CYCLONE

Complete Dynamic Multi-cloud Application Management

Project no. 644925

Innovation Action

Co-funded by the Horizon 2020 Framework Programme of the European Union

Call identifier: H2020-ICT-2014-1

Topic: ICT-07-2014 – Advanced Cloud Infrastructures and Services

Start date of project: January 1st, 2015 (36 months duration)

Deliverable D5.1

Functional Specification of the E2E Network Service Model

Due date: 31/07/2015

Submission date: 01/09/2015

Deliverable leader: José Aznar (I2CAT)

Dissemination Level

☒	PU:	Public
☐	PP:	Restricted to other programme participants (including the Commission Services)
☐	RE:	Restricted to a group specified by the consortium (including the Commission Services)
☐	CO:	Confidential, only for members of the consortium (including the Commission Services)

List of Contributors

Participant	Short Name	Contributor(s)
Interoute S.P.A.	IRT	Domenico Gallico, Matteo Biancani
SixSq SARL	SIXSQ	Charles Loomis
QSC Ag	QSC	Maria Kourkouli
Technische Universitaet Berlin	TUB	Mathias Slawik, Begüm İlke Zilci
Fundacio Privada I2CAT, Internet I Innovacio Digital A Catalunya	I2CAT	José Aznar, Isart Canyameres, Eduard Escalona, Javier Fernández
CNRS French Institute of Bioinformatics	CNRS-IFB	Christophe Blanchet, Jean-François Gibrat

Document history

Version	Date	Partners	Description/Comments
0.1	25/05/2015	I2CAT	ToC First version document for open discussion
0.2	02/06/2015	I2CAT	Second ToC version based on previous comments
0.3	18/06/2015	ALL	First round of contributions
0.4	24/06/2015	I2CAT	First integrated version
0.5	13/07/2015	ALL	Second round of contributions
0.6	24/07/2015	I2CAT	Second integrated version
0.7	17/08/2015	I2CAT SixSQ TUB	Third round of contributions
0.8	24/08/2015	I2CAT	Fourth integrated version
0.9	28/08/2015	ALL	Final review
0.9	30/08/2015	I2CAT	Quality review
1.0	31/08/2015	I2CAT	Final version release

Table of Contents

Executive Summary

The primary objective of WP5 is to innovate federated cloud environments by bridging network service capabilities to cloud based services. Thus, besides the control, management and visibility on the Infrastructure as a Service (IaaS) part of the federated infrastructure, tenants also demand to be capable of controlling specific networking aspects of such federation frameworks in coordination with the pure IT resources, enabling added-value functionalities while operating dynamic deployments of complex distributed applications.

This document reports on most relevant networking requirements imposed by Cloud federation environments and presents the OpenNaaS network service management architecture and the functional specification of the End-to-End network service model, to integrate the CYCLONE requirements and features in terms of networking while deploying such complex applications.

1. Making the most of the network in Cloud federated environments

Future Internet (FI) frameworks and architectures are adopting service-oriented approaches in which IT and Network coordination is a must. More specifically, in Cloud federated scenarios, a group of Cloud infrastructure providers are federated and trade their surplus resources amongst each other to gain economies of scale, efficient use of their assets, and expansion of their capabilities, e.g. to overcome resource limitations (scalability), avoid vendor lock-in, ensure availability and recovery, provide with geographic distribution to ensure low latency, regulatory constraints, cost efficiency and energy saving, etc. [Nadjaran]. Thus, this model enables to deliver computing facilities to Service Providers (SPs) using resources of either one infrastructure provider, or combination of different providers.

Such a scenario imposes challenging requirements and essential characteristics that fall on the network aspects of service delivery in terms of service delivery automation, resource management or on-demand creation of network connectivity between Communication Service Providers (CSPs) and Communication Service Consumers (CSCs), within Data Centers (DCs) or between the computing nodes of a CSPs infrastructure. All these capabilities to offer network connectivity as a Service is what in the literature refers to “Network-as-a-Service” (NaaS). NaaS is a business and service delivery model related to network infrastructure for providing network services with dynamic and scalable, yet secure and isolated, access to networks for multiple tenants. So far, Application Service Providers (ASPs) have barely been in control of the network resources, limiting tenants’ flexibility while deploying such applications. Also Network Service Providers (NSPs) become limited, since they experience little control or complete lack of knowledge on the semantics of the application data and on the network requirements that the applications may request as part of the application deployment. Additionally, the variation of network configuration capabilities among cloud platforms entails significant constraints while provisioning inter-cloud network federated resources. Previous limitations highlight the important role that the network plays in cloud federated scenarios. The possibility to specify network requirements while deploying complex applications opens a wide range of possibilities for SPs.

Thus, WP5 is mainly devoted to bring innovation at federated environments by bridging the network service capabilities to cloud based services, extending the IaaS model and integrating the networking aspects and requirements of cloud federations in order to provide enable users with the means to include also network management options in the services’ requests. In CYCLONE, such innovation is introduced by the OpenNaaS [OpenNaaS] service management platform. OpenNaaS is able to provide flexible deployment, operation and customized network services according to applications requirements

In this deliverable, CYCLONE partners provide the functional specification of the OpenNaaS End-to-End (E2E) network service model that allows to model network requirements into network services. The OpenNaaS network model should be flexible enough to procure different designs and services implementations coming from new requirements that applications may impose, but fixed enough so common tools can be built and reused across plugins by other components (in this case CYCLONE components). Starting from the

discussion of network requirements in cloud federation scenarios, it will be explained how the OpenNaaS model can be extended to integrate the requirements imposed by the other CYCLONE components, stakeholders and use cases.

The structure of the document is organized as follows:

- Section 2 reports and specifies a set of cloud federation network requirements retrieved from (1) the analysis of the use cases in Milestone MS11 (*Plan for deployment and testing of selected use cases in Year 1)* [MS11], (2) the specific needs of the other CYCLONE and (3) some additional generic cloud federation requirements coming from CSPs and IaaS providers. The process of requirement gathering and definition will go on for the whole project lifespan, since additional requirements may appear at later stage while identifying new use cases that could produce additional features to be modelled within OpenNaaS.
- Section 3 presents the OpenNaaS network service management platform: the overall architecture, its main building blocks, roles, and interfaces. Moreover, it describes the network model that OpenNaaS uses to define network resources and services and how this may be applied to implement the required networking functionalities and services (the full specification of the required extensions and interfaces that will be developed is part of the MS19 document, due by month 10).
- Finally, Section 4 provides some concluding remarks and next steps towards the adoption of networking services and features while specifying the deployment of complex applications.

2. Network Service requirements

While integrating networking capabilities within the CYCLONE federated framework, several networking features that are desirable to be included as part of the solution have been identified. In a first approach, the network is not intended to be directly handled by CYCLONE end-users (e.g., Application Service Providers) in a stand-alone manner, but as an integrated component to be consumed by other CYCLONE components to either enrich the component itself, or to enhance the capabilities such component is bringing to CYCLONE. Thus, partners have considered both general requirements that any federated environment may entail for the network (and which may not bring too much innovation but is nevertheless required) and novel requirements derived from the specifics of the CYCLONE solution components and CYLONE use cases. More specifically, this chapter covers the identified network service requirements set by:

- The CYCLONE security framework.
- The Cloud application provisioning tools, i.e. Slipstream [Slipstream].
- The IaaS provisioning platforms, with a major focus on StratusLab [StratusLab].
- The CYCLONE use cases (i.e. Bioinformatics' and Energy use cases).

2.1. Generic scenario footprint

Before entering into the details of the network requirements that the different CYCLONE components may impose to gain a better control and management of the network, it is important to have a clear view of the different stakeholders that are somehow involved:

- Application Service developers/providers (**ASPs**) are the end users of CYCLONE. ASPs create the application and specify how the applications should be deployed in the Cloud federation.
- Cloud Application Managers (**CAMs**) carry out the deployment and management of such applications carrying out a comprehensive functional description of the cloud resources required for an application and the configuration of each of those resources. **SlipStream** is reference CAM tool in CYCLONE.
- Cloud Service Providers (**CSPs**) are the managers of the Cloud infrastructure. CSPs deploy cloud resources according to the specifications provided by the application deployment tool. For simplicity in this document we will consider that CSPs are the infrastructure owners, consequently acting as **IaaS providers** too.
- Network Service Providers (**NSPs**) are responsible for network service integration in Cloud federations, aiming at enhancing current applications' performance by offering networking services and network management capabilities to both CSPs and CAMs. The reference Network service framework in CYCLONE is the **OpenNaaS** platform and the different network requirements coming from the rest of CYCLONE components will be implemented in OpenNaaS. OpenNaaS will integrate with different CSPs in order to expose such network services and network management capabilities. To achieve this target, it is required that OpenNaaS is able to interact with CSPs network resources. Thus, in the short-term we will consider the integration of OpenNaaS with **StratusLab** (as it offers APIs and connectors to access networking resources) and will consider the integration with other public and private CSPs in the mid-term.

Figure 1 shows the relation between previously identified stakeholders.

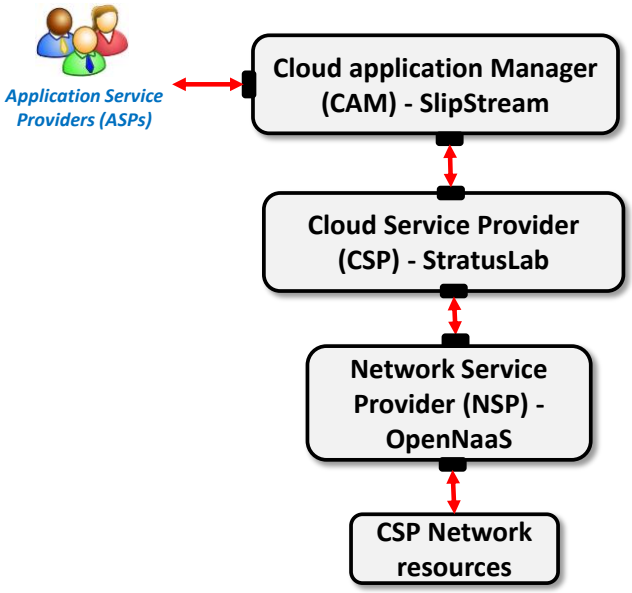

Figure 1: High level view on the CYCLONE components involved in the provisioning of network services

2.2. Network requirements set by the security framework

There are no specific network requirements coming from the security part of CYCLONE but one general feature beneficial to the whole security ecosystem, which is enabling "*distributed logging*": OpenNaaS network service management tool should be able to log its messages to the Elasticsearch [Elasticsearch], Logstash [Logstash] & Kibana [Kibana] (ELK) logging stack, formerly developed by TUB for TRESOR, so that all logs can be uniformly managed.

2.3. Network requirements set by ASPs and CAM tools

2.3.1. Integration with Cloud service management and operation standard models

CAMs (e.g. SlipStream) deploy cloud applications onto one or more IaaS cloud infrastructures and to manage the cloud resources allocated to the running cloud applications. In order to specify applications' needs, these managers generally rely on uniform interoperable management standard interfaces that ensure a proper description of the requirements towards the different IaaS providers. Nevertheless, the considered CYCLONE reference application models (see Deliverable D6.1 – *Complex Application Description Specification*) [D6.1], include very generic options while specifying networking options; thus preventing applications to specify networking capabilities and enhance applications' performance.

This is the case of CIMI (Cloud Infrastructure Management Interface) specification standardized by DMTF and endorsed by ISO/IEC; in its current standard version (v1.0) [CIMI-v1.0] it includes a very limited number of network resources and relationships that enable restricted networking configuration options. The TOSCA model (Topology and Orchestration Specification for Cloud) [TOSCA], does include some options to specify network connectivity; In June 2015, it has been released a first draft version of the TOSCA profile for NFV (Network Function Virtualization) adoption [TOSCA-NFV] which aims to define virtual application topologies, NFV dependencies and relationships, actions to be performed as part of a lifecycle. Nevertheless, it is still a draft version. Also the network type of the OCCI (Open Cloud Computing Interface) [OCCI] model is a L2 networking entity (e.g. a virtual switch) the use of which could be extended using to support L3/L4

capabilities such as TCP/IP etc. It includes only three attributes (802.1q VLAN Identifier, Tag based VLANs and network state (*active*/*inactive*) with two basic instances enabled (*up* and *down*). Finally, Docker containers [DOCKER] allow to configure port connection (in order to interact with services and applications running inside Docker containers), to set bridge instances, to set up IPv4 and IPv6 options and Domain Name System (DNS) configuration. Nevertheless, even if some networking features are gradually being included, previous mentioned standards are in general far from satisfying what applications demand.

In order to satisfy the deployment and management of applications in terms of networking, the same approach proposed in section 4 of Deliverable D6.1 will be followed. The reference CAM in CYCLONE is SlipStream; the SlipStream model already supports full multi-cloud application deployments but does not currently incorporate networking options while describing the deployment of applications. In the pursuit of interoperability, SlipStream is gradually moving from its own RESTful API (Application Programming Interface) to the standardized CIMI API. The rationale for choosing the CIMI model and API is detailed in section 4 of the same D6.1 technical report.

CYCLONE cloud federated environment relies on OpenNaaS network management platform and its integration with CSP platforms to make available networking options to the cloud ASPs by means of the Cloud Applications Managers. Thus, in the short term, the CSP-OpenNaaS integration will adapt the current SlipStream model to incorporate the desired network features and migrate incrementally to the CIMI model in the mid-term. Independently of the model adopted to specify the network resources required by the applications, there are two potential implementation strategies that could be adopted:

A. OpenNaaS could implement previous models and directly accept SlipStream or CIMI resources. In the case of CIMI, it should also be extended to satisfy CYCLONE networking capabilities while deploying Cloud-based applications and lastly integrate them in OpenNaaS as shown in Figure 2.

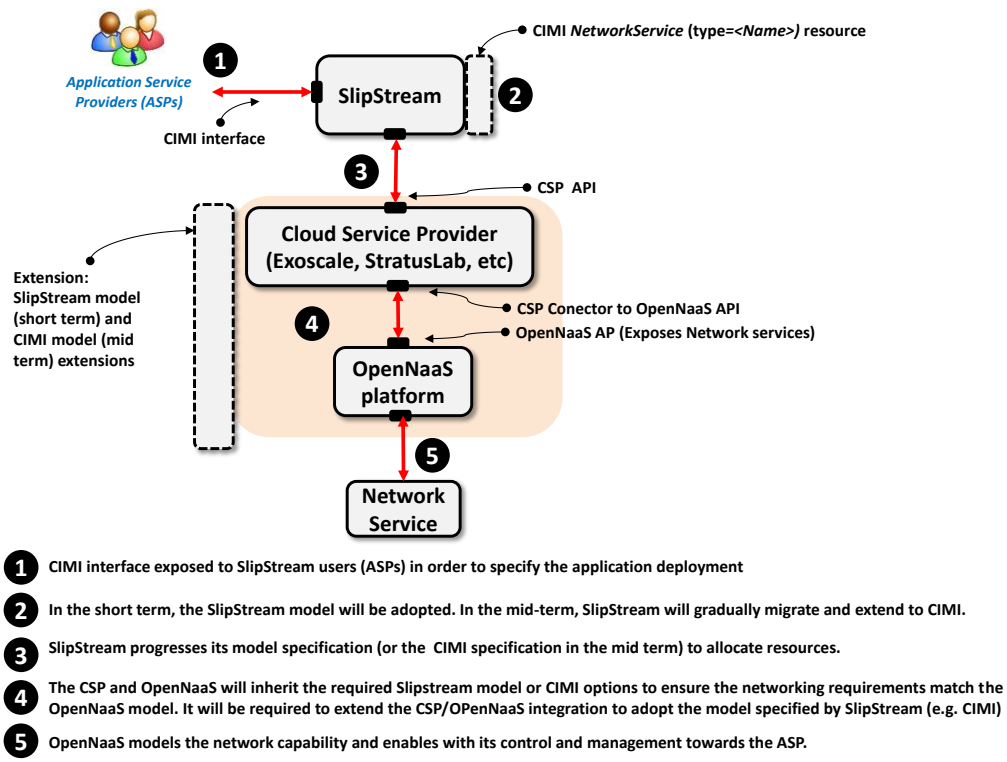

Figure 2: Option "A" - SlipStream forwards the model specification towards the CSP-OpenNaaS.

B. SlipStream Cloud application manager could modify the cloud connector (or create a new one) to adapt to the API of the CSP-OpenNaaS integration. For this option, there is also an open question on whether the OpenNaaS interface will be hidden behind the CSP interface or exposed directly. Figure 3 shows this second alternative.

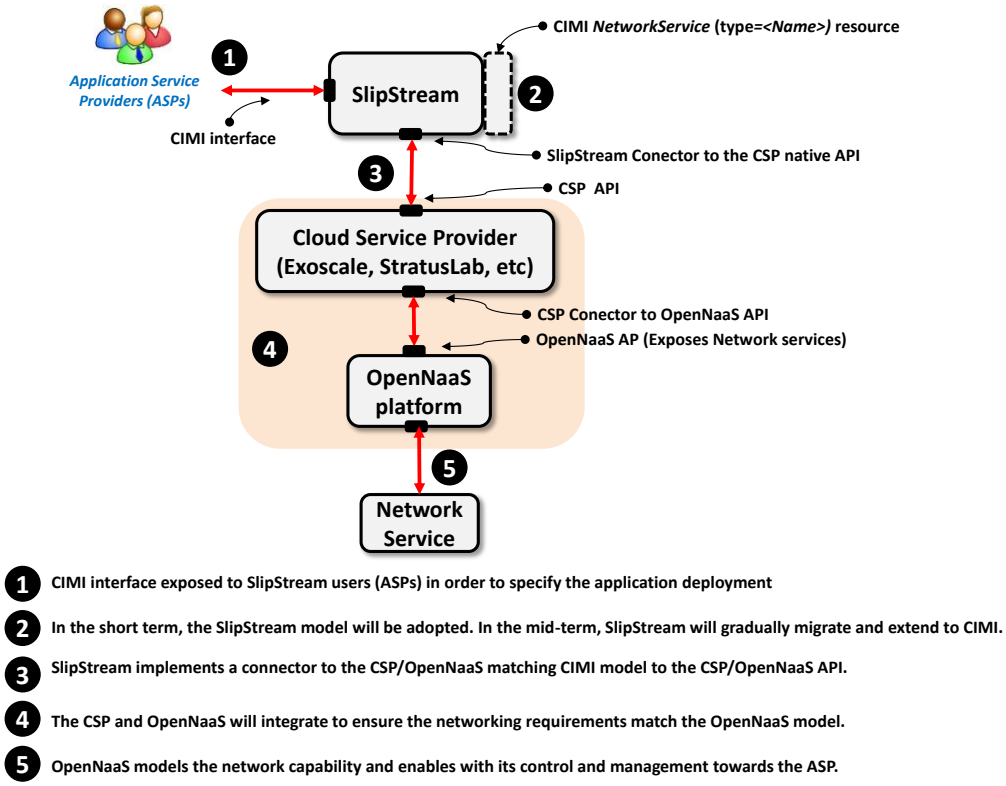

Figure 3: Option B - SlipStream implements a connector to match the CSP-OpenNaaS model.

It will be a matter of choice to select either one or the other. Obviously option “A” is easer from SlipStream point of view if the CSP-OpenNaaS accepts directly the CIMI resource and option “B” is easier from the CSP point of view that would entail SlipStream changing to the CSP-OpenNaaS formats while specifying the application deployment.

OpenNaaS network service management framework will implement specific extensions in its network model so that they can be integrated as part of IaaS CSPs features. We will initially start integrating OpenNaaS with StratusLab, while integrations with other CSPs will be considered in later stages of the project. Additionally, at the application description model level, these modelled network capabilities will be exposed while specifying the description of complex applications in CYCLONE environment.

2.3.2. Firewalling mechanisms

When deploying applications, the current version of SlipStream does not perform firewall management and the firewalling options have to be implemented manually. CSPs usually incorporate firewalling mechanisms at different levels providing also automatic real-time reconfiguration features, triggered by the needs of applications. The deployment of such firewalls is limited to the CSPs or IaaS providers, but the configuration and management can be performed by different stakeholders of the Cloud Federation as shown in Figure 4 and Figure 5.

- The cloud infrastructure (IaaS-level) firewall (marked as “1”): It is controlled by the CSP and configures the rules according to applications’ needs. This firewall is usually open, although a CSP may proactively block access from/to certain addresses, for example in response to an active DDoS (Distributed Denial of Service) attack.
- The application-level firewall (marked as “2”): Rules for this firewall are managed by Cloud Application Developers through the Cloud Service Providers’ APIs (and hence through SlipStream).
- The VM-level firewall (marked as “3”): The Cloud Application Provider/Developer has complete control over the firewall within the Virtual Machines (VMs). CSPs do not control or manage these firewalls explicitly. At this level, the Cloud Application Developer, has full control on the firewalling configuration.

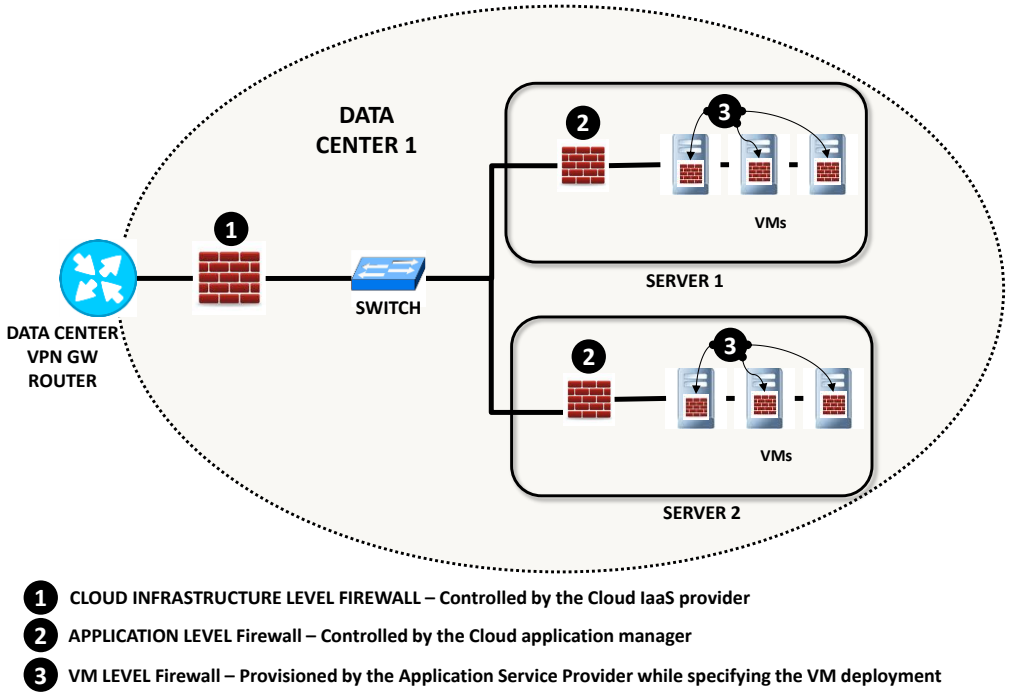

Figure 4: Cloud Infrastructure considered firewalling mechanisms.

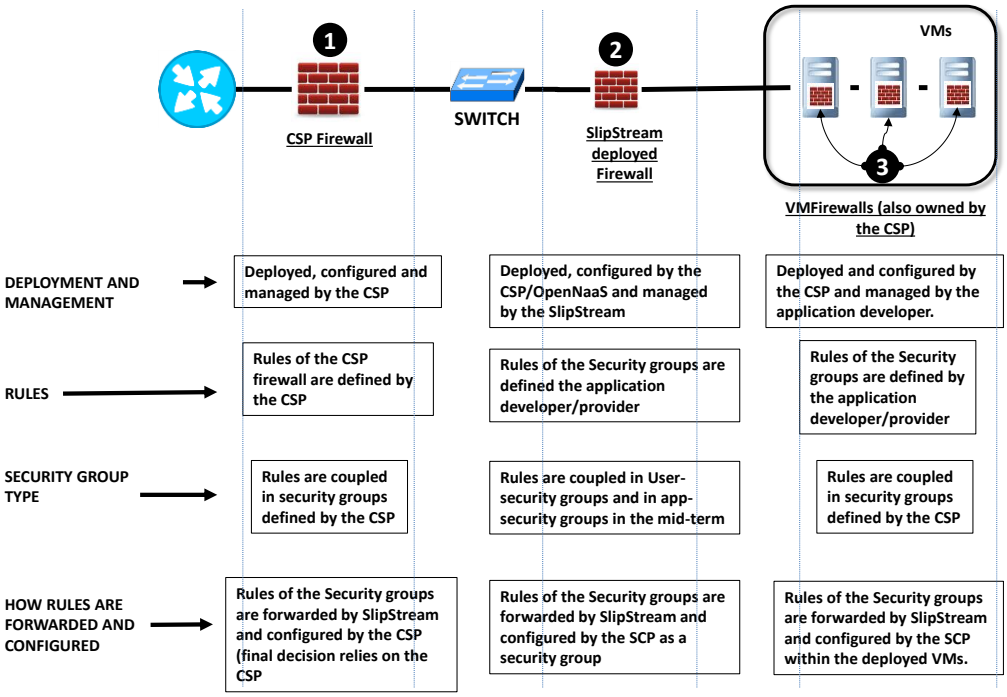

Figure 5: Rules definition, deployment, configuration and management depending on the type of firewall.

SlipStream is not able to exert direct control over firewalls "1" and "3" but only over firewall "2". The objective is therefore to enable Cloud Application Developers through the SlipStream APIs with the automated management of the application-level firewall (firewall 2 in Figure 4).

To this target, OpenNaaS network model will be extended in order to enable the management of this application-level firewall by SlipStream users. Ideally, in order for SlipStream to be able to manage the firewall, it would be desirable to have one separated firewall for each of the running applications, so that each application could own its own firewalling rules. Nevertheless, most cloud platforms limit the number of security groups (standard firewall rules usually refers to protocols, sources (host/ports) and destinations (host/ports)) to a small number. Consequently, the quota would be quickly achieved, greatly complicating deployment and management of applications. In order to enable the firewall management, there have been identified two critical things on the OpenNaaS side:

1. Firstly, in the short term, most realistic approach is creating and using the cloud security groups. OpenNaaS should allow a large number of **user-defined firewalls** (to avoid quota issues) in order to provide an efficient way to assign access to resources on a network. In this situation one single security group covers all user's applications within a cloud and permissions are assigned on a security group basis for each resource and defining also the access level, i.e. Full Control. In the mid-term, OpenNaaS should aim for a **per-application firewall** (security groups) like in other cloud platforms.
2. Secondly, the firewall must be **dynamic**, that is, it must be possible that any modification of the firewall specifications to be applied immediately for running applications. At least, the following properties it would be demanded to be dynamically modified:
    - "*Which ports*"
    - "*From where*"
    - "*To where*"

Concerning the specification of the firewalling mechanism, the same approach described in the previous section will be adopted: we will start from the SlipStream model and gradually migrate towards the CIMI

specification. In the CIMI specification, the firewall options can be found in the *NetworkService resource*, which has been introduced with version 2.0.0 draft [CIMI - v2.0]. This feature associates policies with a given type of network service (i.e. firewall, Virtual Private Network (VPN), Network Address Translation (NAT) and others are specifically named). It will be therefore required to map the *NetworkService resource* to the OpenNaaS model so that the control and management of the firewall "2" can be achieved by SlipStream.

The figure below shows the workflow while specifying, configuring and managing the different firewalls for the particular case of using StratusLab as CSP (for other CSPs it will require a deeper analysis on the options that their APIs enable).

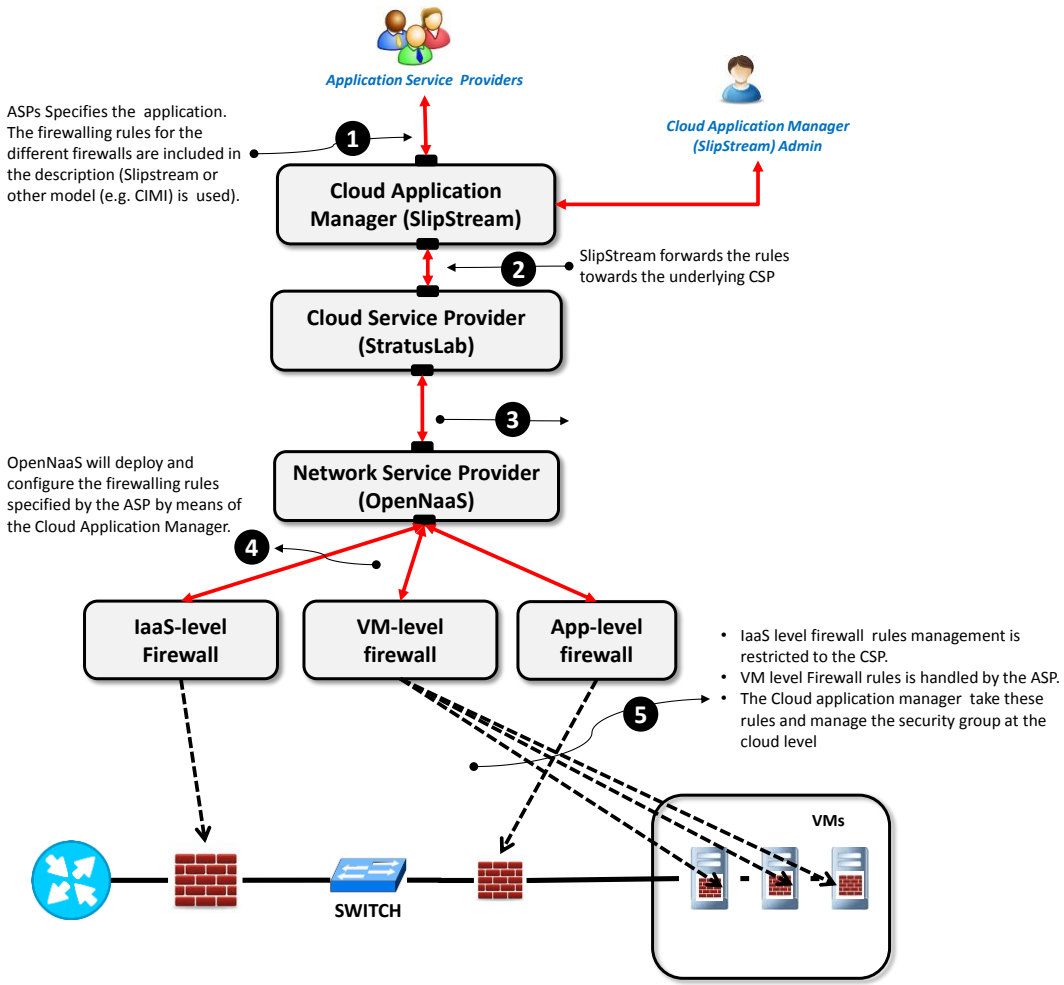

Figure 6: Firewalling rules configuration, deployment and management workflow

A previous discussion on firewalling mechanisms has been conducted for a single CSP working on a single DC for simplicity reasons. Nevertheless, this is also extensible to scenarios in which applications require distributed resources across several IaaS providers. The automated management of per-user and per-application security groups should be deployed at each of the infrastructures that the brokering mechanism has identified as suitable to deploy the application.

2.3.3. VPN automated configuration

In order for users to access their deployed applications in remote cloud facilities, secured access must be guaranteed. Usually this is based on VPN mechanisms; so that the application owner is capable of sending data between two computers or two networks across a shared or public network in a manner that

emulates a point-to-point private link. Thus, the system would set up the VPN tunnel and configure the servers on both ends of the cloud service providers. VPN classification is based on the type of deployment criteria and includes VPN configuration connecting remote clients to a private intranet (Figure 7A) and VPN configuration connecting remote sites across the Internet (Figure 7B). There exists also the possibility to create VPN communities[1] . From the users' point of view while accessing their applications deployed in Cloud federated scenarios, VPN for connecting remote clients to an intranet seems to be the most relevant approach.

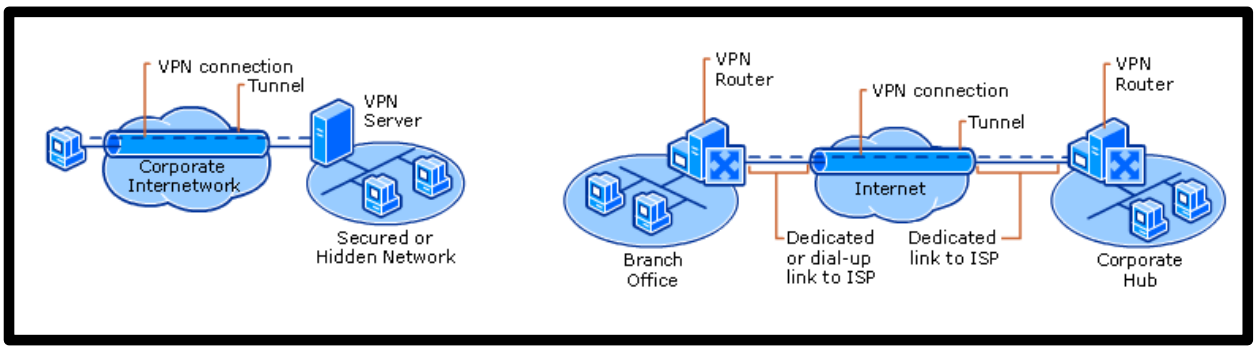

Figure 7: Most extended usage of VPN configurations. (A) Connection of a remote client to a secured intranet. (B) Site to Site VPN Connection between remote branch offices.

VPN server configuration relies on CSPs, which will have to choose between a policy-based VPN or a Router-based VPN solutions depending on the type of VPN level to be deployed (Layer 4/7 VPN (e.g. WebVPN), layer 3 VPN (e.g. IPSec, GREoIPSec) or layer 2 VPN (e.g. L2TP, PPTP, MPPE)). On the other hand, there are a number of configuration options that could be delegated (shared at least) with the Cloud Application Manager in order to gain some management over the VPN service required by a certain application. Some examples of such configuration options on VPN servers are:

- Creation of users and groups of users for remote VPN clients.
- Limit the maximum number of VPN clients allowed.
- Verify and monitor VPN connectivity (Verify that all users have user accounts, monitor remote access usage, authentication attempts, etc.)
- VPN Quarantine control to quarantine each VPN client when it connects in order to ensure that it complies with your security policy
- Configure the number of VPN ports (to add or remove VPN ports)
- Other VPN server specific features to configure the VPN server.

Similar to previous firewall mechanism, SlipStream does not currently support VPNs management. In the context of CYCLONE, it would be desirable:

1. To **enable** SlipStream (through the OpenNaaS Network service platform) to handle specific **VPN configuration options**.
2. To enable **automated VPN server configuration**: SlipStream should ideally be able to configure the server part of a VPN connection specified in the application deployment recipe.

OpenNaaS will integrate with CSP (Starting with StratusLab and studying other CSP platforms in the mid-term) to gain control over specific VPN configuration options and be able to expose them towards the

[1] VPN communities are based on Star and Mesh topologies. In a Mesh community, there are VPN connections between each Security Gateway.

SlipStream application manager. As in previous cases, OpenNaaS will extend the platform creating a connector to access VPN server options as well as the required APIs to expose the management. Figure 8 summarizes the high-level workflow overview on the VPN connections configuration: (1) the ASP specifies the application deployment to SlipStream application manager. (2) The application management includes the request of a VPN remote connection so that the application can be remotely accessed in a secured way. SlipStream matches the request and forwards (among other application requirements) to the CSP the VPN configuration rules to create the VPN. (3) As in previous situations, the CSP will utilize the OpenNaaS network service platform in order to configure, control and manage the networking resources (in this case, specific VPN configuration options). (4) OpenNaaS will model the VPN server as a network resource, handling the options enabled by the CSP VPN GW (Gateway) router of the figure. At the same time, it offers to the SlipStream Admin a management interface to handle the VPN connection management. (5) According to the application specification, a VPN entry point in the VPN server will be created so that the ASP is able to remotely access the application through a VPN client.

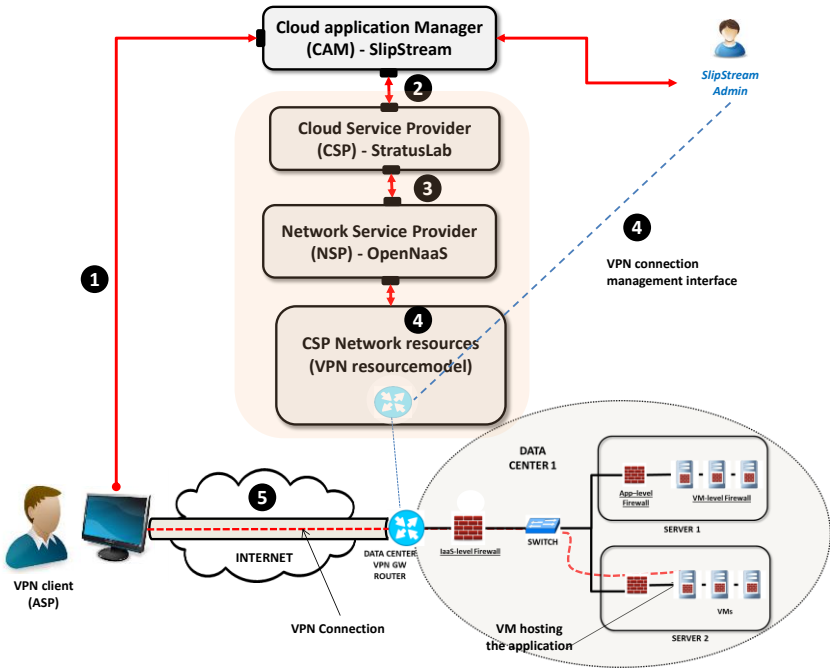

Figure 8: VPN network resource modelling, configuration and management delegation scheme.

As it has been discussed, a first attempt will be carried out while integrating OpenNaaS with StratusLab CSP, analyzing the VPN capabilities that are available to be modelled by OpenNaaS and consequently to be exposed for automated management at the cloud application management level.

2.3.4. Load balancing

In its most basic form, Cloud load balancing provides an organization with the ability to distribute application requests across any number of application deployments located in data centers and through cloud-computing providers. Cloud load balancing enables to extend the application architectural deployment model increasing the number of available choices for organizations when determining from where a given application should be delivered. Besides, IT organizations consider valuable the ability to redirect, split or rate-shape application traffic between DCs, when selecting a cloud provider.

Load balancing capabilities have been largely integrated as part of the suite of Cloud IaaS providers: Amazon web services includes the so-called "*Elastic Load Balancing*" [AWS-LB], Microsoft Azure [Azure-LB] load balancing services also includes a number of features (*e.g.* Automatic reconfiguration on scale

out/down, Direct Port Endpoint Support, etc.). Google Compute Engine offers server-side load balancing [Google-LB] so incoming network traffic can be distributed across multiple virtual machine instances.

Therefore, it is also desirable to include the possibility to **specify load balancing configurations while describing application deployments with SlipStream** in Cloud federated environments. Some potential load balancing configuration options to be handled by OpenNaaS are:

- Configure network Load Balancing host parameters (e.g. priority, IP address, subnet mask, etc.).
- Configure network Load Balancing cluster parameters (e.g. IP address, subnet mask, full Internet name, and cluster operation mode, etc.).
- Configure network Load Balancing manager log settings (to change the explicit permissions on the file to restrict read and write access).
- Specify the load value for an IP address or for a range of IP addresses.
- Select Load balancing algorithms.
- Define the type of Load Balancing for registered services and applications.
- Create/remove/edit network Load Balancing port rules.

Again, the number of load balancing configuration options that can be exposed to the cloud application manager depends on the load balancing configuration options enabled by the CSP. The load balancing options that StratusLab and other CSPs enable will be analised. OpenNaaS will extend the platform creating a connector to access Load Balancer options as well as the required APIs to expose the management. Figure 9 shows the high-level overview and enables Load Balancing configuration options at the Cloud Application Manager level: (1) OpenNaaS should have access to the CSP Load Balancer to be able to model the different load balancing configuration options that the CSP may enable to share with the application manager in terms of configuration. (2) OpenNaaS will model the Load balancer and the allowed options (capabilities). (3) Additionally, OpenNaaS will expose a REST API to enable the SlipStream Cloud Application Manager to handle the Load Balancing options.

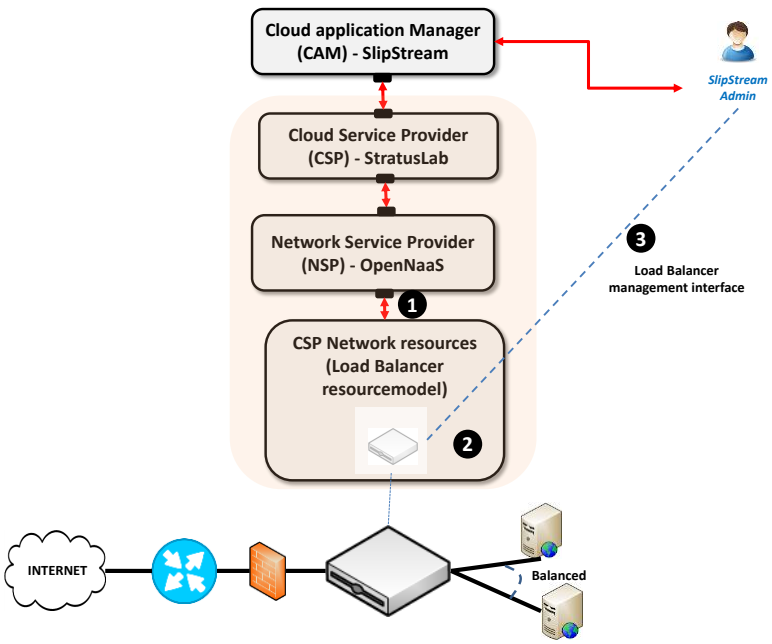

Figure 9: Load Balancer resource modelling, to enable SlipStream with Load Balancing configuration options.

2.3.5. Additional "nice to have" networking requirements while specifying Cloud application deployments

Complimentarily to the previously described networking requirements, some other requirements have been identified that would be desirable to satisfy from the perspective of Application Service Providers and Cloud Application Managers. The following subsections briefly describe each of the remaining requirements that will be considered at later stages of the project (conferring priority to previously stated ones).

2.3.5.1.Guarantee applications' QoS while specifying network configuration.

It would be desirable that Cloud applications could request specific QoS parameters tied to certain pre-established SLAs. In terms of bandwidth (BW), it would be desirable to extend the applications' description with the means to specify the bandwidth to be allocated while being deployed in the Cloud federation. Also, in terms of latency, the network that connects the user and the private or public cloud should be capable of delivering an experience to the user that makes their separation from those resources completely transparent enabling low-latency configuration options. From the applications' point of view, this would consist of a parameter which could be allocated in the application description recipe via application manager.

2.3.5.2. Automated DNS and DHCP services deployment and configuration

These are services that could be automatically configured and deployed as application components assuming that the necessary network connectivity is in place. *Zero Touch provisioning* of these services is also a desirable requirement while deploying the applications.

2.4. Network requirements imposed by IaaS and CSPs.

In the previous section a list of requirements coming from the CYCLONE environment and the integration of its components have been presented in order to achieve suitable complex application deployments and management. In this section, we summarize the requirements retrieved from the CSPs and IaaS providers that participate in CYCLONE.

2.4.1. Network virtualization and network abstraction

IaaS providers periodically get new network equipment supporting their IaaS services. This leads to a number of network resources from different vendors. IaaS owners complain about the large amount of time spent on plenty of operational networking tasks and difficult manual configurations, which in the end turns into a waste of effort and money. By introducing network abstraction, network resource configuration and management could be automated and handled in an easier way, relaxing the required skills and knowledge on network resources (specially in multi-vendor environments) and reducing the effort dedicated to networking tasks. Thus, network abstraction, constitutes a valuable networking CYCLONE requirement.

OpenNaaS allows automated configuration of dynamic network resources in DCs and defines vendor-independent interfaces to access services provided by these resources. This is enabled by using virtualization and abstraction of underlying network technologies.

The OpenNaaS network service platform relies on network virtualization and lightweight network abstraction. Thus, OpenNaaS enables automated configuration of dynamic network resources in DCs and defines vendor-independent interfaces to access the services provided by these resources. The level of abstraction provided by the OpenNaaS API can be customized, limiting (or extending) the range of options to be managed. Figure 10 shows a lightweight representation of some generic physical network resources (lower part of the image) and the virtualized view of such resources (upper part). It highlights that different

virtualized resources (green items) support different features (red items) depending on the degree of abstraction level.

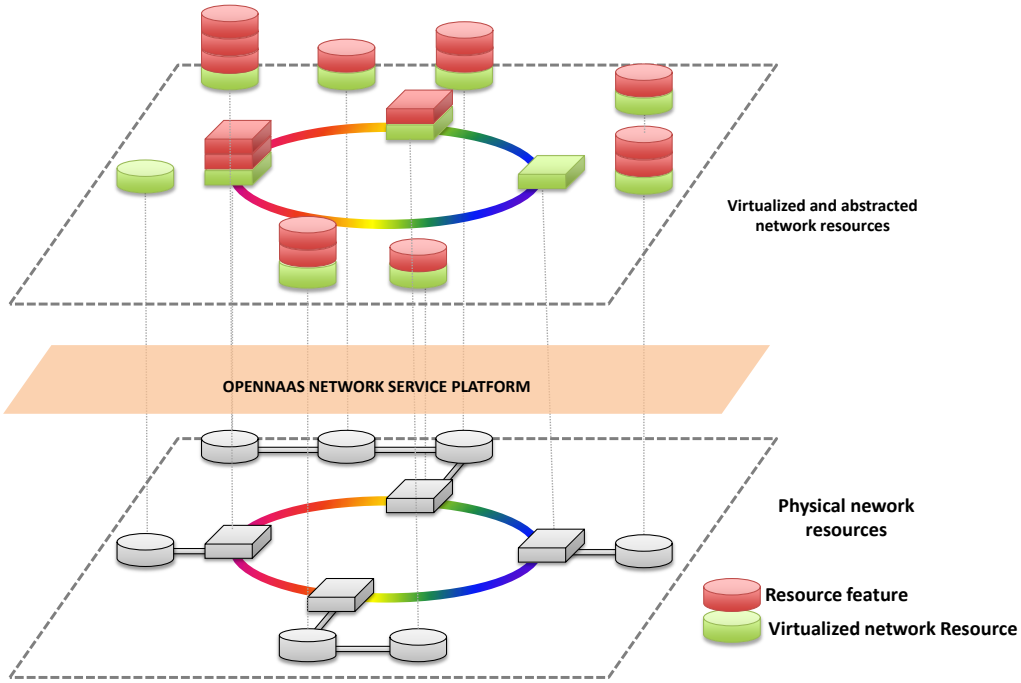

Figure 10: Lightweight representation of OpenNaaS virtualization and abstraction.

2.4.2. Network isolation

The description format should allow the application developer to specify the characteristics of isolated network(s) to be created for a given application. This network isolation may be extended to machines outside of the application, such as a user's workstation. The isolation of network resources can be achieved based on partitioning also improving the efficiency of the network infrastructure utilization by allowing multiple federated cloud users to run their different complex applications on top of their logical infrastructure compositions.

2.4.3. Multi-tenancy support

Highly related to the previous requirement, the possibility to enable different types of tenants over the same network resources is achieved by means of network virtualization and network isolation. Enabling different application deployments utilizing common network resources constitutes an added-value feature to IaaS providers' since it enables a better utilization of system resources. This feature applies not only to intra-DC domains but it is also extensible to inter-DC networks, thus benefiting ISPs and network operators. Figure 11 shows the example of two network overlays running on the same physical resources. Based on virtualization and abstraction mechanisms, it is possible to isolate the network overlays at the time multi-tenancy is enabled. The traffic of the overlay 1 that corresponds to business applications cannot be seen by the administrators of the second overlay running research oriented applications.

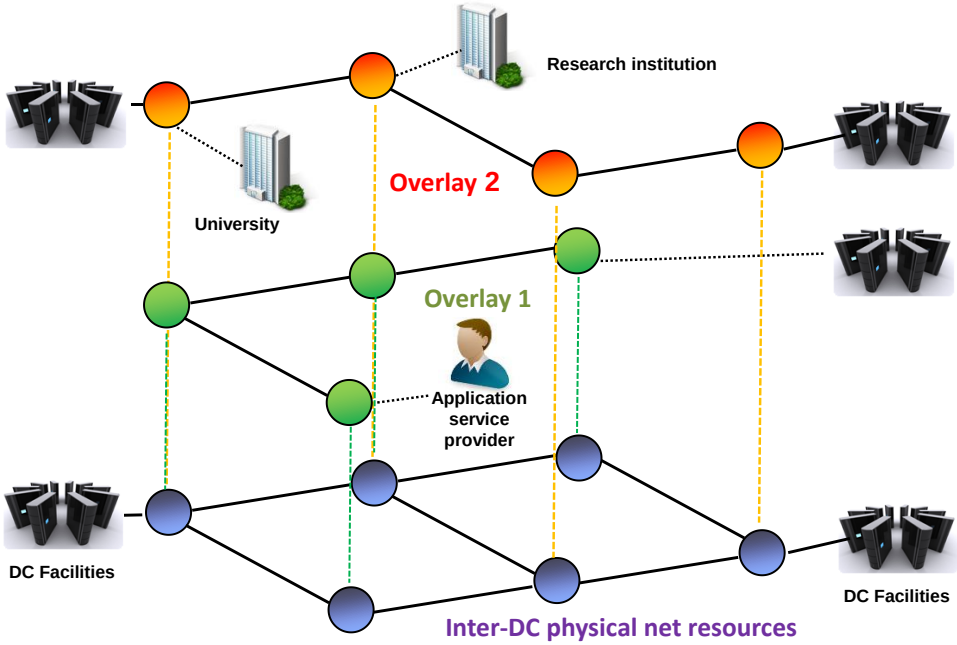

Figure 11: Inter-DC multi-tenant isolated support over the same physical resources.

2.4.4. Dynamic network resource discovery

While handling network resources of a DC, CSP administrators may demand to automate the discovery of new network resources. The utilization of Link Layer Discovery Protocol (LLDP) [LLDP] or other protocols constitute a valuable tool for IaaS providers to be aware of the network resources active in the cloud federation and identify potential cutoffs.

OpenNaaS could bring the automated discovery of the network resources by "reading" the LLDP message interchanges.

2.4.5. Monitoring of network resources

Very close to previous requirements, and in order to ensure network infrastructure reliability, monitoring functions have to be enabled from physical to application level to monitor a high number of service metrics (e.g. network availability and performances, etc.). The IaaS provider has to be able to retrieve data from all the virtual instances that have been deployed. This fine-grained control and monitoring will permit data analytics, long-time planning and short-time re-configuration actions.

2.4.6. Delegation of network functions management

Besides specifying and requesting specific network resources, the network model for Cloud federations should also include the possibility for application owners to modify the performance characteristics (for instance in terms of switching or bandwidth allocation) once the application has been deployed. In this way, the slice(s) of network resources that the IaaS providers have put at a certain application's disposal, could be also modified by such application. This requirement has been already described for some specific components, e.g., Firewall, VPN server, Load balancer, Dynamic Host Configuration Protocol (DHCP) server and DNS network resources in section 2.3. Figure 12 shows two of the management interfaces that OpenNaaS is able to provide (in this case to SlipStream administrator) to enable the management of certain network resources (load balancer on the left and VPN server on the right).

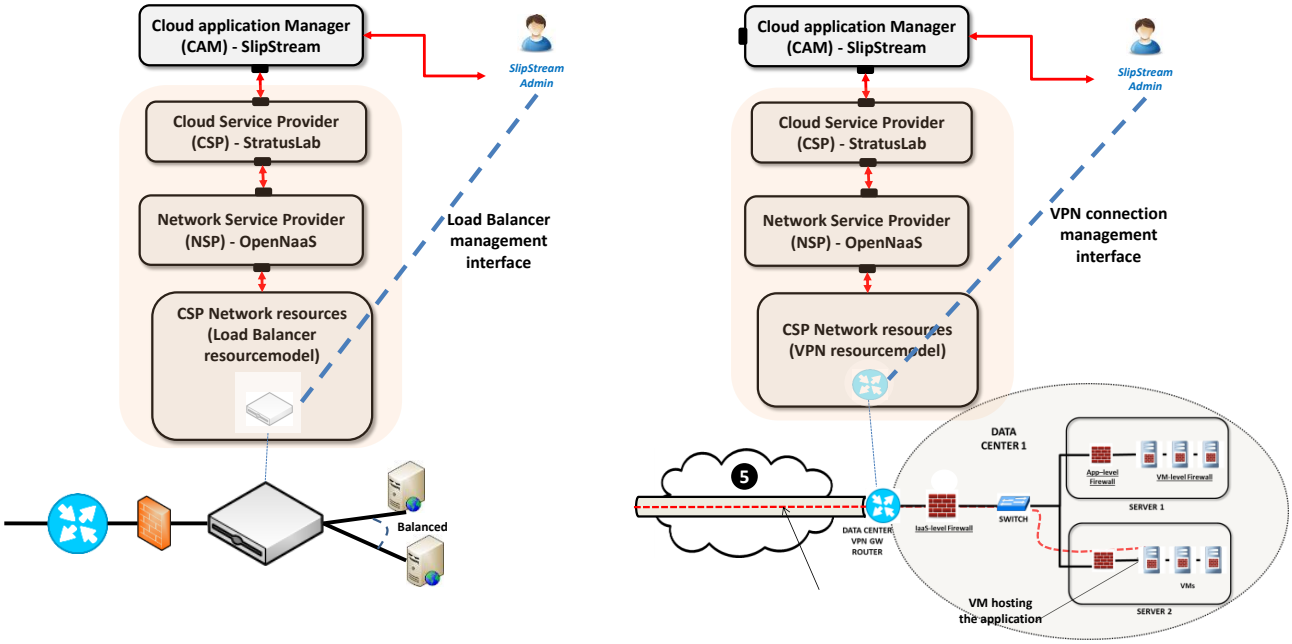

Figure 12: Two examples of interfaces facilitated by OpenNaaS to enable with the management of network resources (load balancing on the left and VPN server on the right).

2.4.7. SDN/OpenFlow support

With the recent adoption of Software Defined Networking (SDN) based technologies within Data Centers, it is also desirable to include the possibility to configure and manage networking resources which operate based on SDN/OpenFlow (OF). For instance, the support of Open virtual switches (OvS) [OvS] could be considered. OvS is an open source OpenFlow capable virtual switch, typically used with hypervisors to interconnect VMs within a host and also different hosts across networks, or any other network resources that implement an OF agent to which OpenNaaS is able to access for management purposes. OvS instances enable to shift all the management of networking functions out of the networking resources into an appropriate network service controller and management platform (e.g., OpenNaaS).

For example, here are some available features of OvS switches that would be interesting to be supported by OpenNaaS:

- VLAN tagging and 802.1q trunking
- LACP protocol
- Port mirroring
- Flow export
- Tunneling (GRE, VXLAN, IPSEC)
- Quality of Service (QoS) control

2.4.10. Network resources elasticity

The IaaS Network resources have to be provisioned in a flexible manner. The Network infrastructure should ideally scale-up or scale-down automatically or on-demand following the user's needs and be able to adapt to the different and dynamic characteristics of the applications running on top of the virtual infrastructures. This procedure should be initiated on-demand, through explicit user's requests, or triggered automatically when some thresholds are exceeded but, in any case, it should not impact on service continuity. By enabling the management of network resources, it would be desirable that the OpenNaaS network service

platform could provide such elasticity, attending to the application specified requirements and established thresholds.

2.4.11. Inter DC connectivity

Although previously mentioned in some other requirements, inter-DC connectivity requires a separate subsection. Federated Cloud services may need to rely on dedicated inter-DC connectivity services that can either based on connection-oriented paradigms and QoS in controlled and deterministic ways (e.g., through IP/VPN, MPLS, WAN, dedicated leased lines, or optical circuits). Dedicated connection-oriented inter-DC connectivity may guarantee the fulfilment of fast service deployment and access despite the location of the target DC. Also, dedicated inter-DC connections can allow matching the KPIs on Recovery Time Objective (RTO) and Recovery Point Objective (RPO) set in case of Disaster Recovery services for the applications running in the IaaS environment. Thus, it is highly desirable to consider also the inter-DC segment of the solution as part of the networking resources to be controlled. Several inter-DC connectivity options are being studied in WP7 in order to interconnect the facilities brought to the project by CYCLONE partners. Based on the inter-DC scenario set up, the possibility to monitor, configure and manage the inter-DC network connectivity services will be analysed.

2.5. Specific Network requirements set by the use cases

The project has identified two flagship applications: an academic cloud platform and associated services for bioinformatics research and a commercial deployment from the energy sector for future energy management. Each of them has been split into 3 micro use cases, and for all these six use cases, different requirements have been defined. Among all of these requirements, several are related to the network and are detailed in this document.

ID	Title	Bioinformatics	Energy	Network
1	End-to-end secure data management	✓	✓	✓
2	VM isolated network	✓		✓
3	VPN connectivity services	✓		✓
4	Multi-clouds distribution of community reference datasets	✓		✓
5	Dynamic network resource allocation	✓	✓	✓
6	Multi-clouds distribution of user data	✓	✓	✓
7	WAN high bandwidth links	✓	✓	✓
8	Guaranteed network performances (QoS)	✓	✓	✓

Table 1 : Network implications coming from the use cases requirements.

2.5.1. End-to-end secure data management

The data must be secure throughout an application to avoid inappropriate access to data in transit or in storage. The securing mechanisms should be embedded with the VM creation process and totally transparent to the life science researchers. The connection between the cloud infrastructure and the user computer need to be secured by default. This requirement highly relates to the requirements presented in section 2.3.2 (firewalling mechanisms) and section 2.3.3 (VPN mechanisms) with regards to the way users may access their applications in a secured way so that it will be addressed with a similar approach.

2.5.2. VM isolated network

The VMs of a user must be deployed in a dedicated and isolated network that could be reached only by its owner, according to its identity and access credentials. This will isolate its own resources (VMs and data) from other users, and also avoid any impact with other external VMs. This requirement is closely related to

the "*Network* isolation" requirements presented in section 2.4.2 and the same partitioning mechanisms facilitated by OpenNaaS may apply, to isolate use cases' dedicated VMs.

2.5.3. VPN connectivity services

A life science researcher needs to access all its VMs in a simple manner through a single point of entry such as VPN services. Such features should be presented to the user ideally through a web service. Clearly this requirement will follow the same approach described in Section 2.3.3.

2.5.4. Multi-clouds distribution of community reference datasets

The deployment of bioinformatics workflows over two or more cloud infrastructures requires that the collections of public reference data used during the treatment are available in all of these clouds. These public datasets are, for example, the European Nucleotide Archive, the Ensemble Genomes resource, the Human Genome version 19 hg19 [hg19] or GRCh37 [GRCh37] and the UniProt (Universal Protein Resource) catalogue of information on proteins.

These datasets are public data that does not need to have security rules associated to its access or transfer. However, they require automatic replication mechanisms to be deployed on several cloud infrastructures and to be updated each time the reference dataset is updated on the reference bioinformatics site. Every user of the bioinformatics appliances in the CYCLONE infrastructure must have full access to the data. This access must be transparent to the VM users and, for compatibly reasons with the bioinformatics tools, realised in a file system mode.

In terms of inter-DCs networks, that means having a high performance network to replicate hundredths of gigabytes and to pass the firewalls without performance loss between the different sites. In terms of local networks, a high-performance and distributed file system should be mounted in all the cloud users VMs.

2.5.5. Dynamic network resource allocation

For enabling the aggregation, calculation and visualization of the bioinformatics and energy applications to the distributed multi-cloud environment, different network resources need to be allocated dynamically according to the requirements of the different steps of the application workflow.

In the Energy domain, the adequate networking capability is needed for multiple connections to the platform to collect data from distributed resources.

2.5.6. Multi-clouds distribution of user data

In the "Cloud virtual pipeline for microbial genomes analysis" use case, deploying the complex bioinformatics application requires the distribution of the user data in a secure way through several cloud infrastructures. The user data can be represented in files and relational (*postgreSQL*) or *noSQL* databases.

User data can represent tenths of gigabytes and needs to be accessed by all the VMs of the deployed complex application, even on several DCs. Regardless if they are flat files or databases (relational, noSQL), data needs to be accessed remotely or replicated. In both cases, that means having a high bandwidth network between DCs. Since we are dealing with user data, the associated security (integrity, confidentiality and logging) must be preserved on the different sites where the data is used. Moreover, the choice of the cloud sites to use for the complex application deployment is also subject to data authorization.

2.5.7. WAN high bandwidth links

Both domains of the use cases require the dynamic allocation of high-bandwidth links between the data producers and the cloud storage. For example, bioinformatics users deal with the collection and efficient analysis of biological data, particularly genomic information from deoxyribonucleic acid (DNA) sequencers. The terabytes of raw data produced by the sequencing centres need a high-bandwidth connection to the

cloud storage. This will enable "*live data processing*" between the sequencers and the computing resources to fully analyse the data as it is produced.

Also, real time capability should be guaranteed in the energy domain due to the need to react to events and changing conditions in real-time. To collect distributed data in the Energy domain and process it in a secure way in several cloud infrastructures high bandwidth is required.

In order to dynamically modify link bandwidth, OpenNaaS should be able to access the configuration features of the services provisioning the connections between the sequencers and the computing resources (for the bioinformatics use case) and the connectivity between the sources of energy Data and the processing facilities (for the energy use case).

2.5.8. Guaranteed network performance (QoS)

Some bioinformatics tools used to analyse genomics data require graphical display with X11 technology, for example the software IGV – Integrative Genomics Viewer. Associated QoS should be enabled to the link between the user LAN and the DC, to satisfy the X11 remote display requirements in terms stability, priority and performances. In the energy domain, a connection that satisfies the QoS parameters in terms of bandwidth and latency is required for the communication with the SCI-platform.

Guaranteeing the QoS (mainly BW and latency) is a requirement coming from the applications' specifications that entail traffic engineering mechanisms at the network level.

2.6. Summary of Network identified requirements:

Origin of the requirement	Item	Requirement	OpenNaaS foreseen extensions	Priority level (High, Medium Low)
CYCLONE framework component (SlipStream)	SlipStream Cloud Application management description model	Integrate with SlipStream application description model (networking features)	Implement the connector or API between SlipStream and OpenNaaS (or between the CSP and OpenNaaS) and integrate.	High
CYCLONE framework component (SlipStream)	Standard application description model adoption: CIMI	Integrate with CIMI model for networking purposes.	Implement the CIMI connector or API between SlipStream and OpenNaaS or between the CSP and OpenNaaS	High
CYCLONE framework component (SlipStream)	Firewalling mechanism	Enable a large quota of per-User security groups	Implement per-User security groups in the Firewall Network service.	High
		Enable a large quota of per-application security groups	Implement per-application security groups in the Firewall Network service	High-Medium
		Enable automatic firewalling capability	Automate application firewall configuration	High
CYCLONE	VPN	Enable automatic VPN	Implement VPN network	High-

framework component (SlipStream)	configuration and management	configuration options	service and REST API. Automate the VPN connections provisioning.	Medium
CYCLONE framework component (SlipStream)	Load balancing configuration and management	Enable load balancing configuration options	Implement VPN network service and REST API.	Medium
CYCLONE framework component (SlipStream)	Guarantee network QoS level	Guarantee QOS SLAs in multi-cloud application deployments	Implement QoS network service according to the application requirements specified	High
CYCLONE framework component (SlipStream)	DHCP configuration and management	Enable automatic DHCP configuration options	Implement DHCP network service and REST API.	Medium-Low
CYCLONE framework component (SlipStream)	DNS configuration and management	Enable automatic DNS configuration options	Implement DNS network service and REST API.	Medium-Low
CYCLONE framework component (TRESOR)	Network logging	OpenNaaS network service management tool should be able to log its messages to the ELK logging stack	Connect OpenNaaS logs to ELK stack	High
CYCLONE framework component (CSPs)	Network virtualization	Virtualize network resources participating of the CYCLONE federation	Extend OpenNaaS network model to implement the connector that enables network resource virtualization and abstraction.	High
CYCLONE framework component (CSPs)	Network abstraction	Abstract network resources participating of the CYCLONE federation to reduce network complexity	Extend OpenNaaS network model to implement the connector that enables network resource virtualization and abstraction.	High
CYCLONE framework component (CSPs)	Network isolation	Ensure applications' traffic isolation while sharing same networking resources	Implement isolation mechanisms to keep applications' traffic isolated	High
CYCLONE framework component (CSPs)	Multi-tenancy support	Enable several applications to share the same networking resources	Enable the support of applications and network services running on top of the same infrastructure and enable with separated tenant control of such applications to the	High

			corresponding tenant	
CYCLONE framework component (CSPs)	Dynamic network discovery	Auto-discovery of networking resources entering/leaving the CYCLONE network infrastructure	Listen LLDP messages and dynamically update the network topology map if a new network resource is set up/disconnected in the CYCLONE network topology	Medium
CYCLONE framework component (CSPs)	Management delegation	Network management and configuration options should be enabled to different CYCLONE stakeholders (mainly CSPs and cloud application managers)	Implement and expose towards other CYCLONE components and stakeholders, the APIs that enable to control and manage network resources options.	High
CYCLONE framework component (CSPs)	SDN/OF support	Support of OF based network resources	Implement an OpenNaaS network connector to handle OF based resources (OF v1.3 preferably)	High
CYCLONE framework component (CSPs)	Elasticity	Enable network elasticity	Extend the network model and implement a service to enable CYCLONE network scale and adapt to match the ever-changing applications' demands.	Medium
CYCLONE framework component (CSPs)	Inter DC connectivity	Provide with network inter-DC service management	Extend OpenNaaS model to control and manage the inter-DC connectivity services decided in WP7 and expose it´s management and monitoring.	Medium
CYCLONE Use cases (Bioinformatics and Energy)	End-to-end secure data management	The connection between the cloud infrastructure and the user computer need to be secured	The firewalling and VPN options previously described will contribute to secure the network part of the CYCLONE solution.	High
CYCLONE Use cases (Bioinformatics)	VM isolated network	The VMs of a user must be deployed in a dedicated and isolated network	Implement isolation mechanisms to keep applications' traffic isolated. VM firewalling mechanisms could also be specified while describing the application (this firewall is not related to network).	High

CYCLONE Use cases (Bioinformatics)	VPN connectivity services	Provide access to all VMs of an user in a simple manner	Implement VPN network service and REST API. Automate the VPN connections provisioning.	High-Medium
CYCLONE Use cases (Bioinformatics)	Multi-clouds distribution of community reference datasets	Provide replication mechanisms for public reference datasets	Implement inter-DC mechanisms and firewalling rules that enable the traffic with high performance.	High-Medium
CYCLONE Use cases (Bioinformatics and Energy)	Dynamic network resource allocation	Provide Dynamic network resource allocation according to the steps of the application workflow	Include OpenNaaS management options to monitor and show network resources utilization and the means to increase/reduce network resources in a dynamic and automated way.	High
CYCLONE Use cases (Bioinformatics and Energy)	Multi-clouds distribution of user data	Distribute the user data in several cloud infrastructures	In terms of networking this entails to ensure the possibility to provide with high inter-DC BW and the security associated to the inter-DC connection. Thus implement flexible inter-DC network services and incorporate specific security items that attain to the network (firewalling mechanisms).	High-Medium
CYCLONE Use cases (Bioinformatics and Energy)	WAN high bandwidth links	Provide dynamic allocation of high-bandwidth links between the data producers and the cloud infrastructures	Extend OpenNaaS model to control and manage the inter-DC connectivity services decided in WP7 in terms of BW.	High
CYCLONE Use cases (Bioinformatics and Energy)	Guaranteed network performances (QoS)	Guarantee the network performances for some applications features (remote display, real-time)	Implement QoS traffic engineering. Extend OpenNaaS to control and configure inter-DC connectivity services.	High-Medium

Table 2: Summary of retrieved requirements and foreseen extensions to the OpenNaaS network model.

2.7. Potential limitations while integrating networking services in CYCLONE environment

Part of CYCLONE innovation consists on integrating networking provisioning as part of the Cloud federated solution. Previous sections have reported and detailed a large number of requirements retrieved from different Cloud federation stakeholders and the particularities of the CYCLONE framework and components. On one hand, the requirements coming from the CYCLONE components are quite specific and the foreseen extensions to the networking platform are enclosed. On the other hand, requirements coming from CSPs and IaaS are quite general and attend to usual DC requirements that also apply to federated Cloud environments.

Nevertheless, a number of potential limitations have been identified that may represent clear showstoppers for integrating networking options and services in the federation.

2.7.1. Access to IaaS and CSPs network resources

This is clearly the most critical item that may limit CYCLONE networking performance. In order for OpenNaaS to model and expose network services, it is fundamental that the tool has access to any network resource (either physical or logical network resource) in order to gain control and management over the provisioned network services. For example, Table 3 presents a router (*type = router*) network resource together with some of its network functionalities that OpenNaaS would likely handle and expose as services to the CYCLONE federation.

Network resource (physical or virtualized)	Network functionality
Router	Chassis
	IP
	GRE
	OSPF
	OSPFv3
	Static route
	VRRP
	Queue
	Power supply
	Power monitoring
	Power management

Table 3: Example of a router network resource and network functionalities.

While working with CYCLONE partners' testbeds, it is expected to be able to gain control on them and be able to control and manage the network functions they present. In the case of private cloud services (Azure, AWS, etc.) the range of networking possibilities that may be exposed while deploying resources as part of the Cloud federation is not so clear. It may even be the case that such networking services are totally hidden and may not be available, thus preventing OpenNaaS to be able to offer the "*network as service*" segment of private Cloud platforms.

2.7.2. Inter-DC network service connectivity

It is part of WP7 to set up the CYCLONE testbed infrastructure and carry out its operation and maintenance. Several network connectivity services may fit while establishing the inter-DC network segments, which is also a fundamental part of the Cloud federation. In case the communication between DC facilities is carried out using the best effort public Internet or establishing static pipelines, it will be difficult for OpenNaaS to provide customised networking services complying with applications' requirements, for instance, in terms of network latency, bandwidth, etc.

Thus, from the networking management point of view, it would be highly desirable that the networking configurations related to inter-DC connectivity services would use endpoints that can be tuned and customized with the OpenNaaS network management tool.

2.7.3. Limited number of OpenNaaS Network model extensions

As already discussed, OpenNaaS network model is flexible enough to enable different designs and network services implementations. Nevertheless, it would not be reasonable trying to implement all the networking features, functionalities and protocols that the CYCLONE federation network's resources may offer. Thus, partners will be limiting the implementation effort of those characteristics and requirements that directly impact federated Cloud environments. Each of the previously reported requirements will be analysed in detail to study their implementation feasibility, considering the use cases workflows defined by WP3 and the CYCLONE testbed set up established in WP7.

3. OpenNaaS: A network management platform for tailoring network services

This section provides an overview of OpenNaaS covering its main building blocks and interfaces. It also describes the current OpenNaaS network abstraction and resource partitioning model which will be extended while implementing additional services, components, drivers and REST APIs in order to address previously defined requirements.

Open Network as a Service (OpenNaaS) is an open-source framework which provides tools for managing the different resources available in any network infrastructure. The software platform has been created in order to offer a neutral tool to the different stakeholders comprising heterogeneous, converged networks. It allows them to contribute and benefit from a common NaaS software-oriented stack for both applications and services. OpenNaaS is based on a lightweight and abstracted operational model which is decoupled from actual vendors' specific details and is flexible enough to accommodate different designs and orientations. In fact, the OpenNaaS framework provides tools to implement the logic of an SDN-like control and management plane on top of the lightweight abstracted model.

OpenNaaS aims to:

- Impact network and service provisioning models overcoming the current mismatch between network, applications and users.
- Provide an integrated approach for network management, enabling the orchestration of composable services and the management of the lifecycle of all resources and network service instances.
- Provide automation, security, service optimisation and monitoring mechanisms to current and future networking services.
- Integrate virtualization and abstraction, hiding network complexity to tenants and enabling the provisioning of customized services over heterogeneous infrastructure resources.
- Create a rich catalog of NaaS and IaaS services end-users can deploy with the click of a button.

OpenNaaS' software features appeal to a wide variety of users:

- Application service providers and software developers gain the ability to easily provide new network services and applications and enlarge a catalogue of network services that end-users can deploy with the click of a button.
- Network service providers may use OpenNaaS as an application provisioning tool to deploy modular network services on top of a network infrastructure and provide it to their clients.
- Network administrators can rely on OpenNaaS for network configuration, management and resource provisioning.
- DevOps practitioners can take advantage of OpenNaaS automation and scaling capabilities to deploy, maintain, and optimize applications deployed on top of the platform.

In order to do so, OpenNaaS is based on an existing open-source platform, Apache Karaf [Karaf]. Karaf is a small OSGi [OSGi] based runtime which provides a lightweight container onto which various components and applications can be deployed. OpenNaaS builds upon the dynamic features of Karaf and deploys its building blocks and particular implementations on top.

OpenNaaS is constantly evolving and it is currently experiencing a renovation of its core architecture. OpenNaaS developers are continuously evolving the tool, releasing new versions matching users' requirements and expectations. The following subsections provide an overview of the OpenNaaS architecture and its building blocks.

3.1. OpenNaaS architecture and main building blocks

OpenNaaS architecture is built around the concepts of **resources**, **services and capabilities**. There are different reusable building blocks, common to all the extensions and abstractions. In essence, a resource represents a manageable unit inside the NaaS concept. Each resource holds a list of capabilities, which are the list of different actions that can be performed upon each resource. OpenNaaS allows creating a software resource (e.g. Devices, Networks, Network Functions, etc.) and managing the offered services.

3.1.1. OpenNaaS base concepts

3.1.1.1. Resource

Resources are the main entities OpenNaaS offers, manages and provisions. A resource is a virtual representation of a physical or virtual network component manageable by OpenNaaS capabilities. Common examples are network devices, such as routers or switches, which are classified as Root Resources inside the framework. However, the main elements of those network devices can also be treated as resources, for example, device ports and network links, among others. These entities are not classified as Root Resources but as (regular) Resources.

3.1.1.2.Capability

Capabilities are directly linked to resources. A capability is a set of services bound to a specific resource, providing new capabilities and functionalities to this resource. Depending on the features it provides, capabilities can be classified in two different groups:

- Management Capability: Provides new resources to the system. Those new resources are managed by the capability and persisted as children of the resource the capability is bound to. For example: a network resource contains an *IRootResourceAdministration* capability, which allows creating child resources inside this network (router, switch, network, …)
- Regular Capability: Provides services to configure the resource it's bound to. For example, the *LinkAdministration* capability allows to specify the source and destination port of a link, which are resources managed by the *IPortManagement* capability.

3.1.1.3.Service

Services and resources are the two main concepts of the OpenNaaS framework. All other components and features of the system are built around them. A service is a software component providing the ability to configure and manage one or multiple resources. Services are executed by the system inside a specific context, and optionally inside a transaction system, offering a way to execute a chain of services, but also managing the set of services as a single one. Examples of services are: *createLink*, *getPorts*, *configureOSPF*, etc.

3.1.2. Core OpenNaaS services

3.1.2.1.BindingManagement

OpenNaaS manages service and resource interdependencies by means of the *BindingManagement*. This component manages the lifecycle of all resources and service instances within the ecosystem, and maintains their state in a Resource tree. Figure 13 shows an example of the service enabled for a *router* resource and its resource tree diagram. The left part of the figure shows the core services (needed for any

type of resource modelling) and the router core services (which map the usual functionalities that comprise a router logic). Since a router must be enabled with interfaces, some basic services (IP assignment and VLAN Tagging) have been also instantiated. The right part of the figure shows the resource dependency tree and the logic of the management while running the router lifecycle.

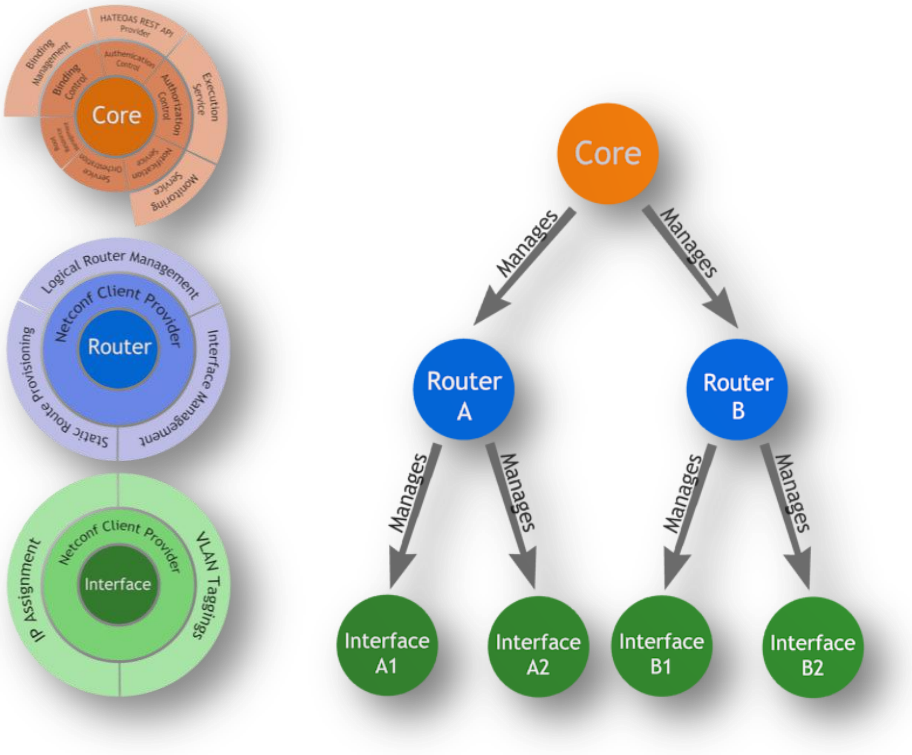

Figure 13: Resource tree diagram

Responsibilities of the *BindingManagement* are:

- Determine which capabilities are bound to a resource.
- Instantiate capability instances and binding them to the resource.
- Monitor resources and their capabilities' lifecycle.
- Interact with resource tree once resources/capabilities are added/ removed.

Whenever a new resource is added to the system, the *BindingManagement* determines which capabilities should be bound to the resource, instantiates them, performs the binding, and communicates to the system which capabilities have been created and bound. Figure 14 shows the binding management process workflow for a generic *router* resource example.

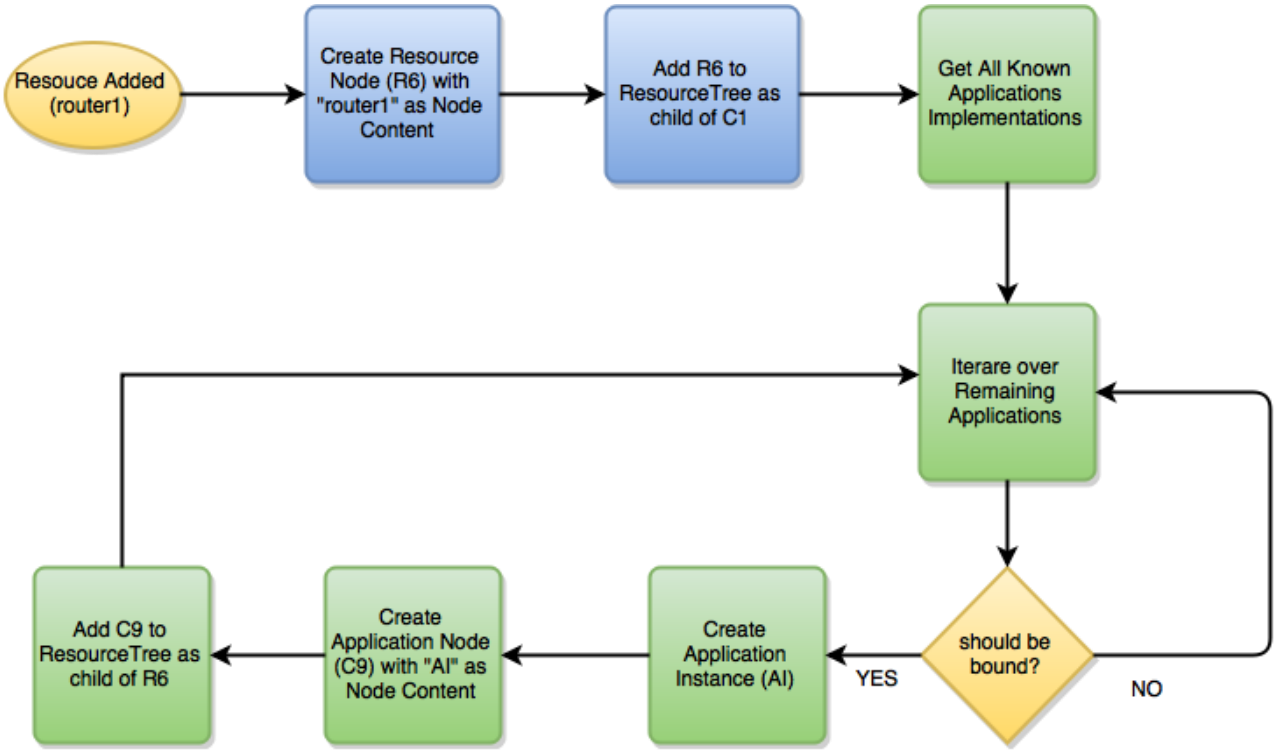

Figure 14: Binding management workflow diagram

The inverse process occurs when an existing resource is removed. A process similar to the depicted one is launched whenever a capability implementation is added to the system. The only change is that iteration goes over existing resources.

3.1.2.2.Dependency resolution and injection service

OpenNaaS provides a mechanism to resolve dependencies between capabilities and their services. Capabilities may define dependencies on other capabilities by means of annotations in their source code. These annotations can be seen as filters used by OpenNaaS to determine, among the available capability instances, which ones are suitable for satisfying the dependencies.

Whenever a capability is instantiated, the dependency resolution mechanism scans the class for dependency annotations and identifies its dependencies. Afterwards a two-side resolution process is launched. This process attempts to resolve the new capability with already existing capabilities, and also to resolve already existing unresolved ones with it, taking care of cycles. As part of the resolution process, a capability which is selected to resolve a particular dependency is injected to the dependent capability, so it can access services in its public interface.

It has to be taken into account that a capability is only operative and accessible to the user once all its dependencies have been satisfied with operative capabilities, that is, after it is completely resolved.

A similar process is launched whenever a capability instance leaves the system, which potentially could cause operative capabilities to become unresolved. In this case, however, a new attempt to resolve it is made to ensure that others may take the place of the newly lost one in case they are ready.

3.1.2.3.Execution service

The execution service is responsible for the execution of any other service in the platform. Its design allows running the execution inside a specific context where services can get data from, and also inside a

transaction. The execution service also offers an observation mechanism that can be used for other components of the platform to react to the execution of particular services.

Any call to a service, whether it is invoked through the remote API or directly in the source code of a capability implementation, is transparently run by the execution service. This is achieved by means of capability proxies automatically build by the system.

3.1.3. Additional services offered by the platform

Besides the services previously described, there are also a number of additional services that are important for the normal OpenNaaS operation.

3.1.3.1.Client Provider

In order to establish a communication between the physical device and the virtual representation inside the OpenNaaS framework, capabilities require a specific client. In order to retrieve it, they should ask for it to the client provider. The client provider offers a mechanism to build and retrieve clients for a specific resource whenever they're required, using the proper protocol (e.g., HTTP, SSH, NetConf) and attaching user credentials if that is the case.

However, the client provider is an internal feature oriented to developers. This process is transparent to the final user, who needs only to provide the device endpoint and their credentials. The capability developer is responsible for taking this information from the resource, choosing the protocol to use, and finally invoking a client provider to get the desired client.

The flexibility of the client provider consists in building any type of client the developer requires, thus uncoupling the underlying protocol from the application level: the same application interface can be used to both generate a REST API client and also to generate a Netconf client. The developer needs only to choose the desired one, according to the resource endpoint.

3.1.3.2.API Provider

The northbound API, which will be explained in following sections, is automatically built by the API Provider module. The API provider exposes the capabilities and the application interfaces through the REST API. Therefore, it analyses the services they declare and builds a REST endpoint for each service, by automatically choosing the URL, the HTTP method, and additional HTTP headers, such as the content-type.

The structure of the northbound API is built from the state of the resource tree. As mentioned in previous sections, the resource tree is a set of nodes defining the framework model state in terms of available resources and capabilities, in which Resources contain Capabilities as child nodes and vice-versa. This semantic creates a resource-capability-resource structure that is reflected in the REST API as well.

After the binding process (see section 3.1.2.1) the API Provider component searches for which new capabilities have been instantiated, but also under which context (i.e., to which resource they have been bound). The new services endpoints are published according to these two characteristics:

- The capability location or “context”: Used to build the endpoint URL, it is related to the capability location in the Resource Tree. The URL is made by adding all parent nodes as parts of the URL path. For example, let's assume that network ”A” contains a Router ”B” resource, and the API provider has to publish the methods of its *IPortManagement* Capability. The capability base URL would look like http://..../networkA/IRootResourceAdministration/routerB/IPortManagement/.
- The service contract: The API distinguishes between getters and setters in order to choose the proper HTTP method (mainly GET, PUT and POST) and the relative path inside the capability context, but also analyses the expected message parameters (query parameters, path parameters, body...) to follow the RESTful style.

These endpoints are published only when capabilities have already been resolved. Figure 15 shows the complete workflow diagram.

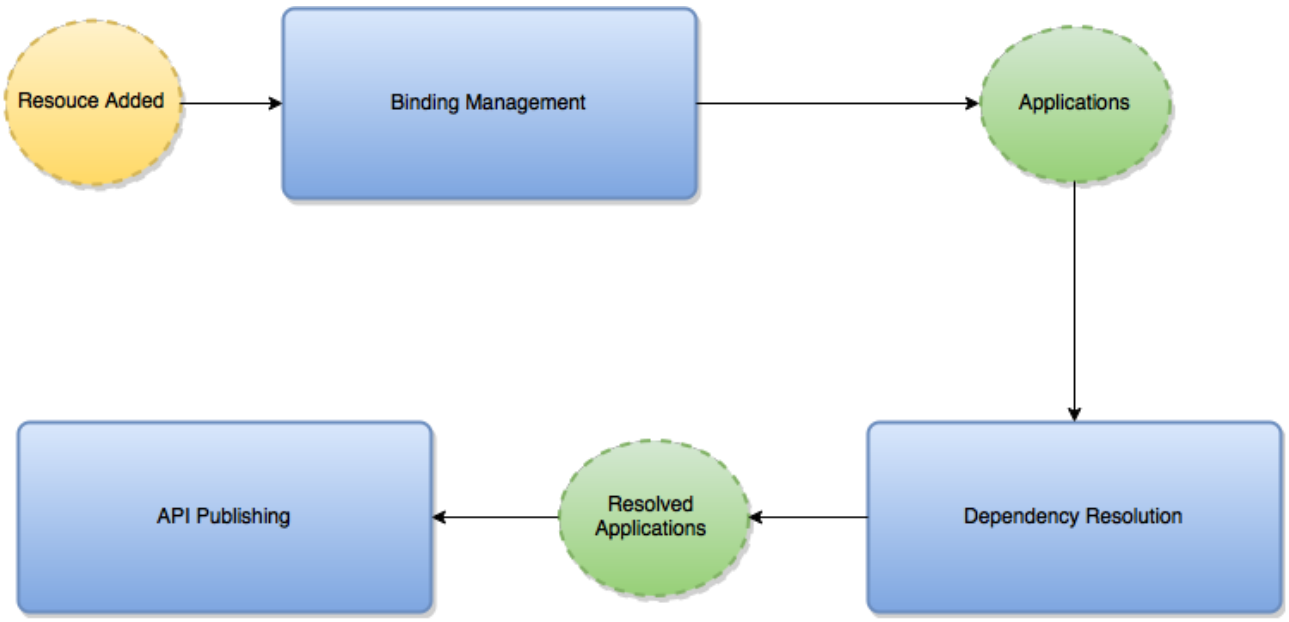

Figure 15: API publishing workflow starting from the addition of a new resource.

3.1.4. Available extensions

OpenNaaS features are provided in form of so-called *extensions*. These are pieces of software that may be deployed in the platform and commonly provide a set of resources, capabilities and services with a particular goal. OpenNaaS already includes a number of extensions available that are detailed in the following:

3.1.4.1.Network extension

To make OpenNaaS a Network Management platform, the first goal was to be able to handle networks as resources. With that in mind, the network extension was created. It provides services to manage a network resource composed of other networking devices (including other networks) with their ports and the links between them.

3.1.4.2.Resource partitioning (slicing) service

This extension provides the feature of virtually cutting resources in independent pieces (slices). It provides a generic abstraction based on the resource partitioning model described in section 4.4. The main idea is to create a virtual replica of the original resource but with a limited subset of *subresources* e.g., only a couple of ports, or a limited range of *vlanId* tags. Particular implementations of the slicing mechanism must be developed for every type of device in order to enforce the application of the generic abstraction to any protocol used to communicate with that device.

3.1.4.3.Other provider specific services

Actually, in order to control networking devices, extensions implementing functional capabilities for those particular network devices must be developed. These implementations are commonly based on a protocol client handling the communication with the remote device and are tailored to the specifics of such remote devices. OpenNaaS currently supports, at least partially, the following types of network services and devices:

Device	Protocol	Supported features
OpenStack Icehouse	REST API over HTTP	Client using JClouds library, retrieval of VMs location and statistics
OpenDayLight Helium	REST API over HTTP	Forwarding rules configuration, retrieval of port statistics
Ethernity Networks CPE	REST API over HTTP	Slicing, raw client for complete API
PT Inovação Sistemas OLT v1.1.0	REST API over HTTP	Slicing, raw client for complete API
UTH Nitos testbed scheduler	REST API over HTTPS	Slicing, network resources reservation
TSON	REST API over HTTP	Slicing

Table 4: Network services and devices currently supported by the OpenNaaS model

3.1.5. Modelling CYCLONE extensions in OpenNaaS

Previous extensions offer a good view on the way network services and resources are modelled within OpenNaaS. In the context of CYCLONE, it will be analyzed how the functionalities and services that the requirements stated in section 2 should be mapped to the OpenNaaS framework, and modelled as extensions to be plugged in and out to the OpenNaaS instance running in the CYCLONE Cloud federation. The generalized way used to represent network resources and services in OpenNaaS makes it possible to model any type of them, as long as the *Client Provider* service is able to establish a communication with the physical device and access its configuration and management stack. This is the way WP5 partners aim to bring CYCLONE network features to application deployment.

3.2. OpenNaaS interfaces

3.2.1. Northbound interface

Each capability is free to define its own API and model. However, OpenNaaS platform encourages developers to make use of the API Provider, detailed in section 3.1.3.2, who offers a mechanism to automatically publish a REST API in a structured way. This is likely to be the followed approach in CYCLONE.

3.2.1.1.Core services API

All core services are mostly internal and their functionality is not exposed in the API as callable services. However, most of them have an API to retrieve information about the state of the component. These APIs are published by means of the already mentioned API Provider, and therefore are exposed following a common pattern.

- *RootResourceProvider* and *RootResourceAdministration* manage *RootResources* in the resource they are present. When they are bound to the core, they administer the set of resources the platform directly manages. *RootResourceProvider* exposes a single service to retrieve created *RootResources* while *RootResourceAdministration* is responsible for creating and removing *RootResources*.
- *ServiceProvider* is exposed as a core service listing all services available in any resource in the system.

Table 5 shows the HTTP methods of the OpenNaaS Northbound API interface.

HTTP method	URL	Body	Response
List *IRootResources* directly managed by the core			
GET	…/mqnaas/IRootResourceProvider	-	<IRootResource> <IRootResourceId>resourceId-1</IRootResourceId> <IRootResourceId>resourceId-2</IRootResourceId> </IRootResource>
Create a new *IRootResource* (a network in this case) directly managed by the core			
PUT	…/mqnaas/IRootResourceAdministration	<ns2:rootResourceDescriptor xmlns:ns2="org.mqnaas"> <specification> <type>NETWORK</type> <model>Internal</model> <version>1.0</version> </specification> </ns2:rootResourceDescriptor>	Network-Internal-1.0-2 The resource id
Delete an existing *IRootResource*) directly managed by the core			
DELETE	…/mqnaas/IRootResourceAdministration/{RESOURCE-ID}	-	-
List services of a resource (grouped by capability)			
GET	http://localhost:9000/mqnaas/IServiceProvider/services?arg0={RESOURCE-ID}	-	<services> <capability name="org.mqnaas.core.api.IAttributeStore"> <service>org.mqnaas.core.impl.AttributeStore:setAttribute</service> <service>org.mqnaas.core.impl.AttributeStore:getAttribute</service> <service>org.mqnaas.core.impl.AttributeStore:getAttributes</service> </capability> <capability name="org.mqnaas.network.api.topology.port.INetworkPortManagement"> <service>org.mqnaas.network.impl.topology.port.NetworkPortManagement:removePort</service> <service>org.mqnaas.network.impl.topology.port.NetworkPortManagement:getPorts</service> <service>org.mqnaas.network.impl.topology.port.NetworkPortManagement:addPort</service> <service>org.mqnaas.network.impl.topology.port.NetworkPortManagement:activate</service> <service>org.mqnaas.network.impl.topology.port.NetworkPortManagement:deactivate</service> </capability> <capability name="org.mqnaas.network.api.topology.link.ILinkManagement"> … </capability> <capability name="org.mqnaas.core.api.IRootResourceProvider"> … </capability> <capability name="org.mqnaas.core.api.IRootResourceAdministration"> … </capability> … </services>

Table 5: HTTP methods of the OpenNaaS Northbound API.

3.2.2. Southbound interface

The southbound API consists of the protocol clients that are instantiated by capability implementations. These clients may open sessions with a remote device. These sessions can be used by capabilities for sending commands to the device, but also by devices for communicating with the platform.

The OpenNaaS platform encourages developers to use *ClientProvider*, which offers a mechanism for capabilities to automatically resolve required clients.

Currently, most of the available clients are REST clients for particular APIs. It is envisioned to enlarge the set of supported devices by having a session based client such as *Netconf* and OVS-DB client protocols CYCLONE, will develop the required APIs so that the different CYCLONE components and services can access the expected network services and resources.

3.3. OpenNaaS network abstraction model

The OpenNaaS abstraction model is based in the triad made of **Resource–Capability–Capability Implementation**. Capabilities may adopt any model that complies with this schema.

The OpenNaaS network abstraction model is made of a very simple set of concepts. It was built specifically to satisfy previous requirements, that is, to display a physical network topology having the degree of detail expected while hiding the networking complexity.

The model defines the following concepts:

- Network: The abstract grouping of all the other elements. Contains *RootResources* and links, accessible through *RootResourceManagement* and *LinkManagement* respectively.
- *RootResource*: A device in the network. Contains ports, accessible through *PortManagement*.
- Port: A network port serving as a link endpoint. Properties are accessible through *AttributeMapper*.
- Link: Represent a connection in the network. Links two ports between them, accessible through *LinkAdministration*.

The following diagram depicts the network model composed by the mentioned resources and their capabilities forming a tree.

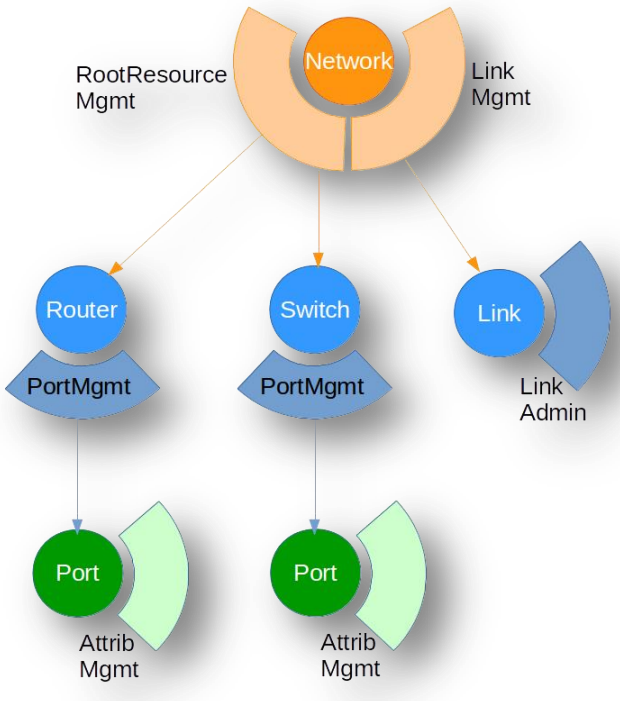

Figure 16: OpenNaaS network abstraction model

3.4. OpenNaaS resource partitioning model

Slicing can be defined as the partition of a physical infrastructure into different isolated virtual infrastructures built upon it. Each of these portions is called a “slice”.

OpenNaaS contains a slicing mechanism that can slice virtual resources as well. Therefore, it defines an abstract model valid for all types of resources, based on three main concepts: slicing units, unit ranges and slice cubes.

Slicing units are also resources inside the OpenNaaS framework. A slicing unit defines into which parts the resource could be divided. For example, a switch could contain “ports” and “VLANs” as slice units, which will be used to create isolated virtual switches. The slicing capacity of a resource is defined by the capacity of each unit, i.e., the unit range. For example, the total number of ports, the maximum number of VLANs, etc.

The last and most complex concept is the slice cube. The slice cube is the n-dimensional space formed by the amount of ranges of the resource's slice units. Let's think in an *n-dimensional* space, where “*n*” is the number of slice units of the resource. Each axis represents a different unit, whose length is determined by its range. Each coordinate of the space constitutes a sliceable element.

For example, given a switch resource, which contains two ports and 4096 VLANs, we would say:

- Switch contains two slice units: *port* and *VLAN*.
- Port unit range would be [0, 1]
- *VLAN* unit range would be [0, 4095]

So, we have a bi-dimensional space, where the x-axis would contain a length of 2 and the y-axis would contain a length of 4096. Each coordinate of this bi-dimensional space can be used to create new slices:

- Slice 1: *port* 0, *VLAN* [496, 511]
- Slice 2: *port* 1, *VLAN* [496, 511]
- Slice 3: *port* 0, *VLAN* [1024, 2047]

Of course, each element of the space could only be part of one slice at once, since it's the main idea behind the isolation concept.

Slices are managed, mainly, by three capabilities.

- The *SliceProvider* capability, which is used to distinguish which resource can be sliced.
- *SliceAdministration* capability, which provides services to define the slice unit and resource ranges.
- *SlicingCapability*, responsible of creating new slices according to *SliceAdministration* information.

The slicing capability analyses the requirements of the virtual resource to be created by checking the state and information provided by the *SliceAdministration* capability. This new resource would contain, additionally to all proper capabilities, all elements of the n-dimensional space selected to be of it. So, taking as reference previous “slice 2”, new virtual resource would contain 1 port, and 16 VLANs.

The isolation of the different slices is specifically made using a proxy that filters all queries and responses according to assigned slicing cubes. The proxy API duplicates the resource API and ensures isolation by checking and translating the elements of the slice cube into the mapped resource. For example, “slice 2” proxy would check that only port-1 is accessible for the final user, and will translate it to the port name of the real device, for example, eth-1.

In order to manage the slicing-related concepts, the API exposes services to manage the different slices of a resource. When a resource is loaded in the platform, its slicing units, their ranges and the appropriate cubes are loaded. This is done in the *SliceProvider* capability. Then, the user may query the

SliceAdministration API for availability of desired cubes and launch a slicing request to the *SlicingCapability* of the resource using selected values as parameters. With that information, the slicing capability creates the resource filtering proxy, configures it with appropriated cubes, marks the cubes as not available in the parent resource and returns the new resource back to the caller. There are also methods to retrieve the slices already created from a parent resource.

Figure 17 depicts the slicing model composed of mentioned resources and capabilities in the form of a tree. Cubes and Ranges are stored as attributes in *SlideAdmin* and *UnitAdmin* capabilities respectively, instead as being resources themselves. This is done for simplicity.

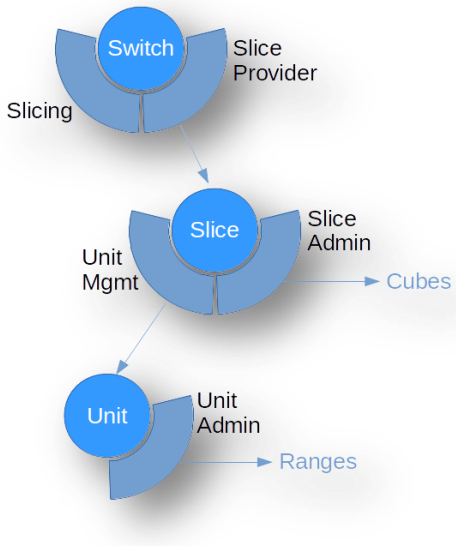

Figure 17: OpenNaaS resource partitioning model

4. Conclusions

The OpenNaaS architecture and its building blocks, abstraction and partitioning models constitute the basis for implementing network functionalities and services required by CYCLONE. It is the aim of OpenNaaS to impact the networking provisioning in Cloud federated scenarios by bringing the management aspects of networking closer to the CYCLONE stakeholders, which will in the end benefit from tailored solutions that can be offered "*as a service*".

As it was stated in the Description of Action document, CYCLONE networking implementation efforts will follow an incremental approach. We also expect that new network requirements will arise in the next project's phases as a consequence of newly identified use cases requirements and networking features adding value to other CYCLONE software.

Besides implementing the required extension to the OpenNaaS model, it will be also required to integrate with CYCLONE principal components i.e. SlipStream application manager (in collaboration with WP6), StratusLab Cloud Service provisioning tool (in collaboration with WP3), and the security framework (in collaboration with WP4) as well as the network resources and inter-DC connectivity services defined in WP7.

The specification of the functional OpenNaaS extensions and interfaces to satisfy the requirements identified in section 2 will be reported in following Deliverables, together with the specification of the network service management features to be implemented as part of the complete E2E federated Cloud solution.

5. References

[AWS-LB] http://aws.amazon.com/elasticloadbalancing/?nc2=h_ls

[Azure-LB] https://azure.microsoft.com/en-us/documentation/articles/load-balancer-overview/

[CIMI – v1.0] http://dmtf.org/sites/default/files/standards/documents/DSP0263_1.0.0.pdf

[CIMI – v2.0] http://dmtf.org/sites/default/files/standards/documents/DSP0263_2.0.0c.pdf

[D6.1] CYCLONE Deliverable D6.1: Complex Application Description Specification (pending of approval). Available at: http://www.cyclone-project.eu/deliverables.html

[DOCKERS] https://docs.docker.com/articles/networking/

[Elasticsearch] https://www.elastic.co/products/elasticsearch

[Google-LB] https://cloud.google.com/compute/docs/load-balancing/#network_load_balancing

[GRCh37] http://www.snpedia.com/index.php/GRCh37

[hg19] http://hgdownload.cse.ucsc.edu/goldenPath/proteinDB/proteins140122/database/

[Karaf] http://karaf.apache.org/

[Kibana] https://www.elastic.co/products/kibana

[LLDP] http://www.ieee802.org/1/files/public/docs2002/lldp-protocol-00.pdf

[Logstash] https://www.elastic.co/products/logstash

[MS11] CYCLONE Milestone MS11: Plan for deployment and testing of selected use cases in Year 1. Confidential CYCLONE document.

[Nadjaran] http://www.cloudbus.org/papers/InterCloud-ACMCS.pdf

[OCCI] https://www.ogf.org/documents/GFD.184.pdf

[OpenNaaS] http://opennaas.org/

[OSGi] http://www.osgi.org/Specifications/HomePage

[OvS] http://openvswitch.org/

[SLIPSTREAM] http://sixsq.com/products/slipstream/

[StratusLab]c http://www.stratuslab.eu/

[TOSCA] http://docs.oasis-open.org/tosca/TOSCA/v1.0/TOSCA-v1.0.pdf

[TOSCA - NFV] http://docs.oasis-open.org/tosca/tosca-nfv/v1.0/csd01/tosca-nfv-v1.0-csd01.pdf

6. Acronyms and Definitions

6.1. Definitions

No specific definitions are required in this document.

6.2. Acronyms

API	Application Programming Interface
ASP	Application Service Provider
BW	BandWidth
CAM	Cloud Application Manager
CIMI	Cloud Infrastructure Management Interface
CSC	Communication Service Consumer
CSP	Cloud Service Provider
DC	Data Center
DDoS	Distributed Denial of Service
DevOps	Development and operations
DHCP	Dynamic Host Configuration Protocol
DMTF	Distributed Management Task Force
DNA	deoxyribonucleic acid
DNS	Domain Name System
E2E	End to End
FI	Future Internet
GRE	Generic Routing Encapsulation
GW	Gateway
HTTP	Hypertext Transfer Protocol
IaaS	Infrastructure as a Service
LLDP	Link Layer Discovery Protocol
MPLS	Multiprotocol Label Switching
NaaS	Network as a Service
NAT	Network address translation
NetCONF	Network Configuration Protocol
NFV	Network Function Virtualization
NSP	Network Service Provider
OCCI	Open Cloud Computing Interface
OF	OpenFlow
OSGi	Open Services Gateway initiative
OVS	Open virtual Switch
QoS	Quality of Service
REST	Representational State Transfer
RPO	Recovery Point Objective
RTO	Recovery Time Objective
SDN	Software Defined Networking
SQL	Structured Query Language

SSH	Secure SHell
SP	Service Provider
TOSCA	Topology and Orchestration Specification for Cloud Applications
UniProt	Universal Protein Resource
VLAN	Virtual LAN
VPN	Virtual Private Network
VM	Virtual Machine