stored on the node does not change).

This forces use of a higher level "identifiers" within the mesh to retain node identity. We have to relearn routing tables when the IP address changes, as the IP routing processes are unaware of these higher level "identifiers", the mesh network must maintain a set of "multi-homed" IP addresses for each node, and re-implement custom distance vector and cost functions to work out an optimal route.

2.3.2. Routing

There is currently no hierarchical routing in re6net. Because of this, the routing table size and the routing information exchanged between each neighbors is in O(n). Implementing a hierarchical routing is essential to allow re6net to scale since it would reduce the size of the routing table and the information exchanged to O(ln(n)).

Usually, subnets are assigned statically and manually at the creation of a network. But in re6net, when a node asks to join, there is no information about the node neither about

the connectivity graph of the network so it is impossible to choose in which subnet the new node should be attached to. Moreover, the overlay network might evolve in a very different way. Last but not least, since nodes from a single subnet must stay connected, having a static addressing would add a constraint and reduce resilience. Thus a dynamic addressing capable of following the evolutions of the network's connectivity graph and to recover from a failure is required.

Even if the addressing is dynamic, some services still need to have a static IP. To achieve this each peer must run a NAT and have both a static and a dynamic IP. A packet is sent to an other machine using its static IP. The NAT of the sender changes the static IP of the destination to its dynamic IPs and the NAT of the receiver changes it back to the static IP. This way, the overlay network only sees dynamic IPs and machines only see static IPs. The index necessary to store the indirection between static and dynamic IPs can be stored in a DHT with a cache on each peer. When a node changes its IP, it can keep both the old and the new one until the information has propagated.

Multiple algorithms and their implementations exist to automatically build a hierarchical mesh network [lopumo], but some work might be needed to adapt them to re6net since most of them have been developed for specific environments like ad hoc wireless networks. Every subnet could contain 100 machines or less. Inside a subnet the routing will be optimal and outside a subnet, a packet will be routed on the shortest path to enter to another subnet (and not on the shortest path to its destination). Since the routing will not be optimal, latency will be affected; however if the number of machines in a subnet is big enough, the effect should be small.

The parts of VIFIB's algorithm relying on the routing table (described before) will have to be adapted to work with a hierarchical routing. If for each subnet in its routing table, a peer knows the number of peers it contains, then it can count the number of peers accessed through each tunnel. It can also send a message to an uniformly chosen peer : it chooses a subnet (or peer) S in its routing table with a probability proportional to the number of peers inside S. If S is a machine, it only need to send the packet to it. If S is a subnet, it anycasts the packet to this subnet. When the packet is received by a node of S it is forwarded to a randomly chosen machine inside S using the same technique recursively.

The number of nodes in a subnet can be transmitted at the same time than routes. A node starts by advertising its own subnet saying there is one node in it. Then, whenever a node receives a route to a subnet, it will learn the number of nodes inside it. When a node aggregates some routes, it just has to add the number of nodes in the routes aggregated.

2.3.3. Security

VIFIB currently controls who is connecting to our network using OpenSSL certificates for OpenVPN. But if an evil peer manages to get a certificate, it may damage the network. If any of VIFIB's hosts is attacked, then one can broadcast in re6stnet "wrong" babeld routing information and propagates wrong routes that will result in the disconnection of services between 2 nodes. The problem is even worse if VIFIB is opened to the public and people can add their PC to the re6stnet mesh. Anyone can then join re6stnet and put it down by broadcasting wrong babeld protocol information. Some security issues have to be solved to answer this problem.

The first thing necessary is the possibility to revoke some certificates. This could be achieved by storing the revoked certificates in a server contacted once a day by peers to update their list of revoked certificates (registry). An algorithm to detect peers behaving badly (for example a node refusing to forward traffic) would also be necessary.

re6net also needs to ensure that external nodes cannot send false information about entry-points inside the network using a peer with a badly configured firewall or that a single node cannot advertise many false entry-points. This can be done by signing every message sent for entry-point advertising. This way, re6net can check that the message was sent by someone with a certificate granted by the registry and that it only advertises one entry-point.

2.3.4. Isolation of service trees

Not necessarily a limitation of the overlay, since it could be addressed on top, but a limitation of VIFIB's service. Compared to current cloud computing offerings, there are some limitations in the service provided by VIFIB to customers when looking at how services are interconnected. Services belonging to the same tree have to be able to connect to each other, but it would be desirable to isolate these services from the other trees and - moreover - offer them a customized networking environment (where routing, data transfer, authentication, encryption etc. can be programmed). Currently VIFIB can provide authentication and encryption between services through the use of Stunnel, but cannot go beyond that.

It is also hard to segregate traffic from different customers on the "mesh" network. This leads to potential congestion issues as one customer floods the network with data, and this impacts other customers. This problem is caused by the lack of a distinct enrollment process for IP connectivity. All nodes can send traffic without requesting explicit permission to do so.

2.3.5. Maturity of IPv6 deployment

IPv6 is not implemented well by all ISP providers. For example, certain IPv6 addresses become unusable after a certain amount of time, etc.

2.4. Applying RINA to the use case: requirements analysis

2.4.1. Overview

RINA as a direct replacement for the overlay layer

The first immediate application of RINA to the distributed cloud use case comes with the replacement of the re6net overlay (babel daemons + openVPN tunnels) with a RINA DIF, as seen in the following Figure. We have called this DIF the SlapOS base DIF (SOS-DIF), since it provides connectivity to all the VIFIB nodes, the registry(ies) and the gateway(s). The characteristics of this DIF should be similar to those of the re6net overlay improving it with extra functionality as explained in the requirements analysis section below.

The gateway will have to be adapted to become a RINA-IPv6 gateway: it must terminate TCP/UDP flows on the IPv6 Internet side and map them to RINA flows in the "SlapOS base DIF", and vice-versa. The globally routable IPv6 address associated to a service will be used as its application name.

Services will be able to interact with other services and external users in the Internet via the faux sockets API - therefore they don't require modifications. This faux-sockets API maps the calls to the sockets API to calls to the native RINA API, using IP addresses as application names in the RINA DIF.

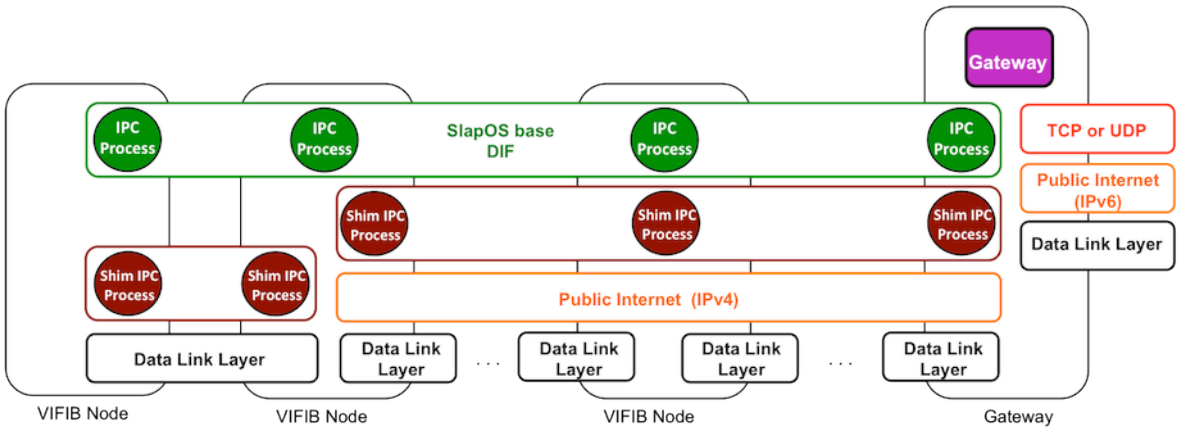

Figure 14. RINA applied to the distributed cloud use case: SOS-DIF

The Figure below illustrates an example of the SOS-DIF connectivity graph. It is a random mesh with a very high degree of connectivity, since each IPC Processes has a number of N-1 flows to different neighbors that varies between 10 and 30. This allows to keep the average number of hops between IPC Processes down to 1.7.

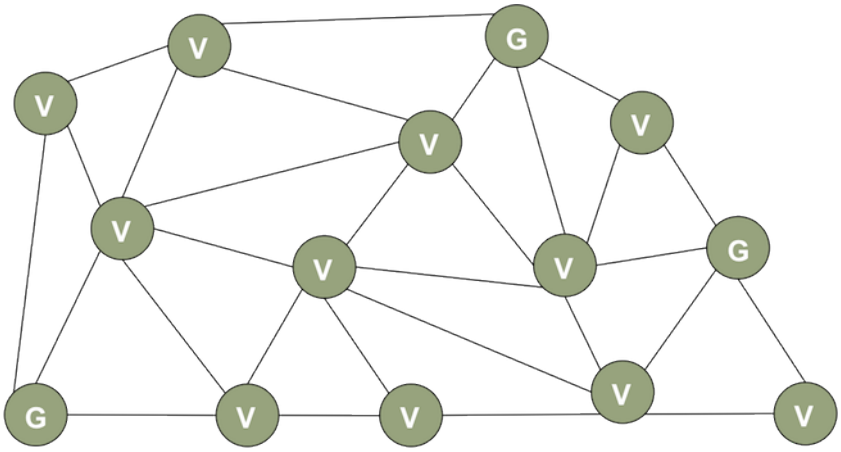

Figure 15. Example connectivity graph of the SOS DIF. V = VIFIB node, G = Gateway.

Service-tree DIFs

Another possibility different from the previous scenario is to consider the deployment of service-tree specific DIFs (ST-DIFs) that isolate service trees from each other and at the same time provide a highly customized networking environment to each service-tree. There are two ways of using service-tree DIFs, explained in the following subsections.

Service-tree DIFs over the SOS DIF (ST-DIFa)

Since services are directly instantiated as processes in the VIFIB nodes they all have direct access to the SOS-DIF and therefore are one hop away from each other. With this configuration it may still make sense to create DIFs that isolate the different service trees so that user applications cannot have direct access to the SOS-DIF. Since the SOS-DIF causes all the VIFIB nodes to be one hop away from each other, the connectivity graph of the SOS-DIF can just reflect the connectivity requirements of the service tree. For example, if the service tree is a web application, traffic may always flow from a gateway to a load balancer (in one VIFIB node), and then from the load balancer to different instances of the web application (in one or more VIFIB nodes).

All the service-tree DIFs would be interconnected to the gateway, which would provide connectivity with the IPv6 Internet. The gateway would work as in the previous case, except for the fact that it would need to know what DIF to use to reach a certain service.

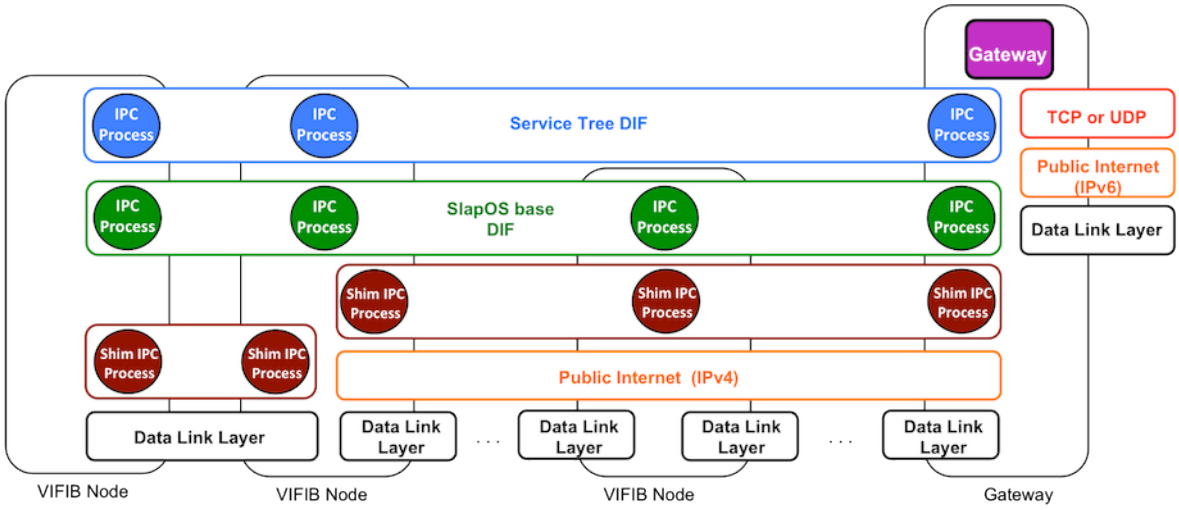

Figure 16. RINA applied to the distributed cloud use case: SOS-DIF and ST-DIFa with gateway

Another factor that adds interest to this approach is the case where the users of the service tree - which are not part of the VIFIB infrastructure - also support RINA. In this case - illustrated in the Figure below - the service tree DIF can span to the user systems, creating an arbitrarily complicated DIF with policies tailored to the users of the service tree. In this case no gateways would be required.

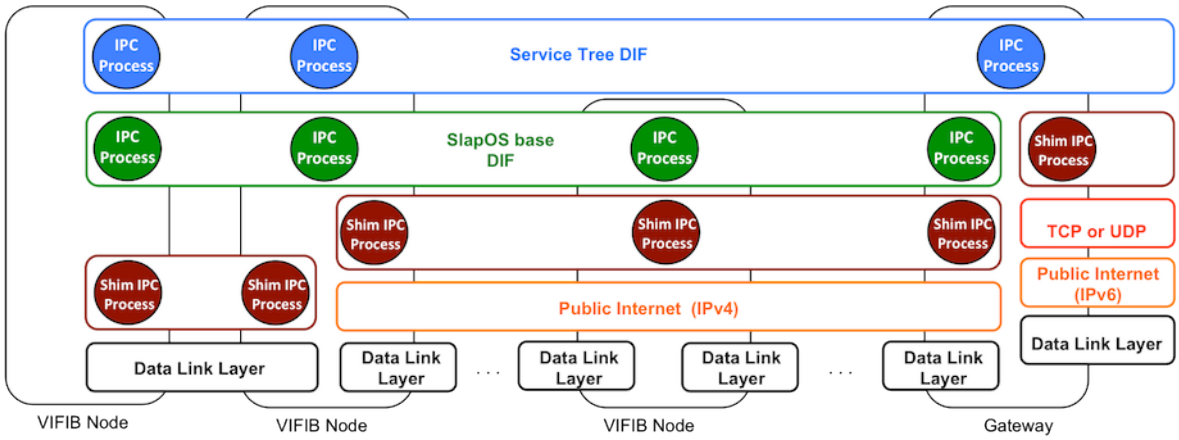

Figure 17. RINA applied to the distributed cloud use case: SOS-DIF and ST-DIFa without gateway

Service-tree DIFs without the SOS DIF (ST-DIFb)

In this scenario there is no SOS-DIF that interconnects all the VIFIB nodes together. Instead different ST-DIFs are allocated for different customers interconnecting only the resources that are required to provide the required service to each particular request of the customer. The Network Management - Distributed Management System (NM-DMS) would work in close cooperation with the SlapOS master node in order to create

the required IPC Processes when a customer requested resources to deploy a service tree.

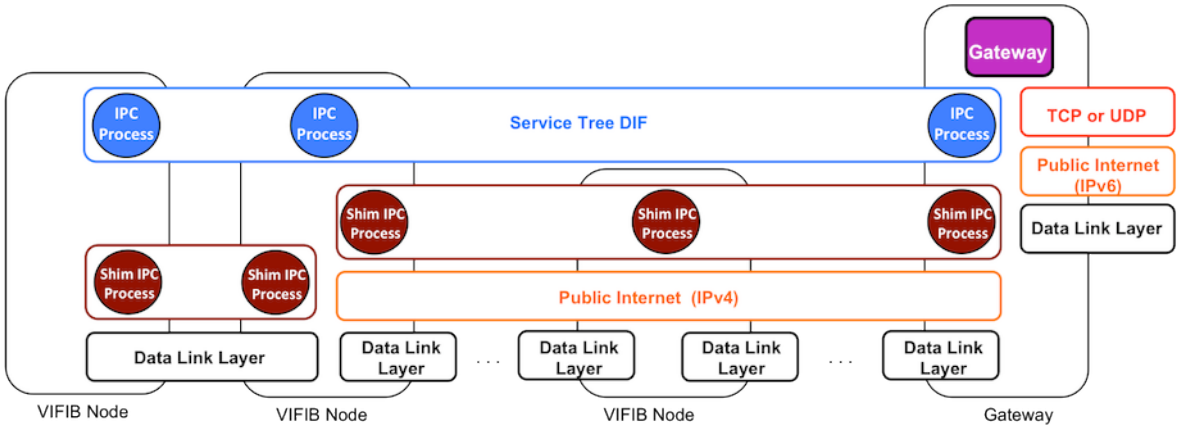

Figure 18. RINA applied to the distributed cloud use case: ST-DIFb with gateway

This scenario has the potential to provide less overhead to applications, since there is one level of DIFs less, while keeping service-trees completely separate (isolated in their own DIFs). However, the creation of service-tree DIFs -something that must be done dynamically, on a per-customer request basis- becomes more complicated. This is due to the fact that i) service trees cannot rely on the SOS-DIF and therefore must implement their own routing and resource allocation strategies and ii) service-tree DIFs will have to be larger because VIFIB nodes are no longer one hop away.

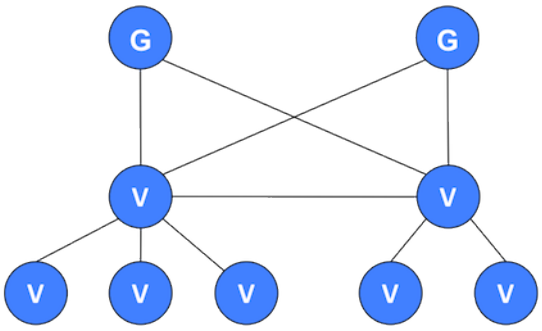

Figure 19. Example connectivity graph of the ST-DIFb. V = VIFIB node, G = Gateway.

This scenario still requires the SlapOS Master and NM-DMS to be able to connect to all the VIFIB nodes for management purposes; therefore a management DIF that complies with this purpose must exist. However, this Management DIF may have very different characteristics than the ones of the SOS-DIF.

2.4.2. Requirements analysis

Once we have described the distributed cloud use case, identified its limitations and studied different options to apply RINA in VIFIB's infrastructure describing the

different DIF configurations, it's necessary to study the requirements that emanate from the different DIFs in order to define the goals of its future design.

In the following, the requirements for the different DIFs are described and numerated to ease future references.

SlapOS base DIF (SOS-DIF)

This is the DIF that provides connectivity to the distributed cloud environment, joining together all the VIFIB nodes, gateways, registries and management systems. This DIF is not directly visible to the services accessed by the user. Those are confined to its own environment through the use of dedicated service-tree DIFs over the SOS-DIF.

The SOS-DIF has to accomplish the following requirements:

- **SOS-DIF-1**: In order to preserve the reliability of the overlay, the connectivity graph of the DIF should be generated the same way re6net's connectivity is generated, as explained in the section Connectivity Graph of the detailed description of this use case.
- **SOS-DIF-2**. The Babel protocol must be adapted to the RINA environment as the way to generate the PDU Forwarding table of the IPC Processes in the SOS-DIF. A first approach could go for a flat addressing structure (similar to what re6net is using today).
- **SOS-DIF-3**. PDUs in the base SOS-DIF must be encrypted hop by hop, as it is done in the OpenVPN tunnels used in re6net.
- **SOS-DIF-4**. IPC processes have to authenticate via X.509 certificates when joining the DIF, as it happens in the re6net overlay.
- **SOS-DIF-6**. IPC Processes in the SOS-DIF need to monitor the sanity and characteristics of the N-1 flows, like delay or error rate.

Complying with the previous requirements would make the SOS-DIF behave like the re6net overlay. The following list enumerates a number of improvements over this basic scenario that can be considered by the PRISTINE project.

- **SOS-DIF-7**. Support of hierarchical routing to allow the SOS-DIF to scale better. Study of other methods to compute the PDU Forwarding Table such as Link-State routing.
- **SOS-DIF-8**. Dynamic addressing capable of following the evolutions of the network's connectivity graph and to recover from a failure is required.

- **SOS-DIF-9**. Deploy resource allocation schemes to better support the flows of service trees (bounds on loss and delay), and to allocate overlay resources in a more efficient way (by knowing in advance the capacity and maximum delay required by individual applications to work).
- **SOS-DIF-10**. Deploy a congestion avoidance scheme in order to back-off flows causing congestion in buffers of the IPC Processes in the DIF.
- **SOS-DIF-11**. Assuming a rogue member has joined this DIF, investigate measures to discover who the rogue member is and how it can be isolated from the other DIF members.
- **SOS-DIF-12**. If management traffic is also carried by this DIF, provide the adequate means to separate management traffic from user traffic in order to guarantee the manageability of the VIFIB infrastructure.

Service Tree DIF (ST-DIF)

Service-tree specific DIFs isolate service trees from each other and at the same time provide a highly customized networking environment to each service-tree. ST-DIFs are the equivalent of tenant DIFs in the DC networking use case. Some of the requirements for this type of DIFs are highly dependent on the characteristics of the applications (services) the DIF has to support, but some of them are related to the basic service level that VIFIB wants to provide to its customers.

ST-DIF have to accomplish the following requirements:

- **ST-DIF-1**. Support for reliable flows: in-order delivery of SDUs, no SDUs lost.
- **ST-DIF-2**. Congestion avoidance scheme that protects the resources in the ST-DIF from becoming congested.
- **ST-DIF-3**. Support for dynamic computation of PDU forwarding table, probably using different types of approaches.
- **ST-DIF-4**. Optional use of authentication when enrolling with the DIF, using X.509 certificates (equivalent to stunnel usage between services in re6net).
- **ST-DIF-5**. Automatic address assignment.
- **ST-DIF-6**. Optional use of hop by hop encryption (equivalent to stunnel usage between services in re6net).
- **ST-DIF-7**. The DIF must be able to scale up and down dynamically, meaning that new members can be added or removed at any time.

- **ST-DIF-8**. The connectivity graph of the DIF will change depending on the application requirements, but each ST-DIF needs to have one policy for the generation of the connectivity graph.
- **ST-DIF-9**. IPC Processes in the ST-DIF need to monitor the sanity and characteristics of the N-1 flows, like delay or error rate.
- **ST-DIF-10**. Allows applications (services) to request flows with upper bounds on loss and delay.

Shim DIF(s)

In the Distributed Cloud scenario RINA DIFs are overlaid over two types of environments: Local Area Networks and the IPv4 Internet. Therefore, the following two types of shim DIFs will be used in the analysis, implementation and demonstration of this use case:

- **Shim DIF over Ethernet**. Allow RINA DIFs to be deployed over plain Ethernet or 802.1q layers (VLANs).
- **Shim DIF over TCP/UDP**. Allow DIFs to be deployed over IPv4 and IPv6 layers using TCP and UDP.

Network Management - Distributed Management System (NM-DMS)

In order to match the Network Management scenario with the scope of the project we assume that all the VIFIB infrastructure is a single management domain. We also assume a scenario in which there is logically centralized *Manager* process configuring and monitoring the VIFIB nodes via management agents deployed at each node. There are several options for the communication between the *Manager* process and the Management agents, depending on the presence and characteristics of the SOS-DIF.

- The *Manager* process can communicate with each ones of the agents via flows provided by the SOS-DIF. This is the approach currently used by VIFIB. In order to ensure that *user traffic* does not starve management traffic, the SOS DIF should be able to isolate management traffic and always guarantee a minimum level of service.
- The *Manager* process can communicate with the Management Agents via a separate DIF, dedicated to the NM-DMS. This is an option if the SOS-DIF is present, and a requirement in the case that Service Tree DIFs are put directly over the shim DIFs (ST-DIFb).

The requirements for the NM-DMS are the following:

- **NM-DISCLOUD-1**. The NM-DMS needs to interact with the VIFIB registry and the SlapOS Master in order to keep track of the nodes that are joining and leaving the SOS-DIF.
- **NM-DISCLOUD-2**. Each VIFIB node needs to run a Management Agent.
- **NM-DISCLOUD-3**. The Manager process needs to be able to authenticate Management Agents, and viceversa.
- **NM-DISCLOUD-4**. Data exchanged between the Manager and Management agents must be encrypted.
- **NM-DISCLOUD-5**. The NM-DMS must be able to monitor the state of the IPC Processes in the SOS-DIF and the ST-DIFs.
- **NM-DISCLOUD-6**. The NM-DMS must be able to create and destroy IPC Processes in the ST-DIFs.
- **NM-DISCLOUD-7**. The NM-DMS must be able to modify the configuration of the IPC Processes in the SOS-DIF and of IPC Processes in the ST-DIFs.
- **NM-DISCLOUD-8**. The NM-DMS must be able to isolate a system from the VIFIB infrastructure (by destroying IPC Processes or deallocating N-1 flows).
- **NM-DISCLOUD-9**. VIFIB nodes must be able to operate without the NM-DMS.
- **NM-DISCLOUD-10**. The NM-DMS must have a complete view of the VIFIB infrastructure.
- **NM-DISCLOUD-11**. There is the need to maintain a namespace to assign names to IPC Processes.
- **NM-DISCLOUD-12**. There is the need to maintain a namespace to assign Distributed Application Names to DIFs.

Gateway

The gateways of the VIFIB infrastructure allow customer traffic from the IPv6 Internet to access the services deployed in VIFIB's distributed cloud. After applying RINA, the gateway needs to maintain a mapping between IPv6 addresses of services deployed in the VIFIB infrastructure and the DIF names that these services are available through. Using this information the gateway will:

- Terminate incoming TCP or UDP flows.
- Check the destination IPv6 address, and find out which is the DIF through which the service with this IPv6 address is available (the IPv6 address of the service is used as the application process name).

- Allocate a flow to the service over the service-tree DIF identified in the previous step.
- Write the data from the TCP or UDP flow to the RINA flow, and vice-versa.
- When the TCP or UDP flow are terminated, deallocate the flow in the service-tree DIF.

Application APIs

Since the applications deployed in this use case cannot be modified, the VIFIB nodes running the RINA stack need to support the *faux sockets API*, which converts the calls to sockets into invocations to the native RINA API. Applications can be used on top of DIFs untouched, at the price of keeping the limitations of the sockets API.

3. Datacentre networking

The datacenter space is one of the areas that has seen more virtual networking innovations during the last few years, fueled by the flexibility requirements of cloud computing. A myriad of SDN-based virtual network solutions, usually providing L2 over L3 or L4 tunnels and a control plane, are available in the market (VXLAN, NVGRE, STT, etc). PRISTINE will investigate and trial the use of RINA-based solutions for intra- as well as inter-datacenter networking. Important issues to be addressed in a datacenter environment are the mobility of Virtual Machines to allow an efficient utilization of datacenter resources as well as high reliability; multi-homing support; guaranteeing the level of service in inter-data center communications and flexible allocation of flows supporting computer and storage resources.

3.1. Introduction and Motivation

3.1.1. Introduction

Since the appearance of computers, datacenters (DCs) have been present in some way or another. The first DCs were isolated groups of computers interconnected in order to carry out different calculations mainly in universities and large companies. These old DCs were mostly isolated, and it was later with the appearance of the Internet that they started to become what we understand as a datacenter now. Nowadays, when we refer to a datacenter, we are referring to a facility used to host computer systems, interconnected among them in order to carry out different tasks. Typically this facility is located in one room or one dedicated building, which in turn can be interconnected with other external datacenter facilities.

In the nineties, emerging Internet companies were demanding and developing DCs, but it was in 1999 when Google founders faced the necessity of a very large amount of servers for their brand new company. In 2006 Google built its first datacenter which was a whole dedicated building housing thousands of servers. Google's infrastructure has improved since then, and is currently one of the largest along with Facebook, Amazon, and so on. These datacenter infrastructures host several hundreds of thousands of servers, and this number is increasing day by day.

The datacenter network was not a critical aspect in those first days. The amount of traffic was not very high and the applications were not too complex. The datacenter network was basically a tree structure interconnected by switches, where the upper links (those closer to the Internet access point) had more capacity than the lower links.

The applications and dependent components were located nearby, typically within the same machine or within the same rack of servers (a set of servers directly connected by one switch). Back then there was little traffic, both external and internal within the datacenter.

The main issues DCs face today include energy, traffic, security, reliability and QoS issues. DCs try to optimize the energy they spend, utilizing efficient cooling systems. The traffic has increased enormously and now routing and load balancing mechanisms play a key role, since the amount of both intra and extra datacenter traffic reaches challenging peaks. And finally, datacenter users expect high reliable and secure services with acceptable QoS. All this represents the challenges DCs must face today and which they must address with current technologies. Implementing RINA may be a good approach to overcome these issues, which we will see in the remainder of this use case.

3.1.2. Motivation

RINA implementation in DCs represents a very interesting use case to study since DCs can leverage from the RINA architecture in a wide variety of aspects. Moreover, DCs allow the inclusion of RINA-based solutions more easily than other existing infrastructures.

The main aspects that motivate the implementation of RINA in datacenter facilities are the following:

- DC usage is expanding and DC technologies are evolving continuously. Gartner [sperling] suggests any datacenter more than seven years old is obsolete. If DCs are being built from scratch continuously, that represents an opportunity to implement innovative network technologies (such as RINA) that would be more difficult to deploy in an existing infrastructure.
- DCs are expensive to build. Datacenter building cost depends basically on the number of servers, nodes and network links present in the DC. Reducing the amount of the datacenter components improves the datacenter deployment cost. RINA based solutions may help in this regard. Specific designed routing and load balancing policies may help reducing the weight of the network infrastructure.
- DCs are expensive to operate and maintain. Operational costs depend basically on energy consumed and DC failures. A high scalable, flexible and modular solution like RINA can represent an advantage improving DC operation costs, for example, reducing the amount of intra datacenter traffic and implementing specific policies to handle datacenter failures appropriately.

- Multi-tenancy -the ability to support independent customers using the same infrastructure- demands strict flow isolation, both from a security and resource allocation point of view. TCP provides poor flow isolation, as by design flows compete for the same resources, interfering with each other. Security is complicated since it is expressed in terms of IP addresses and ports, instead of application names (updating the rules is cumbersome in a changing environment, such as the DC one). With RINA different congestion control and resource allocation strategies can be deployed in the different DIFs of the DC, providing strong flow isolation and allowing the DC network to stay within the predictable range of operation at higher compared to TCP/IP. This should result in direct savings in infrastructure, which in turn causes savings in energy consumption and DC operation.
- DCs need to support different applications with changing requirements. This fact requires the DC network to provide different levels of services, backed by different resource allocation techniques, which IP doesn't support. Moreover, for the DC to make an efficient use of its resources and to support the high availability of applications, it is necessary to relocate running VMs to different physical machines, sometimes in another physical DC. The fact that IP doesn't support application mobility complicates VM mobility a lot, usually restricting the movement of a VM within the same IP subnet. In RINA mobility is a non-issue, since it is inherently supported by a complete naming and addressing scheme, therefore managing the mobility of VMs is much simpler than in the case of IP. Moreover the internal policies of DIFs can be tailored to different applications, providing different levels of service but still keeping the same API; thus not increasing the DC management complexity.

3.2. Detailed description

The DC Networking use case description intends to provide the initial input for the issues to address in this use case. The goal of this first contribution is to set the DC aspects that may have an impact in a future deployment of RINA and describe how they are addressed today with current technologies. An important outcome of the trials may be the evidences of the DC performance improvement/enhancement when it's implementing RINA compared to current technologies.

3.2.1. Topologies

Large DCs housing a large amount of servers demand a tiered network structure. They use to have a 2 or 3 tiered network interconnected by level 2 or level 3 switches. The basic layers of a datacenter are the core layer (that closer to the external access point)

and the edge layer (that closer to the datacenter hosts). In case of a 3 layered datacenter, an additional aggregation layer is introduced on top of the edge layer.

Canonical topology

The traditional tree network structure of the DC is the canonical topology. This is the typical topology present in the Internet DC nowadays, which fulfill the typical Internet applications, such as serving web pages, files or any other content. The next figure shows the conventional network architecture for data centers (adapted from figure by Cisco [cisco]). This Figure shows the canonical tree structure where the nodes are duplicated for redundancy issues.

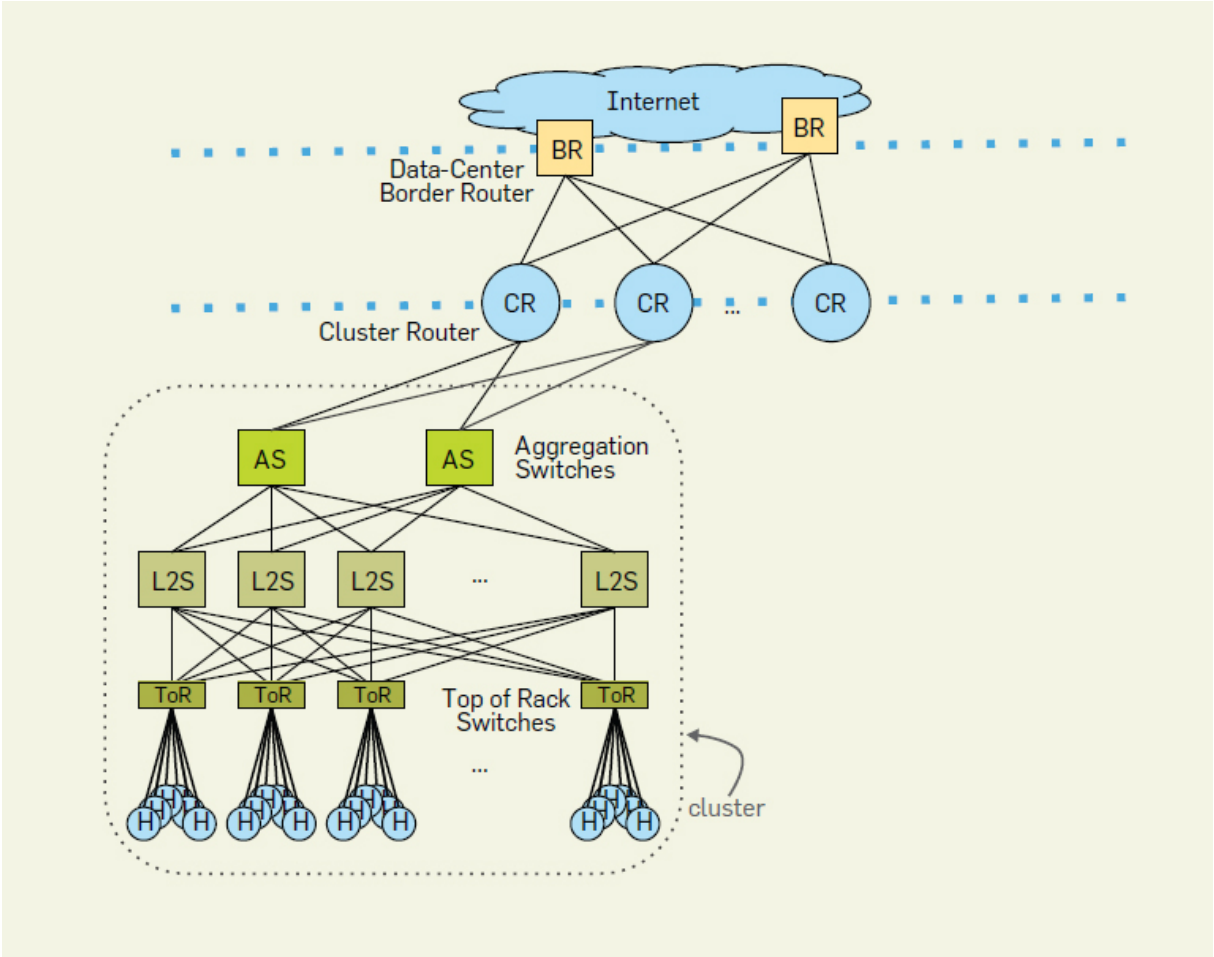

Figure 20. Canonical topology

Oversubscription

Many canonical DCs introduce oversubscription as a means to lower the total cost of the design. We can define the term oversubscription as the ratio of the worst-case achievable aggregate bandwidth among the end hosts to the total bisection bandwidth of a particular communication topology [guodcell]. The bisection bandwidth at a

certain level in a DC network is the aggregate of all the link capacity between the consecutive layers in that level.

An oversubscription of 1:1 indicates that all hosts may potentially communicate with arbitrary other hosts using the full bandwidth of their network interface. An oversubscription of 5:1 means that only 20% of the available host bandwidth is available when the other hosts in the rack are transmitting at full rate.

In canonical DCs with oversubscription, it is an engineering pattern to locate the application elements and dependent components nearby, typically within the same rack of servers. This way, the oversubscription introduced does not affect the performance, since in this case inter DC traffic is in fact intra-rack traffic, and servers do not intercommunicate with servers outside their racks.

As long as we work with traditional applications (serving typical web pages) the oversubscription does not represent a problem and the canonical topology would have a good performance with a relatively low cost. But the applications performed by the data centers today require many server-to-server intercommunications (as explained later in section “Traffic”), so the oversubscription limits the data center performance and the canonical topology does not fit well in this case.

To overcome this problem, one solution is to reduce or eliminate the oversubscription in the canonical data center, but this leads to high costs. We would have to add a large number of switches per layer to fulfill the requirements, so the cost of the data center would increase dramatically. Another solution is to employ another data center topology instead of the canonical topology. To that end new topologies have been proposed, which are described in the following sections.

New topologies

The aim of these topologies is to satisfy the high bandwidth requirements introducing no oversubscription. They achieve it basically increasing the number of switches and links in the data center network. The VL2 data center [greenberg], is a data center network architecture proposal that enables multi-path and scalability with layer 2 semantics. This proposal uses virtualization. Virtual Machines, located in the hosts, add a layer to the network. When services or Virtual Machines wish to send traffic, the packets are encapsulated in an IP-in-IP tunnel, and sent to the destination.

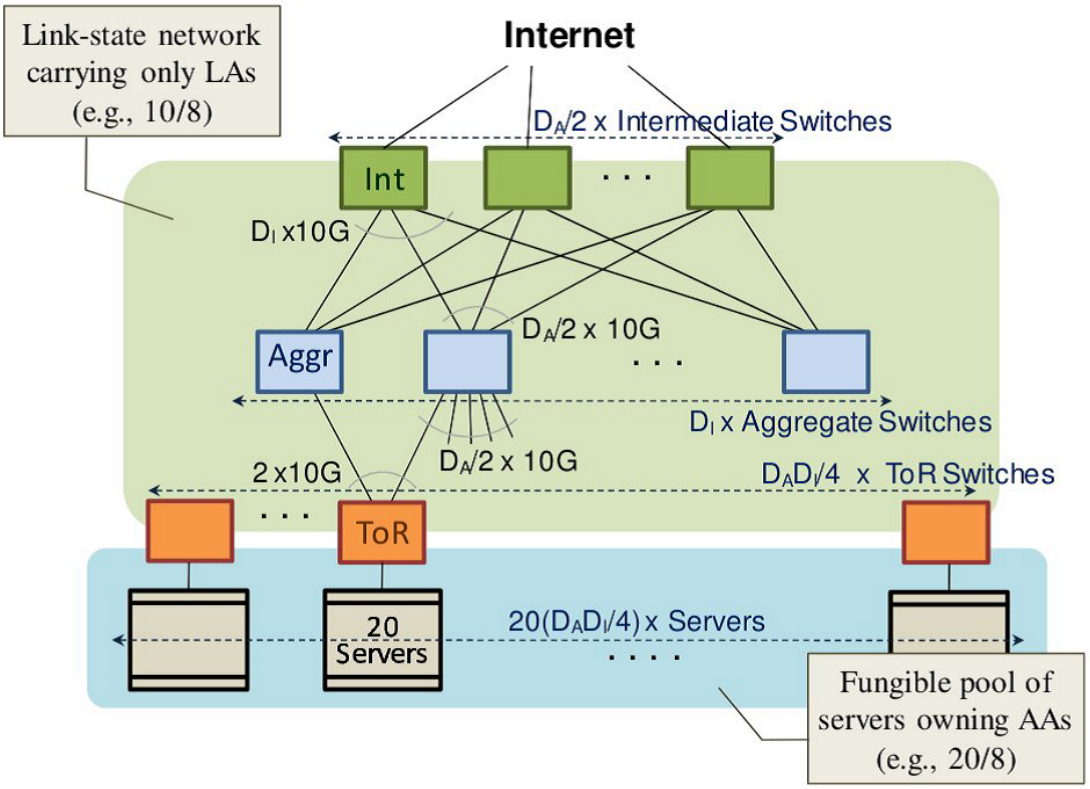

Figure 21. VL2 topology

This topology uses a maximum of 20 servers per rack and core and aggregation links of 10 Gbps and edge links of 1 Gbps. The number of switches per layer depends on a certain established ratios. We use DA port aggregation switches and DI port core switches. We have DA/2 intermediate switches, DI aggregation switches and DADI/4 edge switches.

The fat tree Clos network topology is organized in a k-ary structure. There are k pods, each containing two layers of k/2 switches. Each k-port switch in the lower layer is directly connected to k/2 hosts. Each of the remaining k/2 ports is connected to k/2 of the k ports in the aggregation layer. There are $(k/2)^2$ k-port core switches. Each core switch has one port connected to each of k pods. The i-th port of any core switch is connected to pod i such that consecutive ports in the aggregation layer of each pod switch are connected to core switches on k/2 strides. A fat tree Clos network built with k-port switches supports $k^3/4$ hosts. The following figure depicts a 4-ary fat tree Clos Network.

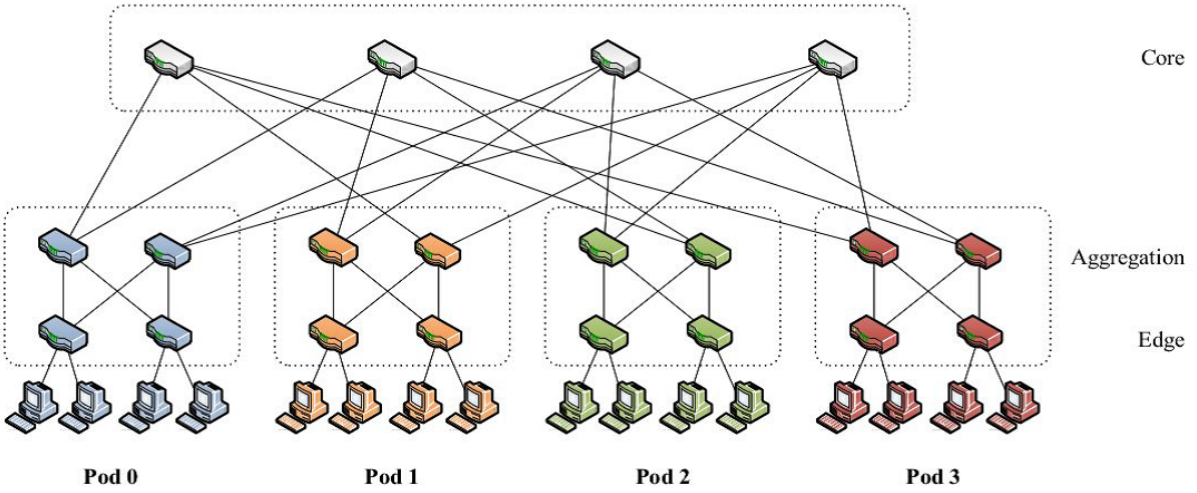

Figure 22. Portland topology

One peculiarity of this topology is that the number of links per layer is much larger than the other topologies. We can observe that this topology has the same number of core links than edge links, all of them with the same capacity and connecting switches with the same capacity.

These data center topologies are the so called switch-centric structures. In this kind of networks the switches are the only relay nodes. There are other data center network proposals, which are the so called server-centric structures [chen]. In these structures, servers act not only as end hosts but also perform relay functions. Data center networks such as BCube [guobcube] and DCell [guodcell] fall into this category. The next figure depicts the BCube topology.

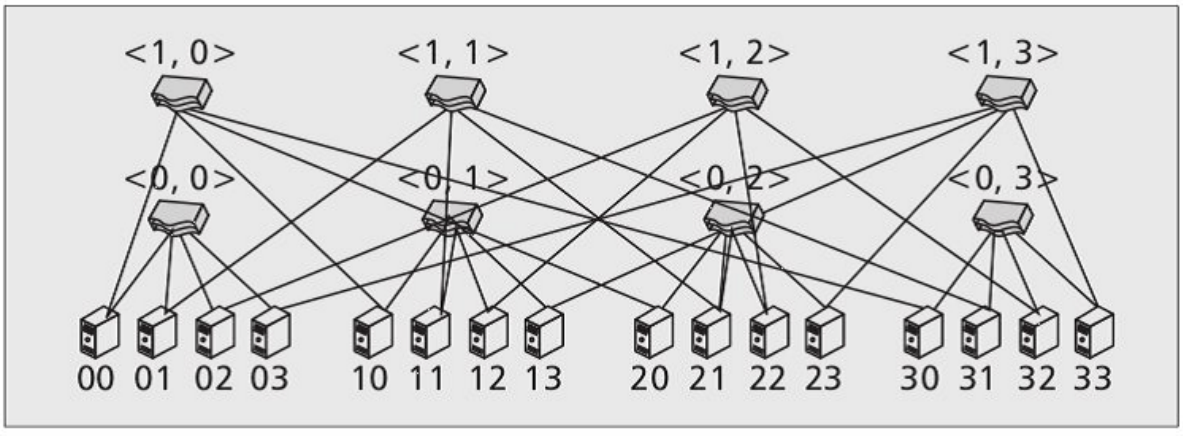

Figure 23. Bcube

That server-centric topologies have some advantages regarding the inter data center communications. Servers can interconnect with others in fewer hops than in the switch-centric ones, and they are more flexible when performing routing decisions.

Pod concept

The Portland architecture [niranjan] (Fat Tree Clos Network topology) introduces a very interesting concept: the pod. A pod can be defined as a set of aggregation and edge switches fully interconnected which are independent from other switches in other pods. In turn, pods only interconnect with one another through the core switches.

The next figure represents the pods in the PortLand network. The different pods are highlighted in different colors.

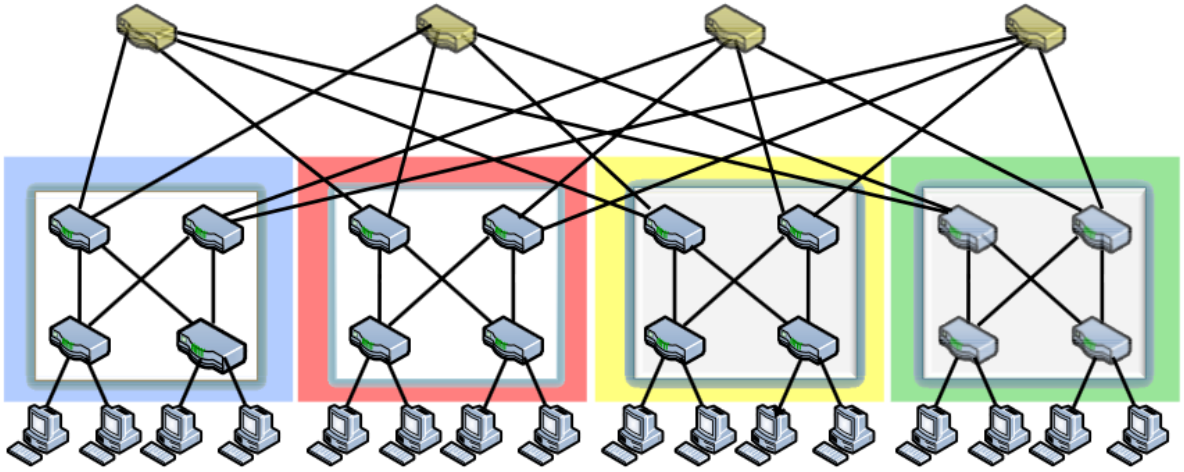

Figure 24. Pod concept

This concept can be also extended to other topologies. This is interesting because every topology can be parameterized by means of the pod concept. For example, the next figure depicts the VL2 topology where the pods are highlighted with a dashed red line.

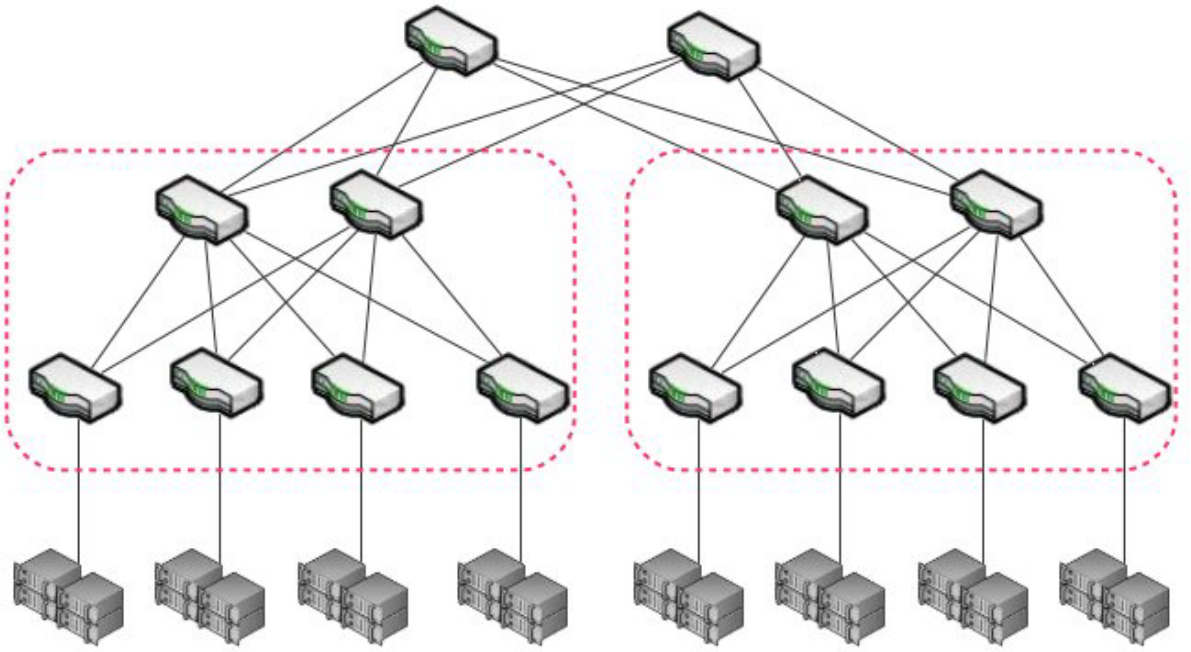

Figure 25. Pods in VL2 topology

Just as an example, a possible parameterization of a DC topology (assuming core and aggregation layers are fully interconnected) could be carried out specifying the following parameters:

- Number of core switches
- Number of pods
- Number of aggregation switches per pod
- Number of edge switches per pod
- Number of servers per rack

Naming & Name Resolution

An important aspect to study having in mind the later RINA implementation in a DC is how naming and name resolution are addressed in today's DC topologies. Today's DC node names are basically addresses (IP, MAC, ...) and the "addressing" term is the one used when referring to the assignment of addresses to nodes. This aspect is important to study, since RINA separates names and addresses while today's addressing schemas do not differentiate it and addresses can be regarded as the names of the nodes.

In today's topologies level 2 or Level 3 addresses must be given to the nodes and hosts in the data center network. There exist multiple approaches to DC addressing

and address space allocation. An address space is the range allocated for all possible addresses within the datacenter network. In turn, sub-networks within the datacenter may allocate an independent address space. Address space may refer to a range of either physical or virtual addresses that as unique identifiers of single entities, each address specifies an entity's location.

Two main approaches can be differentiated for addressing: flat addressing and hierarchical addressing. Addresses in a flat address space are expressed starting at a specific initial or starting address and continuing as incrementally increasing units until the end of the address space. Hierarchical addressing assigns addresses depending on the topology or location in a hierarchical fashion. IP, MAC, and ATM all use a flat addressing scheme. The telephone numbering scheme is a hierarchal addressing scheme. However, IP address spaces can be assigned hierarchically following the datacenter topology. Flat addressing advantages can be easily administered, but a hierarchical addressing can simplify the routing.

The next figures show an addressing plan in a fat tree Clos network which assigns addresses spaces hierarchically depending on the sub-net (pod) and assigns addresses depending on the position of the host in the DC topology. This depicts the Portland particular case, in which Pseudo-MAC addresses are used to denote the position of the nodes with respect to their position in the pods.

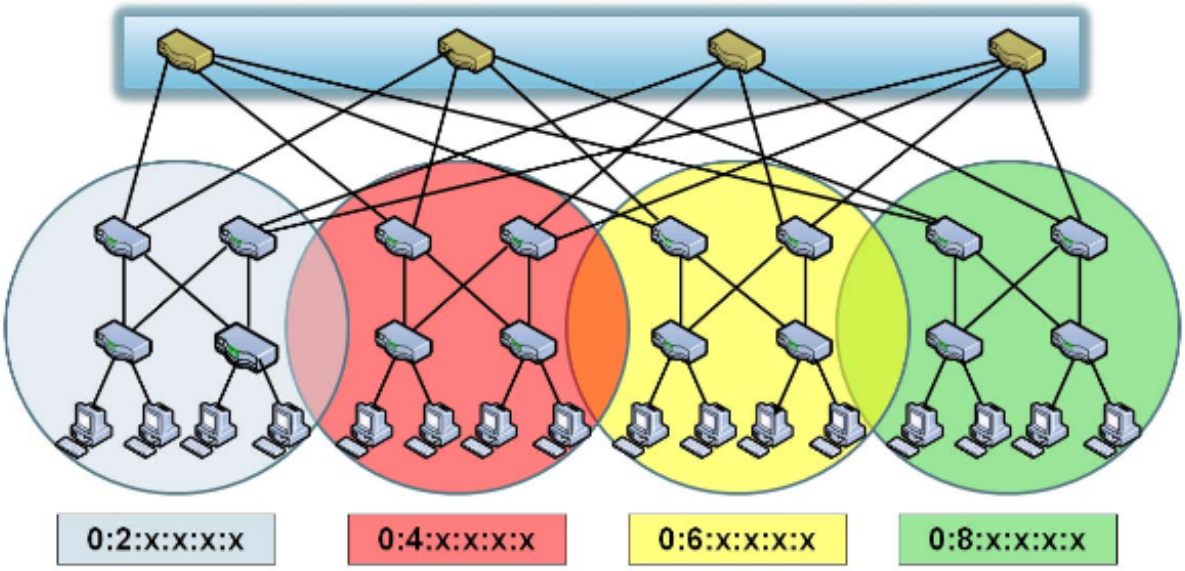

Figure 26. Portland addressing (I)

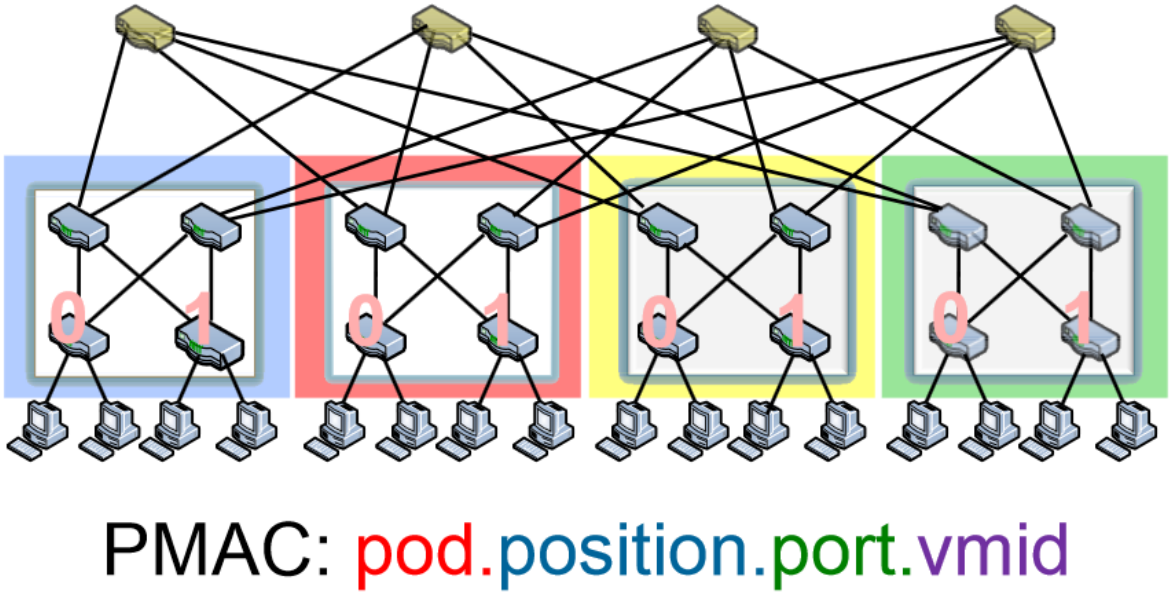

Figure 27. Portland addressing (II)

In turn, addresses can either be layer 2 or layer 3 addresses. The next figure depicts the comparison of L2 and L3 DC fabrics as specified in the PortLand architecture [niranjan], which internally separates host identity from host location (it uses IP address as host identifier and MAC addresses as network endpoint).

Table 1. Comparison of L2 and L3 DC fabrics

Technique	Plug and Play	Scalability	Small Switch State	Seamless VM Migration
Layer 2: Flat MAC addresses	+	-	-	+
Layer 3: IP addresses	-	+	+	-

Resource Discovery

For the DC network to be modified dynamically, mechanisms must be allowed to notify the existing network elements to know the insertion of new devices in the network. The basic approach to do so consists on advertising messages sent by the new devices to the neighbor nodes.

The Link Layer Discovery Protocol (LLDP) is a vendor-neutral link layer protocol in the Internet Protocol Suite used by network devices for advertising their identity, capabilities, and neighbors on an IEEE 802 local area network, principally wired Ethernet. The protocol is formally referred to by the IEEE as Station and Media

Access Control Connectivity Discovery specified in standards document IEEE 802.1AB [802.1AB]. Similar proprietary protocols do exist, such as Cisco Discovery Protocol (CDP) [ciscodp], which perform a similar functionality.

3.2.2. Traffic

DC traffic now is different from the DC traffic present in the first days. Nowadays new requirements are arising since large companies, such as Google, Facebook, Amazon and so on, demand other kind of applications. For instance, a typical web search request may touch more than a thousand servers hosting the inverted index to return the most relevant results [chen]. Moreover, other sources affirm that it often requires parallel communication with every node in the data center [alfares].

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

The DC traffic behavior is rather complex. For different topologies we have a different traffic scenarios. Some topologies are more indicated to traditional computations, in which the elements and dependent components are located nearby; however, another topologies are more appropriated to the new bandwidth requirements allowing inter communications at full rate at any moment between any servers.

One critical characteristic is the link capacity of each layer. The worst case scenario is when there are overloaded links and congestion points, which lead to losses and higher latency. New topologies overcome this problem by increasing the number of links and nodes in the network. RINA could help in this regard, with specific routing, load balancing and congestion control mechanisms dedicated to overcome these DC issues.

Load balancing

In every DC topology traffic must be spread evenly among the servers. That is the so called load balancing, which balances the traffic going through the links, avoiding having some links and servers overloaded while others remain idle.

Load balancing strategies are closely related to routing. The theoretical approach is that all servers (physical nodes) that attend a certain kind of application requests must serve an even amount of the total traffic load. This is true in a symmetric topology when there are not imbalances among nodes in the DC. In an asymmetric topology (or application

distribution), the optimal load balancing mechanisms could be different, not focusing on the distribution of application requests, but optimizing the link utilization or any other metric. To that end, appropriate routing strategies must be used.

Load balancing can be performed in different ways. One possible way is to use a central scheduler. Another one is to route the external requests downwards randomly across the switches to reach one random host each time. This is up to the design of the data center, and a different technique can be used depending on the preferences and requirements.

Routing

As seen in the previous section, routing mechanisms can be used to balance the traffic load among the DC links and nodes, or for any other purposes that involve the existence of more than one possible path from the source and destination nodes.

To make a simple differentiation with load balancing we can state that load balancing mechanism are intended to distribute traffic when several destination nodes are possible and the traffic must be distributed among them, while routing mechanisms apply when several paths are possible between two certain nodes and a decision must be made regarding what path to choose.

Single-path and Multi-path routing infrastructures

Single-path routing infrastructures are those in which only a single path exists between any two servers within the DC network (e.g. the canonical topology). These topologies simplify the routing tables and the packet flow paths but they are not fault tolerant. When a link breaks or a router crahses, the traffic cannot be delivered sucessfully to the nodes below the broken link or to the crashed server.

Multipath routing infrastructures are those in which multiple paths exist between servers within the DC network. These infrastructures are fault tolerant, i.e. dynamic routing protocols, such as OSPF, balance the traffic load across the multiple paths (taking into account metric values). Multipath infrastructures are more complex and routing loops can appear when using distance as metric for vector–based routing protocols.

Flat and hierarchical routing infrastructures

Flat routing infrastructures are those in which each address is represented individually in the routing table. The addresses do not represent subnetworks structures (e.g. by

means of prefixes) and sets of addresses cannot be aggregated into simpler addressing references.

Hierarchical routing infrastructures are those in which groups or sets of network addresses (representing a subnetwork) can be represented as a single entry in the routing tables. These routing infrastructures simplify routing tables requiring lower amount of routing information to be exchanged among nodes. In hierarchical routing infrastructures the DC network can be divided into routing domains, which typically comprise a subnetwork region whose nodes (routers and servers) share the same routing information within the domain.

Routing strategies

Many routing strategies exist today, such as ECMP (Equal-Cost Multi-Path) and VLB (Valiant Load Balancing) [greenberg]. ECMP is a routing strategy where next-hop packet forwarding to a single destination can occur over multiple "best paths" which tie for top place in routing metric calculations. Multipath routing can be used in conjunction with most routing protocols, since it is a per-hop decision that is limited to a single router. VLB is a per flow based strategy in which for each new flow arriving at a switch, if more than one possible output ports are present, then the flow is forwarded through a random port among the set of output ports for the destination. That way each flow follows a different random path from the source and the destination, and thus, the traffic is balanced among the network.

Loop free routing is another routing strategy in which switches populate their forwarding tables after establishing local positions in the following way: 1) Core switches forward according to pod numbers; 2) Aggregation switches forward packets destined to the same pod to edge switches, to other pods to core switches; 3) Edge switches forward packets to the corresponding hosts.

Fault Tolerant routing is another routing strategy in which network failures are overcome using a central scheduler in the following way: 1) Nodes exchange keep-alive messages; 2) A switch reports a dead link to the central scheduler; 3) The central scheduler updates its faulty link matrix, and informs affected switches the failure; 4) Affected switches reconfigure their forwarding tables to bypass the failed link.

Intra-DC traffic

Perhaps the most important problem known with intra-DC traffic is the "incast problem", which is a result of sending traffic to many nodes and getting results back

in one place with MapReduce-like communication patterns. Various mechanisms to address this problem have been proposed, often focusing on changes to TCP congestion control. One particularly well known example is Datacenter TCP (DCTCP) which is a simple change to the sender-side congestion control and can be deployed by configuring a standard Active Queue Management mechanism on all routers "strangely", to use the instantaneous queue instead of an average. DCTCP can greatly improve the performance of the network and achieve near-zero queuing delays. DCTCP does, however, not incorporate knowledge about flow deadlines. When deadlines are known, (DC)TCP's efforts to fairly allocate the rates do not match the optimization goal, and improvements that assign the bandwidth to flows in a more suitable matter have been proposed, culminating in the most recent and near-optimal proposal "PDQ" [liu].

Inter-DC traffic

The performance of Inter-DC traffic is related to the network infrastructure between the DCs. If it is the Internet, it is not under control of the operator(s) of the datacenter(s). Then, the best one can do is to avoid that the flows from one datacenter to another do not badly interfere with one another, and ensure that the overall traffic is reacting reasonably well to conditions between the DCs. This can be achieved by bundling the flows together, combining their congestion controls, ensuring prioritization between flows etc., and / or applying a suitable overall congestion control on the tunnel. Such functions are carried out by products such as Microsoft's SeaWall [shieh], for example.

Multi-tenancy and QoS

Multitenancy refers to a principle in which multiple tenants (clients) are served by a single software instance running on a single server. In a DC multitenancy applies when different tenants are served by different servers running instances of the same application. The difference of multitenancy with multi-instance architectures is that in multi-instance architectures independent software instances operate on behalf of different tenants, e.g. in virtualized infrastructures, where different Virtual Machines serve different tenants operating their VM independently.

QoS describes the overall system performance based on quantitative and objective metrics that measure different network and system aspects (e.g. delay, jitter, packet loss, bandwidth, transmission rate, etc). In multi-instance systems the resource management can be performed in advance in a more controlled fashion since the resources are dedicated and allocated to users. In multi-tenant systems the QoS can be affected by the users behavior and can vary unpredictably. So QoS and multi-tenancy are intimately correlated. An ideal scenario would be the one in which the

traffic belonging to different tenants could be managed independently, thus allowing the possibility of offering specific QoS to each of them.

Current QoS solutions in the Internet (TCP/IP) are based on a family of standards that focus on giving preferential treatment to certain types of IP traffic, ameliorating the effects of variable queueing delays and congestion issues. QoS management in current network infrastructures allows to:

- Regulate the amount of traffic differentiated by type;
- Use marked packets so that routers can deliver the traffic according to certain policies;
- Reserve resources by means of reservation requests from receivers and announce sender sessions available for resource reservation requests.

The two main Internet QoS models are the so called "integrated services" and "differentiated services", which enhance the traditional best-effort service model of the Internet.

Integrated services

Integrated Services (IS) refers to a dynamic resource reservation model in which hosts use the signaling protocol RSVP (Resource ReSerVation Protocol) in order to request a specific QoS dinamically. RSVP messages carry the QoS parameters while each node along the path store the parameters so that the requested QoS can be obtained.

IS handles two defined services: "guaranteed services" and "controlled-load services". The signaling is done for each traffic flow and reservations are carried out at each hop along the route. This model is well-suited for handling the dynamic needs of applications but it faces some scaling issues when employed in networks in which single routers handle many concurrent flows.

Differentiated services

Differentiated Services (DS) refers to a QoS model in which the per-flow and per-hop scalability issues of ISs are removed. Instead of using flows, they are replaced with a simplified mechanism of classifying packets which uses bits (DS codepoint) in the IP Type Of Service (TOS) byte to differentiate packets into classes. The DS codepoint is used by routers to define the QoS delivered at a particular hop, in a similar way than routers do IP forwarding using routing table lookups. The treatment given to a packet with a certain DS codepoint is called Per-Hop Behavior (PHB) and it’s handled

independently at each network node. The concatenated functionality and effects of the individual PHBs from the so called "end-to-end service".

Three PHBs are defined: Expedited Forwarding (EF) PHB, Assured Forwarding (AF) PHB group, and Default (DE) PHB. EF PHB forwards packets with higher priority and can be used to achieve low latency, low jitter and low losses for and end-to-end service. AF is a family of PHBs that classifies the drop precedence level of the packets. The drop precedence is assigned determining the relative priority of a packet within the AF family. The DE PHB is the traditional Internet best-effort service model.

Data Center TCP

DCTCP [alizadeh] is an enhancement to the TCP congestion control algorithm for data center networks. It leverages Explicit Congestion Notification (ECN), a feature which is increasingly becoming available in modern data center switches.

Explicit Congestion Notification (ECN) is an extension to the Internet Protocol and to the Transmission Control Protocol and is defined in RFC 3168 (2001). ECN allows end-to-end notification of network congestion without dropping packets. ECN is an optional feature that is only used when both endpoints support it and are willing to use it. It is only effective when supported by the underlying network. Conventionally, TCP/IP networks signal congestion by dropping packets. When ECN is successfully negotiated, an ECN-aware router may set a mark in the IP header instead of dropping a packet in order to signal impending congestion. The receiver of the packet echoes the congestion indication to the sender, which reduces its transmission rate as though it detected a dropped packet.

DCTCP sources extract multi-bit feedback on congestion from the single-bit stream of ECN marks by estimating the fraction of marked packets. In doing so, DCTCP sources react to the extent of congestion, not just the presence of congestion as in TCP. This finer level of control allows DCTCP to operate with very low buffer occupancies while simultaneously achieving high throughput.

3.2.3. Virtualization

Resource utilization

One of the main purposes of virtualization in DCs, along with elasticity (VM mobility), is the improvement of the physical resources utilization. Estimations about not virtualized distributed servers indicate an average capacity utilization ranging from 8 to 12% [daniels], so we can state that most distributed computing environments

underutilize physical server capacity when it's not virtualized. Virtualization allows devices to be used at their full potential and also improves costs, since less physical devices can be used to achieve the same performance. [oguchi] summarizes the background and effects of server virtualization.

VM mobility

VM mobility is a key aspect in today's DC operations. The process of moving a VM between hosts is called VM migration, which basically is the ability to move an entire virtual machine instance to another machine and transparently resume operation. Migration processes transfer the entire virtual machine (the in-memory state of the kernel, all processes, and all application states) from one host to another.

Migration may be either live or cold. In a live migration, the VM continues to run during transfer and downtime is kept to a minimum. In a cold migration, the virtual machine is paused, saved (the context is conveyed to persistent storage), and sent to another physical machine.

The migrated VM will expect its IP address and ARP cache to work on the new subnet since the in-memory state of the network stack persists unchanged. Attempts to initiate live migration between different layer 2 subnets will fail outright. Cold migration between different subnets will work, in that the VM will successfully transfer but will most likely need to have its networking reconfigured.

Cold Migration

The most basic way to move a VM from one physical machine to another is to stop it completely, move its backing storage, and re-create the VM on the remote host. This requires a full shutdown and reboot cycle for the VM. Migration begins by saving the VM. The administrator manually moves the saved file and the domain's underlying storage over to the new machine and restores the domain state. Since the underlying block device is moved over manually, there's no need to have the same file system accessible from both machines.

Live Migration

Live migration is the ability to move a domain from one physical machine to another transparently, that is, imperceptibly to the outside world. In contrast to cold migration, live migration transfers the domain's configuration as part of its state; it doesn't require the administrator to manually copy it. Manual copying is, in fact, not required at all. Of course, the initial configuration is still needed in case the domain is recreated from scratch on the new machine.

Live migration has some extra prerequisites. It relies on the VM's storage being accessible from both machines since it is not migrated and the source and destination machines access the same storage location. It also demands both machines being on the same subnet. Moreover, since the copy phase occurs automatically over the network, the machines must run a network service.

Live migration is based on the basic idea of save and restore only in the most general sense. The machine doesn't stop until the very last phase of the migration, and it comes back out almost immediately.

Some live migration techniques, such as those implemented by Xen hypervisor (one of the most widely used today), begin by sending a request (or reservation) to the target specifying the resources the migrating VM will need. If the target accepts the request, the source begins the iterative precopy phase of migration. During this step, pages of memory are copied to the destination host. While this is happening, pages that change are marked as dirty and then recopied. The machine iterates until the remaining pages are those which change very frequently, at which point it begins the stop and copy phase. Then the VM is stopped and the remaining memory pages are copied to the destination host. Finally, the VM starts executing on the new machine. After migration, the domain sends an unsolicited ARP (address request protocol) reply to advertise its new location. The migrating instance can only maintain its network connections if it's migrated to a machine on the same physical subnet since its IP address remains unchanged.

VM mobility over Wide Area Networks

VM migration among nodes that are not in the same LAN involves data transmission over Wide Area Networks such as the Internet. Cold migration can be carried out easily as long as IP reconfiguration takes place, mainly performed manually by the system admistrator. Other issues can arise when the VMs must be connected by external elements that must be aware of its location, but in reality these issues are regarded as operational and do not influence the migration feasibility.

Live migration becomes more challenging. WAN delays affect the migration process heavily, mainly because all the VM storage must be migrated along with the VM state and memory, since solutions such as shared storage are not feasible for machines that are not in the same LAN.

A system to support live migration over a WAN should have the following qualities [harney]:

- Continuous service: Any services provided by the virtual machine should sustain minimal downtime and interruption when the virtual machine is migrated.
- Permanent address for the VM: The network address of the virtual machine should not change after migration occurs.
- Interoperability with IP protocols: A virtual machine should be able to be migrated over a WAN connection without interfering with application-level communications. TCP and UDP should behave normally, with minimal side effects introduced by the migration or virtualization system, so that clients accessing the virtual machine are not affected. Additionally, the migration should be able to occur between any two hosts on the WAN.

Some research work has studied techniques for VM migration over WAN. [harney] studies a technique to enable live migration of virtual machines over the Internet. The method assumes the network supports Mobile IPv6. IPv6 standard enables a machine to maintain the same IPv6 address when moving between networks. This is done by creating a dynamic binding between a machine moving between networks and its home network. As the machine moves between networks, packets sent to its original address are either tunneled or rerouted to its new location. This approach eliminates the need for virtual networking schemes to interconnect the VM platforms. The sources of delay associated with the live migration are identified and the conclusion is that as long as migrations occur relatively infrequently, live migration over the Internet is practical.

[bradford] performs a bulk transfer of the virtual machine image before the actual migration takes place, and then transfers small deltas of the changing parts of the disk. Dynamic DNS (dynDNS) allow the VM's open network connections to continue even after migration and to ensure that new connections are made seamlessly to its new IP address. This work designs and evaluates a system that enables live migration of VMs that use local storage, and have open network connections, without severely disrupting their live services, even across the Internet. The system pre-copies local persistent state. It transfers it from the source to the destination while the VM operates on the source host. This allows for a substantial reduction in the amount of downtime caused by the time it takes to transfer the VM image during migration.

Overlay Virtual Networks

Motivation

DCs providing cloud services need to host multiple application stacks belonging to different tenants. Each application stack needs network services: layer 2 or layer 3 connectivity within the datacenter, firewalls, load balancers, connectivity with

the Internet, NAT and others. Since the applications belong to different tenants - sometimes even multiple applications from the same tenant but that require different security zones - the datacenter must provide proper isolation between them; both in terms of visibility and quality of service. What is more, the migration of an existing application into the cloud should be as seamless as possible, therefore the application stack must be able to maintain its existing internal addressing, network services and security model.

The usual cloud requirements of elasticity - the number of supported application instances and tenants must be able to scale up and down -, agility - the resources required for the deployment of an application stack should be provisioned in seconds - and flexibility - the placement of resources within the DC should not be static, but modifiable in real time without disrupting the applications supported by those resources - also need to be taken into account when designing the architecture of a DC that needs to provide cloud services.

If we focus on the requirements imposed to the networking side, it is clear that the proper support for multiple tenants or multiple application stacks per tenant with different security requires the provisioning of multiple network segments: ideally each application stack should be dedicated its own isolated virtual network segments with QoS guarantees. The traditional way of having multiple independent network segments over the same physical infrastructure is through the use of VLANs: each VLAN provides an isolated broadcast domain. Since VLANs provide layer 2 domains they also support VM migration within a single segment. This approach works very well in small DCs, but it suffers from scalability problems:

- Only 4096 VLANs can be used over the same L2 network (limitation on number of tenants)
- VLANs need to be manually provisioned at the physical switches of the DC substrate and at the hypervisor switches … (limitation on flexibility)
- … unless all VLANs are configured in all server ports, which causes the hypervisor to process all the multicast traffic in all VLANs - such as ARP requests - (limitation on scalability)
- Full routing is not supported within a VLAN. The Spanning Tree Protocol (STP) guarantees the creation of a loop-free connectivity graph within the VLAN, but it prevents hosts attached to the VLAN from exploiting multiple paths (limitation on efficient use of resources)
- All the DC relies on a single L2 network, which is a single failure domain (limitation on reliability)

One way of mitigating these scalability issues is to deploy different solutions within the physical network to overcome or mitigate them. As explained in [pepelnjak], the networking industry is reacting with a number of proposals:

- VM-aware networking, in particular IEEE 802.1Qbg [802.1Qbg], replaces the simple Hypervisor switch with another Ethernet switch that collaborates with the physical switches. The host can tell the switch which VLAN the VM needs and which MAC address (or set of MAC addresses) the VM uses, this way the edge switch can drop the broadcast packets of the VLANs that are not relevant to the host.
- In order to overcome the limitations of STP, a full-fledged routing protocol is being introduced at layer 2: Intermediate System to Intermediate System (IS-IS) as proposed by TRILL (Transparent Interconnection of Lots of Links) [rfc6325] or SPB (Shortest Path Bridging) [802.1aq].
- IEEE 802.1ad (also known as *q in q*) [802.1ad] can be used to overcome the 4096 VLAN limitation, by adding another VLAN tag over the original one (that allows to scale up to 4096*4096 VLANs).
- If there are limitations in the number of MAC addresses available within the L2 domain, TRILL (Transparent Interconnection of Lots of Links) or PBB (Provider Backbone Bridging) [802.1ah] allow for MAC-in-MAC encapsulation.

The fact that some of these technologies are still under development and not always commercially available, that each one only partially addresses the scalability limitations and therefore a deployment requires a combination of them, and that there is still only a single failure domain (layer 2) makes this approach costly and not well-suited to all DC environments. An alternative is gaining momentum, based on decoupling virtual networking (implemented at the hosts) from physical transport (implemented at the hosts and physical networking gear): overlay virtual networking.

Description

Virtual overlay networks comprise a number of similar technologies that share the same assumptions: the datacenter *physical* network provides simple IP connectivity between hypervisor hosts, which tunnel the traffic of virtual networks running in the virtual machines they host using different tunneling protocols (mainly VXLAN [mahalingam], STT [davie] or NVGRE [sridharan]). The datacenter physical IP network remains unchanged, as well as the Operating Systems running in the Virtual Machines. All the complex processing and tunneling is performed in the Hypervisors machines, sometimes with the assistance of additional "service" machines depending on whether the virtual overlay network is running a control plane.

The next Figure shows an example of a virtual overlay network using VXLAN as the encapsulation protocol and no control plane (it instead relies on IP Multicast at the DC network). The figure assumes that all the TCP connections, UDP connections and tunnels are already setup. When application A1 wants to communicate with application A2, it sends a TCP packet with the IP address of the destination VM where A2 is. The Ethernet header in the packet contains the MAC address of the virtual interface on the VM. This packet will arrive at the Hypervisor machine through the Virtual NIC device driver, and assigned to VXLAN processing via a previously configured software bridge (not shown in the Figure). The VXLAN processing entity at each Hypervisor is called Virtual Tunnel EndPoint (VTEP). Each VTEP function has two interfaces: One is a switch interface on the local LAN segment to support local endpoint communication through bridging, and the other is an IP interface to the transport IP network. The IP interface has a unique IP address that identifies the VTEP device on the transport IP network (in this case the DC network).

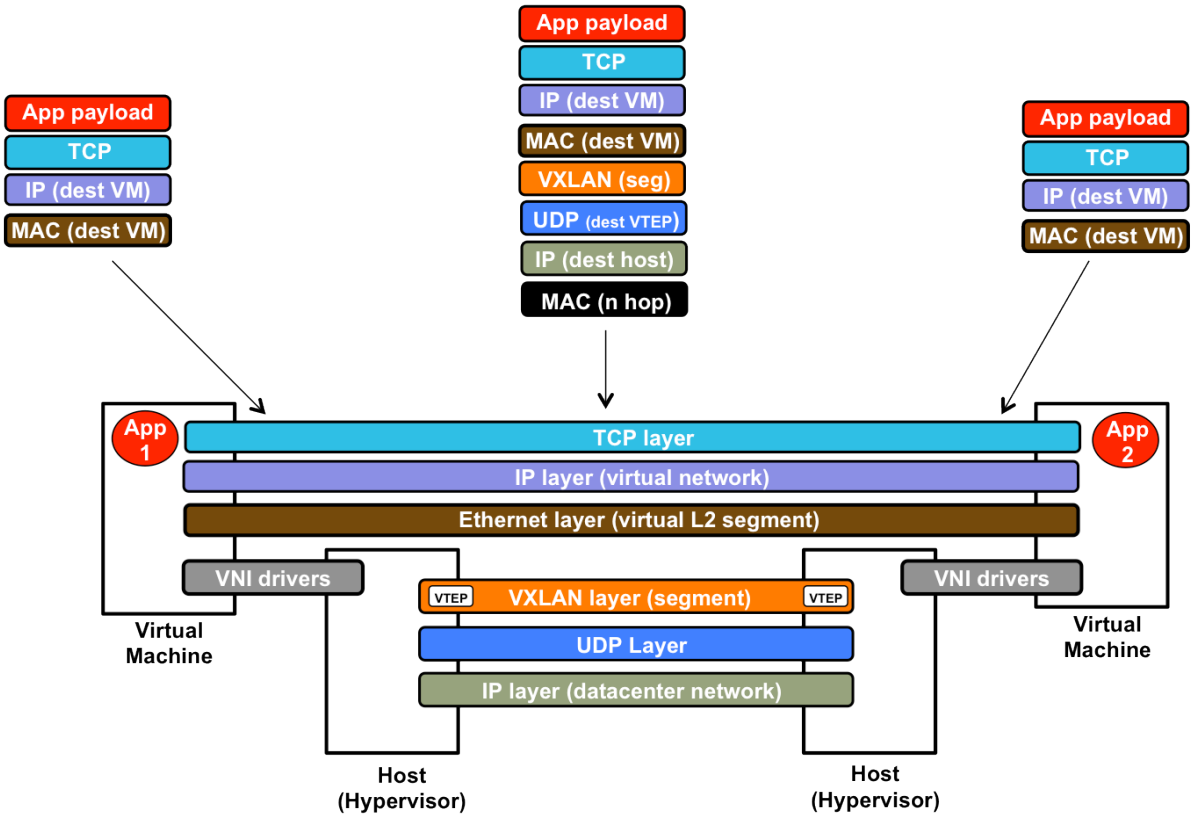

Figure 28. Overlay virtual network implemented with VXLAN

Once the packet reaches the VTEP, this has to decide to what VTEP it has to forward it to. Each VTEP has a forwarding table that maps a destination VM MAC address (in a virtual Ethernet segment) to a VTEP IP address (in the DC network). If there is an entry for the destination VM MAC address, the VTEP adds the VXLAN and UDP headers to the packet and requests the Hypervisor IP stack to deliver the packet to the

destination VTEP IP address. If there are no entries associated to the destination VM MAC address, the VTEP will request the IP stack to send a multicast packet to a pre-configured multicast address. When the packet arrives at the destination hypervisor host, the kernel IP stack delivers the packet to the VTEP, who strips the UDP and VXLAN headers, checks that the destination VM MAC address is available through its software bridge and delivers it to the VM. The packet reaches the VM through the VNI device driver, is processed by the VM OS and delivered to application A2.

Other encapsulation protocols work in a similar way: NVGRE uses its own header directly over IP, and STT uses the fields in the TCP header in order to take advantage of TCP offload techniques to increase the performance. The real differentiator in Overlay Virtual Networks is how to acquire and manage the distributed state required to make the system work. For example, how is the VTEP "forwarding table" (mapping of VM MAC address to VTEP IP address) generated and updated? A simple but not very scalable way is to emulate a "MAC learning switch" and rely on flooding/multicast to learn the mappings. Another way is to use an external control plane that can maintain the shared state and "download" this information to the VTEPs. Since there are trade-offs involved, no solution is ideal: it depends on the scalability requirements of the use case; for example: number of tenants to be supported, number of VMs per tenant.

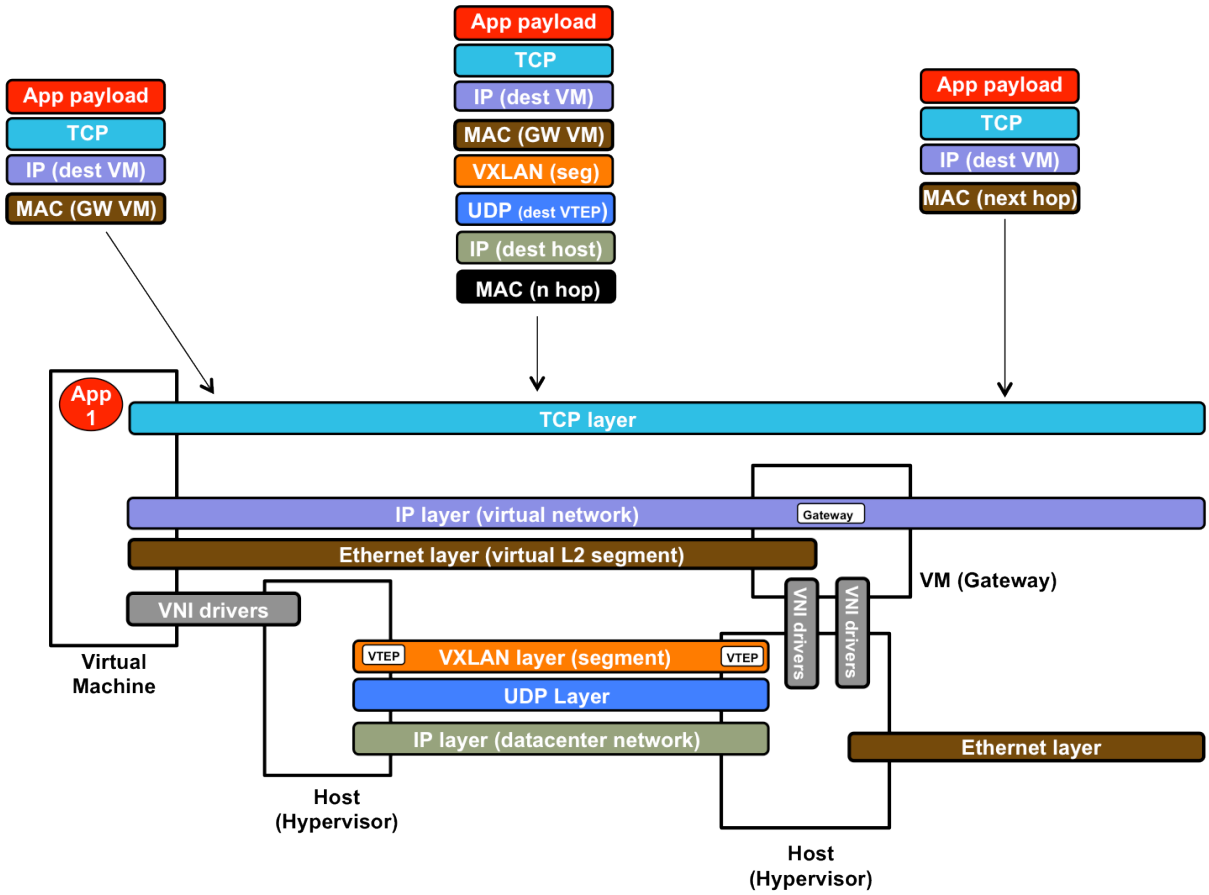

Figure 29. Overlay virtual network getting out of the datacenter

The scope of an overlay virtual network doesn't need to be restricted to a single datacenter, as shown in the Figure above. The use of gateways can interconnect the overlay virtual network with the external world (be it the public Internet, another DC or a customer VPN). The example depicted in the Figure uses VXLAN technology and a software gateway running in a Virtual Machine. This gateway - which can perform routing, firewalling and NATing functions - has, at least two interfaces: one that is part of the virtual layer 2 segment, and another one that is part of an outer layer 2 segment - allowing the gateway to forward traffic in and out of the DC.

Software-defined Data Center

Software-defined data center (SDDC) [volk1] is the term used to refer to a data center where all infrastructure is virtualized and delivered as a service. Control of the data center is fully automated by software, meaning hardware configuration is maintained through intelligent software systems.

SDDC extends the well-known virtualization concepts - abstraction, pooling and automation - to all data center resources and services and its ultimate goal is to centrally control all aspects of the data center – compute, networking, storage – through

hardware-independent management and virtualization software. This software will also provide the advanced features that currently constitute the main differentiators for most hardware vendors. This is in contrast to traditional data centers where the infrastructure is typically defined by hardware and devices.

SDDCs are considered by many to be the next step in the evolution of virtualization and cloud computing as it provides a solution to support both legacy enterprise applications and new cloud computing services. Software-defined data center is a relatively new enterprise computing phrase, but a number of vendors have announced software-defined data center products, including the VMware vCloud Suite [vmware].

The SDDC constitutes a grand vision of what enterprise IT would have to look like to optimally serve the business. The key idea behind this ideal state is to allow the application to define its own resource requirements – compute, network, storage, software – based on corporate SLA and compliance policies. In order to ensure the scalability, flexibility, and agility that will ultimately translate into a significantly reduced OPEX, it is essential to note that these requirement definitions are based on business logic, instead of technical provisioning instructions. These business logic elements are then translated into a set of API instructions that enable the management and virtualization software to provision, configure, move, manage, and retire the resources relevant to the business service. In short, the SDDC transforms the traditionally infrastructure-centric data center, with its focus on ensuring the proper operation of compute, network, and storage elements, into an application or business service focused environment. This transformation constitutes a radical paradigm shift, changing the role of IT staff from reactive service providers to proactive change agents ensuring capacities for future workloads. The SDDC purely revolves around application workload demands, allowing business users to deploy and run their applications in the most efficient and SLA compliant manner.

By definition, the SDDC is future-proof, as it relies on requirement definitions that are abstracted from the actual data center automation and orchestration layer. There is no army of clever coders at the heart of the SDDC, who simply tweak the automation and orchestration tool each time a new application workload, SLA, or corporate policy comes along. The SDDC has to be able to "autonomously" provide the type of application environment that ensures the reliability, performance, and security required by the business users.

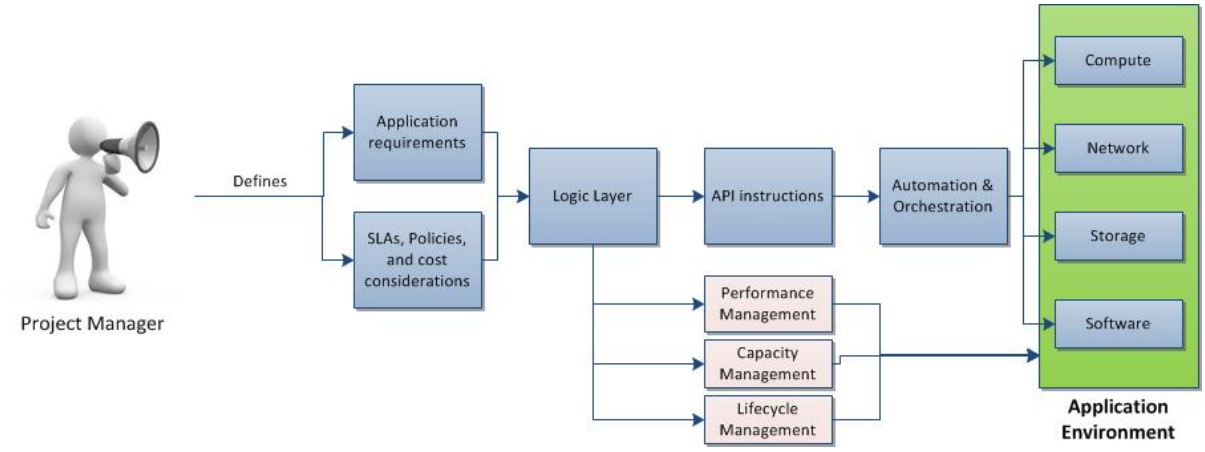

Figure 30. Software Defined datacenter

Google and Amazon are generally envied for their incredibly low OPEX per server and for their extreme scalability. However, we must not forget that simply creating a Google or Amazon-like private cloud is not the answer for most organizations, as there are many existing workloads that will not run on this type of highly standardized environment. Organizations often do not even have the source code available to recompile certain legacy applications to run in the cloud. In contrast, the SDDC – with some input by the project manager – will be able to determine a specific legacy application's requirements from a technical and SLA perspective and create the appropriate runtime environment that simply emulates the non-virtualized legacy infrastructure, where the application used to run on for years or even decades. To take advantage of Amazon's or Google's economies of scale, the SDDC can place certain workloads to the public cloud, as long as there is a sufficiently granular and rigid SLA available.

Core components

In the following sub-sections, we will take a look at the core components of the SDDC [volk2] and provide a brief evaluation of how mature these components currently are.

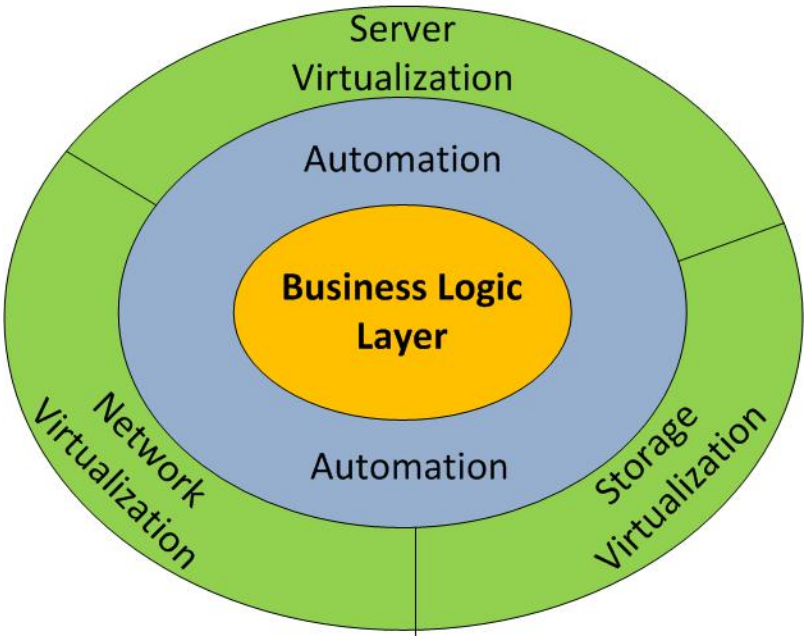

Figure 31. Core components of the SDC

Network virtualization

Creating the required network is a common reason for high provisioning times of new application environments. While it typically takes minutes to provision even a very large number of virtual machines, the requested networking resources often have to be created and configured at least semi-manually from multiple management interfaces. Not only does this process require advanced networking skills, but it can also lead to provisioning errors that can cause security issues. Software-defined networking (SDN) allows the user to simply specify which servers have to be connected and what the relevant SLAs are. The software then figures out the most efficient way of fulfilling these requirements without the typical configuration-intensive process.

Software defined networking (SDN) and network virtualization is one of today's hottest topics in enterprise IT. Most of the larger cloud platform vendors have launched or are in the process of launching their own capabilities for creating virtual networks that are abstracted from the underlying hardware and offer northbound API's for maximum application awareness.

Server virtualization

Pioneered by VMware over a decade ago, server virtualization is the most mature of the three SDDC components. Today, we see a trend toward organizations adopting multi-hypervisor strategies, in order not to depend on any one virtualization vendor and to take advantage of different cost and workload characteristics of the various hypervisor platforms. Configuration management, operating system image lifecycle management,

application performance management, and resource decommissioning are currently the most significant server virtualization challenges. Many enterprise vendors, such as HP, CA Technologies, IBM, BMC, and VMware, currently offer solutions addressing these challenges.

Storage virtualization

Similar to SDN, storage provisioning has traditionally constituted a significant obstacle for many IT projects. This is due to the often complex provisioning process that involves many manual communication steps among application owner, systems administrator, and the storage team. The latter has to ensure storage capacity, availability, performance, and disaster recovery capabilities. Often storage is overprovisioned to "be on the safe side." Overprovisioning storage is expensive, as most organizations pay a significant premium to purchase a brand-name SAN. Today, when buying SAN storage, there typically is a significant brand loyalty and very little abstraction of storage hardware from management and features. As in the instance of networking, when bundling hardware and software, vendors are able to achieve significantly higher markups compared to cases where hardware and management software are unbundled. Hardware independent storage virtualization is finally catching on.

Automation

Automating service provisioning and management processes requires the orchestration of numerous enterprise IT systems, such as server, network and storage automation, job scheduling, application management, database management and many more. The SDDC has to seamlessly connect these applications and "understand" dependencies in terms of availability, performance, security and cost. Only when the various systems that are composing a business service work together seamlessly, can new services be deployed and managed dynamically and intelligently, based on central governance rules.

Business logic layer

A business logic layer is required to translate application requirements, SLAs, policies, and cost considerations into provisioning and management instructions that are passed on to the automation and orchestration solution. This business layer is a key requirement for the SDDC, as it ensures scalability and compatibility with future enterprise applications, so that customers do not have to manually create new automation workflows for each existing or new application. The business logic layer is the "secret sauce" that is essential for tying together network virtualization, server virtualization, and storage virtualization into the SDDC. Without the ability to

automatically translate application requirements into API instructions for datacenter automation and orchestration software, there will be too many manual management and maintenance tasks involved to dynamically and efficiently place workloads within the programmable network, server, and storage infrastructure.

What's currently missing

A number of aspects are missing with regards the maturity of the individual components required for the SDDC [volk3].

- Server virtualization: An engine that dynamically places workloads in the hypervisor environments where they can run in the most efficient manner. This is partially due to customers not being at a point where they are looking for this type of dynamic workload placement, but also due to vendors not yet offering the flexibility required to dynamically shift workloads between hypervisors based on cost and policy requirements.
- Network virtualization: As more and more hardware vendors support open networking standards such as OpenFlow and OpenStack's Quantum and offer virtualization aware switches and WAN gateways, customers are starting to evaluate these technologies. However, it is essential to remember that we are only at the very beginning of customer adoption of software defined networking and network virtualization.
- Storage virtualization: All this progress must not divert our view from the fact that the interface between storage engineers and virtualization administrators is still a significant challenge, as very few customers are even aware of the capabilities of hardware independent storage virtualization and automation. The key problem is that storage vendors have no interest to cut into their own margins by promoting the capabilities of hardware independent storage virtualization. However, in today's real-time IT environments, end users will no longer accept the traditional painful storage provisioning process. This pressure from the customer side is in the process of changing the storage market place, forcing the large storage vendors such as EMC, NetApp and Hitachi to respond or lose market share.
- Central governance: Today, there are mostly point solutions out there addressing the governance challenge. By definition, point solutions are not able to globally tie together private and public cloud environments, under the roof of a central governance layer. ServiceMesh, as one of EMA's Vendors to Watch in March of 2013, exclusively focuses at offering this governance layer to enable developers to rapidly deploy compliant environments, while IT operations staff centrally controls the blueprints these environments are built upon. In order to truly achieve the

SDDC, significant maturation of the market place for central governance solutions is required.

- Dynamic workload placement: Similar to the "central governance" challenge, the dynamic workload placement problem is mostly unsolved. Most organizations do not fully trust current placement technologies, such as vMotion, which leaves them with static workloads. There are few solutions, such as VMTurbo that understand the connection between workload performance and resource utilization metrics and are able to automatically move workloads or add new resources before problems occur. This proactive approach to workload placement is not yet common in production environments and will have to be addressed by any successful cloud vendor.
- Automation and orchestration: While automation and orchestration are recognized as some of the most important IT topics today, IT silos are still common. Based on EMA research, almost half of enterprises believe that technology silos and a lack of automation constitute a significant problem, preventing many of their IT projects from achieving their maximum ROI.

3.2.4. Security

DCs use a multi-layered approach to security, combining physical security controls with information security and business continuity management. Physical security controls are used to prevent unauthorised physical access to the data centre. The location of the data centre can contribute to its physical security. It is carefully considered to ensure the data centre is not susceptible to environmental hazards and to make it easier to control access to the building [scalet]. For this reason, DCs are often located in remote, difficult to access areas, hidden from view and in nondescript facilities [amazon]. The first line of defense is to use perimeter protection to delay an intruder for as long as possible. Both natural and structural barriers are used, such as fences, bollards, landscaping and even locating the data centre in deserts or mountains [scalet]. Physical security controls are also used inside the building. Entry points are tightly controlled with access and visitor log, turnstiles and mantraps, and physical access controls, e.g. electronic cards, biometric multi-factor authentication. Intrusion detection controls, e.g. surveillance cameras, sensors and monitors, are used both at the perimeter and inside the building, are used both as a deterrent and to discover intruders.

Disaster recovery and business continuity management are central to ensuring the availability and resilience of the data centre [amazon], [google]. Multiple, geographically distributed DCs ensures that data can be quickly moved in the event of failure of a data centre. Uninterruptible power supplies and back up generators are

used to provide emergency power to the data centre should the mains power supply fail. DC providers have a business continuity plan in place, which contains strategies and processes for resuming critical business activities in the event of a disruptive incident.

Information security controls are used to protect the confidentiality, integrity and availability (CIA) of the data stored in the data centre. Encryption is used to provide confidentiality. Google now encrypts all Gmail data as it moves between DCs and uses TLS to provide encrypted connections to the data centre [lidzborski]. Encryption can also be used to protect stored data. Amazon Web Services (AWS) supports client-side data encryption, offers a CloudHSM to store keys and supports server-side data encryption [beer].

Access control is also used to protect confidentiality of the stored data. User authentication is used to authenticate both the data owners and the DC administrators. AWS relies on user IDs, passwords and Kerberos to authenticate administrators [amazon], while Google uses multi-factor authentication, e.g. certificates and one-time password generators [google]. Role based Access Control combined with the concepts of need-to-know and least privilege are used to authorise access. Accounting is used to log access to the data centre systems and to data. The resulting audit trail is regularly reviewed to detect unauthorised access.

Virtualisation is used in DCs to provide multi-tenancy and to allow more efficient use of resources. Hypervisors are used to enforce partitions and to isolate virtual machine instances running on the same physical machine. To ensure availability, data is replicated to both multiple systems within a data centre and to multiple DCs. Google uses a distributed file system, which chunks a user's data, replicates it and distributes it across multiple servers and multiple locations. They also obfuscate the file name of these chunks [google].

Multiple layers of network security controls are used in DCs to prevent unauthorised access to the network and to detect intruders. Intrusion detection and network monitoring are used to monitor the network for malicious activity. Firewall and guards are used both at the external network boundary and at key boundaries within the network to monitor and control communications. These devices use rule sets and access control lists to enforce the flow of information to specific services. DC networks are partitioned into networks containing sensitive data, e.g. management traffic, and those containing non-sensitive data. These segments can be physically separate, using separate infrastructure, or logically separate, e.g. private VLANs. Segmenting networks allows the security focus to be placed on the most critical areas and enables greater granularity in access rules.

3.3. Issues and limitations

3.3.1. Datacenter network issues and limitations

DCs today involve a number of issues that limit their performance. These issues may be related to the topology, technologies, devices or network links employed. RINA poses improvements to these limitations from the architectural point of view. Implementing the RINA architecture in a DC along with the corresponding policies would allow to overcome current datacenter limitations enhancing the DC performance, avoiding the challenge that supposes addressing that issues with today's DC approaches.

In the following, datacenter issues and limitations encountered in our analysis are described.

Oversubscription

As explained in the detailed description, certain topologies introduce a certain oversubscription ratio, limiting the maximum server transmission rate. This limitation is imposed by low link capacity, which is a topological issue. Current solutions address this issue increasing the number of links per layer of the topology without exploring other solutions, mainly due to the routing challenges that a non-hierarchical topology imposes.

Resources associated to users

Tenants todays ask for VM with certain characteristics, e.g. number of cores, RAM, storage. In the future they may also ask for network resources in terms of switches and links. As of today, it is unclear how virtualization will tackle the issue of having N control loops making resource allocation over the same substrate without "talking with each other".

Multi-tenancy and flow isolation

Multi-tenant applications with strict QoS demand strict flow isolation to control and enforce the QoS requirements. As explained in the detailed description, existing solutions are based on overlays that only address the issue partially and complicates DC management.

Multi-level services

The difficulties that current datacenter technologies have, impact also the multi-level service management. Having different service levels allows the existence of

dynamic applications with changing requirements, optimizing the system performance by resource allocation techniques, which is not supported by current technologies employed in datacenters such as IP.

VM Mobility

As described in the detailed description, VM mobility in current datacenters is restricted by the datacenter boundaries. Efficient live migration is only possible within the own datacenter infrastructure, while VM migration between different datacenters involves challenging migrations over WAN networks. Moreover, the fact that IP does not support mobility difficulties VM migration specially out of the datacenter infrastructure.

Security

Current DC networks have limited flexibility. Tenants may require different security controls and levels of security. This is currently difficult to manage. There is a complex interdependence of security mechanisms to ensure data protection in transit, storage, etc. This makes it difficult to configure security mechanisms correctly, resulting in configuration errors that can make the data centre vulnerable.

There is a reliance on logical (weak) separation of data on internal DC networks. Traditional DCs do not limit the bandwidth usage of applications, making them susceptible to denial of service attacks and performance interference attacks from applications [bari]. There is also a dependence on hypervisors for partitioning, making the DC susceptible to security vulnerabilities in virtualisation technologies.

3.3.2. Datacenter network requirements

In order to pave the way for the RINA implementation to address its application in the datacenter networking appropriately, the following requirements should be fulfilled in a RINA DC:

Uniform high capacity

The capacity between any two servers within a DC should be limited only by their Network Interface Card (NIC) transmission rate. The topology link's capacity should not limit the maximum intra-datacenter traffic rate. This is a topological issue, but the RINA implementation should be focused on improving this aspect. For example, using novel routing strategies that optimize the intra-DC traffic rate without the need of overpopulating the upper layers with additional links.

Performance isolation

The QoS provided by a service should be unaffected by other services' traffic. That involves aspects such as flow isolation and resource reservation that RINA must fulfill accordingly. Resource reservation should be decoupled from "physical" resources, e.g. a certain ensured bandwidth between two nodes should not involve any specific path between them.

Ease of management

A DC infrastructure should be easy to manage. Management aspects that should be addressed are:

- “Plug-&-Play”. The connection of new servers and nodes should be managed automatically by the network. Not special configuration must be needed before deployment.
- Dynamic addressing. Any server or node’s address/name must be dynamically assigned and managed by the network.
- The server configuration should be the same as in a LAN.
- Topology changes should be efficiently disseminated. Routing mechanisms should be highly convergent.
- Legacy applications depending on broadcast must work.

VM mobility

Any VM should be migrated to any physical machine within the DC infrastructure or to an associated external datacenter without a change in its name/address. In the same way, routing towards the new destination should converge seamlessly allowing new connections to the new destination in a transparent way.

Fault Tolerance

Potential failures should be detected, addressed and solved efficiently. Typical failures such as forwarding loops, etc. should be addressed.

Security

A RINA DC should allow tenants to configure security controls for their applications and it should offer secure, simple, uniform interfaces for configuring security policies and mechanisms. This should make it easier to configure security controls, reducing the

likelihood of configuration errors that lead to security vulnerabilities. Cryptographic protection should be used to separate data on internal networks and to protect control and management information exchanges with distributed facilities as part of RINA Management plane (e.g. name, DIF and network management).

3.4. Applying RINA to the use case: requirements analysis

This section describes how RINA can be applied to the datacenter networking use case. Based on the previous sections - namely detailed descriptions and limitations - the application of RINA is intended to improve the current datacenter characteristics and overcome the current datacenter limitations.

We start by describing how RINA can be applied to the DC networking use case, showing and describing the different options that we have considered for the different DIF's scope. Then we extract the requirements that these options demand from the different DIFs and which will drive the future DIF's design.

3.4.1. Overview

As for the application of RINA to the DC networking use case, two main scenarios are considered, the so called "everything supports RINA" and "whole DC supports RINA". In the former scenario all the network supports RINA, including the Internet. In the latter scenario, only the DC supports RINA internally while the Internet network operates with the current TCP/IP protocol stack.

Everything supports RINA

This section describes the case in which the Internet network the DC is connected to supports RINA. In this case we consider two main DIFs for the Internet architecture.

- Public Internet DIF: this DIF is the one that supports the connections throughout the Internet. It spans from the core switches to the opposite Internet connection end.
- Top Level ISP DIF: this DIF would be provided by the Internet Service Provider that is providing the connection to the DC. It spans from the core switches to the connection boundaries of the ISP.

In the following two options with different datacenter DIF configurations are studied.

Option 1

The following Figure depicts the first option.

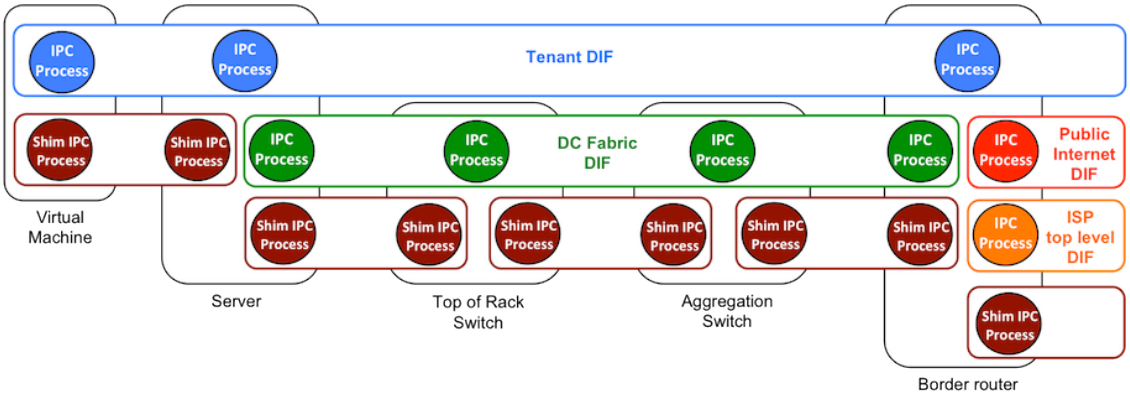

Figure 32. RINA applied to the DC use case

In this option the Intra data center connectivity is provided by a single DIF. The detailed DIF descriptions are the following:

- **Datacentre fabric DIF**: Connects together all the hosts in the DC, providing homogeneous bandwidth, non-blocking, etc. Uses a leaf-spine or multi-staged clos network topology (as show on previous slide). It allows the provider to allocate DC resources efficiently based on the needs of its tenants. Could be broken down in more DIFs.
- **Tenant DIFs**: Connects together all the VMs specified by a tenant, isolating them from other tenants or from other DIFs of the same tenant. Can also connect out of the datacenter (to customers or other sites of the tenant or to other DCs), for example over the public Internet (as shown in the Figure). In a sense, with this DIF the provider's domain becomes an independent unit of resource allocation.

The Figure below depicts the connectivity graph from the Datacenter Fabric DIF point of view. As this DIF spans all the datacenter components, the connectivity graph represents the datacenter topology.

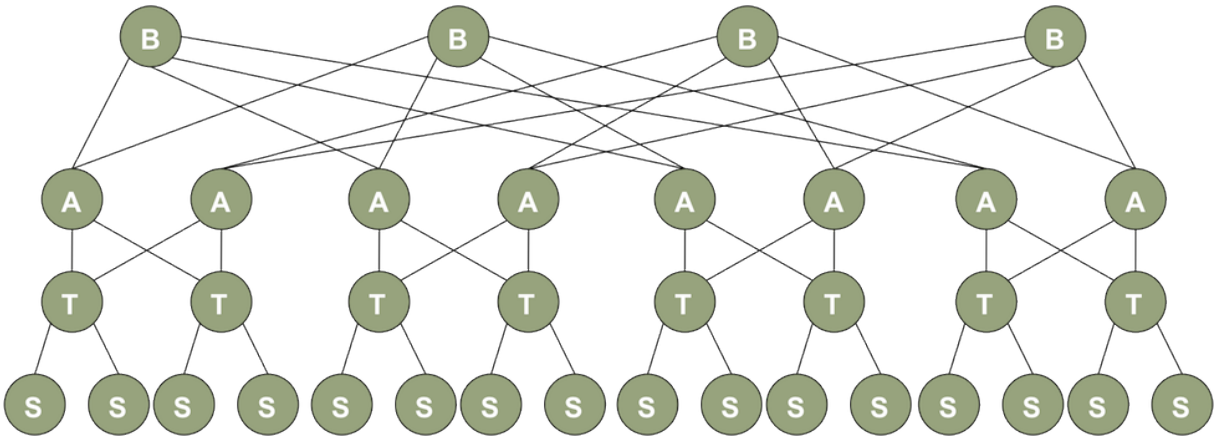

Figure 33. Example connectivity graph of the DCF DIF (S=Server, T=Top of Rack Switch, A=Aggregation Switch, B=Border Router)

The following Figure represents an example of the connectivity graph of a Tenant DIF, interconnecting 5 VMs between them and outside of the DC. This tenant DIF assumes that there will be traffic between VMs of the same server, between VMs of the different servers and between VMs and other resources located outside of the DC.

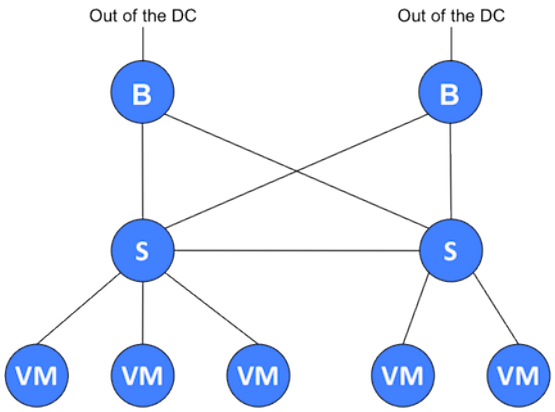

Figure 34. Example connectivity graph of the Tenant DIF (VM=Virtual Machine, S=Server, B=Border Router)

Option 2

In this scenario a new DIF is introduced to minimize the number of IPC Processes and N-1 flows required in the DataCentre Fabric DIF, removing one stage from its connectivity graph without requiring the Aggregation Switches to have more ports. This is accomplished by introducing a new type of DIF: the POD DIF.

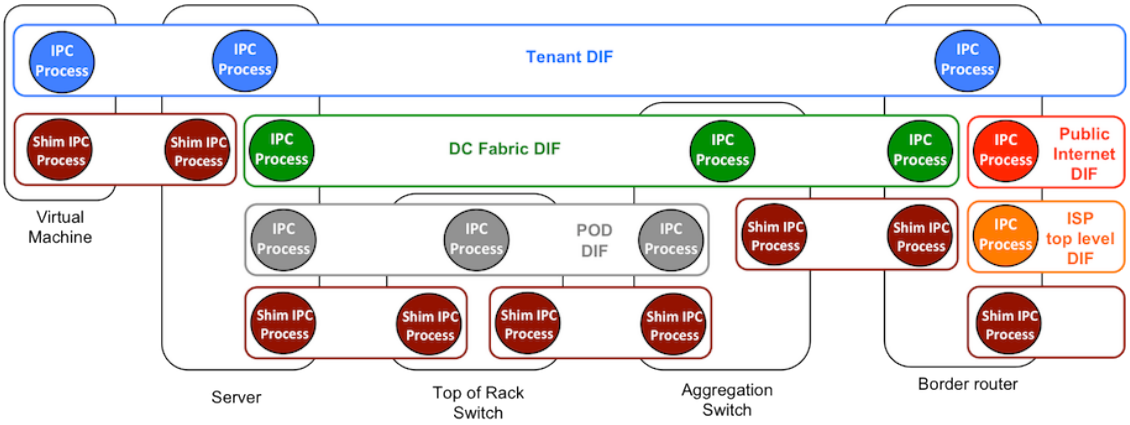

Figure 35. RINA applied to the DC use case with POD DIFs

The DC fabric DIF and Tenant DIFs are equivalent to the option 1, the only difference being that the DC fabric DIF will have less IPC Process since it is now overlaid on top of POD DIFs. The description of the POD DIF is the following:

- POD DIF: Connects together all the hosts, edge switches and aggregation switches in a POD. Could be used in case the Datacentre fabric DIF spanning all the switches

had scalability issues (if we use the POD DIF, edge switches are no longer part of the Datacentre fabric DIF)

The Figure below depicts the connectivity graph from the POD DIF point of view. As the POD DIF spans through a pod, the connectivity graph depicts the POD topology.

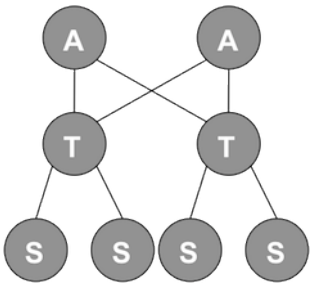

Figure 36. Example connectivity graph of the POD DIF (S=Server, T=Top of Rack Switch, A=Aggregation Switch)

The next Figure depicts the connectivity graph from the DCF DIF point of view. In this case the edge switches are transparent to the DCF DIF, and therefore the hosts see that they are connected through one less hop than in the previous case.

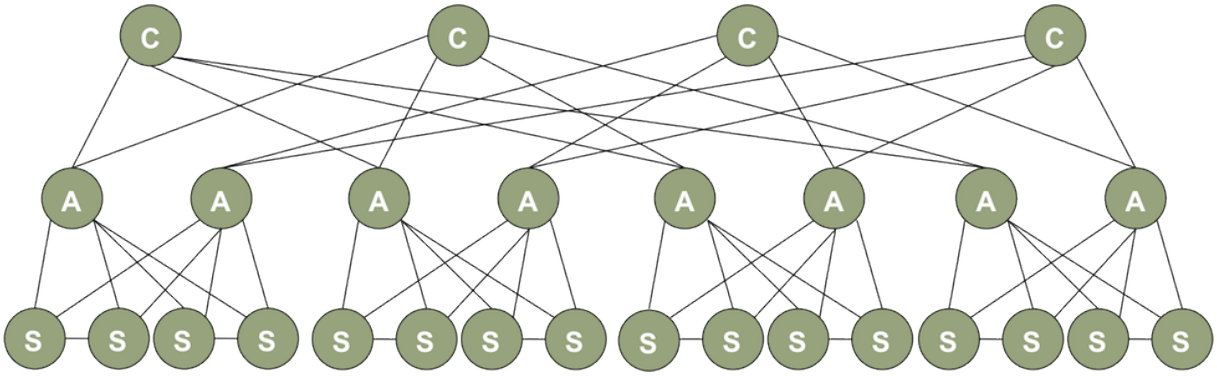

Figure 37. Example connectivity graph of the DCF DIF (S=Server, A=Aggregation Switch, B=Border Router)

Whole DC supports RINA

In this case RINA is only supported within the datacenter infrastructure while the Internet network connecting the datacenter works with the current TCP/IP stack. This case is important to study and consider, since this is the case that is more feasible to implement in the real world in short term.

For the internal datacenter configuration we have only depicted the case in which a datacenter fabric DIF spans along all components, but the case with the POD DIF (as seen in previous option 2) is also possible.

Two options are described for this case: in the first one the Tenant DIF spans to RINA-capable devices, which means that the devices at the other end of the network are RINA compliant; in the second one the Tenant DIF spans to non RINA-capable devices, which means that the devices at the other end of the network are not RINA compliant and work with the typical TCP/IP stack.

Tenant DIF spans to RINA-capable devices

In this option the components at the other end of the network which are connecting with the datacenter are RINA-capable devices. This means that they are the termination of a DIF that spans to the virtual machines in the datacenter. The Figure below depicts the DIF structure for this case.

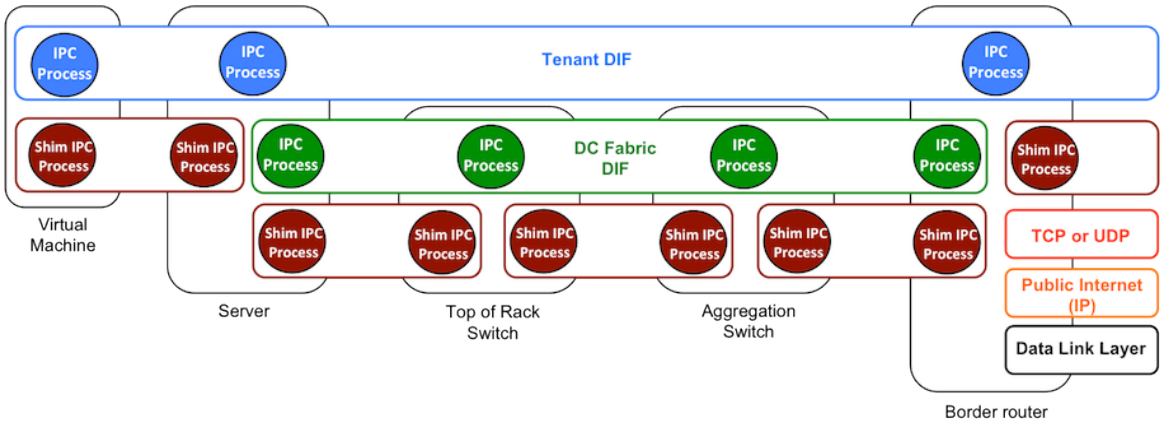

Figure 38. Tenant DIFs with RINA-capable customers

The DCF DIF and Tenant specific DIFs are equivalent to the "everything supports RINA" case. The difference is that the Tenant DIF is collocated over a shim DIF over UDP along the Internet, since the Internet network do not support RINA and a shim DIF is needed, spanning from the core switch to the external device connecting to the DC.

Tenant DIF spans to non RINA-capable devices

In this option the connecting components at the other side of the network are not RINA-capable, which means that they cannot be the termination of a DIF. For a non RINA-capable device to connect to the datacenter, protocol translation is needed. At some point of the connection path the RINA protocols must be translated into current Internet protocols. The following Figure depicts the DIF structure for this option.

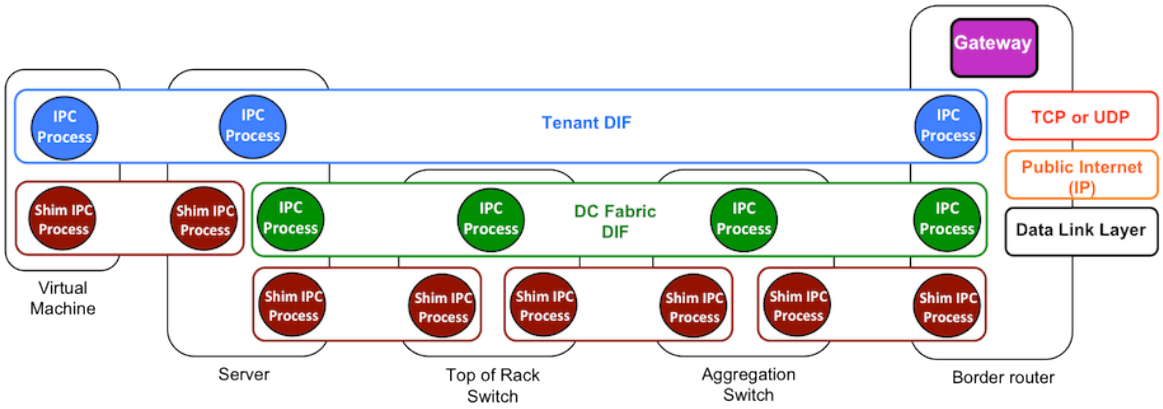

Figure 39. Tenant DIFs with RINA non-capable customers

In this option it is not possible for the DIFs to span from the datacenter VMs to the external end devices, and they terminate at the core switches.

The DCF DIF is equivalent to the previous cases. The Tenant A DIF is now restricted within the datacenter infrastructure, so the DIF functionality of connecting the Tenant's VMs applies only in the case there are more than one Tenant VMs in the datacenter. Therefore, in this option a gateway collocated in the core switches to perform protocol translation is needed. This gateway will interface with the multiple DC Tenant DIFs and the current Internet protocol stack.

3.4.2. Requirements analysis

Once we have described the current datacenter characteristics, identified its limitations and studied different options to apply RINA in a datacenter infrastructure describing the different DIF configurations, it's necessary to study the requirements that emanate from the different DIFs in order to define the goals of its future design.

In the following, the requirements for the different DIFs are described and numerated to ease future references.

Datacentre Fabric DIF (DCF DIF)

The DCF DIF interconnects all the servers and storage devices in the DC effectively creating a resource pool of computing and storage power. The DCF DIF also provides connectivity to the DC core routers, which connect to one or more external Network Service Providers and/or inter-data centre DIFs.

DCF DIFs have to accomplish the following requirements:

- **DCF-DIF-1**: The DCF DIF should allow to utilize any available capacity between two IPC Processes among all the possible paths, including the possibility of using

reserved resources that are not being used at a certain moment (allocating them again to the rightful tenant as soon as it utilizes it).

- **DCF-DIF-2**: Changes in the connectivity graph should be efficiently disseminated. Routing mechanisms should be highly convergent.
- **DCF-DIF-3**: The DCF DIF must provide differentiated QoS levels. The level of service provided to a flow should be unaffected by other flows' traffic.
- **DCF-DIF-4**: The connection of new servers and other systems should be managed automatically by the DCF DIF. Not special configuration must be needed before deployment.
- **DCF-DIF-5**: Dynamic addressing. Any IPC Process address must be dynamically assigned and managed by the DCF DIF.
- **DCF-DIF-6**: Any VM should be migrated to any physical machine within the datacenter infrastructure or to an associated external datacenter without a change in its name. In the same way, routing towards the new destination should converge seamlessly allowing new connections to the new destination in a transparent way.
- **DCF-DIF-7**: Potential failures should be detected, addressed and solved efficiently. Typical failures such as forwarding loops, etc. should be addressed.
- **DCF-DIF-8**: Traffic from different tenants must be properly isolated so that they can be managed independently to handle the QoS provided to different tenants.
- **DCF-DIF-9**: Users of the DIF can request flows with a certain capacity and upper bounds on loss and delay. Adequate mechanisms should be implemented to provide and manage the QoS required by DCF DIF users (data transfer, scheduling, resource allocation and routing mechanisms can influence the quality of the service experienced by the user of a flow).
- **DCF-DIF-10**: the DCF DIF shall be able to effectively multiplex traffic with different levels of burstiness. Therefore the DIF can decide whether to forward traffic using a more 'connectionless-like approach" (i.e. short, bursty flows) or use "connection oriented-like" mechanisms (i.e. long, stable-rate flows).
- **DCF-DIF-11**: The DIF shall run a congestion avoidance scheme in order to protect the resources in his layer.
- **DCF-DIF-12**: The DIF can be enabled to distribute traffic over multiple paths to balance the load and re-balance autonomously if needed.
- **DCF-DIF-13**: any IPC Process joining the DCF DIF needs to be authenticated to avoid spoofing.

POD DIF

The POD DIF interconnects all the systems within a POD (servers, edge and aggregation switches). This DIF is used by the DCF DIF directly, and not exposed to tenants, therefore its security and isolation requirements may be less stringent.

The POD DIF has to comply with the following requirements:

- **POD-DIF-1**: The POD DIF should allow to utilize any available capacity between two IPC Processes among all the possible paths within the POD, including the possibility of using reserved resources that are not being used at a certain moment.
- **POD-DIF-2**: The connection of new servers and other systems should be managed automatically by the network. No special configuration must be needed before deployment.
- **POD-DIF-3**: Dynamic addressing. Any IPC Process address must be dynamically assigned and managed by the network. Potential failures should be detected, addressed and solved efficiently. Typical failures such as forwarding loops, etc. should be addressed.
- **POD-DIF-4**: Only IPC Processes from the DCF-DIF can register and allocate flows in a POD-DIF.
- **POD-DIF-5**: Traffic from different flows must be properly isolated.
- **POD-DIF-6**: IPC Processes joining the POD-DIF must be authenticated.
- **POD-DIF-7**: The DIF shall run a congestion avoidance scheme in order to protect the resources in his layer.
- **POD-DIF-8**: IPC Processes in the POD-DIFs need to monitor the sanity and characteristics of the N-1 flows, like delay or error rate.
- **POD-DIF-9**: Users of the DIF can request flows with a certain capacity and upper bounds on loss and delay.
- **POD-DIF-10**: The POD DIF shall be able to effectively multiplex traffic with different levels of burstiness. Therefore the DIF can decide whether to forward traffic using a more 'connectionless-like approach" (i.e. short, bursty flows) or use "connection oriented-like" mechanisms (i.e. long, stable-rate flows).

Tenant DIF

Tenant DIFs are used to isolate the traffic between different VMs belonging to a single tenant. The characteristics of this DIF are therefore tailored to the requirements of the

tenant deployment scenario: connectivity graph, routing, resource allocation, security, etc.

The following examples illustrate the Tenant DIFs functionality.

- *Datacenter providing IaaS cloud services*. In this case VMs are deployed using the cloud provider IaaS services transparently to the tenants: the tenants that make use of the VMs connect to them without being aware of their location. The Tenant DIF is the one providing this functionality: applications on the tenant side connect with applications in the DC side making use of the services provided by the Tenant DIF. An example of this is a typical web application. The client enters the "website name" in a browser interfacing with the Tenant DIF. The request reaches the VM side. Now the application in the VM can make internal requests to another VMs (or databases) within the DC also making use of the Tenant DIF. And finally the response is sent to the client.
- *Datacenter distributed computing*. Datacenters are also used to perform computations in a distributed manner. The computation is distributed among different VMs. In this case the messages passed among the applications in different VMs involved in the distributed computation are transmitted making use of the Tenant DIF functionality from one VM to another. This Tenant DIF would delimit the network of interconnected VMs dedicated to the distributed computation.

The Tenant DIFs have to accomplish the following requirements:

- **TENANT-DIF-1**: Dynamic addressing. Any IPC Process address must be dynamically assigned and managed by the network. Potential failures should be detected, addressed and solved efficiently. Typical failures such as forwarding loops, etc. should be addressed.
- **TENANT-DIF-2**: The DIF should be able to scale up/down dynamically, with minimal management intervention. Therefore the connection of new servers and other systems should be managed automatically by the network. Not special configuration must be needed before deployment.
- **TENANT-DIF-3**: IPC Processes joining the DIF must be authenticated.
- **TENANT-DIF-4**: The DIF may support hop-by-hop encryption of PDUs.
- **TENANT-DIF-5**: The DIF must support reliable flows (in-order-delivery and o SDU loss).
- **TENANT-DIF-6**: The DIF shall run a congestion avoidance scheme in order to protect the resources in his layer.

- **TENANT-DIF-7**: The DIF must be able to scale up and down dynamically, meaning that new members can be added or removed at any time.
- **TENANT-DIF-8**: IPC Processes in the TENANT-DIFs need to monitor the sanity and characteristics of the N-1 flows, like delay or error rate.
- **TENANT-DIF-9**: Users of the DIF can request flows with a certain capacity and upper bounds on loss and delay. The tenant DIF can rely on characteristics of the flows provided by the DCF DIF to comply with the requirements of the TENANT-DIF users.

Shim DIFs

In the DC Networking scenario RINA DIFs are overlaid over three types of environments: Point to Point Ethernet (plain Ethernet or 802.1q), shared memory between "user" and "privileged" VMs within a computer and the public Internet. Therefore, the following three types of shim DIFs will be used in the analysis, implementation and demonstration of this use case:

- Shim DIF over Ethernet. Allow RINA DIFs to be deployed over plain Ethernet or 802.1q layers (VLANs).
- Shim DIF for Hypervisors. Allow "user" VMs to communicate with the "privileged" VM within a single server, using the XEN or KVM Virtualization technologies.
- Shim DIF over TCP/UDP. Allow DIFs to be deployed over IPv4 and IPv6 layers using TCP and UDP.

Network Management - Distributed Management System (NM-DMS)

In order to match the Network Management scenario with the scope of the project we assume that the DC infrastructure is a single management domain. We also assume a scenario in which there is logically centralized Manager process configuring and monitoring the DC nodes via management agents deployed at each node. The Manager process can communicate with each ones of the agents via a separate DIF dedicated to the NM-DMS system. This DIF can run over physically or logically separated infrastructure. The assumption for the PRISTINE project is that management traffic is both not accessible and not affected by user traffic.

- **NM-DC-1**. There is the need to maintain a namespace to assign names to IPC Processes.
- **NM-DC-2**. There is the need to maintain a namespace to assign Distributed Application Names to DIFs.

- **NM-DC-3**. The NM-DMS must differentiate between different types of users: administrator vs. tenants.
- **NM-DC-4**. The administrator user must be able to manage all the DIFs in the DC.
- **NM-DC-5**. The tenant user should be able to manage the tenant DIFs that it owns. This means reconfiguring IPC Processes, querying their state, getting statistics about their performance, etc.
- **NM-DC-6**. The NM-DMS must be able to dynamically allocate tenant DIFs as a response of a user request. This means creating and configuring the IPC Processes of the tenant DIFs in the most adequate servers, request the necessary N-1 links over the DCF DIF and trigger the enrollment procedures.
- **NM-DC-7**. The NM-DMS must work in close cooperation with the DC management system. This DC Management system could be proprietary or based on open-source solutions (such as OpenStack).
- **NM-DC-8**. The Manager process must be able to authenticate the Management Agents.
- **NM-DC-9**. The Manager process must be able to receive notifications indicating problems that require an action by the Manager (failures, performance degradations, suspicion of attacks, etc.)
- **NM-DC-10**. The Manager process must be able to isolate misbehaving IPC Processes or even complete systems from the DC network.
- **NM-DC-11**. The NM-DMS should be able to encrypt the traffic between the Manager and the Management Agents.

Gateway

The gateway is the component dedicated to interface between the RINA-capable datacenter network and the non RINA-capable Internet network. The gateway have to perform the following functions:

- **GW-DC-1**. Terminate incoming TCP or UDP flows.
- **GW-DC-2**. Check the destination IP address, and find out which is the DIF through which the service with this IP address is available (the IP address of the service is used as the application process name).
- **GW-DC-3**. Allocate a flow to the application over the tenant DIF identified in the previous step.

- **GW-DC-4**. Write the data from the TCP or UDP flow to the RINA flow, and vice-versa.
- **GW-DC-5**. When the TCP or UDP flow are terminated, deallocate the flow in the service-tree DIF.

Application APIs

The usage of two APIs is foreseen for this use case.

- For applications that cannot be modified, Virtual Machines running the RINA stack need to support the *faux sockets API*, which converts the calls to sockets into invocations to the native RINA API. Applications can be used on top of DIFs untouched, at the price of keeping the limitations of the sockets API.
- In order to better evaluate the different capabilities provided by the DC Networking use case DIF, simple test applications that use the *native RINA API* and are capable of requesting specific characteristics for a flow will be used.

4. Network Service Provider

4.1. Introduction and Motivation

4.1.1. Introduction

The goals of this scenario are to investigate and trial the benefits of the use of the RINA technology by a Network Service Provider (NSP), and to analyze RINA as an essential component of the Network Functions Virtualization (NFV) [etsinfv] concept within an operator network. NFV contemplates the provision of network services by decoupling network capacity, implemented in homogenous hardware, and network functionality, provided by software that is dynamically instantiated on the hardware substrate.

We see RINA as an essential technology for realizing NFV within an operator's networks. Thus RINA can provide an enhanced support for any of the use cases being considered as target for NFV, such as Customer Premises Equipment (CEP, for instance in a virtual home setting), access network virtualization (for instance providing the means to virtualize mobile nodes), and virtualizing an operator's mobile core (for instance virtualizing IMS or EPC).

Virtualization *eliminates* the dependency between a Network Function (NF) and the hardware it runs on, as seen in typical physical network appliances. This is done by providing a *homogeneous* execution environment based on regular (if not fully standardized) hardware and supporting software, and the necessary management interfaces to allow the elastic deployment of Virtualized Network Functions (VNFs) over the appropriate elements of the homogeneous supporting infrastructure. Further pooling of the infrastructure and the VNFs themselves facilitates a massive and agile sharing of NFV Infrastructure (NFVI) resources by the VNFs, and new resiliency and load distribution possibilities. This creates new architectural and business opportunities analogous to the cloud computing service models (IaaS, PaaS and SaaS). For example, a VNF owner does not necessarily own the NFVI needed for the proper functioning and operation of the VNF, or a NSP can construct a network service by composing VNFs not necessarily of its own.

4.1.2. Motivation

NFV is in essence a recursive architecture and can greatly benefit from RINA. The realisation of a VNF can be seen as one or more Virtual Machines (VMs) that execute the software performing a certain set of the whole VNF functionality. These VMs are termed VNF Components (VNFC) [nfvarch]. Those VNFCs are interconnected by

means of an infrastructure network to guarantee the functional and non-functional requirements of the associated network function. The VNFs built in this way become part of network services constructed by composing them in a service graph. These services can constitute the underlying network layer supporting a VNF in an upper layer. The following figure shows the layers taking part in the building of a network service. At each layer, the provider *abstracts* the underlying elements providing a control and management interface and takes care of the orchestration of the involved resources.

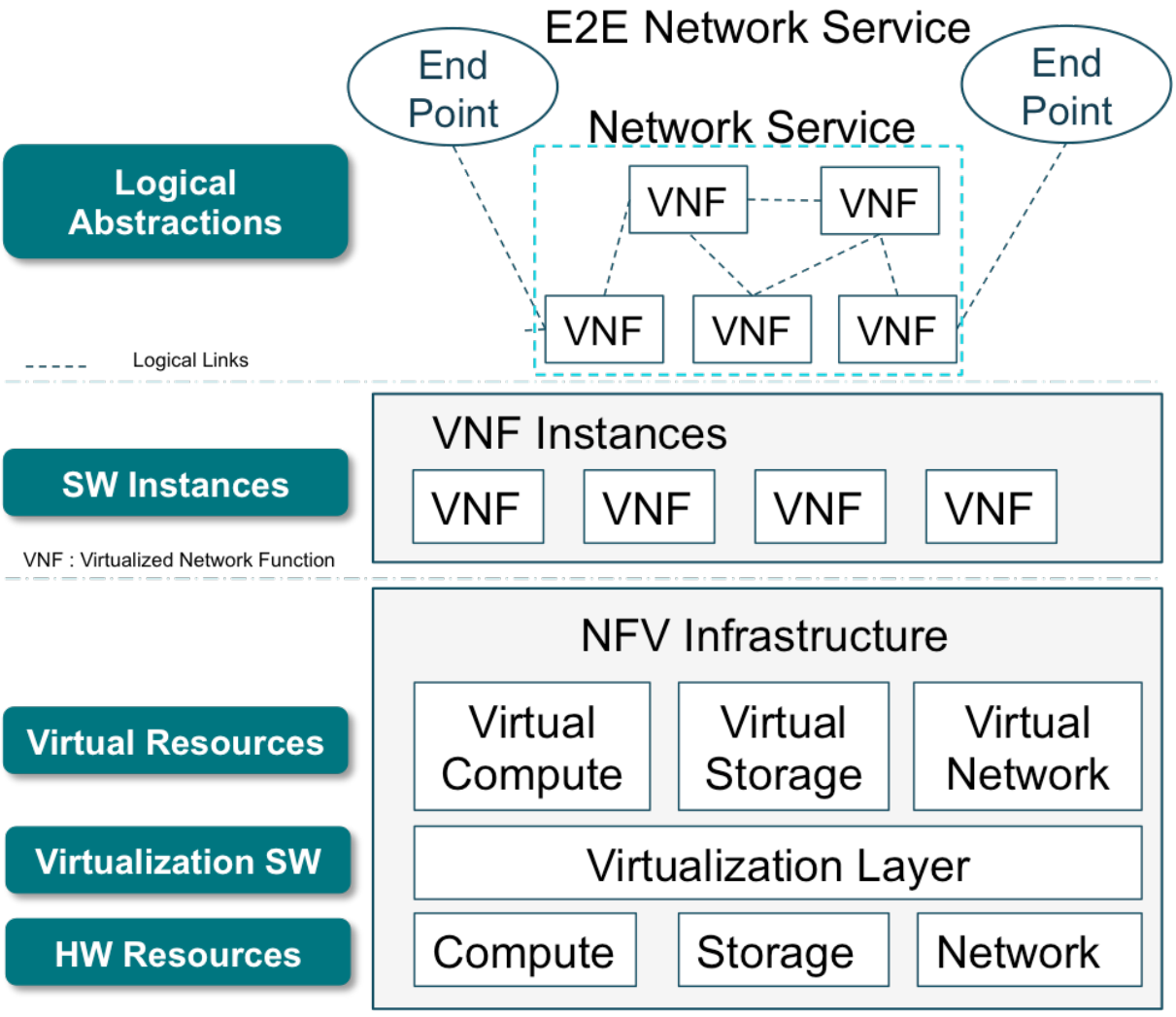

Figure 40. Layers of the NFV architecture

In more detail, RINA's DIF model seems to have a lot of promising answers to problems arising with the adoption of NFV in an operator's environment: security beyond known locations and static nodes, resilience based on mechanisms beyond the physical node availability, and the elasticity that constitute the core feature of virtualization techniques. Furthermore, NFV requires a coherent abstraction layer capable to support unified interactions of (potentially recursive) services and components.

4.2. Detailed description

It is obvious that a disruptive, clean-slate technology like RINA would have a difficult way within a NSP environment, very much oriented towards service provisioning and stability, and the phased incorporation of thoroughly tested technologies. But the recent advent of the NFV proposal opens a very interesting window of opportunity for demonstrating RINA in the NSP environment, and showcase the advantages that the combination of both approaches (NFV and RINA) can bring to network service design, management, and operation. To achieve this showcase effect, we considered a scenario where RINA will be applied to build the service graphs and VNF internal connections, as well as providing support for resiliency mechanisms. RINA will not be applied in this scenario to the external attachment points to the NSP network of the service, that will be out of the scope of this use case.

4.2.1. Defining the scope

To define the precise scope of this use case let's get back to the diagram describing the NFV layers and analyze where RINA is intended to be applied:

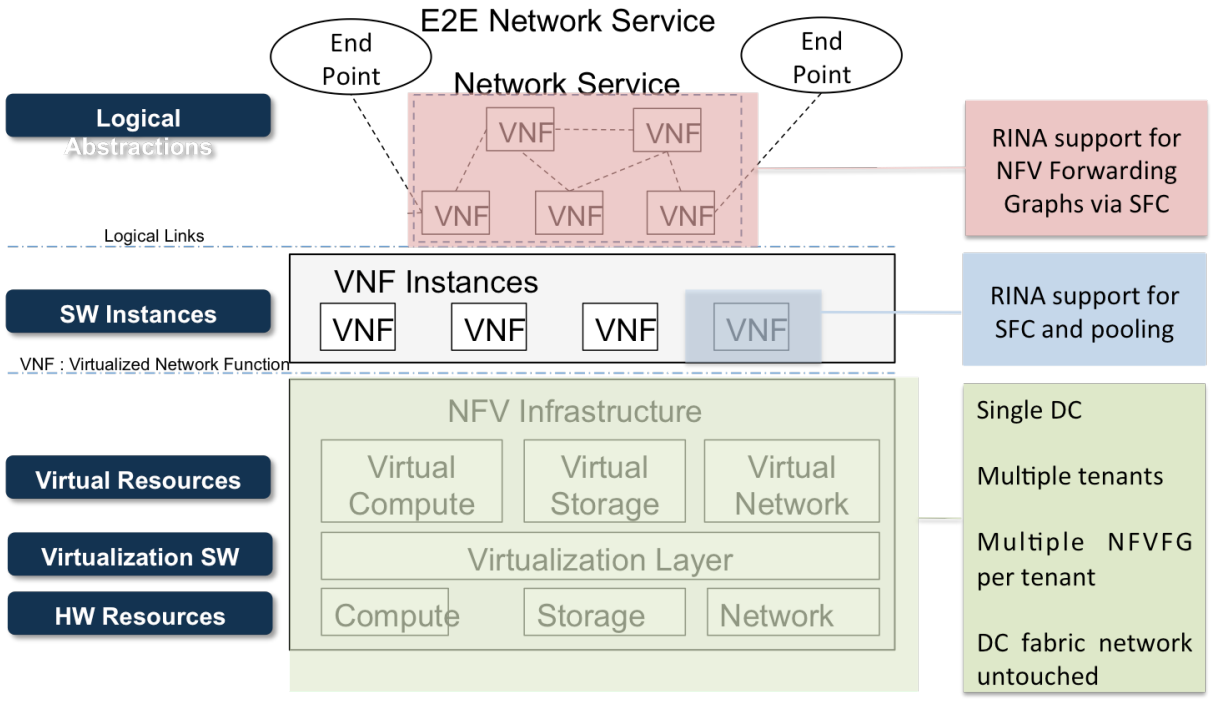

Figure 41. NFV layers and RINA

At the top layer of the NFV framework, RINA is intended to support service construction by VNF composition, and we have chosen to explore the most promising technique for achieving it: a recently new activity within IETF known as *Service Function Chaining* (SFC) [ietfsfc]. In the SFC framework, network services are built

by paths of *service nodes* attached to the rest of the network by means of *boundary* (*ingress* and *egress*) nodes.

At the intermediate layer of the NFV framework, we will assume that each service node will be implemented by a VNF and RINA will be applied to facilitate the necessary management traffic to orchestrate and manage the different VNF components.

Finally, at the infrastructure layer of the NFV framework our intention is to demonstrate the applicability of RINA to implement a seamless pooling mechanism to support the enhanced scalability and resiliency that constitute one of the key features NFV will bring to network infrastructures and services.

The use case will assume all VNFs are deployed on a single datacenter infrastructure, independently of the connection fabric substrate, whether it is based on RINA as well or not (the appropriate DIFs, native or shim, will be available) This single datacenter is the only limitation we have consider for the sake of simplicity. Multiple tenants for the network services, as well as multiple network services and VNFs per tenant are within the scope, as the support for tenant separation and security is one of the key aspects we want to demonstrate.

Implementing Network Services by SFC and Pooling

As said above, a network service is implemented by chaining a set of service nodes (all supported by a single datacenter infrastructure in this use case) The chain is attached to the network by two boundary nodes, one for ingress and another one for egress. These nodes have a few special characteristics:

- They are the only nodes in any service chain connected to the rest of the NSP network.
- As they participate in the external NSP network, they are in charge of the adaptation of the external traffic (non-RINA: IP, MPLS…) to the internal RINA-based traffic.
- While the rest of nodes are required to be implemented by means of VNFs, boundary nodes can be (partially) supported by non-virtualized elements (as Physical Network Function, PNF).

According to the SFC architecture, each service node has an element in charge of applying the appropriate policies to construct dynamically the path in the chain for each traffic flow, plus several elements implementing the actual functionality of the node and oblivious to their participation in the chain(s) they are included within. The former element is known as Service Function Forwarder (SFF), while the elements

implementing node functionality are referred simply as Service Function (SF). In this use case we consider each service node to be mapped to a VNF, so the SFF and the corresponding SFs within a node will become VNFCs, that is: each element within a service node will be implemented by a VM hosted in the datacenter. It is worth noting that this service node architecture can collapse to a single element, implementing the SFF and a unique SF.

To leverage the characteristics of the virtualized infrastructure in order to provide enhanced resiliency and elasticity a pooling mechanism will be applied. Each VNFC will be implemented by a pool of similar VMs, generated by instantiating and starting the same software image. A Pool Manager (PM) will monitor the pool status and allocate, according to policy requirements and allocate the resources of the pool to the VNF (and therefore the service) it is participating in.

The following diagram illustrates these three layers:

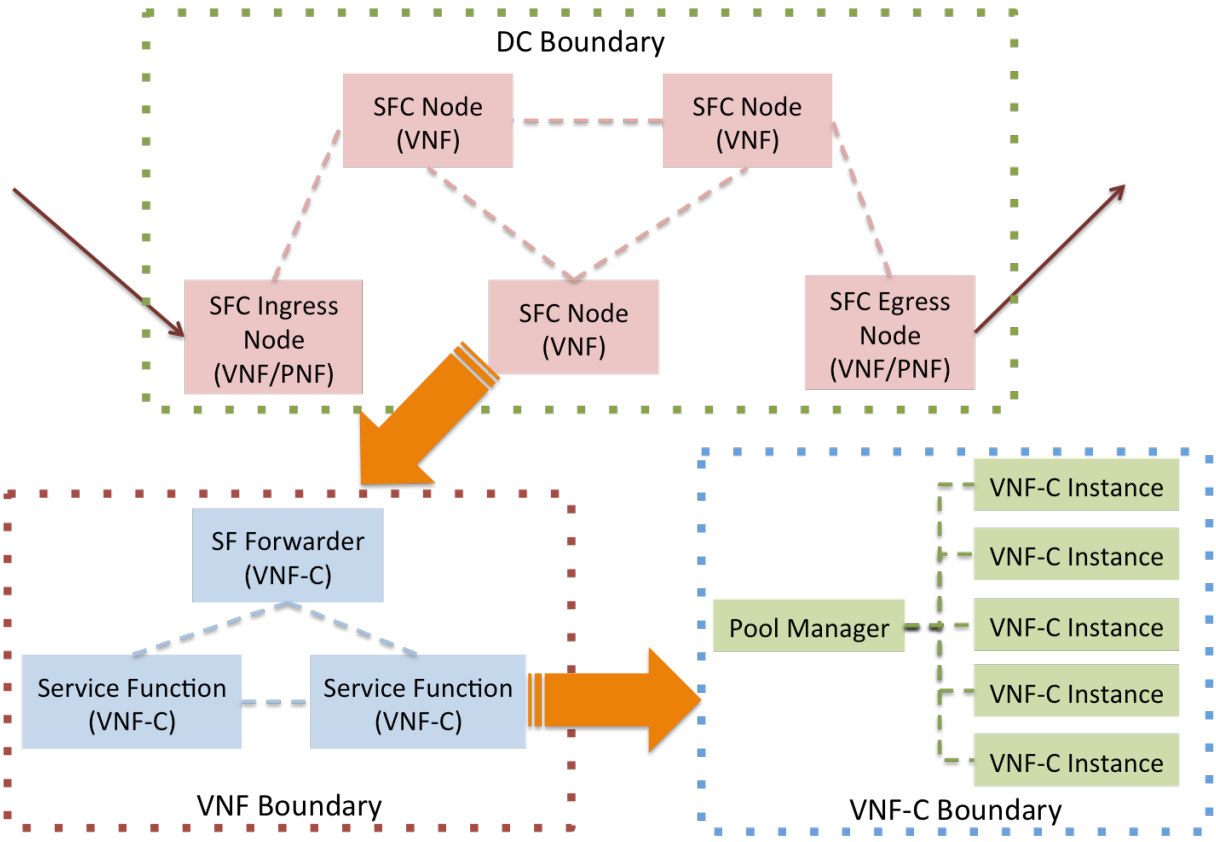

Figure 42. SFC pooling layers

Orchestration

The following figure depicts the NFV reference architecture, showing in the right hand side the elements contributing to the Management and Orchestration (MANO) of the NFV framework

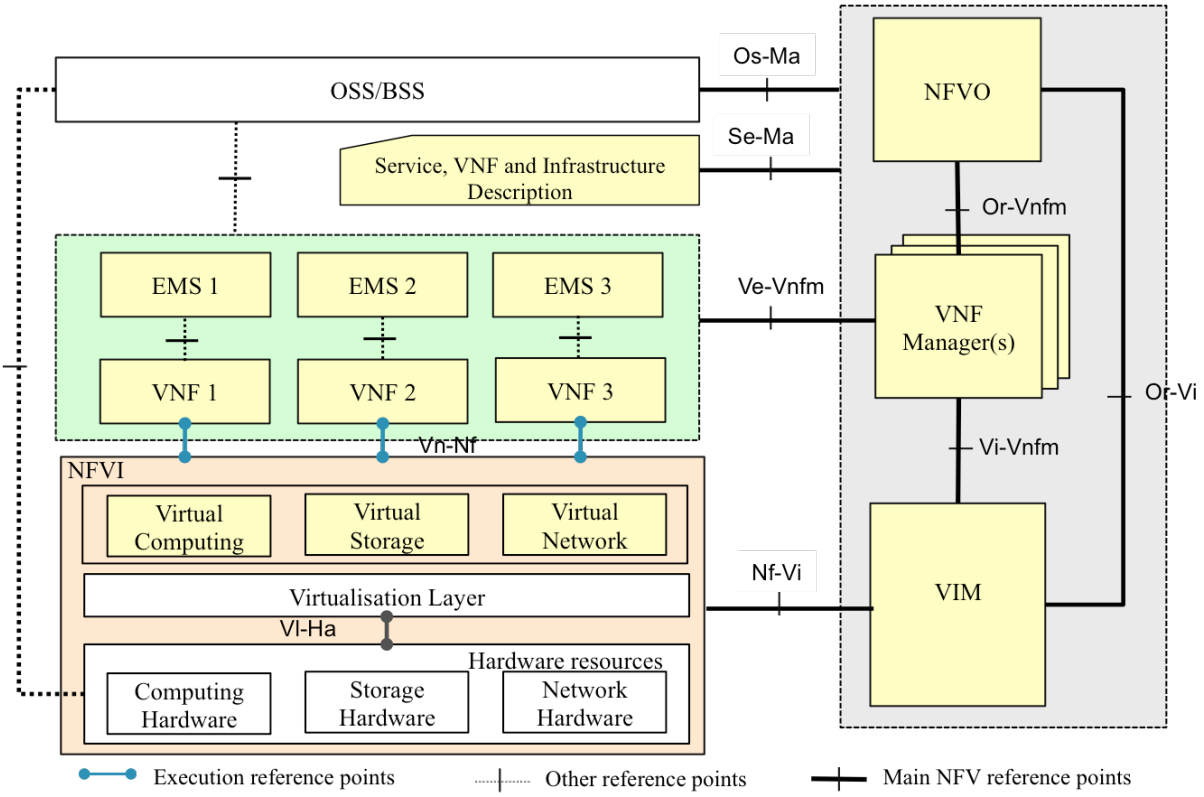

Figure 43. NFV reference architecture

The role of the different MANO elements in this context is as follows:

- The NFVO has to
  - Manage service chain creation, and the registration of required VNF instances to them
  - Configure the boundary nodes so that they inject traffic to the right service chains
  - Communicate policies to the intervening VNFMs
  - Oversee SLA and performance
- The VNFM has to
  - Manage VNF instantiation, including the required components and their interconnection
  - Configure the SFF to perform appropriate internal and chain forwarding
  - Communicate policies to the PMs of the VNFCs
  - Manage VNF lifecycle events
- The VIM has to
  - Support the PMs for monitoring and scaling of the pool
  - Instantiate the necessary infrastructure resources

- Forward to the PMs events and faults affecting VNFC (VM) instances

One of the main advantages we foresee for RINA is the possibility of simplifying the interactions among the different MANO entities, and the infrastructural and functional elements.

4.2.2. Impact of this Use Case

This use case investigates the following aspects while trying to use RINA for NVF:

- VNF construction: what DIF models and policies are required?
- VNF composition: what DIF models and policies are required?
- Non-functional VNF and service requirements: how does RINA satisfy them
  - Tenant separation, no longer available by physical isolation
  - Resiliency, by automated reconfiguration around failures
  - Elasticity, by seamless adaptation to scaling events of the virtualized components
- VNF programming/execution environment: how can this be supported by / harmonized with RINA architectural concepts?
- NVF orchestration: to what degree will RINA simplify it?
- Authentication
  - Authenticating IPCs as part of enrolment in a DIF at runtime
  - Authenticating network administrators to applications for configuring the network
- Access Control
  - Deciding the mechanisms for an IPC to enrol in a DIF according to service and VNF policies
  - User access to admin/configuration applications
  - VNF instances DIF to VM slices DIF
- Content-based Security
  - Integrity of control and management messages in terms of checking authenticity and integrity of messages

The answers to the above aspects should result in at least one NVF proof of concept demonstration.

4.3. Issues and limitations

4.3.1. Extending DC limitations

The NFV infrastructure is essentially based on DC technologies and their orchestration mechanisms, so all the limitations described for the DC use case do apply here, with some specific emphasis in some of them.

Oversubscription

The nature of network functions and their strict requirements on response times and resiliency makes oversubscription a natural trend. This is made stronger by the, so to say, cultural tradition of using oversubscription in traditional current network deployments based on physical functions.

Multi-tenancy and Flow Isolation

All network functions are, as defined in the DC use case, applications with strict QoS demand. The considerations for supporting multi-tenancy and reliable flow separation apply to all the NFV implementations.

4.3.2. Multi-layer Mapping

The construction of network services by composing virtualised network functions that are implemented by pools of VMs requires two mappings: from the SFC layer to the VNF internals, and from the VNF layer to the particular element in the pool. Given that the current techniques proposed to implement the layers are essentially based on overlays, there is a need of implementing overlay mapping, increasing complexity and adding the risk of "least common multiple" design.

4.3.3. Security Issues

The virtualisation of network functions challenges many of the common practices in network security, very often based in physical isolation of certain components.

Topologies must be validated and enforced: A network operator will need to be able to validate that the connectivity of all the virtualised functions matches the intended topology. If port A must connect to port B via a firewall function then a shaping function, it must be possible to validate that.

It is also necessary to be able to check for any connectivity that should not be present. But that is not enough – it must also be possible to prevent unauthorised connectivity

being added, and to prove that it cannot be added by an unauthorised party, avoiding breaches in the control plane.

Maintaining an appropriately isolated management network infrastructure is another key challenge, otherwise events notifying problems cannot be conveyed to the management system, and the management system has no way to remotely re-start failed services. Access control and guaranteed minimal properties are essential.

The infrastructure network must provide support for secure boot and secure crash of the software images implementing the virtualised functions. Think of functions needing network access for its own config. It would certainly be dangerous to start giving virtualised processes access to the separate management network. On the other hand, a crash of one node can leave dangling references to the location where the virtual function was before it crashed on surrounding network elements.

Authentication, Authorisation, and Accounting

The introduction of NFV brings new issues when it comes to AAA, as it implies using the current identity and accounting facilities at least at two layers: the network infrastructure (identifying the tenant) and the network function (identifying the actual user). What is more: the multi-layer approach proposed to support composition and pooling suggests that the stacking of identities can occur at more layers.

A generalized AAA schema for identifying users utilizing a particular tenant infrastructure and/or a tenant acting on behalf of a user is required to support these patterns and the new operational business models they will bring. Current AAA mechanisms assume there will be a single identity, single policy decision and enforcement points, a single level of policy, and a single accounting infrastructure. Even if a strict separation by tunneling (as current practice seems to suggest) would be feasible without breaking some of the promised NFV enhancements in what relates to scalability, agility and resiliency, there exist risks related at each one of the three AAA functions.

4.3.4. Trombone Routing

Many of the techniques originally proposed for achieving service chaining imply different degrees of trombone routing (or tromboning for short), where packets are sent back and forth through the same link to allow the different function elements to process them. Several proposals to alleviate tromboning has been made, but they imply either additional limitations to compositional patterns or they bring into play some kind

of *omniscient* entity. This can become exacerbated by the application of the pooling mechanisms, as they can require an additional lay er for redirections.

4.3.5. Complex Orchestration

Achieving enhanced elasticity and resiliency requires an appropriate orchestration of the virtualised infrastructures, down from the service definition layer. As said above, different overlay techniques are applied at the different layers and that translate into the need for adaptation in the communication between the different MANO entities, with the corresponding additional complexity this implies.

4.4. Applying RINA to the use case: requirements analysis

4.4.1. Overview

This section presents a possible implementation of the distribution of legacy traffic (e.g. IPv4) to the NFV chains using RINA. The proposal is to model all the different functional elements of the NFV architecture, Orchestrator, Pool manager, DFF and DFC nodes, as application processes (APs).

Some APs such as SFF or SFCin, SFCout may run indistinctly in either VM or physical appliances. In order to simplify diagrams though, they are drawn always either in a VM or directly in a physical server. Other simplifications made in the diagram are having a single SFCin and SFCout AP, or having a very simplified DIF structure (e.g. shim DIFs).

The design is split in the two different planes:

- The *Management stack*: in charge of NFV chain life-cycle management (instantiation, configuration, deployment, monitoring and undeployment).
- The *Service stack*: in charge of the transport along of the legacy traffic through the NFV chain, and the processing.

Service stack

This deliverable exposes two different models of the service stack. These are not the only possibilities. The service stack is critical in terms of reliability, performance - such as bandwidth -, and it can be sensitive to delay and jitter, depending on the VNFs.

Service stack, model A

In this model each NFV chain is modeled as an heterogeneous DAF, the so called chain DAF (blue Service Chain 1 DAF).

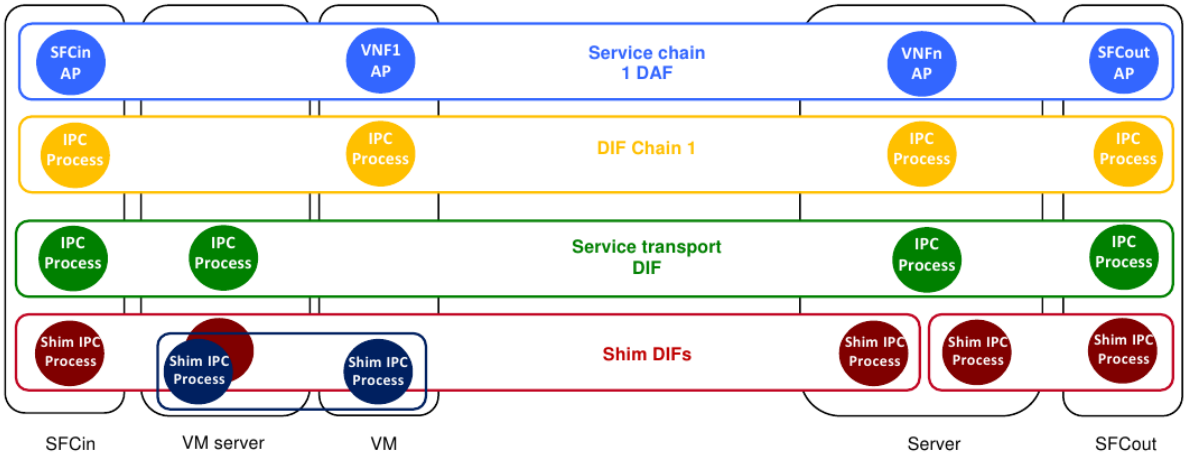

Figure 44. Service stack model A

This DAF contains fundamentally the following APs:

- the ingress AP or SFCin
- the different Virtual Network Functions(VNFs) APs, VNF1...VNFn
- the egress AP or SFCout

At the same time, each VNF is also modeled as a DAF (VNF1...VNFn):

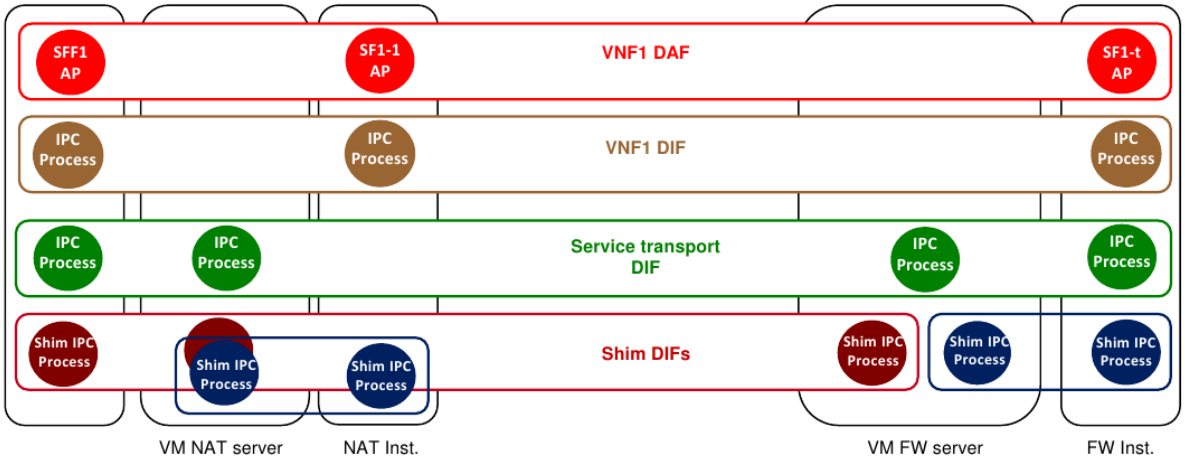

Figure 45. Service stack model A

In the example there is a VNF1 composed by a NAT instance (SF1-1), a certain number of other SFs and a final firewalling instance (SF1-t). The total number of DAFs for each chain is therefore n+1, being n the number of VNFs in the chain, and 1 for the chain DAF itself.

Packet flow

From the RINA point of view, the legacy traffic (IPv4/Ipv6) is received from the ingress point(SFCin) and encapsulated as a PDU. From what regards to the RINA domain, the legacy traffic is nothing but application payload. SFCin immediately delivers the PDU to the first VNF(VNF1) of the chain. The AP VNF1 will internally process the PDU according to the function needs. To do so, the AP VNF1 sends the PDU to the SFF1 AP that belongs to the VNF1 DAF. Eventually a specific DAF between *only* VNF1 AP and SFF1 AP, and a supporting shim DIF may be created to allow both APs to exchange PDUs.

It is important to clarify that the component called Service Function Forwarders (SFF) in NFV SFC architecture is split between the VNFi and SFFi APs. VNFi handles the PDU to its corresponding SFFi, and is in charge of forwarding the PDU to the next VNFi +1. For security reasons it is proposed to instantiate a different SFF in each service chain((VNFi + SFFi) instead of having one per function. In this way any failures or attacks that the SFFi could suffer would affect only the chain where it belongs to.

Once the PDU enters the VNF DAF, SFF decides the particular path over the Service Functions (SF) that the packet needs to be processed by. This is pre-configured by the management DAF, and/or may eventually, depending on the SF, depend upon the the contents of legacy packet(IPv4/IPv6) itself. Each SF decapsulates the legacy traffic packet, processes it, and eventually re-encapsulates it in another PDU to be sent back to the SFFi. At any time, one of these SFs may also decide to drop the PDU.

Once the PDU has fully been processed by a VNF, e.g. VNF1, the AP VNF1 decides to send to the next AP VNF2, and so forth. The last element of the chain, will eventually send the packet to the SFCout for decapsulation and re-injection in legacy networks.

Service stack model B

The design of the service stack model B embeds the VNF APs (all) in a single much more heterogeneous than in model A, DAF; the service chain DAF. For simplicity the colors from the APs that were belonging to the VNF1..N DAFs are the same.

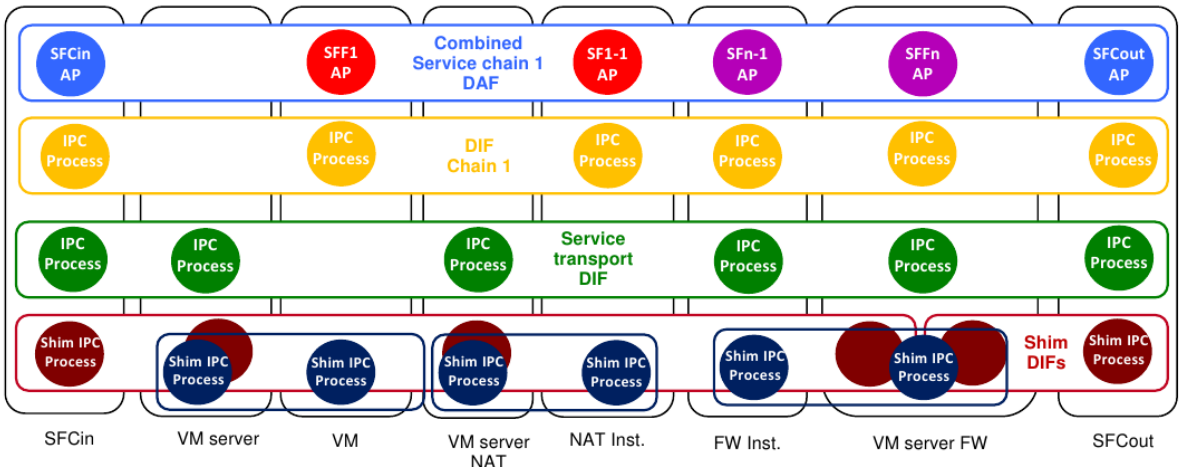

Figure 46. Service stack model B

This model presents a simplified layer structure, at the cost of having a more complicated security management, since not all the APs can communicate (i.e. allocate flows) to each other. A clear example are the VNFi APs (SFi-1…SFi-t) can only communicate among each other and to the SFFi.

Service stack common elements

Both service stack models are supported by the same transport DIFs. Underneath every service chain DAF there is a DIF chain (yellow) that contains only the IPC processes of the chain. The inherent RINA layered design allows the different NFV chains to be isolated between each other, and hence protect themselves from misbehaving elements in other chains

The chain DIF is supported by a service transport DIF (green) that is unique for all the chains. It is in this DIF where the relaying between the lowest level shim DIFs, physical (red), and virtual (blue), is done.

The exact configuration of the transport DIFs is very much dependent of the deployment scenario, e.g. which APs run in a VM or in a physical appliance, physical topology… It is important to note that these shim DIFs, due to its nature could map directly over Ethernet, but they could perfectly be deployed on top TCP/IP, MPLS, GMPLS or any other legacy technology, as well as rawly on top of the hardware (native RINA).

Management stack

The management stack is in charge of allowing communication of the different APs that are in charge of managing the service chains. The management is a completely separate Distributed Application Facility, purely in charge of bootstrapping, instantiating,

configuring and deallocation VNF chain resources. These are, among others: the orchestrator, the SFCin and SFCout nodes, the necessary VM servers that support non-SF functions, the specific VM servers that support SF functions, the pool managers and the SF VMs.

Therefore, the design defines all these elements required for proper functioning of the system as APs processes that belong to the same DAF, in this case the management DAF. The Figure below shows how this DAF is build.

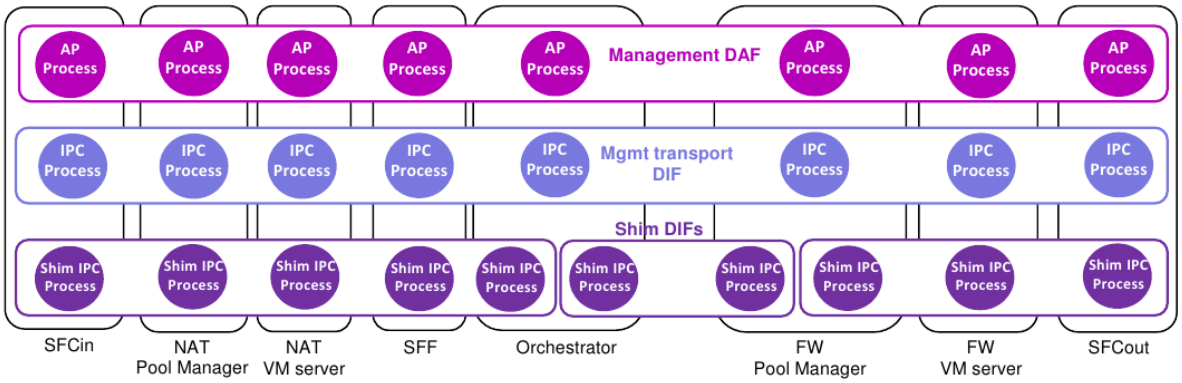

Figure 47. Management stack

The APs in this DAF consume the services of the transport DIFs. In general, this DIF (or DIFs) may be built on top of existing transport DIFs (e.g. Service stack transport DIf), or dedicated out-of-band management DIFs, as shown in the diagram. However, due to its requirements in terms of reliability, it is assumed that there will always be an out-of-band, perhaps redundant (not shown in the diagram), to ensure management over the different components can always be performed.

4.4.2. Requirements analysis

Once the mapping of the NFV into the RINA architecture is described and the roles of the DIFs and DAFs involved defined we must list what are the specific requirements that each type of them must meet.

Shim DIFs

The shim DIFs could in principle be on top of Ethernet, or TCP/IP or any other raw hardware. Another type of Shim DIFs are the ones that connect the VM with the Servers.

- **Shim DIF over Ethernet**. Allow RINA DIFs to be deployed over plain Ethernet or 802.1q layers (VLANs). Should support QoS (at least bandwidth allocation).
- **Shim DIF over TCP/UDP**. Allow DIFs to be deployed over IPv4 and IPv6 layers using TCP and UDP.

- **Shim DIF for VM**. Allow communication to the deployed VMs. Should support QoS (at least bandwidth allocation).

Service Transport DIF (ST-DIF)

The scope of this type of DIF is the Data Center. It is supposed to lay on top of different lower DIFs (shim DIFs) and provide a way to communicate between all the elements (servers and VMs).

- **ST-DIF-1**. The DIF needs to authenticate the joining IPC Processes.
- **ST-DIF-2**. The DIF needs to provide reliable traffic, deliver PDUs in order and without loss.
- **ST-DIF-3**. The DIF will provide the requested QoS (BW, delay and jitter) by the upper layer DIFs, given that the resources are available.
- **ST-DIF-4**. A congestion avoidance scheme must be run to protect the resources of the layer.
- **ST-DIF-5**. The DIF should support scaling up and down. Adding new Servers or VMs should be handled with no pre-configuration.
- **ST-DIF-6**. The DIF must have a routing algorithm with fast convergence so it supports failures as well as the addition/removal of new machines.
- **ST-DIF-7**. The DIF must provide addressing dynamically to the new IPCs.
- **ST-DIF-8**. The DIF must support multiple paths and proper balancing over them.
- **ST-DIF-9**. The different DIFs that lay upon this one must be properly isolated.

Service Chain DIF (SC-DIF) - model A

- **SC-DIF-1**. The DIF needs to authenticate the joining IPC Processes.
- **SC-DIF-2**. The DIF needs to provide reliable traffic, deliver PDUs in order and without loss.
- **SC-DIF-3**. The DIF will provide the requested QoS (BW, delay and jitter) by the upper layer DIFs, given that the resources are available.
- **SC-DIF-4**. A congestion avoidance scheme must be run to protect the resources of the layer.
- **SC-DIF-5**. The DIF should support scaling up/down and manage the adding/ removal of IPCs.

- **SC-DIF-6**. The DIF must provide addressing dynamically to the new IPCs.

Service Chain DAF (SC-DAF) - models A and B

- **SC-DAF-1**. The DAF needs to authenticate the joining Application Processes.
- **SC-DAF-2**. The DAF requires reliable traffic, deliver PDUs in order and without loss.
- **SC-DAF-3**. The DAF should request the required QoS cube required by the SF in terms of bandwidth, delay, jitter.
- **SC-DAF-4**. A congestion avoidance scheme must be run to protect the resources of the layer.
- **SC-DAF-5**. The DAF should support scaling up/down and manage the adding/ removal of APs.
- **SC-DAF-6**. The DAF must provide addressing dynamically to the new APs.

Virtual Network Function DIF (VNF-DIF) - model B

- **VNF-DIF-1**. The DIF needs to authenticate the joining IPC Processes.
- **VNF-DIF-2**. The DIF needs to provide reliable traffic, deliver PDUs in order and without loss.
- **VNF-DIF-3**. The DIF will provide the requested QoS (BW, delay and jitter) by the upper layer DIFs, given that the resources are available.
- **VNF-DIF-4**. A congestion avoidance scheme must be run to protect the resources of the layer.
- **VNF-DIF-5**. The DIF should support scaling up/down and manage the adding/ removal of IPCs.
- **VNF-DIF-6**. The DIF must provide addressing dynamically to the new IPCs.

Virtual Network Function DAF (VNF-DAF) - model B

- **VNF-DAF-1**. The DAF needs to authenticate the joining Application Processes.
- **VNF-DAF-2**. The DAF requires reliable traffic, deliver PDUs in order and without loss.
- **VNF-DAF-3**. The DAF should request the required QoS cube required by the SF in terms of bandwidth, delay, jitter.

- **VNF-DAF-4**. A congestion avoidance scheme must be run to protect the resources of the layer.
- **VNF-DAF-5**. The DAF should support scaling up/down and manage the adding/removal of APs.
- **VNF-DAF-6**. The DAF must provide addressing dynamically to the new APs.

Combined Service Chain DAF (CSC-DAF) - model B

- **SC-DAF-1**. The DAF needs to authenticate the joining Application Processes.
- **SC-DAF-2**. The DAF requires reliable traffic, deliver PDUs in order and without loss.
- **SC-DAF-3**. The DAF should request the required QoS cube required by the SF in terms of bandwidth, delay, jitter.
- **SC-DAF-4**. A congestion avoidance scheme must be run to protect the resources of the layer.
- **SC-DAF-5**. The DAF should support scaling up/down and manage the adding/removal of IPCs.
- **SC-DAF-6**. The DAF must provide addressing dynamically to the new IPCs.
- **SC-DAF-7**. The DAF must provide mechanisms to only allow to requested flows between certain AP (according to the chain). That means, for example, that flow requests between Service Functions of different Virtual Network Functions should be denied.

Management Transport DIF (MT-DIF)

- **MT-DIF-1**. The DIF needs to authenticate the joining IPC Processes.
- **MT-DIF-2**. The DIF needs to provide reliable traffic, deliver PDUs in order and without loss.
- **MT-DIF-3**. A congestion avoidance scheme must be run to protect the resources of the layer.
- **MT-DIF-4**. The DIF should support scaling up and down. Adding new Servers or VMs should be handled with no pre configuration.
- **MT-DIF-5**. The DIF must have a routing algorithm with fast convergence so it supports failures as well as the addition/removal of new machines.
- **MT-DIF-6**. The DIF must provide addressing dynamically to the new IPCs.

- **MT-DIF-7**. The DIF must support multiple paths and proper balancing over them.

Management DAF (M-DAF)

- **M-DAF-1**. The DAF needs to authenticate the joining Application Processes..
- **M-DAF-2**. The DAF requires reliable traffic, deliver PDUs in order and without loss.
- **M-DAF-3**. The DAF should support scaling up/down and manage the adding/ removal of APs.
- **M-DAF-4**. The DAF must provide addressing dynamically to the new APs.
- **M-DAF-5**. The DAF must be able to create, destroy and configure new Service chain DIFs and DAFs.
- **M-DAF-6**. The DAF must be able to monitor the Service Chain DIFs and DAFs and reconfigure them dynamically.

Bibliography

[abbes08] H. Abbes, C. C´erin, and M Jemni. *Bonjourgrid as a decentralised job scheduler*. Proceedings of the 2008 IEEE Asia-Pacific Services Computing Conference, pages 89–94,Washington, DC, USA, 2008. IEEE Computer Society.

[abbes09] H. Abbes, C. C´erin, and M. Jemni. *Bonjourgrid : Orchestration of multi-instances of grid middlewares on institutional desktop grids*. 3rd Workshop on Desktop Grids and Volunteer Computing Systems (PCGrid2009), May 2009.

[abbes10] H. Abbes, C. C´erin, M. Jemni. *A decentralized and fault-tolerant Desktop Grid system for distributed applications*. Concurrency and Computation: Practice and Experience 22(3): 261-277 (2010).

[albrightson] B. Albrightson, J. Garcia Luna and J. Boyle. 'EIGRP–a Fast Routing Protocol Based on Distance Vectors". Proceedings of Interop 94, 1994.

[alfares] M. Al-Fares et al. *A Scalable, Commodity Data Center Network Architecture*. SIGCOMM, 2008.

[alizadeh] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan. *Data Center TCP (DCTCP)*. SIGCOMM 2010.

[amazon] *Amazon Web Services: Overview of Security Processes (white paper)*. Amazon, November 2013. Available online[7]

[bari] Bari, Boutaba et al. *Datacentre network virtualization: a survey*. IEEE Communications Surveys and Tutorials, Vol. 15, Issue 2, pp.909-928, 2nd Quarter of 2013.

[beaugnon] U. Beaugnon. *Building a resilient overlay network: re6stnet*. Technical report, available online[8]

[beer] K. Beer, R. Holland. *Securing Data at Rest with Encryption (white paper)*, Amazon, November 2013. Available online[9].

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

[7] http://media.amazonwebservices.com/pdf/AWS_Security_Whitepaper.pdf
[8] http://www.nexedi.com/P-ViFiB-Resilient.Overlay.Network/Base_download
[9] http://media.amazonwebservices.com/AWS_Securing_Data_at_Rest_with_Encryption.pdf

[bollobas] B. Bollobas. *Random graphs*. Cambridge University Press, October 2001. ISBN: 9780521797221.

[bradford] R. Bradford, E. Kotsovinos, A. Feldmann and H. Schiöberg. *Live Wide Area migration of virtual machines including local persistent state*, VEE 2007.

[chen] K. Chen et al. *Survey on Routing in Data Centers: Insights and Future Directions*. IEEE Network, Volume 25 Issue 4, 2011.

[cisco] Cisco Systems. *Data center: Load balancing data center services*. 2004. Available online[10]

[ciscodp] Cisco Discovery Protocol. Available online[11].

[courteaud] R. Courteaud and Y. Xu. *Practical solutions for resilience in SlapOS*. 4th IEEE International Conference on Cloud Computing Technology and Science, December 2012. Available online[12].

[daniels] J. Daniels. *Server virtualization technology and implementation*. ACM Crossroads. 2009.

[davie] B. Davie, J. Gross. *A stateless transport tunneling protocol for Network Virtualization*. March 2012. Available online[13].

[delavega] W. Fernandez de la Vega and B. Bollobas. *The diameter of random regular graphs*. Combinatorica, 1981.

[etsinfv] ETSI. *Network Functions Virtualisation – Update*. White Paper. Available: online[14].

[nfvarch] ETSI. *Network Functions Virtualisation (NFV); Architectural Framework, GS NFV 002, V1.1.1*. October 2013. Available online[15].

[google] *Security Whitepaper: Google Apps Messaging and Collaboration Products*. Available online[16]

[10] https://learningnetwork.cisco.com/servlet/JiveServlet/previewBody/3438-102-1-9467/cdccont_0900aecd800eb95a.pdf
[11] http://www.cisco.com/c/en/us/support/docs/network-management/discovery-protocol-cdp/43485-cdponios43485.html
[12] http://www.computer.org/csdl/proceedings/cloudcom/2012/4511/00/06427511-abs.html
[13] http://tools.ietf.org/html/draft-davie-stt-01
[14] http://portal.etsi.org/NFV/NFV_White_Paper2.pdf
[15] http://www.etsi.org/technologies-clusters/technologies/nfv
[16] http://www.google.com/enterprise/apps/business/resources/docs/security-whitepaper.html

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

[gribble] S.D. Gribble, S. Saroiu and P.K. Gummadi. *A measurement study of peer-to-peer file sharing systems*. Proc.Multimedia Computing Networking, 2002

[guobcube] C. Guo et al. *BCube: A High Performance Server-centric Network Architecture for Modular Data Centers*. SIGCOMM, 2008.

[guodcell] C. Guo et al. *DCell: A Scalable Fault Tolerant Network Structure for Data Centers*. SIGCOMM, 2008.

[harney] E. Harney, S. Goasguen, J. Martin, M. Murphy and M. Westall. *The Efficacy of Live Virtual Machine Migrations Over the Internet*. VTDC'07, November 2007.

[hoffman] E. Hoffman, J. Snell, T. Anderson, S. Savage S and A. Collins. *The end-to-end effects of internet path selection*. SIGCOMM, 1999.

[ietfsfc] IETF, *Service Function Chaining Working Group*. Available online[17].

[802.1AB] *IEEE 802.1AB-2012. Station and Media Access Control for connectivity discovery*. Available online[18].

[802.1ad] *IEEE 802.1ad Provider Bridges*. Available online[19].

[802.1ah] *IEEE 802.1ah Provider Backbone Bridges*. Available online[20].

[802.1aq] *IEEE 802.1aq Shortest Path Bridging*. Available online[21].

[802.1Qbg] *IEEE 802.1Qbg Edge Virtual Bridging*. Available online[22].

[lidzborski] N. Lidzborski. *Staying at the forefront of email security and reliability*. Available online[23].

[liu] X. Liu. *Low latency datacentre networking: a short survey* (December 2013). Available online[24].

[17] http://datatracker.ietf.org/wg/sfc/charter/
[18] http://www.ieee802.org/1/pages/802.1ab.html
[19] http://www.ieee802.org/1/pages/802.1ad.html
[20] http://www.ieee802.org/1/pages/802.1ah.html
[21] http://www.ieee802.org/1/pages/802.1aq.html
[22] http://www.ieee802.org/1/pages/802.1bg.html
[23] http://googleblog.blogspot.co.uk/2014/03/staying-at-forefront-of-email-security.html
[24] http://arxiv.org/abs/1312.3455

[lopumo] A. Lo Pumo. *Scalable mesh network and the address space balancing problem*. University of Cambridge, 2010.

[mahalingam] M. Mahalingam, D. Dutt, K. Duda, P. Argawal, L. Kreeger, T. Sridhar, M. Bursell, C. Wright. *VXLAN: A Framework for overlying virtualized L2 networks over L3 networks*. August 2011. Available online[25].

[mahlmann] P. Mahlmann, C. Schindelhauer. *Random Graphs for Peer to Peer Overlays*. Available online[26]

[mazieres] D. Mazieres and P. Maymounkov. *Kademlia : a peer-to-peer information system based on the xor metric*. First International Workshop on Peer-to-Peer Systems, 2001.

[ngo] H.Q. Ngo, G.W. Peck, D.Z. Du and D.F. Hsu. *On connectivity of consecutive-d digraphs*. Discrete Mathematics, 2002

[niranjan] R. Niranjan et al. *PortLand: A Scalable Fault-Tolerant Layer 2 Data Center Network Fabric*. SIGCOMM, 2009.

[oguchi] Y. Oguchi and T. Yamamoto. *Sever virtualization Technology and its Latest Trends*. Fujitsu Sci. Tech J., January 2008.

[openvpn] Openvpn website. Available online[27].

[pepelnjak] I. Pepelnjak. *Overlay Virtual Networking explained*. PLNOG 2011. Available online[28]

[perkins] C. Perkins and P. Bhagwat. *Highly Dynamic Destination-Sequenced Distance-Vector Routing (DSDV) for Mobile Computers*. ACM SIGCOMM'94 Conference on Communications Architectures, Protocols and Applications, 1994.

[rai] R. S. Ganesh, D. Loguinov and A. Kumar. *Graph-theoric analysis of structured peer-to-peer systems: Routing distance and fault resilience*. Texas A & M Technical Report, 2003.

[re6stdoc] *re6st documentation*. Available online[29].

[25] http://tools.ietf.org/html/draft-mahalingam-dutt-dcops-vxlan-00
[26] http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.414.9387
[27] http://openvpn.net/
[28] http://demo.ipspace.net/get/Overlay%20Virtual%20Networking%20Explained%20-%20PLNOG11.pdf
[29] http://git.erp5.org/gitweb/re6stnet.git/blob/HEAD:/README?js=1

[re6stsrc] *re6st source code*. Available online[30].

[rfc6325] R. Perlman, D. Eastlake, D. Dutt, S. Gai, A. Ghanwani. *RFC 6325: Routing Bridges (RBridges). Base protocol specification*. July 2011. Available online[31]

[sayer] P. Sayer. *VIFIB wants you to host cloud computing at home*. Cio.com, June 2010 Avalilable online[32].

[scalet] S. Scalet. *19 Ways to Build Physical Security into a Data Center*. Available online[33].

[shieh] A. Shieh, S. Kandula, A. Greenberg, C. Kim. *Seawall: performance isolation for cloud datacenter networks*. Hotcloud 2010. Available online[34].

[slaposdoc] SlapOS Community. *SlapOS documentation*. Available online[35].

[smets] J. P. Smets, C. Cerin and R. Courteaud. *SlapOS: A multi-purpose cloud operating system based on an ERP billing model*. IEEE 2011 International Workshop on Performance Aspects of Cloud and Service Virtualization, 2011. Available online[36].

[sperling] E. Sperling. *Next-Generation Data Centers*. Forbes, March 2010. Available online[37].

[spring] N. Spring, C. Lumezanu and D. Levin. *Peerwise discovery and negotiation of faster paths*. HotNets, 2007

[sridharan] M. Sridharan, A. Greenberg, Y. Wang, P. Garg, N. Venkataramiah, K. Duda, I. Ganga, G. Lin, M. Pearson, P. Thaler, C. Tumuluri.*NVGRE: Network Virtualization Using Generic Routing Encapsulation*. August 2013. Available online[38].

[30] http://git.erp5.org/gitweb/re6stnet.git/tree?js=1
[31] http://tools.ietf.org/html/rfc6325
[32] http://www.cio.com/article/596689/ViFiB_Wants_You_to_Host_Cloud_Computing_At_Home
[33] http://www.csoonline.com/article/2112402/physical-security/19-ways-to-build-physical-security-into-a-data-center.html
[34] http://research.microsoft.com/en-us/um/people/srikanth/data/hotcloud10_seawall.pdf
[35] http://community.slapos.org/wiki/osoe-Lecture.SlapOS.Extended
[36] http://community.slapos.org/slapos-Wiki.Home/slapos-Smets.Cerin.Courteaud.IEEECloudPerf2011?format=
[37] http://archive.today/DlZQ
[38] http://tools.ietf.org/html/draft-sridharan-virtualization-nvgre-03

[taoup] E. Raymond. *The Art of Unix Programming*. Addison-Wesley. ISBN 0-13-142901-9.

[vifibweb] *VIFIB web page*. Available online[39].

[vmware] *VMWare vCloud suite*. Available online[40].

[volk1] T. Volk. *Software defined datacenter basics*. Available online[41].

[volk2] T. Volk. *Sofware defined datacenter core components*. Available online[42].

[volk3] T. Volk. *Software defined datacenter today*. Available online[43].

[walsh-muellner] N. Walsh and L. Muellner. *DocBook - The Definitive Guide*. O'Reilly & Associates. 1999. ISBN 1-56592-580-7.

[wenedelin] *Wendelin project website*. Available online[44].

[white] J.L. White, "Technical overview of datacenter networking", 2012. Available online[45]

[39] http://www.vifib.com
[40] http://www.vmware.com/products/vcloud-suite
[41] http://blogs.enterprisemanagement.com/torstenvolk/2012/08/16/softwaredefined-datacenter-part-1-4-basics
[42] http://blogs.enterprisemanagement.com/torstenvolk/2012/08/22/softwaredefined-datacenter-part-2-core-components
[43] http://blogs.enterprisemanagement.com/torstenvolk/2013/03/25/software-defined-datacenter-part-4-4-today/
[44] http://www.wendelin.io
[45] http://www.snia.org/sites/default/education/tutorials/2012/fall/networking/JosephWhite_Technical%20Overview%20of%20Data%20Center%20Networks.pdf